1 Inhaltsverzeichnis

2 Einführung

Die Texturadvektion beschreibt den Transport von Bildern oder Dichtefeldern mithilfe eines Vektorfeldes. Dies ist ein essentieller Bestandteil der modernen Computergrafik und Animation. Hierbei wird die Texturadvektion zur Bereicherung von visuellen Details oder zum Transport von physikalischen Daten, wie zum Beispiel Rauchdichte oder Temperatur, verwendet. Die meisten aktuellen Verfahren basieren auf der wiederholten Interpolation von bereits berechneten Daten. Hierdurch wird ein sich wiederholender Tiefpassfilter implementiert, so dass hochfrequente Details ungewollt ausgefiltert werden.

In dieser Diplomarbeit wird ein Vergleich verschiedener echtzeitfähiger Verfahren durchgeführt. Es werden Verfahren untersucht, die die Genauigkeit bei der numerischen Zeitintegration erhöhen. Dies sind beispielsweise das Runge-Kutta Verfahren vierter Ordnung und das Leap-Frog Verfahren. Außerdem werden verschiedene Ansätze, um die numerische Abweichung von einem Integrationsschritt zum nächsten zu minimieren, analysiert. Hierzu wird speziell das BFECC (Back and Forth Error Compensation and Correction) Verfahren betrachtet. Neben den rückwärtsgerichteten Verfahren werden auch vorwärts gerichtete Verfahren betrachtet. Speziell das Verfahren, das in der Arbeit (Qizhi, Neyret, Bruneton, & Holzschuch, 2010) beschrieben wird, wird näher untersucht.

Alle hier untersuchten Verfahren können auf der GPU implementiert werden und sind somit auch echtzeitfähig. Um eine vergleichbare Analyse der einzelnen Verfahren zu ermöglichen, sind die Verfahren hier jedoch auf der CPU implementiert und analysiert.

Die einzelnen Verfahren werden anhand optischer Kriterien und unter Performancegesichtspunkten untersucht. Die optischen Analysen eines Verfahrens werden mithilfe von Differenzbildern zu einer Referenzimplementierung, dem optischen Fluss innerhalb eines Bildes, sowie dem Erhalt des Spektrums durchgeführt. Zusätzlich werden die Verfahren auf ihre Reversibilität untersucht.

Die Untersuchung und der Vergleich der Verfahren werden mittels eines speziell hierfür entwickelten Programms durchgeführt. Das Programm beherrscht sämtliche betrachteten Verfahren und kann diese auf beliebige Bilder und Vektorfelder anwenden. Hierbei können die Unterschiede zwischen den einzelnen Verfahren verdeutlicht werden. Die einzelnen Analysemethoden werden auch mit diesem Programm durchgeführt.

Ziel dieser Diplomarbeit ist es, einen Vergleich aller Verfahren aufzustellen. Dabei werden Verfahren höherer Ordnung mit Verfahren niedrigerer Ordnung verglichen. Außerdem wird untersucht, ob eine Kombination aus verschiedenen Ansätzen zur Verbesserung der Bildqualität erfolgreich ist. Kombinationsmöglichkeiten sind beispielsweise ein Verfahren zur Genauigkeitserhöhung bei der numerischen Zeitintegration, sowie ein Verfahren, das die Abweichung zwischen zwei Integrationsschritten minimiert.

3 Related Work

Das Stabilitätsproblem, das in der Arbeit (Foster & Metaxas, Realistic Animation of Liquids, 1996) entstand, wurde mit dem Semi-Lagrange Verfahren in der Arbeit (Stam, 1999) behoben. Diese Methode ist bei der Simulation von nicht komprimierbaren Gasen, wie Rauch, weit verbreitet. Die Anwendung mit Rauch wurde in der Arbeit (Fedkiw, Stam, & Jensen, 2001) vorgestellt. Diese Methode wird auch bei weiteren einfachen Anwendungen wie in der Arbeit (Foster & Fedkiw, Practical animation of liquids, 2001) oder (Enright, Marschner, & Fedkiw, 2002) verwendet.

Das Semi-Lagrange Verfahren aus der Arbeit (Stam, 1999) erzeugt allerdinge große Fehler, da es nur eindimensional arbeitet. Diese Fehler können verringert werden, indem die Dimension erhöht wird. Dies wurde in der Arbeit (Song, Shin, & Ko, 2005) mithilfe der CIP Methode (Takahashi, et al., 2003) durchgeführt.

Im Gegensatz dazu ist die BFECC Methode (ByungMoon, Yingjie, Ignacio, & Jarek, 2005) einfach zu implementieren und zu verwenden. Diese Methode funktioniert auch bei Rauchsimulationen sehr gut. Eine Vereinfachung ist in (Selle, Fedkiw, ByungMoon, Yingjie, & Jarek, June 2008) beschrieben. Hierbei wird der letzte Schritt durch einen vorherigen, schon berechneten Schritt ersetzt.

Eine mathematische Zusammenfassung der Simulation ist in dem Kurs (Bridson & Müller-Fischer, 2007) zu finden.

Eine Methode, die sowohl das Spektrum als auch den optischen Fluss erhält, wurde in der Arbeit (Qizhi, Neyret, Bruneton, & Holzschuch, 2010) vorgestellt. In dieser Methode wird ein Partikelfeld advektiert. Partikelfelder wurden in der Arbeit (Reeves, 1983) vorgestellt und beschreiben einen Weg, wie feine Details zu einer Animation oder einem Modell hinzugefügt werden können. Ein Beispiel hierzu ist eine Explosion. Seitdem werden Partikelsysteme in der Computergrafik oft bei Spielen oder Spezialeffekten eingesetzt. Dieses Partikelfeld kann schnell mit der Methode aus der Arbeit (Dunbar & Humphreys, 2006) erstellt werden. Es wird verwendet, um das gesamte Bild mit einer Gleichverteilung der Partikel einzudecken.

4 Grundlagen

Für die Advektion werden zwei grundlegende Eingangsdaten benötigt. Auf diesen wird anschließend die Advektionsgleichung möglichst genau gelöst. Bei den Eingangsdaten handelt es sich um ein Vektorfeld in Kombination mit beliebigen Daten. Auf diesen Daten soll die Advektionsgleichung mithilfe des Vektorfelds gelöst werden.

4.1 Formale Definition der Eingangsdaten

4.1.1 Vektorfeld

Ein Vektorfeld ist ein Feld mit einer definierten Größe. Im Allgemeinen ist es ein kontinuierliches Feld, in dem an jeder Stelle die Geschwindigkeit v ablesbar ist. Da dies jedoch nur schwer angegeben werden kann, wird ein diskretes Vektorfeld verwendet. In diesem ist die Geschwindigkeit v an regelmäßigen Stellen gegeben. Die Geschwindigkeit wird mit einem Vektor angegeben. In Abbildung 4-1 sind Visualisierungen solcher Vektorfelder zu sehen.

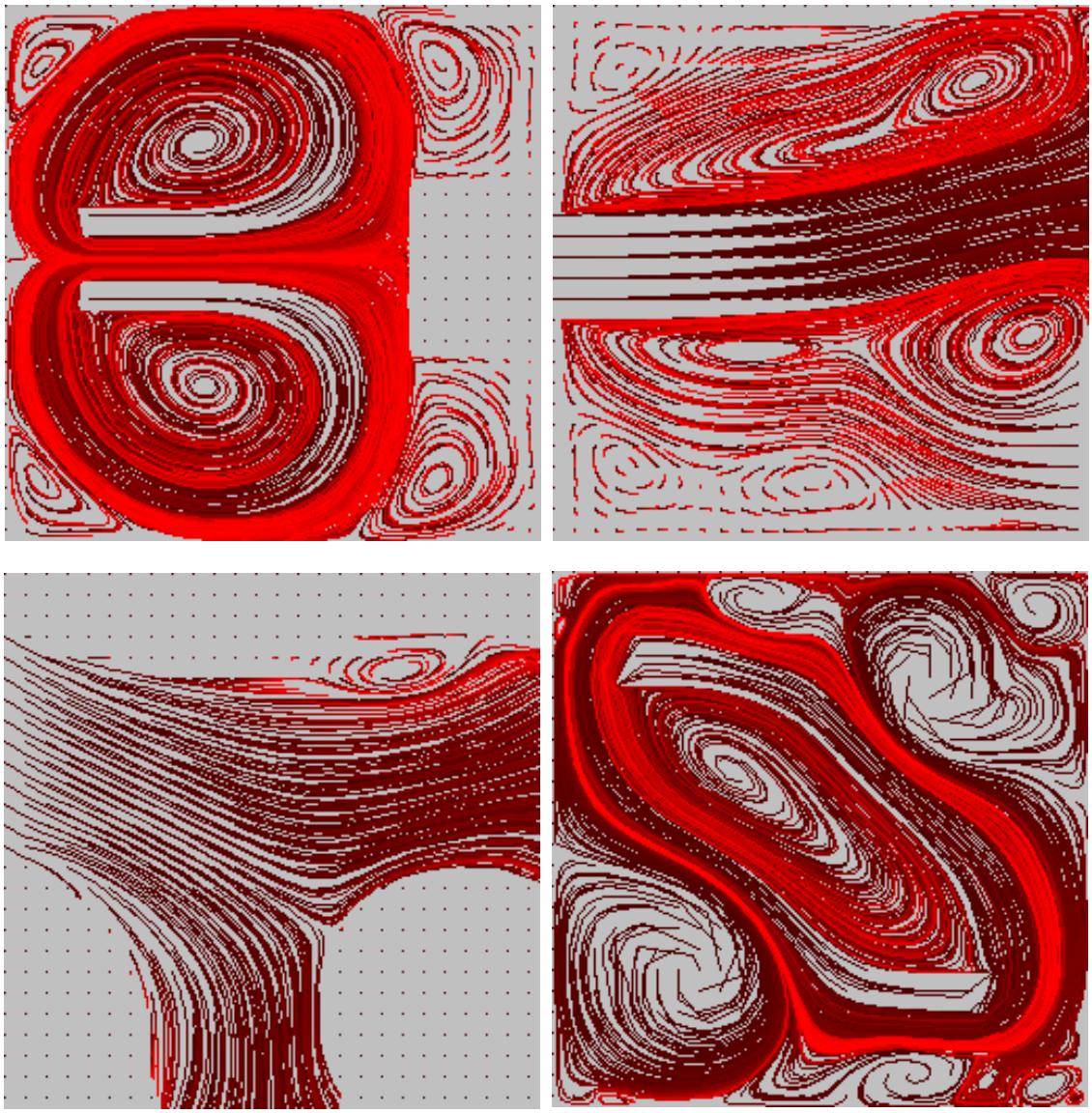


Abbildung 4-1: Visualisierung von Vektorfeldern. Links oben: box1.dat. Rechts oben: high_viscosity.dat. Links unten: bifurcation.dat. Rechts unten: box3.dat.

Die hier verwendeten Vektorfelder sind alle divergenzfrei. Sie entstammen also aus nicht kompressiblen Medien.

Die Vektorfelder können wie folgt formalisiert werden:

$$v(x) = V(x)$$

x ist hierbei die Stelle, an der das Vektorfeld ausgewertet werden soll. x kann jede beliebige Dimension annehmen, solange das Vektorfeld die gleiche Dimension hat. Typischerweise ist dies entweder eindimensional bei Funktionen oder zweidimensional bei Texturen.

V ist die mathematische Abbildung von $R^n \rightarrow R^n$, wobei n die Dimension der Daten ist.

4.1.2 Daten

Die Eingabedaten werden wie folgt formalisiert:

$$T(x) = (a_0(x), a_1(x), a_2(x), \dots, a_j(x), \dots)$$

x ist hierbei die Stelle, an der die Daten ausgewertet werden sollen. x kann wieder jede beliebige Dimension annehmen, solange die Daten die gleiche Dimension haben. Die Daten sollten die gleiche Dimension wie das verwendete Vektorfeld besitzen. Ansonsten muss die Position beim Lösen der Advektionsgleichung in die Dimension des Vektorfelds konvertiert werden.

a_j ist der Kanal der Daten. Diese sind typischerweise r-, g-, b-Farbkomponenten für Bilder, physikalische Größen, wie Dichte oder Temperatur, oder Funktionen, wie beispielsweise eine Rauschfunktion.

Zur Vereinfachung werden die Datenkanäle $a_0, a_1, a_2, \dots, a_j$ in R zusammengefasst. Es folgt also Formel 4-1.

Formel 4-1: Formale Definition der Eingangsdaten.

$$T = (R)$$

4.2 Advektionsgleichung

Um die Advektion durchzuführen, muss die Advektionsgleichung, die in Formel 4-2 gegeben ist, möglichst genau gelöst werden.

Formel 4-2: Advektionsgleichung

$$\frac{\partial \psi}{\partial t} + u(\nabla \psi) = 0$$

Unter der Bedingung, dass $\nabla u = 0$ ist, ist das Vektorfeld divergenzfrei.

u ist das Vektorfeld, ∇ ist der divergence-Operator und ψ ist ein skalarer Wert, der advektiert werden soll. Betrachtet man den divergence-Operator auf einem kartesischen Koordinatensystem und das Vektorfeld u die Komponenten u, v und w in x-, y- und z-Richtung hat, so ergibt sich Formel 4-3.

Formel 4-3: Advektionsoperator auf einem kartesischen Koordinatensystem.

$$u\nabla = u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y} + w\frac{\partial}{\partial z}$$

Im eindimensionalen Raum vereinfacht sich die Advektionsgleichung in Formel 4-2 zu Formel 4-4.

Formel 4-4: Eindimensionale Advektionsgleichung.

$$\frac{\partial \psi}{\partial t} + u\frac{\partial \psi}{\partial x} = 0$$

Um nun eine Advektion auf den Eingangsdaten durchzuführen, muss diese Advektionsgleichung gelöst werden. Es muss also eine Transformation von T nach T' durchgeführt werden. T' enthält die transformierten Ausgangsdaten. Wie in Formel 4-5 zu sehen ist, erfolgt eine parallele Transformation aller Kanäle der Eingangsdaten. Es wird also R nach R' transformiert.

Formel 4-5: Formale Definition der Ausgangsdaten.

$$T' = F(R')$$

Wenn die Advektionsgleichung exakt gelöst wird, dann entsteht ein perfekt advektiertes Bild. Dies ist Ziel der Texturadvektion. Da jedoch die Eingangsdaten nur an diskreten Punkten gegeben sind und die Advektionsgleichung numerisch gelöst wird, entstehen Fehler. Diese Fehler sind in Formel 4-6 dargestellt.

Formel 4-6: Advektionsgleichung mit Fehler.

$$\frac{\partial \psi}{\partial t} + u(\nabla \psi) = C$$

Die Fehler, die mit C bezeichnet werden, verursachen Fehler bei den Ausgangsdaten. Falls C ein Diffusionsterm zweiter Ableitung ist, äußert sich dies durch eine Unschärfe in den Daten. Wenn ein Verfahren iterativ arbeitet und die einmal berechneten Daten für den nächsten Schritt als Eingangsdaten verwendet werden, dann potenziert sich dieser Fehler. Eine advektierte Textur würde hierdurch zu einem mittleren Farbwert tendieren. Es wird ein Tiefpass auf den Daten angewandt.

4.3 Gegenüberstellung der Lagrang'schen und der Euler'schen Sichtweise

In der Lagrang'schen Sichtweise wird das gesamte Feld als Partikelsystem betrachtet. Jeder Punkt in dem Feld wird als eigenständiges Partikel mit einer Position x und einer Geschwindigkeit v gesehen. Die einzelnen Partikel werden anschließend mithilfe des Vektorfelds bewegt. Eine Modifikation hiervon ist die Semi-Lagrang'sche Sichtweise. In dieser Sichtweise werden die Partikel nur einen Schritt lang betrachtet.

Die Euler'sche Sichtweise verwendet im Gegensatz zur Lagrang'schen Sichtweise einen anderen Ansatz. Bei ihr wird die Veränderung der Werte über die Zeit an fixen Punkten betrachtet. Die fixen Punkte werden normalerweise in einem Gitter angeordnet.

Eine genaue Gegenüberstellung beider Sichtweisen ist in Kapitel 1.3 der Arbeit (Bridson & Müller-Fischer, 2007) zu finden.

5 Programm

Um die einzelnen Verfahren zu vergleichen wurde ein Programm namens AdvecUS erstellt. Ein Screenshot ist in Abbildung 5-1 zu sehen. AdvecUS bietet eine einfache Methode die einzelnen Verfahren zu steuern und eine direkte Ausgabe der berechneten Resultate zu bekommen. Die einzelnen Verfahren werden hierbei im Zweidimensionalen auf Texturen berechnet.

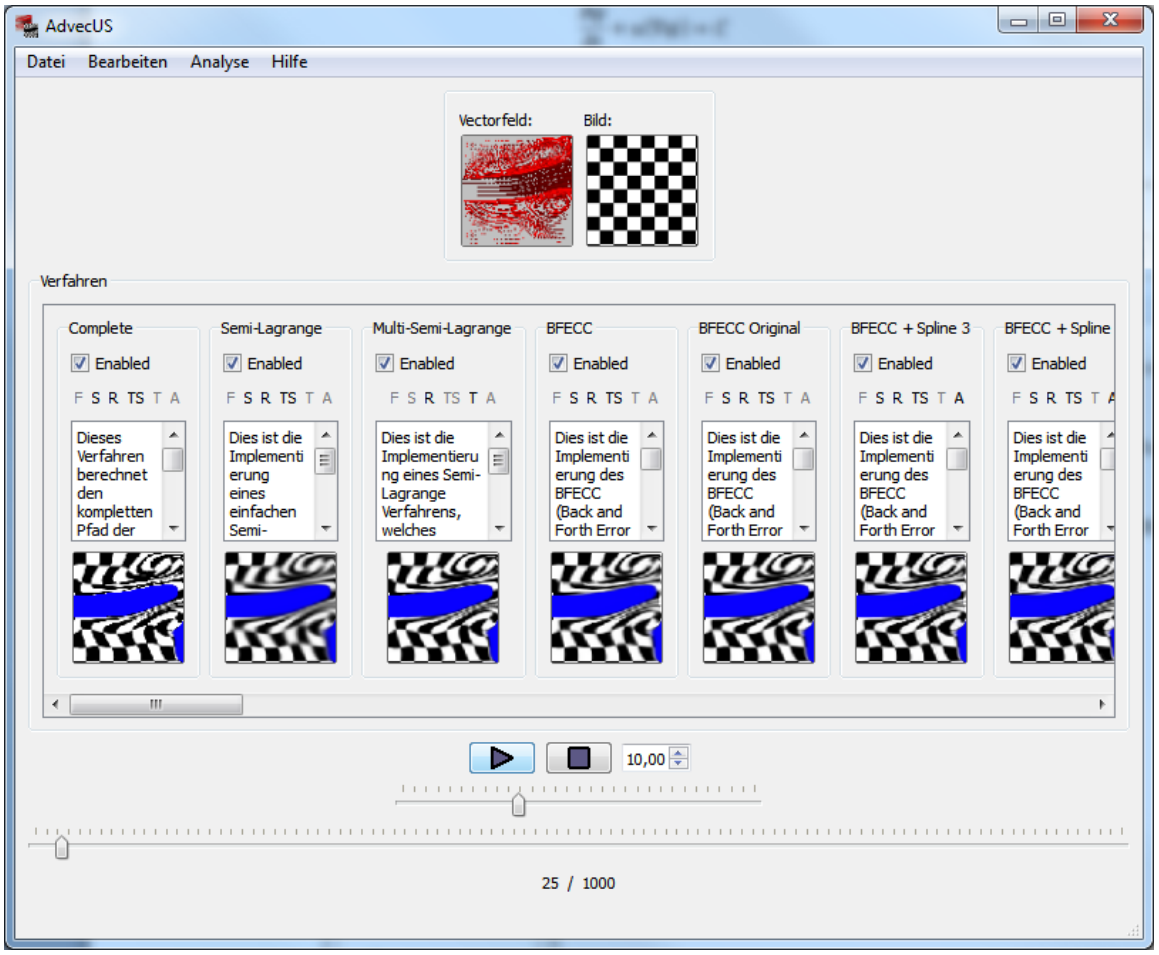


Abbildung 5-1: Programm AdvecUS, das einen Vergleich der verschiedenen Verfahren durchführt.

Das komplette Programm wurde in C++ geschrieben. Zusätzlich zu der Standartlibrary wurden einige externe Libraries verwendet.

5.1 Abhängigkeiten

Die hier verwendeten Libraries stellen grundlegende Verfahren zur Verfügung und sind alle frei verfügbar.

5.1.1 QT

QT ist ein plattformunabhängiges Applikations- und UI-Framework.

Dies steuert das gesamte Fenstermanagement des Programms. Dadurch hat man den Vorteil, dass das Layout des Programms unabhängig vom Betriebssystem ist.

Des Weiteren beinhaltet QT das Signals- and Slots-System. Hierdurch ist es einfach, verschiedene Komponenten miteinander zu verbinden.

QT wird in der Version 4.7.2 eingesetzt.

5.1.2 POCO

POCO ist eine Sammlung von plattformunabhängigen Open-Source Libraries. Es beinhaltet ein breites Spektrum an häufig benötigten Komponenten. POCO steht für POrtable COmponents.

Mit POCO wird der gesamte Zugriff auf das Dateisystem des Betriebssystems erledigt. Dies bringt den Vorteil mit sich, dass der Zugriff sowohl auf Linux als auch auf Windows basierten Systemen unterstützt wird. Neben dem Zugriff auf das Filesystem wird der Zufallszahlengenerator von POCO verwendet, um eine homogene Menge von Zufallszahlen zu produzieren.

POCO wird in der Version 1.3.6 eingesetzt.

5.1.3 OpenCV

OpenCV ist eine plattformunabhängige Library, die Funktionen für die Echtzeit Computer Vision bereitstellt. OpenCV steht für Open Source Computer Vision.

AdvecUS benutzt nur die Funktionen um den Optischen Fluss mittels des Lucas-Kanade Algorithmus zu berechnen.

OpenCV wird in der Version 2.2 eingesetzt.

5.1.4 FreeImage

FreeImage ist eine plattformunabhängige Library zum Laden und Speichern von vielen Bildformaten. Mithilfe dieser Bibliothek werden sämtliche aktuellen Bildformate unterstützt.

FreeImage wird in der Version 3.13.1 eingesetzt.

5.2 Datenformate

Das Programm benötigt ein Vektorfeld und ein Bild als Input. Beide können in diesem Programm aus Dateien geladen werden. Die Dateiformate werden hier kurz beschrieben:

5.2.1 Vektorfeld

Vektorfelder werden in Dateien mit der Endung *.dat gespeichert. Da es noch kein standardisiertes Dateiformat für ein Vektorfeld gibt, wird das Vektorfeld aus einem selbstdefinierten Format geladen. In diesem Dateiformat wird erst die Größe des Feldes in X- und Y-Richtung angegeben. Dies wird in zwei 32 Bit Integern gespeichert. Diesen folgen anschließend die einzelnen 2D-Vektoren, die die Bewegung dieses Punktes repräsentieren. Ein Vektor besteht aus zwei 32 Bit Floating Point Werten. Die Vektoren werden reihenweise gespeichert.

5.2.2 Textur

Als Textur kann jedes beliebige Bild geladen werden. Es werden annähernd alle standardisierten Dateiformate unterstützt. Insbesondere wird auch ein Alphakanal unterstützt. Das Zentrum eines Texels befindet sich im Zentrum eines Pixels (siehe Abbildung 5-2). Dies ist wichtig, sobald man das Wrapping der Texturen einschaltet. Nähere Informationen hierzu sind in der DirectX Dokumentation (Microsoft, 2009) zu finden.

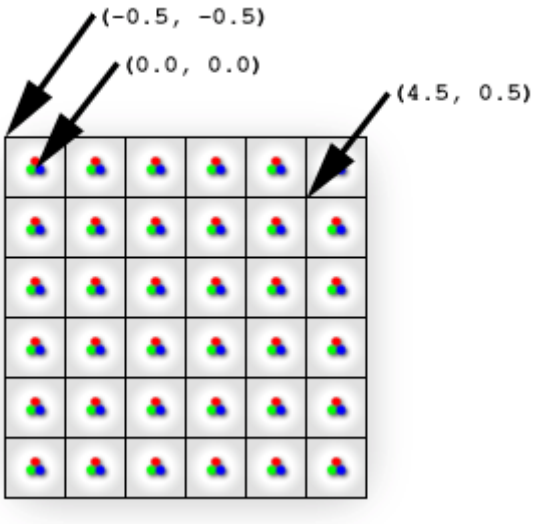


Abbildung 5-2: Zentrum eines Pixels. Quelle: (Microsoft, 2009)

5.3 Aufbau der Kernkomponenten

Das Programm besteht aus verschiedenen Kernkomponenten. Sie sind auf zwei Projekte verteilt, der Utility-Library und dem Hauptprogramm AdvecUS. In Abbildung 5-3 ist zu sehen wie die einzelnen Projekte und Dependencies zusammenhängen.

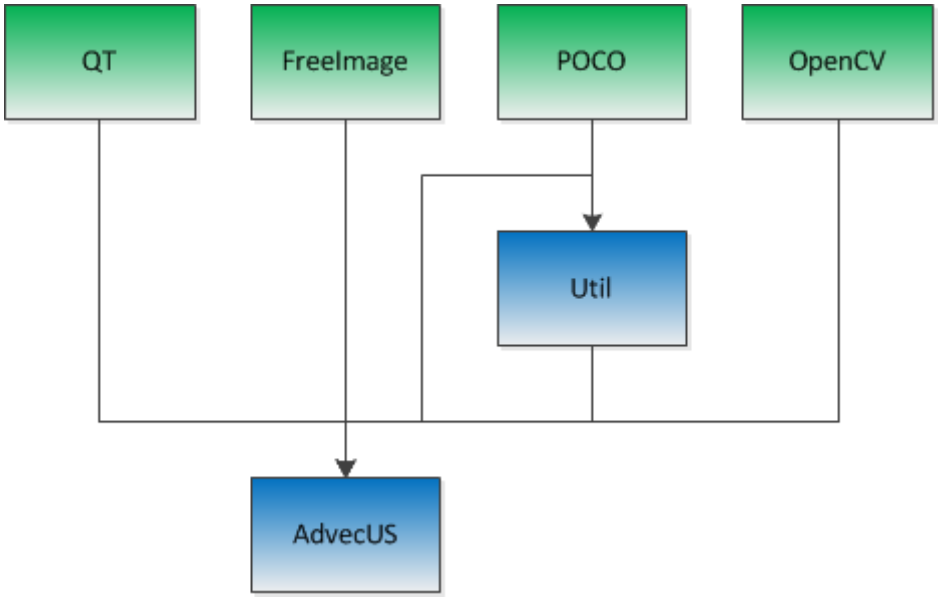


Abbildung 5-3: Diagramm, welcher Teil welche Komponente verwendet. Grün sind Dependencies. Blau sind selbsterstellte Komponenten.

5.3.1 Utility

In der Utility-Library sind verschiedene, oft gebrauchte Komponenten untergebracht. Sämtliche hier enthaltenen Funktionen sind in dem Namespace *Util* zusammengefasst. Die wohl wichtigste Komponente für AdvecUS ist die Mathematik-Funktionalität. Hier ist der 2D-Vektor, der für die Berechnung der Pixelpositionen eingesetzt wird, enthalten. Neben den Mathematikfunktionen beherbergt diese Library auch die Singleton-Basisklasse.

5.3.2 AdvecUS

Dieses Projekt erzeugt die Hauptanwendung. Es wird direkt die ausführbare Datei erzeugt. Sämtliche Klassen, die hierzu gehören, sind im Namespace *AdvecUS* zusammengefasst. Dieses Programm setzt Singleton-Klassen ein um eine einfache Verfügbarkeit der wichtigsten Kernkomponenten herzustellen. Die wichtigsten Kernkomponenten sind:

- MainWindow
  Diese Klasse ist die Hauptklasse der Anwendung. In ihr werden alle anderen Klassen erzeugt. Zusätzlich wird das Hauptfenster der Anwendung geladen und verwaltet.

- Operation::Manager
  Der Operation::Manager verwaltet alle Verfahren in der Anwendung. Jede Operation muss sich bei diesem Manager registrieren und wird anschließend von diesem verwaltet und dargestellt.

- Picture
  Dies ist ein einfaches Bild. Dieses Bild stellt einfache Funktionen zur Manipulation zur Verfügung. Jedes Bild in dem Programm basiert auf dieser Klasse. Um die Bilder in QT anzuzeigen gibt es eine Wrapperklasse namens QTPicture. Sie transformiert bei einer Änderung des Bildes die Bilddaten in ein QT-Bild, welches angezeigt werden kann.

- BasePicture::BasePicture
  Diese Klasse stellt das Basisbild zur Verfügung. Dieses Bild soll von allen Verfahren advektiert werden.

- Vectorfield::Vectorfield
  Genau wie das Basisbild stellt diese Klasse das Vektorfeld, das von allen Verfahren genutzt werden soll, zur Verfügung.

- Generator::Manager
  Dieser Manager verwaltet alle Vektorfeldgeneratoren des Programms. Jeder Generator muss sich hier registrieren ehe er verwendet werden kann.

6 Verfahren zur Texturadvektion

Um die Advektion durchzuführen, muss die Advektionsgleichung in Formel 4-6 gelöst werden. Der Fehler C sollte bei dieser Lösung möglichst gegen 0 gehen. Ansonsten entstehen Folgefehler bei Verfahren, welche auf die Ergebnisse eines vorherigen Schrittes zugreifen. Im Folgenden werden Verfahren mit numerischen Algorithmen vorgestellt, die die Advektionsgleichung lösen.

Bei den Verfahren gibt es zwei grundlegende Unterschiede im Aufbau:

Die erste Gruppe der Verfahren transportiert die Information eines Pixels vorwärts, um die Advektion $T \rightarrow T'$ zu berechnen. Sie werden als Forward-Verfahren bezeichnet. Es entsteht allerdings das Problem, dass der so transportierte Wert nicht auf genau einem neuen Pixel landet. Deshalb muss er aufgeteilt werden um T' an den Gitterpunkten zu erhalten. Diese Aufteilung ist allerdings ein unlösbares Problem. Hierdurch entsteht der Fehler C.

Ein weiteres Problem bei diesem Vorwärtstransport der Informationen ist, dass Gitterpunkte im Ergebnis existieren können, die keine Information zugewiesen bekommen. Solche Löcher vergrößern sich mit steigender Schrittzahl. Allerdings ist bei manchen Verfahren ist das Entstehen von Löchern aufgrund des Verfahrens nicht möglich.

In Abbildung 6-1 ist die Funktionsweise dieser Gruppe an Verfahren verdeutlicht.

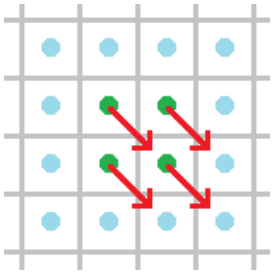


Abbildung 6-1: Gitterstellen die vorwärts transportiert werden.

Die zweite Gruppe der Verfahren sind die Backward-Verfahren. Es handelt es sich um Verfahren, die versuchen, für jeden Pixel des finalen Bildes die advektierten Werte zu berechnen. Es wird also versucht $T \rightarrow T'$ durch Zurückrechnen zu erhalten. Hierdurch wird die Aufteilung der Farbinformation auf die entsprechenden Pixel bei den Forward-Verfahren umgangen. Löcher können auch nicht auftreten, da für jeden Gitterpunkt die Daten berechnet werden. Allerdings erhält man dadurch weitere Probleme. Jetzt muss nicht der Wert auf neue Pixel aufgeteilt werden, sondern der Wert zwischen anderen Pixeln interpoliert werden. In Abbildung 6-2 ist zu sehen, wie diese Verfahren die einzelnen Werte berechnen.

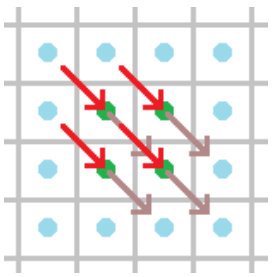


Abbildung 6-2: Verdeutlichung der Funktionsweise von Backward-Verfahren.

Verfahren, welche die Interpolation der Daten T über eine Fläche durchführen, erzeugen hierdurch eine Unschärfe im Ergebnis T' der Berechnung. Dies geschieht durch den iterativen Ansatz, bei welchem im nächsten Schritt wieder der Mittelwert auf den gemittelten Ergebnissen berechnet wird. Bei solchen Verfahren tendiert der Wert der Ergebnisdaten zum Mittelwert der Quelldaten. Es wird also ein Verfahren benötigt, das den Fehler C, der bei der Transformation $T \rightarrow T'$ entsteht möglichst gering hält.

Beide Verfahrensarten haben das Problem, dass aufgrund des Geschwindigkeitsvektors aus dem Vektorfeld eine neue Position berechnet werden muss. Die Schwierigkeit liegt hierbei an Stellen, an denen sich das Vektorfeld krümmt. Die Verfahren, die die neue Position berechnen, müssen in der Lage sein der räumlichen Änderung des Vektorfelds möglichst genau zu folgen und somit die neue Position möglichst genau zu berechnen. Die Schrittweite ist variabel. Je nach Anwendung kann eine dynamische oder eine statische Schrittweite verwendet werden. Bei der Analyse der Verfahren und Komponenten wird eine statische Schrittweite von $h = 1$ verwendet.

Sowohl die Forward- als auch die Backward-Verfahren lassen sich komplett oder teilweise auf die GPU portieren. Das würde die Performance dieser Verfahren deutlich erhöhen, da die Berechnung parallel erfolgen kann.

6.1 Taxonomie

Sowohl die Forward- als auch die Backward-Verfahren bestehen aus getrennten Komponenten. Diese Komponenten machen die Qualität des Verfahrens aus. Sie werden in verschiedenen Verfahren eingesetzt und können innerhalb der Kategorien ausgetauscht werden, um die Verfahren bezüglich Performance oder Qualität zu optimieren. Im Folgenden werden diese Komponenten erläutert.

6.1.1 Dateninterpolation

Jedes Verfahren muss die Quelldaten an einer beliebigen Stelle auswerten können. Das ist wichtig, weil die Quelldatenfelder nicht mit der eigentlichen Größe des berechneten Bildes übereinstimmen müssen und manche Verfahren die Daten an beliebigen Stellen benötigen. An Stellen, an denen kein exakter Wert gegeben ist, muss dieser aus den vorhandenen Werten errechnet werden. Hierzu gibt es verschiedene Verfahren, mit denen dies durchgeführt werden kann. Bei iterativen Verfahren ist der Qualitätsunterschied zwischen den einzelnen Verfahren bedeutend, da sich Fehler aufsummieren. In Abbildung 6-3 ist ein Beispiel der Auswertung der Daten an einer markierten Stelle zu sehen.

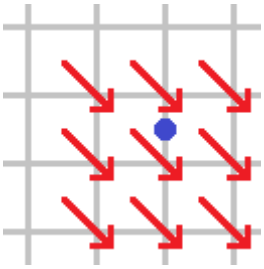


Abbildung 6-3: Vektorfeld mit einer markierten Stelle (blau) an der das Vektorfeld ausgelesen werden soll.

Bei der Texturadvektion gibt es üblicherweise zwei verschiedene Typen von Datenfeldern. Einerseits das Vektorfeld und andererseits die Daten, welche advektiert werden sollen. Bezüglich der Verfahren unterscheiden sich beide Datentypen nicht voneinander. Es existieren lediglich an Positionen außerhalb der Daten verschiedene Methoden wie diese interpretiert werden sollen.

Bei beiden Datentypen gibt es die einfache Möglichkeit, Daten außerhalb des definierten Bereichs als nicht vorhanden anzunehmen und einen Default-Wert zurückzuliefern. Bei dem Vektorfeld wäre dies der Nullvektor. Das bedeutet, dass außerhalb des definierten Bereichs keine Bewegung stattfindet. Bei dem Datenfeld hängt der Default-Wert von der Art der Daten ab. Sind zum Beispiel Farbinformationen gespeichert, so wäre dies entweder schwarz oder eine vollständig transparente Farbe.

Bei dem Vektorfeld kann es jedoch bei manchen Verfahren sinnvoll sein auch außerhalb des Vektorfelds eine Bewegung zu erhalten. In diesem Fall wird der äußere bekannte Wert des Vektorfelds angenommen. Dies ist in Abbildung 6-4 symbolisiert.

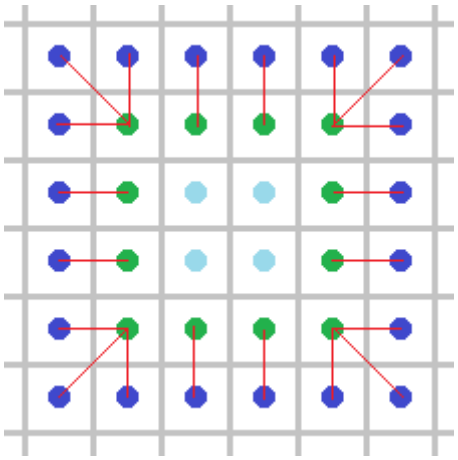


Abbildung 6-4: Diagramm, welche Pixel welche Werte erhalten, wenn die Pixel außerhalb des Vektorfelds (blau) mit Werten gefüllt werden sollen. Die grünen Pixel sind hierbei die äußere Reihe des Vektorfeldes.

Bei dem Datenfeld ist in manchen Situationen auch ein Wrap sinnvoll. Dies bedeutet, dass sich das eigentliche Feld außerhalb des definierten Bereichs immer wiederholt. Eine detaillierte Beschreibung des Wraps ist in der DirectX Dokumentation zu finden (Microsoft, 2009).

6.1.1.1 Nearest Neighbour

Dies ist die einfachste Strategie um die Daten des Felds auszuwerten. Es wird einfach der nächste vorhandene Punkt genommen und zurückgegeben. In Abbildung 6-5 ist das verdeutlicht.

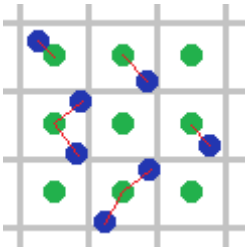


Abbildung 6-5: Nearest Neighbour Verfahren mit Markierung, welche Position von welchem Pixel ausgewertet wird. Grün: Vorhandene Pixel. Blau: Stellen an denen die Daten interpoliert werden müssen.

6.1.1.2 Bilineare Interpolation

Die am häufigsten anzutreffende Strategie ist die bilineare Interpolation zwischen den bekannten Werten. Hierbei wird zwischen den vier bekannten Eckpunkten eine bilineare Interpolation durchgeführt. Diese Interpolation ist in Abbildung 6-6 zu sehen.

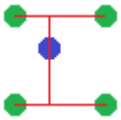


Abbildung 6-6: Bilineare Interpolation um einen Wert zu berechnen.

In Abbildung 6-7 und Formel 6-1 ist die lineare Interpolation für den eindimensionalen Fall zu sehen. Der Abstand zwischen zwei Stützstellen ist $\Delta x = 1$.

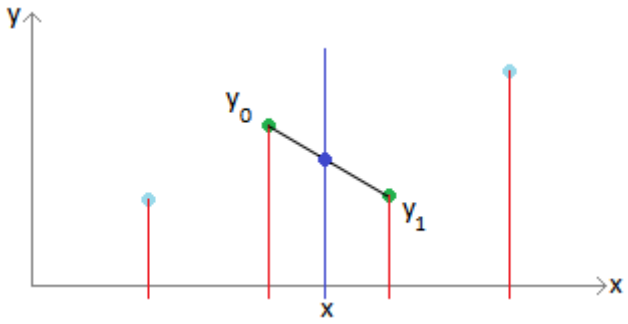


Abbildung 6-7: Lineare Interpolation im eindimensionalen Fall.

Formel 6-1: Lineare Interpolation zwischen zwei Werten.

$$f(x) = (y_1 - y_0)x + y_0$$

6.1.1.3 *Interpolation mit Polynomen dritten Grades (Polynom 3)*

Bei dieser Interpolation werden Polynome dritten Grades verwendet um die Werte zu berechnen. Es werden eindimensionale Polynome auf der X- und Y-Achse verwendet. Für Polynome dritten Grades werden vier Stützstellen benötigt. Diese Stützstellen sind die vier nächsten bekannten Werte. Der Abstand zwischen zwei Stützstellen ist $\Delta x = 1$. Ein Polynom dritten Grades ist in Abbildung 6-8 für einen eindimensionalen Fall zu sehen.

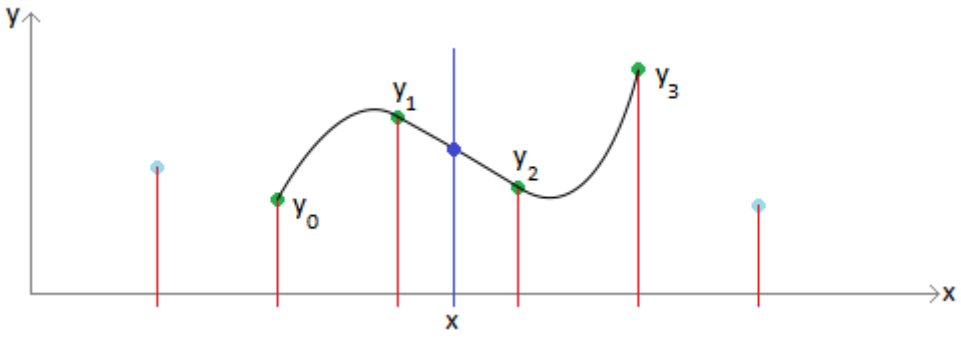


Abbildung 6-8: Eindimensionale Interpolation mit einem Polynom dritten Grades. Nur die vier direkten Nachbarwerte werden für die Interpolation verwendet.

Der Polynom, der hierbei für die Berechnung verwendet wird, ist in Formel 6-2 zu sehen.

Formel 6-2: Polynom dritten Grades für die Interpolation.

$$f(x) = \left(-\frac{1}{6}y_0 + \frac{1}{2}y_1 - \frac{1}{2}y_2 + \frac{1}{6}y_3\right)x^3$$
$$+\left(y_0 - \frac{5}{2}y_1 + 2y_2 - \frac{1}{2}y_3\right)x^2$$
$$+\left(-\frac{11}{6}y_0 + 3y_1 - \frac{9}{6}y_2 + \frac{1}{3}y_3\right)x$$
$$+y_0$$

Da bei einem Polynom auch Werte über dem Maximalwert und unter dem Minimalwert der Stützstellen auftreten können muss eine Limitierung der Werte verwendet werden. Diese Limitierung sorgt dafür, dass die berechneten Werte den Bereich der Daten nicht verlassen. Als Limitierung wird eine Begrenzung des berechneten Wertes auf den Wertebereich zwischen dem Maximum und Minimum der Stützstellen des Polynoms verwendet. Werte außerhalb dieser Begrenzung werden auf das Maximum, beziehungsweise Minimum dieses Bereiches gesetzt.

Um mit eindimensionalen Polynomen einen zweidimensionalen Wert zu interpolieren, wird erst ein Polynom auf der X-Achse verwendet um die Werte auf der X-Achse zu berechnen. Anschließend wird ein Polynom auf der Y-Achse und den eben berechneten Werten verwendet, um den finalen Wert zu erhalten. Abbildung 6-9 verdeutlicht dieses Vorgehen.

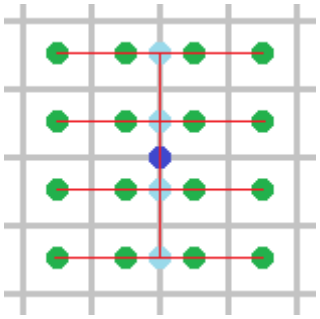


Abbildung 6-9: Interpolation mit Polynomen dritten Grades auf einem zweidimensionalen Datenfeld. Die verwendeten eindimensionalen Polynome sind durch die roten Linien dargestellt.

6.1.1.4 Interpolation mit Polynomen fünften Grades (Polynom 5)

Die Interpolation mit Polynomen fünften Grades ist genauso aufgebaut wie die Interpolation mit Polynomen dritten Grades. Der einzige Unterschied besteht darin, dass sechs statt vier Stützstellen verwendet werden und dadurch die Polynome genauer berechnet werden können. Die Polynome werden hier mit Formel 6-3 berechnet. Ein solches Polynom ist in Abbildung 6-10 zu sehen.

Wie bei den Polynomen dritten Grades muss auch hier eine Limitierung der Werte verwendet werden um die berechneten Werte innerhalb des Wertebereichs zu halten.

Formel 6-3: Polynom fünften Grades für die Interpolation.

$$f(x) = \left(-\frac{1}{120}y_0 + \frac{1}{24}y_1 - \frac{1}{12}y_2 + \frac{1}{12}y_3 - \frac{1}{24}y_4 + \frac{1}{120}y_5\right)x^5$$
$$+\left(\frac{1}{8}y_0 - \frac{7}{12}y_1 + \frac{13}{12}y_2 - y_3 + \frac{11}{24}y_4 - \frac{1}{12}y_5\right)x^4$$
$$+\left(-\frac{17}{24}y_0 + \frac{71}{24}y_1 - \frac{59}{12}y_2 + \frac{49}{12}y_3 - \frac{41}{24}y_4 + \frac{7}{24}y_5\right)x^3$$
$$+\left(\frac{15}{8}y_0 - \frac{77}{12}y_1 + \frac{107}{12}y_2 - \frac{13}{2}y_3 + \frac{61}{24}y_4 - \frac{5}{12}y_5\right)x^2$$
$$+\left(-\frac{137}{60}y_0 + 5y_1 - 5y_2 + \frac{10}{3}y_3 - \frac{5}{4}y_4 + \frac{1}{5}y_5\right)x$$
$$+y_0$$

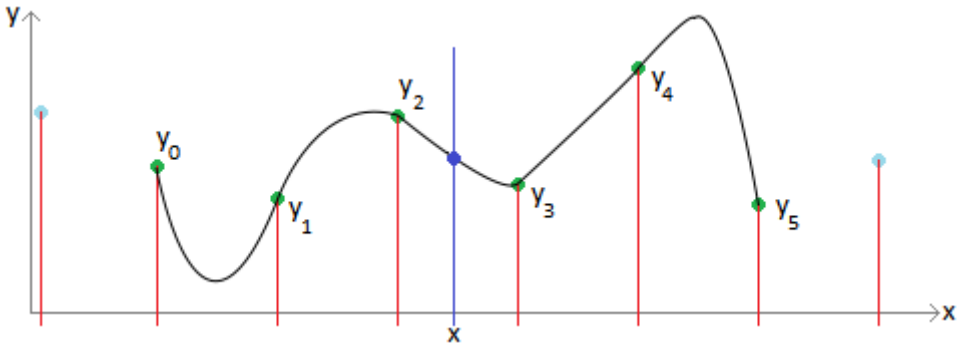


Abbildung 6-10: Eindimensionale Interpolation mit einem Polynom fünften Grades. Nur die sechs direkten Nachbarstellen werden für die Interpolation verwendet.

Die Aufteilung auf ein zweidimensionales Feld funktioniert genauso wie bei den Polynomen dritten Grades.

6.1.1.5 Approximation mit B-Splines dritten Grades (B-Spline 3)

Bei diesem Verfahren werden die Daten mittels eines B-Splines dritten Grades ausgewertet. Im Gegensatz zu den Verfahren mit Polynomen wird hier ein B-Spline durch die Stützstellen gelegt und an der entsprechenden Stelle ausgewertet. Diese Stützstellen sind die vier nächsten bekannten Werte. Der Abstand zwischen zwei Stützstellen ist $\Delta x = 1$. Abbildung 6-11 stellt eine solche eindimensionale Interpolation dar.

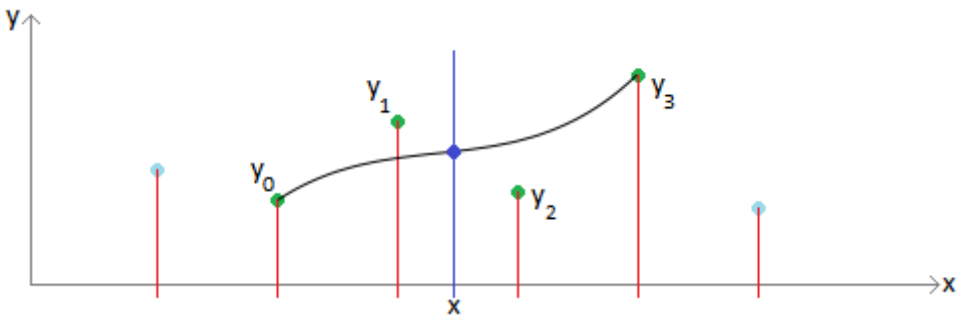


Abbildung 6-11: Eindimensionale Approximation verschiedener Werte mit einem B-Spline dritten Grades. Nur die vier direkten Nachbarstellen werden für die Approximation verwendet.

Die Aufteilung auf ein zweidimensionales Feld funktioniert genauso wie bei den beiden Verfahren für Polynome.

6.1.1.6 Approximation mit B-Splines fünften Grades (B-Spline 5)

Dieses Verfahren ähnelt dem Verfahren, welches B-Splines dritten Grades für die Approximation des Wertes einsetzt. Bei diesem Verfahren werden jedoch statt B-Splines dritten Grades B-Splines fünften Grades eingesetzt. In Abbildung 6-12 ist ein eindimensionales Beispiel zu sehen.

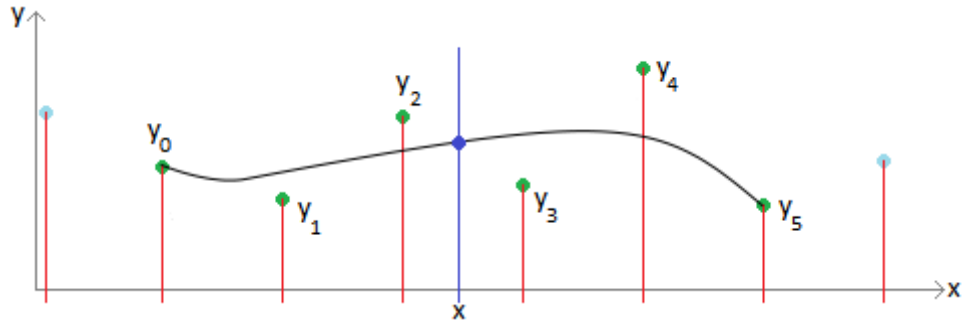


Abbildung 6-12: Eindimensionale Approximation verschiedener Werte mit einem B-Spline fünften Grades. Nur die sechs direkten Nachbarstellen werden für die Approximation verwendet.

6.1.1.7 Kernel

Diese Methode zum Interpolieren der Datenfelder verwendet einen Kernel. Dieser Kernel wird auf die auszulesende Position gelegt (siehe Abbildung 6-13). Anschließend werden alle Stellen innerhalb des Kernels mit einem Gewicht multipliziert und aufeinander addiert. Formel 6-5 ist die hierfür verwendete Formel. Danach muss nur noch der so errechnete Wert normiert werden. Diese Normierung erfolgt in Formel 6-4 mithilfe von Formel 6-6. Die Gewichte werden mit Formel 6-7 berechnet. Bei dieser Formel ist zu beachten, dass Gewichte außerhalb des Kernels 0 sind. Auf diese Weise wird der Wert an der entsprechenden Stelle berechnet. Hierbei ist der Radius des Kernels einstellbar, was sich direkt auf die Qualität in der jeweiligen Situation auswirkt. Die Gewichtsfunktion des Kernels wurde in der Vorlesung (Speith, 2006) vorgestellt.

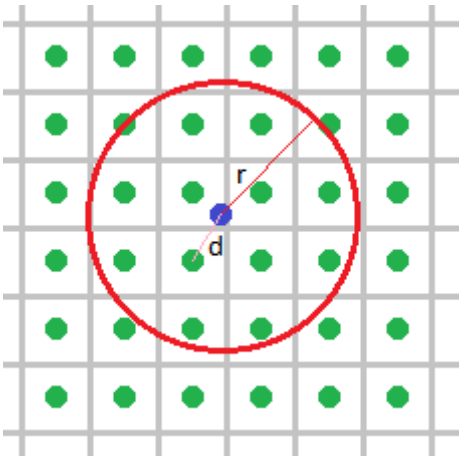


Abbildung 6-13: Verfahren, welches einen Kernel einsetzt um beliebige Stellen zu interpolieren.

Formel 6-4: Berechnung eines Wertes.

$$c = \frac{\tilde{c}}{w_g}$$

Formel 6-5: Addition der verschiedenen Werte unter Berücksichtigung des Gewichts. w(r) ist die Gewichtsfunktion und c(x) ist der Wert an der Stelle x. d(x) ist die Funktion, die den Abstand von x zum Mittelpunkt des Kernels berechnet.

$$\tilde{c} = \sum w(d(x))c(x)$$

Formel 6-6: Berechnung des Gesamtgewichts. w(x) ist die Gewichtsfunktion an der Stelle x.

$$w_g = \sum w(d(x))$$

Formel 6-7: Berechnung der Gewichte. r ist der Radius des Kernels.

$$\delta = \frac{40}{7\pi}$$

$$w(d) = \begin{cases} \frac{\delta}{r^2}\left(6\left(\frac{d}{r}\right)^3 - 6\left(\frac{d}{r}\right)^2 + 1\right), & \frac{d}{r} < \frac{1}{2} \\ \frac{2\delta}{r^2}\left(1 - \frac{d}{r}\right)^3, & \frac{1}{2} \leq \frac{d}{r} \leq 1 \\ 0, & sonst \end{cases}$$

6.1.2 Vektorfeldintegration

Der zweite große Teilbereich der Verfahren ist die numerische Zeitintegration der Vektorfelddaten um eine neue Position zu berechnen. In dem Vektorfeld sind die Geschwindigkeiten an den entsprechenden Stellen gespeichert. Aus diesen Geschwindigkeiten muss eine neue Position des Pixels in diesem Schritt errechnet werden. Das Vektorfeld muss also zeitlich integriert werden. Bei Forward-Verfahren wird die Geschwindigkeit in positiver Richtung, bei Backward-Verfahren in negativer Richtung gesehen.

Bei den Verfahren ist die Position durch s gegeben. $v(x)$ ist eine Funktion, die das Vektorfeld an der Stelle x auswertet. h ist die Schrittweite eines Schrittes.

Die genaue Analyse der Funktions- und Verhaltensweise dieser Verfahren ist in dem Buch (Hairer, Nørsett, & Wanner, 2009) oder einer vergleichbaren Arbeit zu finden.

6.1.2.1 Euler'sche Methode

Die Euler'sche Methode der Zeitintegration des Vektorfelds ist die einfachste Methode um die neue Position zu berechnen. Hierbei wird die neue Position berechnet, indem das Weg-Zeit Gesetz gelöst wird. Dieses ist in Formel 6-8 zu finden.

Formel 6-8: Weg-Zeit Gesetz, um eine neue Position zu berechnen.

$$s_{i+1} = s_i + h * v(s_i)$$

Bei dieser Berechnung wird jedoch von keiner Krümmung des Vektorfeldes und damit der Geschwindigkeit während der Schrittweite ausgegangen. In Abbildung 6-14 ist die Darstellung dieses Verfahrens zu sehen.

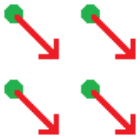


Abbildung 6-14: Pixel, für welche mittels der Euler'schen Methode die neue Position berechnet wird.

Die Größe des Fehlers, welcher in der Advektionsgleichung Formel 4-6 auftritt, hängt hierbei stark von der Schrittweite ab. Geht die Schrittweite gegen 0, so entsteht kein Fehler bei der Lösung der Advektionsgleichung.

6.1.2.2 Leap-Frog Verfahren

Das Leap-Frog Verfahren ist ein Verfahren zweiter Ordnung. Deswegen liefert es genauere Ergebnisse als Verfahren, die nur erster Ordnung sind.

Bei dem Leap-Frog-Verfahren wird die Position und die Geschwindigkeit zu unterschiedlichen Zeitpunkten berechnet und daraus anschließend die neue Position errechnet. Dies geschieht ähnlich wie beim Bockspringen (siehe Formel 6-9). Zur Verdeutlichung des Verfahrens ist Abbildung 6-15 gegeben. Hier sind die einzelnen Positionen und Geschwindigkeiten zu sehen.

Formel 6-9: Leap-Frog Verfahren zur Berechnung einer neuen Position.

$$s_{i-1} = s_i - v(s_i)h$$

$$a_{s_i} = -v(s_{i-1}) + v(s_i)$$

$$s_{i+1} = 2s_i - s_{i-1} + a_{s_i}h^2$$

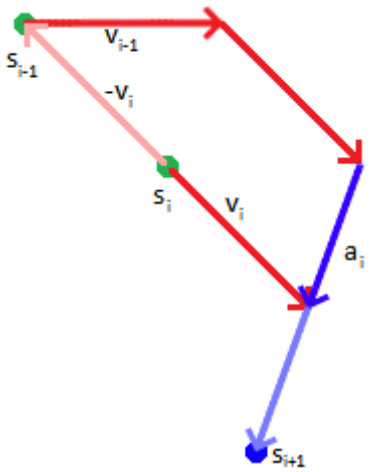


Abbildung 6-15: Leap-Frog Verfahren zur Berechnung der neuen Position. Die Schrittweite wird in dieser Zeichnung als 1 angenommen.

6.1.2.3 Klassisches Runge-Kutta Verfahren (RK 4)

Das klassische Runge-Kutta Verfahren ist eine andere Bezeichnung für das vierstufige Runge-Kutta Verfahren zur numerischen Lösung von Anfangswertproblemen. Hierbei wird versucht die auftretenden Fehler bei nichtlinearen Funktionen durch geeignete Kombination aus verschiedenen Differenzquotienten zu reduzieren. Wie die anderen Verfahren ist dieses Verfahren nur eine Näherung an die tatsächlichen Geschwindigkeitspfade im Vektorfeld.

Bei diesem Verfahren muss das Vektorfeld an vier Stellen ausgewertet werden. Die Positionen, an denen das Feld ausgewertet werden muss, erhält man durch Formel 6-10. Diese verschiedenen Geschwindigkeiten werden anschließend kombiniert um die neue Position zu berechnen. Die Kombination geschieht mit Hilfe von Formel 6-11. In Abbildung 6-16 ist dieses Verfahren im Eindimensionalen zu sehen. Die entsprechenden Schritte mit der Steigung der Funktion sind ebenfalls eingezeichnet.

Formel 6-10: Formeln zur Berechnung der Zwischenschritte im klassischen Runge-Kutta Verfahren.

$$k_1 = h * v(s_i)$$

$$k_2 = h * v\left(s_i + \frac{1}{2}k_1\right)$$

$$k_3 = h * v\left(s_i + \frac{1}{2}k_2\right)$$

$$k_4 = h * v(s_i + k_3)$$

Formel 6-11: Formel für die Berechnung der neuen Position im klassischen Runge-Kutta Verfahren.

$$s_{i+1} = s_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

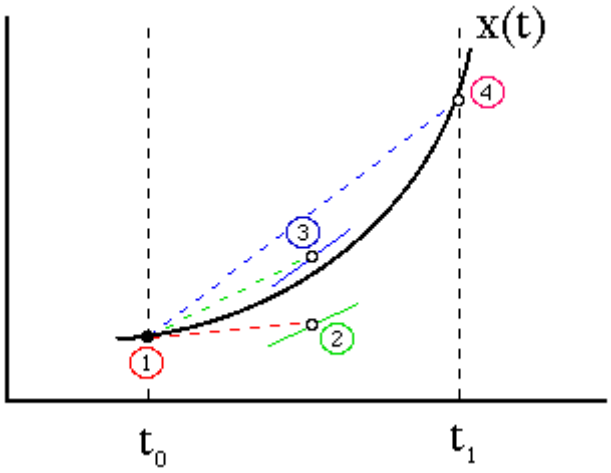


Abbildung 6-16: Klassisches Runge-Kutta Verfahren im Eindimensionalen. In dieser Grafik sind die entsprechenden Schritte und die entsprechende Steigung eingezeichnet. Quelle: (Runge-Kutta, 2011)

6.2 Implementierung der Verfahren

Die meisten Verfahren basieren auf den einzelnen Komponenten. Es gibt jedoch auch Verfahren, die einen komplett anderen Ansatz verfolgen. Die Verfahren werden hier vorgestellt. Die einzelnen Verfahren sind wieder in die zwei Kategorien, Forward- und Backward-Verfahren, aufgeteilt.

6.2.1 Iterative Verfahren

Iterative Verfahren sind Verfahren, deren Berechnungen eines Schrittes auf dem Ergebnis des vorherigen Schrittes beruhen. In jedem Schritt wird das Ergebnis des vorherigen Schrittes als die neuen Quelldaten verwendet. In Abbildung 6-17 ist dieses Verhalten zu sehen.

Abbildung 6-17: Iterationskette eines iterativen Verfahrens. Jeder neue Schritt verwendet das Ergebnis des vorherigen als Quelldaten.

Der Vorteil eines iterativen Verfahrens ist, dass der Rechenaufwand von einem Schritt zum Nächsten konstant ist. In jedem Schritt müssen die gleichen Rechnungen durchgeführt werden. Iterative Verfahren lassen sich somit einfach auf die GPU portieren.

Durch die Iteration entstehen allerdings Probleme bei der Qualität der Verfahren. Fehler, die sich durch Lösen der Advektionsgleichung in Formel 4-6 ergeben, verstärken sich, da der nächste Schritt auf fehlerhaften Daten aufbaut und seinerseits wiederum neue Fehler einbringt.

6.2.2 Backward-Verfahren

Backward-Verfahren sind die Verfahren, die versuchen durch Zurückrechnen auf die Werte eines Pixels zu kommen. Die Verfahren beruhen größtenteils auf der Semi-Lagrang'schen Sichtweise. Die größten Probleme bei diesen Verfahren bestehen in der Interpolation der Werte an einer bestimmten Stelle der Quelldaten und in der Berechnung der Ursprungsposition eines Pixels. h ist also bei den meisten Backward-Verfahren negativ.

Es gibt verschiedene Verfahren, die auf diesem Backward Ansatz basieren.

6.2.2.1 Complete

Dieses Verfahren dient zur Berechnung eines Referenzbildes. Hier wird für jeden Pixel der gesamte Weg zurück zu seinem Ausgangspunkt berechnet. Abbildung 6-18 verdeutlicht dies. Damit gehört dieses Verfahren der Lagrang'schen Sichtweise an. Das Vektorfeld wird mittels der Euler'schen Methode integriert. Das Vektor- und Datenfeld werden mit einem bilinearen Ansatz interpoliert.

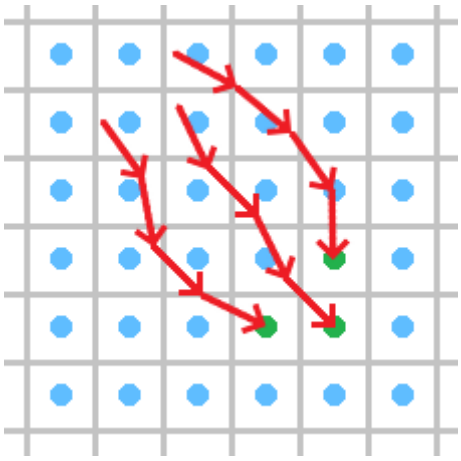


Abbildung 6-18: Komplette Pfadberechnung für vier Schritte.

Dieses Verfahren berechnet für jeden Schritt den gesamten Pfad bis ins Ursprungsbild. Dadurch entstehen keine iterativen Fehler. Allerdings wird viel Rechenaufwand benötigt um den gesamten Pfad zu berechnen. Ein weiterer Nachteil besteht darin, dass der Rechenaufwand mit der Anzahl an Schritten des Verfahrens steigt, da die einzelnen Pfade pro Schritt länger werden.

Aus diesen Gründen wird dieses Verfahren nur dazu eingesetzt um ein Referenzbild zu berechnen. Dieses Verfahren wird bei der Komponentenanalyse als Referenzverfahren eingesetzt.

6.2.2.2 Complete RK 4

Dieses Verfahren funktioniert genauso wie das Complete Verfahren. Einziger Unterschied ist, dass das klassische Runge-Kutta Verfahren zur Integration des Vektorfeldes eingesetzt wird und nicht die

Euler'sche Methode wie beim Complete Verfahren. Dies ermöglicht eine genauere Berechnung der Referenzbilder, da die Integration des Vektorfeldes genauer ist.

Dieses Verfahren wird eingesetzt um die Referenzbilder für andere Verfahren zu berechnen. Andere Verfahren können anschließend mit diesem Referenzbild verglichen werden um die Qualität zu bewerten.

6.2.2.3 Semi-Lagrange

Das Semi-Lagrange Verfahren ist ein iteratives Verfahren. Dieses Verfahren verwendet die einfachsten Methoden um das advektierte Bild zu berechnen. Wie der Name schon sagt gehört dieses Verfahren zur Semi-Lagrang'schen Sichtweise. Die Vektorfeld- und Quelldaten werden mittels der bilinearen Interpolation ausgewertet. Das Verfahren wurde in der Arbeit (Stam, 1999) noch unter dem Namen „Method of Characteristics" vorgestellt. Die Berechnung der neuen Daten einer Position ist in Formel 6-12 zu finden.

Formel 6-12: Berechnung des neuen Wertes einer Position.

$$T'(x) = T(p(x, -h))$$

$T(x)$ wertet hierbei die vorhandenen Daten an der Stelle x aus. Das Ergebnis wird in einem neuen Datenfeld T' gespeichert. T' wird im nächsten Schritt T und wird ausgewertet.

$p(x, h)$ ist die Funktion, die eine Position x integriert und die neue Position im Abstand h berechnet. Die entsprechenden Verfahren für $p(x, h)$ sind in Kapitel 6.1.2 vorgestellt worden. In diesem Verfahren ist h negativ.

6.2.2.4 BFECC–Original

Das Original BFECC (Back and Forth Error Compensation and Correction) Verfahren wurde in der Publication (ByungMoon, Yingjie, Ignacio, & Jarek, 2005) vorgestellt. Dieses Verfahren ist ein iteratives Verfahren. Jeder Schritt basiert auf vier Berechnungsschritten um das Ergebnis zu erhalten.

Der erste Schritt des Verfahrens besteht aus einem einfachen Backward-Schritt. In diesem Schritt wird ein neues Bild berechnet. Die Vektoren werden, wie bei Backward-Verfahren üblich, in die gegengesetzte Richtung integriert.

Der zweite Schritt berechnet wieder ein neues Bild. Diesmal werden die Vektoren jedoch in die korrekte Richtung integriert. Als Quelldaten wird hierbei das Bild aus dem ersten Schritt verwendet.

Anschließend wird die Differenz aus dem Bild des zweiten Schrittes und dem Originalbild berechnet. Diese Differenz ist der Fehler, der durch die einzelnen Schritte zustande kommt. Wenn beide Schritte fehlerfrei funktionieren, ist diese Differenz Null. Aufgrund der Aufteilung auf separate Pixel ist das jedoch nur selten der Fall. Um diesen Fehler zu korrigieren wird das Originalbild korrigiert, indem die Hälfte des Fehlers addiert wird.

Der letzte Schritt verwendet das korrigierte Bild um einen Backward-Schritt durchzuführen. Dieser Schritt liefert das finale Ergebnis.

Das gesamte Verfahren ist nochmals in Formel 6-13 zusammengefasst.

Formel 6-13: Schritte des BFECC-Original Verfahrens

$$\tilde{\varphi} = SingleStep(u, v, \varphi_n)$$

$$\bar{\varphi} = SingleStep(-u, -v, \tilde{\varphi})$$

$$\tilde{\varphi} = \varphi_n + \frac{\varphi_n - \bar{\varphi}}{2}$$

$$\varphi_{n+1} = SingleStep(u, v, \tilde{\varphi})$$

$SingleStep()$ ist hierbei eine Funktion, die einen Schritt des Verfahrens an den Stellen u und v auf dem Datenfeld φ berechnet.

In dem Originalverfahren werden das Vektor- und das Datenfeld bilinearen interpoliert. Die neue Position wird mittels der Euler'schen Methode aus den Geschwindigkeitsvektoren berechnet.

Aufgrund des Korrekturschrittes kann es bei dem Verfahren zu neuen Extremwerten kommen. Dies kann zur Instabilität des Verfahrens führen. Um das zu vermeiden wird der Wert limitiert. Eine einfache Limitierung ist es, den Wert auf das Minimum und Maximum der Werte des vorherigen Schrittes zu beschränken. In Abbildung 6-19 ist dieses Problem und die Lösung mittels Limitierung im Eindimensionalen dargestellt. Eine genaue Betrachtung dieses Problems ist in der Arbeit (Selle, Fedkiw, ByungMoon, Yingjie, & Jarek, June 2008) zu finden.

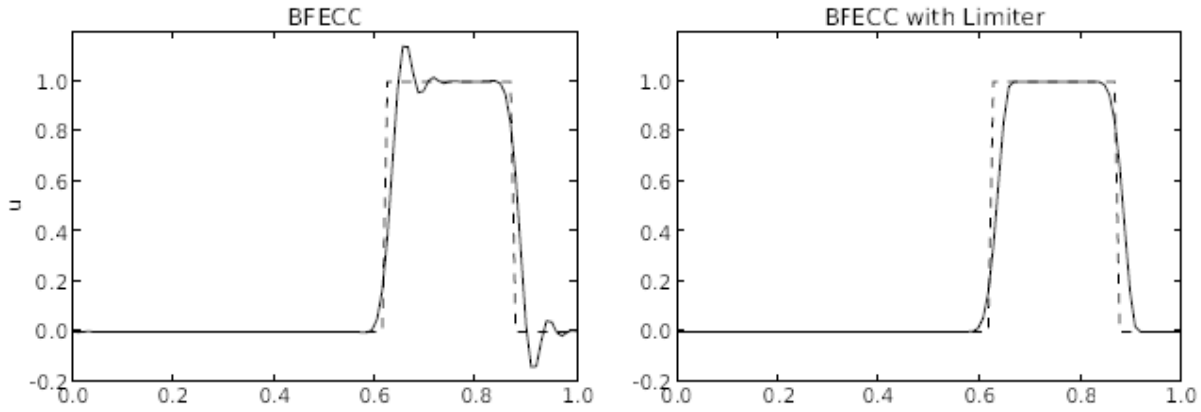


Abbildung 6-19: Eidimensionale Darstellung des Problems der neuen Extremwerte. Links: Darstellung des Original BFECC Verfahrens. Hier sind die neuen Extremwerte zu erkennen. Rechts: Lösung des Problems mithilfe der Limitierung der Werte. Quelle: (Selle, Fedkiw, ByungMoon, Yingjie, & Jarek, June 2008)

Als Limitierung wird in diesem Verfahren der berechnete Wert jedes Gitterpunktes auf den Bereich zwischen dem Maximum und Minimum der Quelldaten, welche für die Berechnung dieses Gitterpunktes verwendet werden, beschränkt. Es gibt jedoch noch weitere Möglichkeiten den Wert zu limitieren. Eine davon ist bei Verlassen des gültigen Bereiches auf das Basisverfahren umzuschwenken und den Wert an diesem Gitterpunkt hiermit zu berechnen. Hierfür wäre keine weitere Berechnung nötig, da diese Berechnung im ersten Schritt des Verfahrens schon durchgeführt ist.

Um die Qualität des Verfahrens weiter zu steigern, gibt es jedoch auch die Möglichkeit andere Methoden für die Auswertung der Felder und Berechnung der neuen Position einzusetzen.

6.2.2.4.1 BFECC-Original + Polynom 3

Bei dieser Modifikation wird statt der bilinearen Interpolation des Datenfelds eine Interpolation mit Polynomen dritten Grades angewendet. Dies erzeugt eine höhere Genauigkeit der Datenwerte.

6.2.2.4.2 BFECC-Original + Polynom 5

Diese Modifikation ist ähnlich der Modifikation mit Polynomen dritten Grades. Hier werden jedoch Polynome fünften Grades eingesetzt um die Genauigkeit weiter zu erhöhen.

6.2.2.4.3 BFECC-Original + Polynom 3 Last

Diese Modifikation setzt das Polynom dritten Grades nur im letzten Schritt des Verfahrens ein. Das bedeutet, dass die Fehlerberechnung und Korrektur weiterhin mittels der bilinearen Interpolation durchgeführt wird. Lediglich die Auswertung der Daten im letzten Schritt wird mit Polynomen dritten Grades bewerkstelligt.

Hierdurch erhält man eine bessere Performance, da die meisten Berechnungen weiterhin mit einer einfachen Methode durchgeführt werden. Die Qualität des Bildes wird jedoch durch den Einsatz der Polynome im letzten Schritt immer noch verbessert.

6.2.2.4.4 BFECC-Original + Polynom 5 Last

Diese Modifikation funktioniert genauso wie die vorherige Modifikation. Hier werden jedoch Polynome fünften Grades im letzten Schritt eingesetzt.

6.2.2.4.5 BFECC-Original + RK 4

Bei dieser Modifikation wird das klassische Runge-Kutta Verfahren angewendet um die neue Position zu berechnen. Die Interpolation des Daten- und Vektorfeldes erfolgt bilinear.

6.2.2.4.6 BFECC-Original + Polynom 3 + RK 4 + Polynom 3

Diese Modifikation besteht aus den Komponenten, welche die besten Ergebnisse geliefert haben und noch mit vernünftiger Performance berechenbar sind.

Hier wird das klassische Runge-Kutta Verfahren eingesetzt um die neue Position aus den Vektorfelddaten zu berechnen. Das Vektor- und Datenfeld werden mithilfe der Polynom 3 Komponente interpoliert.

6.2.2.5 BFECC

Dieses Verfahren ist die Weiterentwicklung des Original BFECC Verfahrens. Es wurde in der Arbeit (Selle, Fedkiw, ByungMoon, Yingjie, & Jarek, June 2008) vorgestellt. Bei diesem Verfahren wird der vierte Schritt des Original BFECC Verfahrens ausgelassen. Stattdessen wird der finale Wert direkt im dritten Schritt berechnet. Grundgedanke ist hierbei, dass der vierte Backward-Schritt im ersten Schritt schon durchgeführt wurde und wiederverwendet werden kann.

Das Verfahren ist in Formel 6-14 dargestellt.

Formel 6-14: Verbessertes BFECC Verfahren.

$$\tilde{\varphi} = SingleStep(u, v, \varphi_n)$$

$$\bar{\varphi} = SingleStep(-u, -v, \tilde{\varphi})$$

$$\varphi_{n+1} = \tilde{\varphi} + \frac{\varphi_n - \bar{\varphi}}{2}$$

Im Gegensatz zum Original BFECC Verfahren wird durch diese Verbesserung des Verfahrens ein Rechenschritt gespart. Allerdings geht durch diese Änderung auch etwas an Genauigkeit verloren, da der Backward-Schritt auf den nicht korrigierten Daten durchgeführt, sondern wiederverwendet wird, um die finalen Daten zu berechnen.

Eine grafische Gegenüberstellung der Verfahren ist in Abbildung 6-20 zu finden.

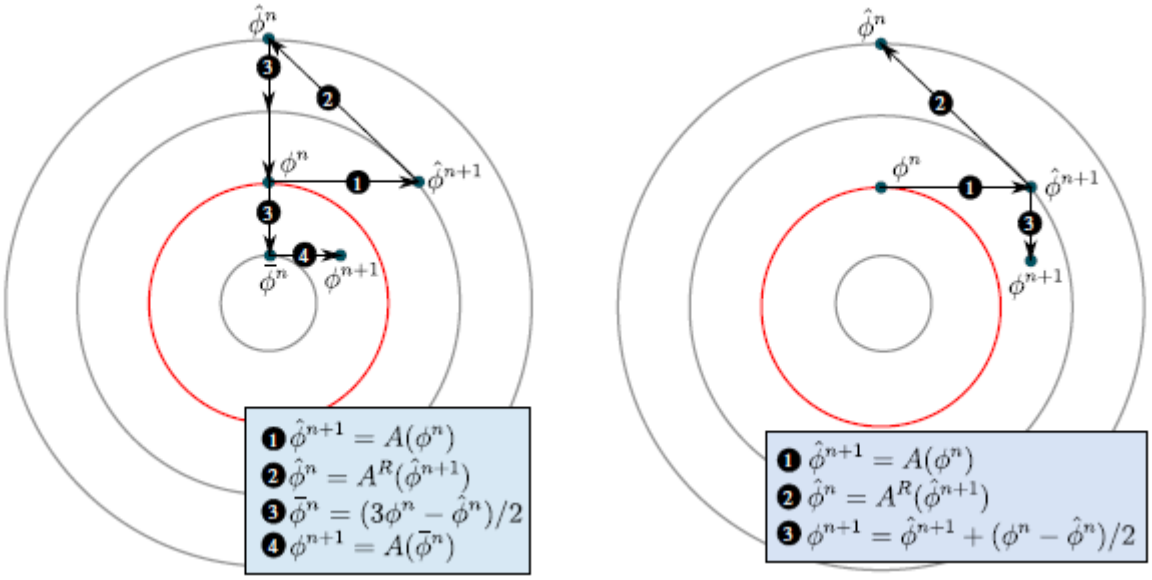


Abbildung 6-20: Gegenüberstellung des Original BFECC Verfahrens (links) mit dem verbesserten BFECC Verfahren (rechts). Quelle: (Selle, Fedkiw, ByungMoon, Yingjie, & Jarek, June 2008)

Das Problem, dass neue Extremwerte durch den Korrekturschritt erzeugt werden, ist wie bei dem BFECC-Original Verfahren vorhanden. Wie in Abbildung 6-21 zu sehen ist, ist das Auftreten der Extremwerte gespiegelt worden. Die Lösung ist wieder durch den Einsatz einer Limitierung erreichbar. Als Grenzwerte werden die gleichen Werte wie beim BFECC-Original Verfahren verwendet. Die genaue Betrachtung des Problems bei diesem Verfahren ist auch in der Arbeit (Selle, Fedkiw, ByungMoon, Yingjie, & Jarek, June 2008) zu finden.

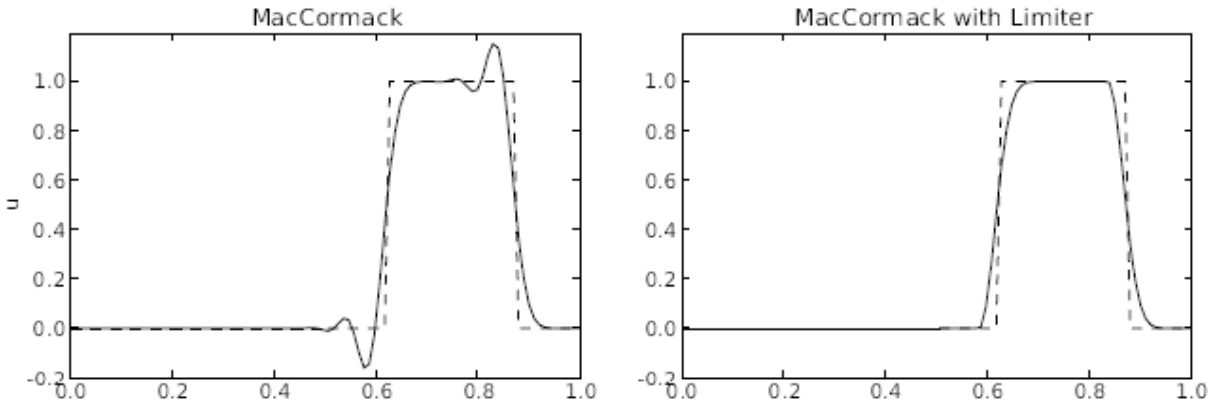


Abbildung 6-21: Eindimensionale Darstellung des Problems, dass neue Extremwerte durch den Korrekturschritt erzeugt werden. Links: Das BFECC Verfahren ohne die Limitierung. Rechts: Lösung des Problems durch eine Limitierung der Extremwerte. Quelle: (Selle, Fedkiw, ByungMoon, Yingjie, & Jarek, June 2008)

Wie bei dem Original BFECC Verfahren existieren einige Modifikationen des Verfahrens.

6.2.2.5.1 BFECC + Polynom 3

Bei dieser Modifikation wird statt der bilinearen Interpolation des Datenfelds eine Interpolation mittels eines Polynoms dritten Grades angewendet. Dies erzeugt eine höhere Genauigkeit der Datenwerte.

6.2.2.5.2 BFECC + Polynom 5

Diese Modifikation ist ähnlich der Modifikation mit Polynomen dritten Grades. Hier werden jedoch Polynome fünften Grades eingesetzt um die Genauigkeit weiter zu erhöhen.

6.2.2.5.3 BFECC + RK 4

Diese Modifikation setzt das klassische Runge-Kutta Verfahren zur Berechnung der neuen Position aus den Vektorfelddaten ein. Das Daten- und Vektorfeld wird bilinear interpoliert.

6.2.2.5.4 BFECC + Polynom 3 + RK 4 + Polynom 3

Diese Modifikation setzt die Verfahren ein, welche das beste Ergebnis ergeben haben.

Um die neue Position zu berechnen wird das klassische Runge-Kutta Verfahren eingesetzt. Die Daten werden mit der Polynom 3 Komponente interpoliert.

6.2.2.6 Bilinear + RK 4 + Polynom 3

Dieses Verfahren ist aus verschiedenen Komponenten zusammengesetzt. Hier wird die bilineare Interpolation eingesetzt um die Vektoren aus dem Vektorfeld auszuwerten. Die neue Position wird mithilfe des klassischen Runge-Kutta aus den Geschwindigkeitsvektoren berechnet. Die Daten aus dem Datenfeld werden mithilfe der Polynom 3 Komponente interpoliert.

6.2.2.7 Bilinear + RK 4 + Polynom 5

In diesem Verfahren wird das Vektorfeld wie im Bilinear + RK 4 + Polynom 3 Verfahren mit der bilinearen Interpolation interpoliert und die neue Position wird mit dem klassischen Runge-Kutta Verfahren berechnet. Die Daten werden jedoch mithilfe von Polynomen fünften Grades interpoliert, was eine höhere Genauigkeit der Daten erwarten lässt.

6.2.3 Forward-Verfahren

Unter Forward-Verfahren versteht man Verfahren, die die Information eines Pixels vorwärts transportieren. Hierbei muss der Wert allerdings auf die verschiedenen Zielpixel aufgeteilt werden. Anschließend müssen die Pixel normiert werden um das finale Ergebnis zu erhalten. Manche der Verfahren dieser Gruppe gehören zur Semi-Lagrang‘schen Sichtweise, andere zur Lagrang’schen.

Die Forward-Verfahren haben allerdings ein Problem. Bei einem Einfluss in das Vektorfeld werden keine Informationen von außerhalb in das Datenfeld hineingetragen, obwohl dies in diesem Fall korrekt wäre. Bei den Forward-Verfahren wird hier immer ein transparenter Wert angenommen. Bei den Backward-Verfahren entsteht dieses Problem nicht, da diese für jeden Pixel die entsprechenden Informationen besorgen. Liegen diese außerhalb, so wird entweder ein Wrapping der Basisdaten durchgeführt oder ein transparenter Wert angenommen.

Wie bei den Backward-Verfahren existieren hierzu verschiedene Verfahren, die diesen Ansatz umsetzen.

6.2.3.1 Forward RK 4

Dies ist das einfachste Forward-Verfahren. Hier wird die Pixelinformation in jedem Schritt weitertransportiert und anschließend auf die vier Nachbarpixel aufgeteilt. Dieser Ansatz ist ein iteratives Verfahren, welches zur Semi-Lagrang'schen Sichtweise gehört.

Die Position des Pixels wird durch das klassische Runge-Kutta Verfahren berechnet. Dieses Verfahren ist in der Lage, dem Vektorfeld ausreichend genau zu folgen.

Die Aufteilung des Pixelwerts erfolgt hierbei mithilfe eines umgekehrten bilinearen Ansatzes. Es wird aufgrund der Position des aufzuteilenden Pixels berechnet auf welchen der Nachbarpixel wie viel des Werts addiert wird. Ist diese Aufteilung für alle Pixel erledigt worden, wird jeder Pixel mit seinem erhaltenen Gesamtgewicht normiert. Die genaue Berechnung, wie viel auf welchen Wert aufgeteilt wird, ist in Formel 6-15 und Abbildung 6-22 zu sehen.

Formel 6-15: Berechnung der Aufteilung des Wertes c auf die vier Nachbarpixel.

$$cTL = (1-x)(1-y)c$$

$$cTR = x(1-y)c$$

$$cBL = (1-x)yc$$

$$cBR = xyc$$

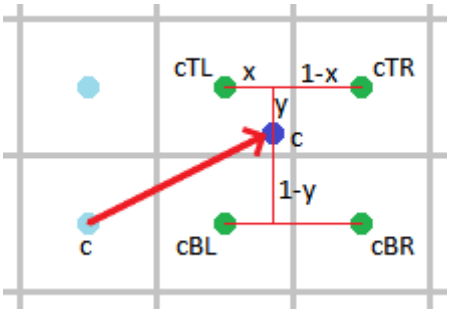


Abbildung 6-22: Graphische Darstellung der Aufteilung des Wertes auf die vier Nachbarpixel.

Dieses Verfahren ist sehr schnell zu berechnen, da die Aufteilung der einzelnen Werte nur sehr wenige Rechenschritte benötigt.

6.2.3.2 Forward RK 4 Particle Kernel

In diesem Verfahren werden die einzelnen Pixel als Partikel gesehen. Es wird also die Lagrang'sche Sichtweise verwendet. Diese Partikel besitzen einen Wert und eine Position im Bild.

Im ersten Schritt wird für jeden Pixel des Quellbildes ein Partikel erzeugt, der die Position und den Wert dieses Pixels enthält. Vollständig transparente Partikel müssen dabei nicht berücksichtigt werden.

Bei diesem Verfahren wird für jeden Partikel der gesamte Pfad berechnet. Da die Partikel eine Position besitzen, kann diese Position einfach verändert werden, um den Partikel mit dem Vektorfeld weiterzubewegen. Jedes Partikel enthält in seiner Position den gesamten Pfad von seinem Startpunkt zu seiner aktuellen Position. Diese Position wird für jeden weiteren Schritt mithilfe des klassischen Runge-Kutta Verfahrens um einen Schritt erweitert. Das Ergebnis eines Schrittes ist also identisch mit

der kompletten Berechnung des Pfades. Abbildung 6-23 demonstriert dieses Verfahren anhand von drei Partikeln.

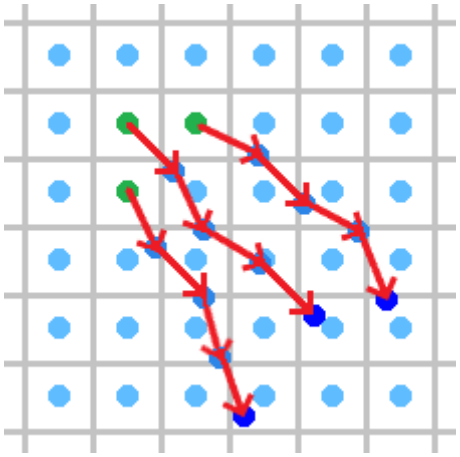


Abbildung 6-23: Forward RK 4 Particle Kernel-Verfahren mit vier Schritten. Die neuen und alten Partikelpositionen sind hier markiert.

Um das Ergebnisbild aus der Partikelwolke herzustellen wird ein Kernel eingesetzt. Dieser Kernel teilt die Daten in dem Partikel auf die entsprechenden Gitterpunkte auf. Als Gewichtsfunktion wird Formel 6-7 eingesetzt. Anschließend müssen die Werte entsprechend der Gewichte normiert werden.

Es ist egal, ob der Kernel auf einem Pixel liegt und die Partikel aufsummiert oder ob der Kernel auf einem Partikel liegt und den Wert auf die Pixel aufsummiert. Beide Varianten ergeben dasselbe Resultat. Dies ist in Abbildung 6-24 und Abbildung 6-25 mittels eines eindimensionalen OBDA-Beweises gezeigt. Hier sind beide Varianten gegenübergestellt. Die verwendeten Kernel sind farbig unter den einzelnen Partikeln dargestellt. In Abbildung 6-24 wird der Kernel auf die einzelnen Pixel angewendet und in Abbildung 6-25 auf die einzelnen Partikel.

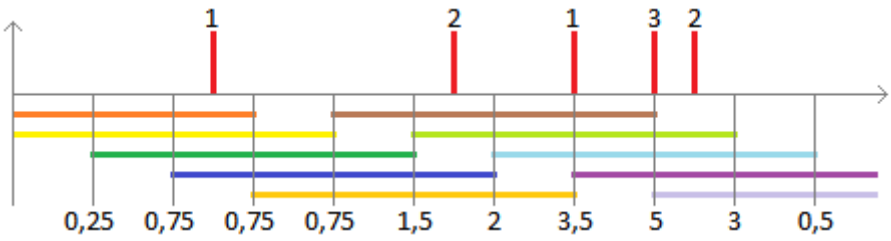


Abbildung 6-24: Der Kernel liegt auf einem Pixel und summiert die Partikel entsprechend der Gewichtsfunktion auf. Die roten Markierungen stellen hier die vorhandenen Partikel dar. Der Wert der Partikel ist über den Markierungen angegeben. Die Zahlen unter den einzelnen Pixeln (grau) sind die Werte, welche diesen zugewiesen werden.

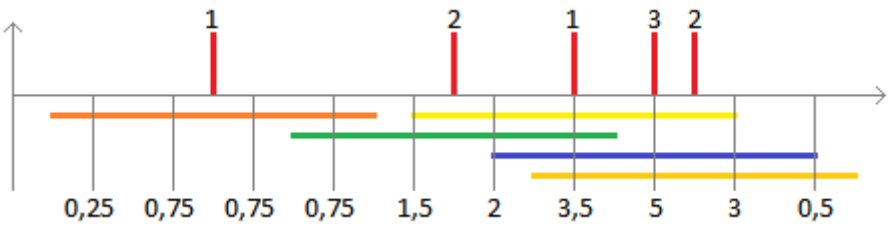


Abbildung 6-25: Der Kernel liegt auf einem Partikel und addiert den Wert entsprechend der Gewichtsfunktion zu den einzelnen Pixeln. Die roten Markierungen stellen hier die vorhandenen Partikel dar. Der Wert der Partikel ist über den Markierungen angegeben. Die Zahlen unter den einzelnen Pixeln (grau) sind die Werte, welche diesen zugewiesen sind.

Bei diesem Vergleich ist ein Kernel mit Radius $r = 2$ verwendet worden. Als Gewichtsfunktion wird ein linearer Abfall innerhalb des Kernels verwendet. Diese Funktion ist in Abbildung 6-26 zu sehen.

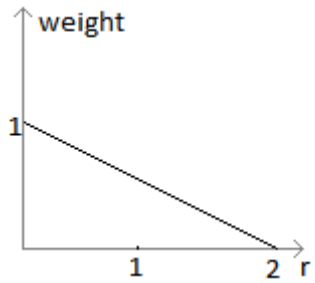


Abbildung 6-26: Verwendete Gewichtsfunktion im OBDA-Beweis.

Bei beiden Varianten sind die Ergebnisse der einzelnen Pixel gleich. Damit ist gezeigt, dass es egal ist, ob der Kernel auf einem Partikel oder einem Pixel liegt.

In diesem Verfahren ist es jedoch geschickter den Kernel auf einen Partikel anzuwenden. Damit können die einzelnen Partikel ihre Werte direkt in ein Datenfeld schreiben und müssen nicht zwischengespeichert werden. Das Suchen nach den Partikeln, welche einen Pixel verändern, entfällt somit ebenfalls.

Partikel, bei denen der gesamte Kernel den betrachteten Bereich des Ergebnisses verlassen hat und sich die Position nicht mehr verändert, können entfernt werden, weil diese keinen Einfluss mehr auf das Ergebnis haben. Da hier der gesamte Pfad eines Partikels berechnet wird, entfällt der Fehler, der sich bei iterativen Verfahren aufsummiert.

Allerdings können bei diesem Verfahren Löcher im Ergebnis auftreten. Dies sind Stellen, denen keine Information zugewiesen wird. Dies kann passieren, da in diesem Verfahren keine neuen Partikel erzeugt werden und es somit möglich ist, dass sich die Partikel an bestimmten Stellen im Vektorfeld sammeln. An solchen Stellen ist die Partikeldichte sehr hoch, während an anderen Stellen die Dichte entsprechend abnimmt.

Die Rechenzeit kann sich mit steigender Schrittzahl in diesem Verfahren verkürzen, da die Anzahl der Partikel abnehmen, jedoch nicht zunehmen kann.

6.2.3.3 Forward Yu

Das Verfahren Forward Yu ist ein Verfahren, welches für die Berechnung des advektierten Bildes eine Gitterstruktur als Lösung einsetzt. Das Verfahren wurde in der Arbeit (Qizhi, Neyret, Bruneton, & Holzschuch, 2010) vorgestellt. Dieses Verfahren ist kein iteratives Verfahren. Jeder Schritt bezieht sich auf die Quelldaten. Hierdurch entsteht kein sich aufsummierender Fehler wie bei iterativen Verfahren. Da in diesem Verfahren neue Partikel an Stellen, die von keinem Partikel mehr erreicht werden, erzeugt werden, entstehen keine Löcher in diesem Verfahren.

Das Verfahren funktioniert in mehreren Schritten. Diese Schritte berechnen das finale Bild aufgrund einer Poisson-Verteilung von Partikeln auf dem Ergebnis. Jeder dieser Partikel besitzt ein Gitter, welches gerendert wird um das finale Ergebnis zu erzeugen. Durch die Poisson-Verteilung wird garantiert, dass keine Löcher im Ergebnis entstehen.

Die Poisson-Verteilung kann mit dem Verfahren aus der Arbeit (Dunbar & Humphreys, 2006) berechnet werden. Dieses Verfahren erzeugt eine homogene Verteilung der Partikel mit wenig

Rechenaufwand. Dies ist in Abbildung 6-27 anhand der grünen Punkte zu sehen. Die minimale Distanz jedes Pixels zu einem Partikel ist höchstens der Radius r der Partikel. Je kleiner dieser Radius r gewählt wird, desto mehr Partikel werden erzeugt und desto genauer ist das berechnete Ergebnis. Allerdings sinkt auch die Performance, da mehr Partikel berechnet werden müssen.

Jeder dieser Partikel besitzt ein Gitter. Das Zentrum dieses Gitters liegt auf dem erzeugten Partikel. Dieses Gitter muss mindestens den gesamten Kreis eines Partikels abdecken. Die Größe des Gitters muss also mindestens $2r$ betragen. Damit die Abdeckung des Partikels auch bei leichten Bewegungen des Gitters gegeben ist, wird ein etwas größeres Gitter gewählt. Diese Vergrößerung kann durch einen Faktor β ausgedrückt werden. Die Größe des Gitters errechnet sich dann mit der Formel $(2 + \beta)r$. Ein β von 0,6 liefert gute Ergebnisse.

Die Auflösung des Gitters hat auch Einfluss auf den Genauigkeitsgrad des Verfahrens. Eine höhere Auflösung geht allerdings auch zu Lasten der Performance. In Abbildung 6-27 ist ein Gitter auf einem Partikel zu sehen.

Die Vertices des Gitters speichern die Texturkoordinaten auf der Originaltextur.

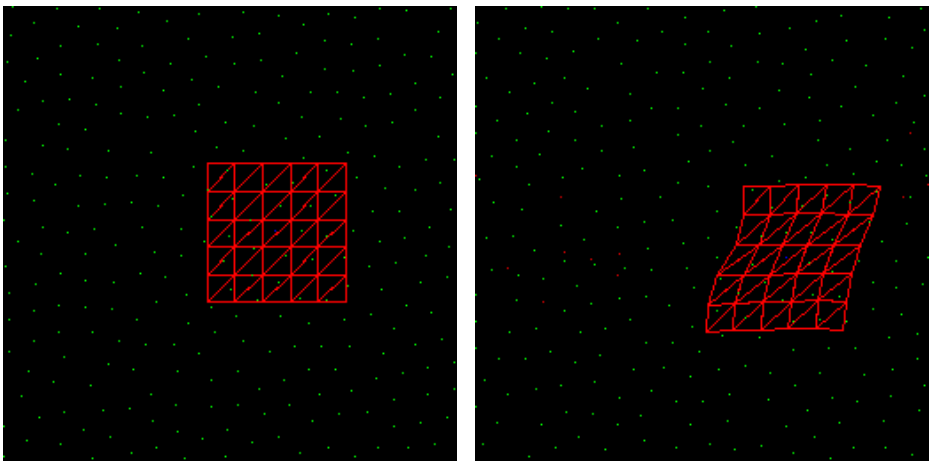


Abbildung 6-27: Erzeugte Partikelverteilung (grün) mit Radius r = 10. Jeder Pixel ist höchstens r von einem Partikel entfernt. Ein Gitter eines Partikels wird dargestellt. Links: Keine Advektion. Rechts: Advektion mit 2 Schritten.

In jedem Schritt werden auf den Partikeln der Poisson-Verteilung folgende drei Schritte durchgeführt:

1. Die einzelnen Partikel werden entsprechend des Vektorfelds advektiert. Hierzu wird die Euler'sche Methode eingesetzt.
2. Partikel in Gebieten, in denen die Dichte der Partikel zu groß wird, werden entfernt.
3. Neue Partikel werden in Gebieten erzeugt, die von keinem Partikel mehr erreicht werden.

Um Endlosschleifen zwischen Erzeugen und Entfernen von Partikeln zu vermeiden, muss beim Entfernen der Partikel ein anderes Kriterium angewendet werden als beim Erzeugen. Ein Partikel wird entfernt, wenn seine Distanz zu einem anderen Partikel kleiner als $(1 - \alpha)r$ ist. 0,25 ist hier ein guter Wert für α.

Um einen sichtbaren Effekt bei erzeugten oder entfernten Partikeln zu vermeiden, werden diese langsam ein- oder ausgeblendet.

Als nächstes müssen die Gitter in jedem Schritt berechnet werden. Hierzu werden für jedes Gitter die folgenden Berechnungen durchgeführt:

1. Die Vertices des Gitters müssen entsprechend des Vektorfelds advektiert werden. Auch hier wird die Euler'sche Methode eingesetzt.
2. Für jedes in diesem Schritt neu eingefügte Partikel muss ein neues Gitter erzeugt werden. Die Position dieses Gitters wird hierbei zufällig auf dem Originalbild gewählt.
3. Partikel, deren Gitter nicht mehr den Qualitätsanforderungen entsprechen, müssen entfernt werden.

Die Kriterien zum Entfernen eines Partikels aufgrund der Qualitätsanforderungen sind:

1. Das Gitter deckt nicht mehr den gesamten Kreis des Partikels ab.
2. Die Verzerrung des Gitters wird zu groß.
3. Das Gitter ist in sich gefaltet.

Die Verzerrung eines Dreiecks wird mit Formel 6-16 berechnet. Diese Formeln verwenden γ_{min} und γ_{max} der Jacobian-Matrix der Transformation zwischen dem Originaldreieck und dem advektierten Dreieck. γ_{min} und γ_{max} repräsentieren die kleinste und größte Skalierung der Transformation und wurden in der Arbeit (Sorkine, Cohen-Or, Goldenthal, & Lischinski, 2002) eingeführt. Die Verzerrung Q_t eines Dreiecks ist 1 für ein unverzerrtes Dreieck und 0 wenn die Verzerrung größer als δ_{max} ist. δ_{max} ist die maximale erlaubte Verzerrung eines Dreiecks. Ein Wert von $\delta_{max} = 0{,}5$ ergibt gute Ergebnisse. Die Verzerrung Q_v eines Vertex des Gitters ist der Durchschnitt der Verzerrung der Dreiecke, die diesen Vertex verwenden. Die Verzerrung des Gitters ist zu groß sobald gilt $Q_v < \frac{1}{2}$. Dies gibt ein Mindestmaß an Qualität für das Ausblenden des Partikels.

Formel 6-16: Berechnung der Verzerrung eines Dreiecks.

$$\delta_t = max\left(\gamma_{max}, \frac{1}{\gamma_{min}}\right)$$

$$Q_t = max\left(\frac{\delta_{max} - \delta_t}{\delta_{max} - 1}, 0\right)$$

Da es bei einem Gitter vorkommen kann, dass einzelne Vertices außerhalb des Vektorfelds liegen, sollte darauf geachtet werden, dass das Vektorfeld auch außerhalb des definierten Bereichs Daten liefert. Dies ist nützlich, damit die Verzerrung des Gitters nicht zu früh eintritt.

In Abbildung 6-27 rechts ist ein advektiertes Gitter zu sehen. Hierbei wurden sowohl das Gitter, als auch die Partikel zwei Schritte lang berechnet.

Um das finale Bild zu erzeugen, werden die Gitter aller Partikel gerendert. Die Texturkoordinaten werden verwendet um die Werte an den entsprechenden Gitterpositionen zu erhalten. Zusätzlich wird für jeden Vertex ein Gewicht berechnet. Dieses Gewicht wird verwendet um das Blending dieses Vertices in dem finalen Bild zu bestimmen. Das Gewicht wird mit Formel 6-17 berechnet.

Formel 6-17: Berechnung des Gewichts eines Vertices. V ist die Position des Vertices. P die Position des Partikels und r der Radius des Partikels. fade ist der Blendfaktor beim Ein- und Ausblenden des Partikels.

$$w_v = K_s(V) * fade$$

$$K_s(V) = \left(1 - \frac{\|V - p\|}{r}\right) d_v Q_v$$

$$d_v = \begin{cases} 0, & V \in Rand\ des\ Gitters \\ 1, & sonst \end{cases}$$

In Abbildung 6-28 ist zu sehen, wie sich das Gewicht auf ein einzelnes Gitter auswirkt.

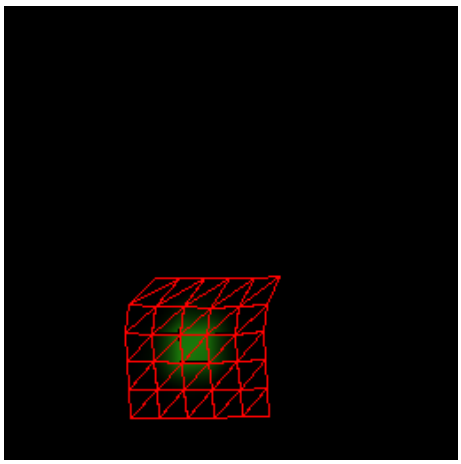


Abbildung 6-28: Darstellung des Gewichts bei einem einzelnen Gitter. Nur der innere Bereich des Gitters wird dargestellt. Am Rand ist das Gewicht 0.

Durch die Verwendung der Texturkoordinaten entstehen keine Fehler in der Textur, da diese sich auf die Quelldaten beziehen und somit kein iterativer Fehler vorhanden ist.

Nachdem alle Gitter auf das finale Bild gerendert sind, muss jeder Pixel noch bezüglich des Gewichts normiert werden.

7 Analyse

Die Analyse der vorgestellten Verfahren bezieht sich auf zwei Teilbereiche. Einerseits werden die verwendeten Komponenten untersucht. Andererseits werden die kompletten Verfahren, in denen verschiedene dieser Komponenten zum Einsatz kommen, analysiert. Bei der Analyse wird eine statische Schrittweite von $h = 1$ verwendet. Es werden verschiedene Methoden für die Analyse eingesetzt.

Als Datenfeld der Analyse wird ein Farbbild eingesetzt.

7.1 Analysemethoden

Es kommen mehrere Analysemethoden zum Einsatz. Welche Methode angewendet wird hängt vom zu untersuchenden Objekt ab. Hierbei wird unterschieden, ob nur Komponenten oder das gesamte Verhalten der Verfahren untersucht wird. Die einzelnen Analysemethoden untersuchen jeweils einen Bereich der Verfahren.

7.1.1 Differenzbilder

Differenzbilder sind die einfachste Analysemethode. Hier wird die Differenz zwischen zwei Bildern berechnet. Aufgrund dieser Differenz kann man den Unterschied zwischen zwei Verfahren deutlich sichtbar machen. Mithilfe eines Referenzverfahrens wird also abgeschätzt wie nahe das betrachtete Verfahren diesem Referenzverfahren kommt. Es zeigt auch welches Verfahren im Vergleich näher an der Referenz liegt.

Die Differenz zwischen zwei Bildern wird pro Pixel berechnet. Das Ergebnis wird wiederum in einem Bild pro Pixel gespeichert. Es wird der absolute Wert der Differenz gespeichert. Die Berechnung der Differenz ist in Formel 7-1 zu sehen.

Formel 7-1: Differenz zwischen zwei Bildern (pic_1 und pic_2). x Ist die Position des Pixels.

$$dif(x) = |pic_1(x) - pic_2(x)|$$

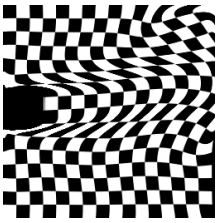
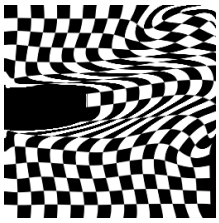
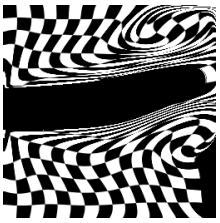
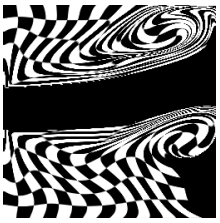
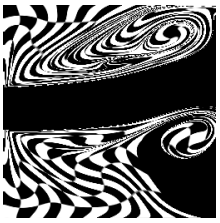
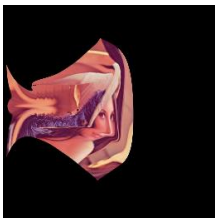
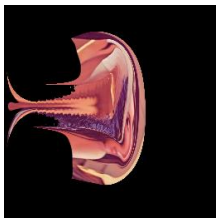
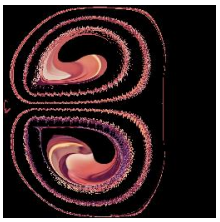
Um einen aussagekräftigen Vergleich der Verfahren zu erhalten, werden Referenzbilder in verschiedenen Schritten benötigt. Diese Referenzbilder werden mithilfe eines Referenzverfahrens berechnet. Als Referenzverfahren wird das Complete RK 4 Verfahren eingesetzt. Dieses Verfahren berechnet den kompletten Pfad eines Pixels bis zu seinem Ursprung.

Diese Analysemethode wird auf den Vektorfeldern high_viscosity.dat und box1.dat eingesetzt. Diese Vektorfelder sind in Abbildung 4-1 rechts oben und links oben dargestellt und besitzen 256 * 256 Datenpunkte. Als Datenfeld wird ein Bild eingesetzt, das auf einem 256 * 256 Pixel großen Datenfeld definiert ist. Die hier eingesetzten Bilder sind in Abbildung 7-1 zu sehen. Die Auswertung der Verfahren findet nach 5, 10, 25, 50 und 100 Schritten statt. In Tabelle 7-1 sind die Referenzbilder nach der entsprechenden Schrittzahl zu finden. Bei der Analyse der einzelnen Verfahren werden Differenzbilder von diesen Bildern berechnet.

Bei Differenzbildern gilt, dass jeder Farbwert einen Fehler im Verfahren bedeutet. Ein komplett schwarzes Bild ist hier ideal, da dieses Verfahren dann Ergebnisse, welche identisch zu dem Referenzverfahren sind, errechnet.

Abbildung 7-1: Originalbilder, welche bei der Analyse eingesetzt werden.

Tabelle 7-1: Referenzbilder nach den entsprechenden Schritten. Oben: Vektorfeld high_viscosity.dat. Unten: Vektorfeld box1.dat.

5	10	25	50	100
				
				

7.1.2 Reversibilität

Unter Reversibilität versteht man erst das Advektieren eines Bildes bis zu einem gewissen Schritt und anschließend diese Advektion umzukehren, um wieder auf das Ausgangsbild zurück zu kommen. Ideal wäre es, das Ausgangsbild wieder zu erhalten. Besonders an Stellen, an denen sich das Vektorfeld räumlich ändert, entstehen häufig Fehler. Diese Fehler werden hierdurch sichtbar gemacht.

Die Reversibilität wird anhand der Bilder aus Abbildung 7-1 untersucht. Diese werden auf die Vektorfelder box3.dat und box1.dat angewandt, welche in Abbildung 4-1 rechts unten und links oben zu sehen sind. Die Advektion wird bis zu den Schritten 5, 10, 25, 50 und 100 berechnet. Anschließend wird die Reversibilität berechnet. Die Ergebnisse werden als Bild dargestellt.

Es entstehen allerdings Fehler an Stellen, an denen Informationen aus dem Vektorfeld hinausfließen, da diese nicht gespeichert werden. Bei solchen Vektorfeldern ist die Reversibilität nicht komplett berechenbar.

7.1.3 Optischer Fluss

Der optische Fluss ist die Bewegung der Punkte im Bild. Diese Bewegung kann zwischen zwei Bildern berechnet werden und sollte der Bewegung aus dem Vektorfeld entsprechen.

Um den optischen Fluss zu berechnen, wird die OpenCV-Library eingesetzt. Der Fluss wird an Punkten, die in einem Gitter angeordnet sind, berechnet. Diese sind in Abbildung 7-2 zu sehen. Die Bewegung dieser markierten Punkte wird in einem neuen Bild dargestellt.

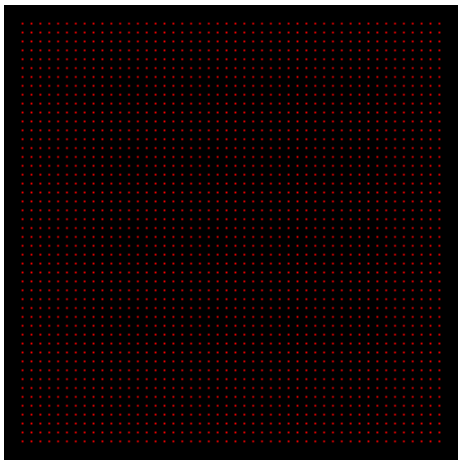


Abbildung 7-2: Stellen, an denen der optische Fluss berechnet wird.

Die Berechnung findet zwischen den Schritten 0 und 1, 25 und 26 und 100 und 101 statt.

Da diese Analysemethode die Bewegung der Punkte aufgrund markanter Stellen in den Quelldaten berechnet, wird eine Rauschtextur als Quellbild eingesetzt. Dies geschieht, weil hier die markanten Stellen gut identifiziert werden können. Die Rauschtextur ist in Abbildung 7-3 zu sehen. Der optische Fluss wird auf den Vektorfeldern box3.dat und box1.dat, welche in Abbildung 4-1 rechts unten und links oben zu sehen sind, berechnet.

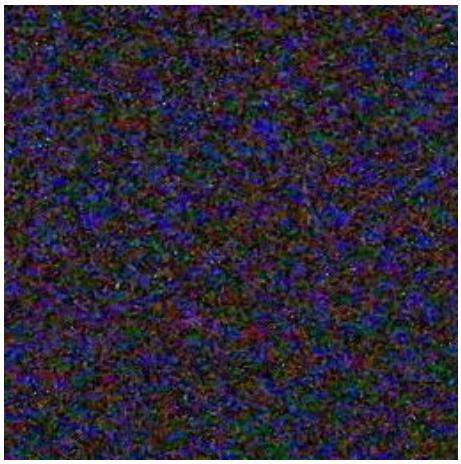


Abbildung 7-3: Rauschtextur, die für die Analyse des optischen Flusses und des Spektrums eingesetzt wird.

Da diese Analysemethode jedoch auf einer optischen Analyse basiert, kann es vorkommen, dass bei vereinzelten Punkten ein falsches oder kein Äquivalent gefunden wird.

7.1.4 Spektrum

Das Spektrum eines Bildes sollte durch eine Advektion konstant bleiben. Diese Analysemethode untersucht dieses Verhalten. Hierzu kann das Spektrum in einem beliebigen Schritt berechnet werden. Das wird anschließend in einem Bild dargestellt. Ein typisches Spektrum ist in Abbildung 7-4 zu sehen.

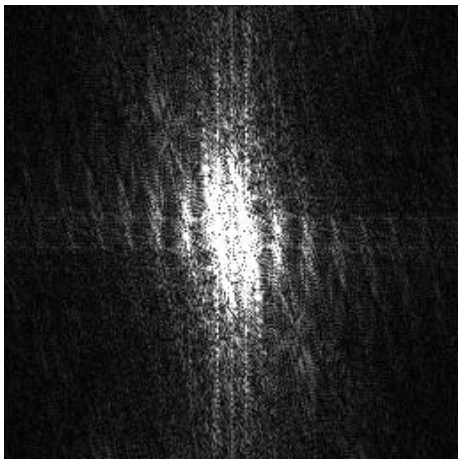


Abbildung 7-4: Typisches Spektrum eines Graustufenbildes.

Um das Spektrum zu berechnen, wird eine 2D FFT (Fast Fourier Transformation) eingesetzt. Die Berechnung findet hierbei auf jedem Farbkanal separat statt. Die Ergebnisse der einzelnen Farbkanäle sind in dem entsprechenden Kanal dargestellt.

Der Idealfall ist, dass sich das Spektrum trotz Advektion nicht ändert.

Das Spektrum wird in den Schritten 5, 25 und 100 berechnet. Als Quelldaten werden die Bilder aus Abbildung 7-3 und Abbildung 7-1 links verwendet. Diese werden auf den Vektorfeldern box3.dat und box1.dat, welche in Abbildung 4-1 rechts unten und links oben zu sehen sind, berechnet. Die berechneten Ergebnisse sind in den Ergebnissen der einzelnen Verfahren angegeben. Auf diesen Ergebnissen wird anschließend das Spektrum berechnet.

Die Spektren der beiden Quellbilder sind in Abbildung 7-5 zu sehen. Im Idealfall ändern sie sich während der Advektion nicht.

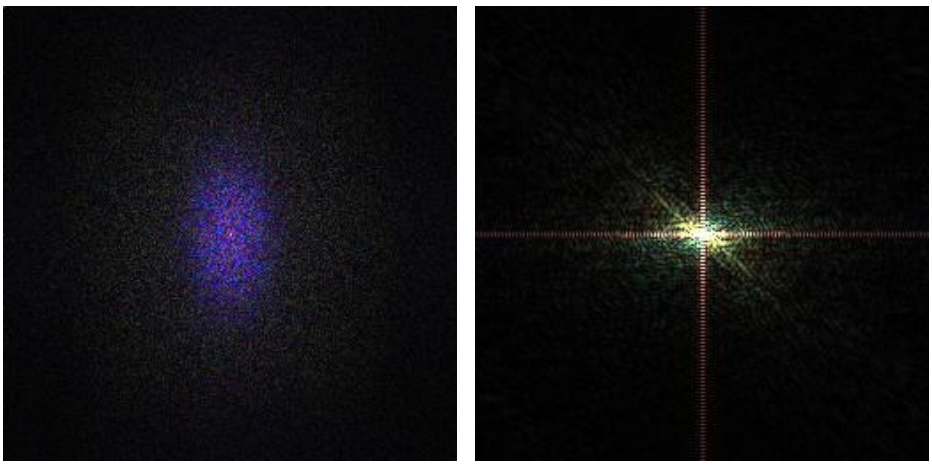


Abbildung 7-5: Spektrum der Quellbilder. Links: box3.dat als Vektorfeld mit einer Rauschtextur (Abbildung 7-3) als Datenfeld. Rechts: box1.dat als Vektorfeld mit einem normalen Bild (Abbildung 7-1 Rechts) als Datenfeld.

7.1.5 Pfaddarstellung

Die Pfaderstellung bezeichnet ein Verfahren, bei dem eine gewisse Anzahl an berechneten Bildern übereinander geblendet wird. Mithilfe dieser Methode kann festgestellt werden, wie gut ein Verfahren die Vektorfelddaten integriert.

Diese Analysemethode wird auf ein kreisförmig generiertes Vektorfeld angewandt. Das Vektorfeld rotiert hier um den Mittelpunkt des Feldes. Als Bild wird ein einfacher Punkt genommen, welcher in Abbildung 7-6 zu sehen ist.

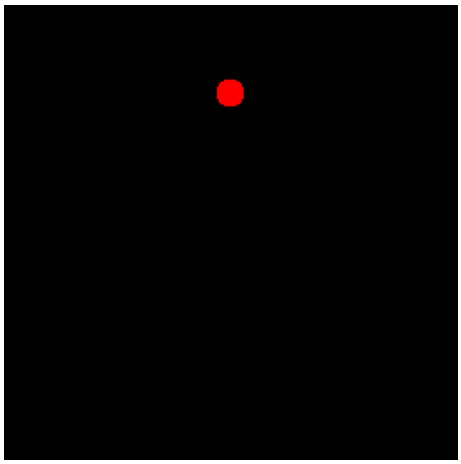


Abbildung 7-6: Quellbild für die Pfadanalyse.

Wenn die Vektorfelddaten ideal integriert werden, müsste der Punkt nach einigen Schritten wieder auf den Punkt im ersten Bild treffen.

Um den Pfad sichtbar zu machen, werden die einzelnen Bilder, welche nach den Berechnungsschritten entstehen, übereinander geblendet. Der Verlauf des Pfades sagt aus wie genau dem Verlauf des Vektorfeldes gefolgt wird.

7.1.6 Performance

Zur Performanceanalyse der Verfahren wird eine integrierte Messeinheit verwendet. Bei dieser Messeinheit werden sämtliche Ausgaben der berechneten Daten unterbunden. Es wird also nur die eigentliche Berechnungszeit, bis das Ergebnis im Speicher liegt, gemessen.

Die Messung der Vergleichswerte erfolgt auf einem Intel Core2 Quad mit 2,7 GHz auf einem Kern. Als Vektorfeld wird das Feld high_viscosity.dat eingesetzt. Dieses ist in Abbildung 4-1 rechts oben zu finden. Als Datenfeld wird das Bild aus Abbildung 7-1 links eingesetzt.

Die Performancemessung findet über 100 Schritte statt. Ungefähr ab 24 FPS (Frames per Second) ist die Berechnung schnell genug um echtzeitfähig zu sein.

7.2 Komponenten

Die Komponenten sind die Grundlage der Verfahren. Die eigentlichen Verfahren setzen sie in bestimmten Bereichen ein. Sie sind in den Verfahren innerhalb einer Kategorie austauschbar. Aus diesem Grund können die einzelnen Komponenten unabhängig von dem eigentlichen Verfahren analysiert werden.

7.2.1 Dateninterpolation

Diese Komponenten beziehen sich auf die Interpolation der Daten aus dem Daten- oder Vektorfeld. Hier wird nur die Interpolation auf dem Datenfeld analysiert. Da die Interpolation auf dem Vektorfeld auf identische Weise erfolgt, sind die Ergebnisse hieraus auch auf das Vektorfeld übertragbar.

Um eine Komponente zu analysieren, wird ein Verfahren benötigt, das diese Komponente enthält. Die Komponente wird anschließend in dem Verfahren ausgetauscht um vergleichbare Werte zu

erhalten. Als Verfahren wird ein Backward-Verfahren eingesetzt. Es wird die bilineare Interpolation verwendet um die Geschwindigkeiten aus dem Vektorfeld zu interpolieren. Die Berechnung eines Punktes wird mit der Euler'schen Methode durchgeführt. Die Interpolation der Farbinformationen erfolgt mit der zu analysierenden Komponente.

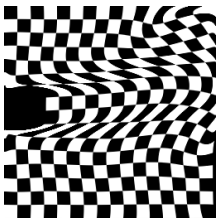
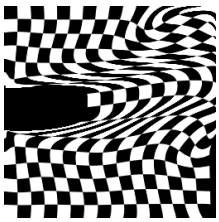
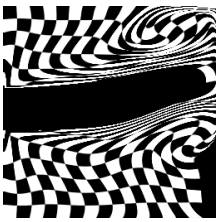
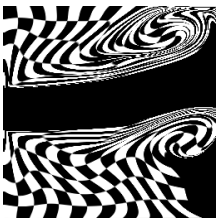
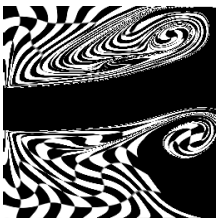
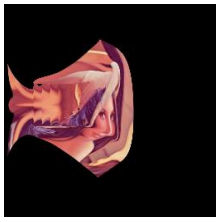
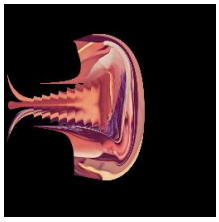
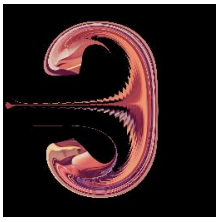
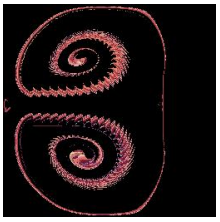
Die Ergebnisse dieser Komponenten basieren auf den Quellbildern aus Abbildung 7-1 und Abbildung 7-3 und sind auf den Vektorfeldern high_viscosity.dat, box1.dat und box3.dat berechnet. Die Vektorfelder sind in Abbildung 4-1 rechts oben, links oben und rechts unten zu sehen. Die Ergebnisse werden für die Schritte 5, 10, 25, 50 und 100 berechnet.

Die Komponenten werden mit den Analysemethoden zur Reversibilität, dem Spektrum, des optischen Flusses und der Differenz zu dem Referenzverfahren analysiert. Die Performance der Komponente wird mithilfe der eingebauten Messeinheit berechnet.

Bei der Berechnung der Differenzbilder wird für die Analyse der Komponenten allerdings ein anderes Referenzverfahren verwendet. Das Complete RK 4 Verfahren liefert bei der Analyse der Komponenten keine guten Differenzbilder, da das Verfahren, welches für die Analyse der Komponenten eingesetzt wird, die Integration des Vektorfeldes mithilfe der Euler'schen Methode durchführt. Dies führt wiederum zu hohen Differenzen aufgrund der verschiedenen Positionsberechnung der Pixel. Aus diesem Grund wird hier als Referenzverfahren das Complete Verfahren eingesetzt. Hierdurch wird die Position der Pixel mit der gleichen Methode berechnet. Die Unterschiede in der Position sind nicht so groß und haben somit keine so große Wirkung auf die Differenzbilder. Der qualitative Unterschied der einzelnen Komponenten ist hierdurch in den Differenzbildern besser zu erkennen. In Tabelle 7-2 sind die verwendeten Referenzbilder zu sehen.

Die Ergebnisse des ersten und zweiten Vektorfeldes werden eingesetzt um die Differenzbilder zu berechnen. Die Ergebnisse des zweiten Vektorfeldes werden auch noch bei der Berechnung der Reversibilität und bei der Berechnung des Spektrums verwendet. Jeweils das zweite Vektorfeld der Analyse verwendet diese Ergebnisse. Die Ergebnisse des dritten Vektorfeldes werden bei der Berechnung des optischen Flusses im ersten Vektorfeld verwendet.

Tabelle 7-2: Referenzbilder um die Komponenten zu analysieren.

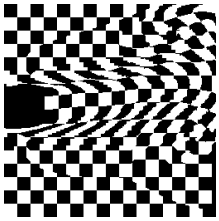
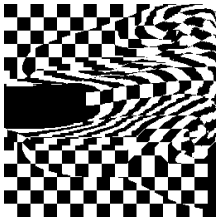
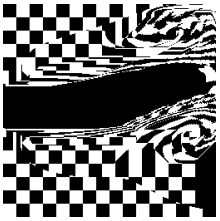
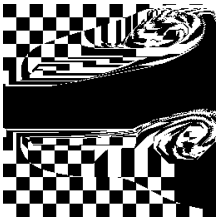
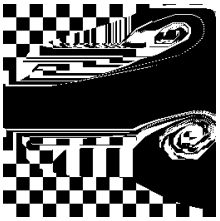
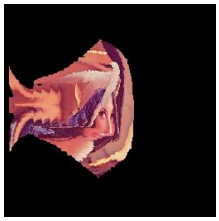
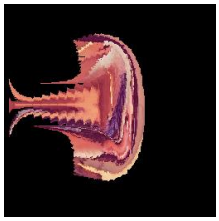
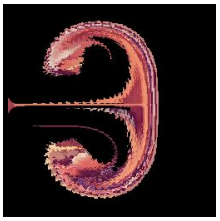
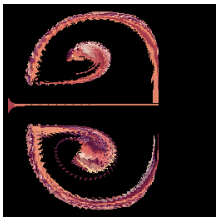
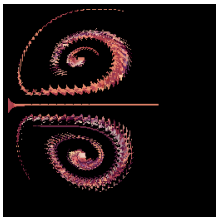
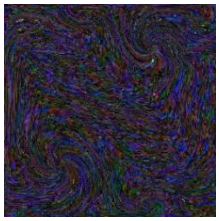
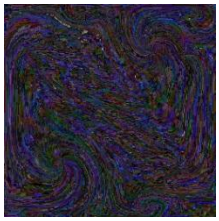
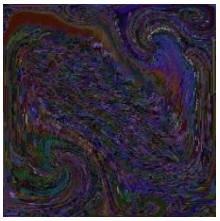
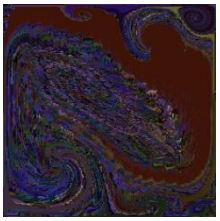
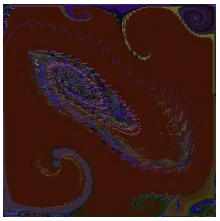
5	10	25	50	100
				
				

7.2.1.1 Nearest Neighbour

Diese Komponente wird in Kapitel 6.1.1.1 beschrieben. In dieser Komponente wird der nächste gegebene Datenpunkt bei der Auswertung zurückgeliefert.

7.2.1.1.1 Ergebnisse

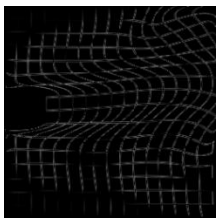
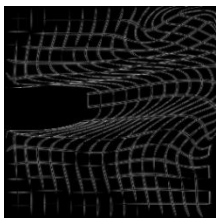
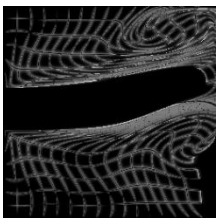
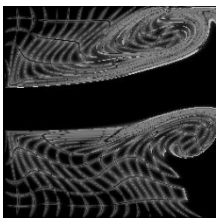
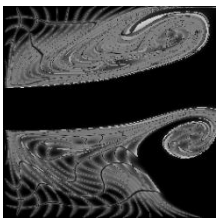
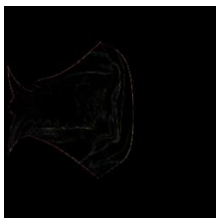
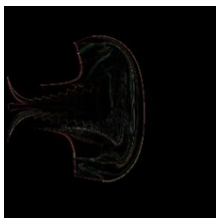
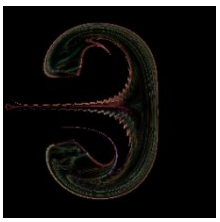
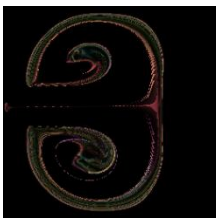
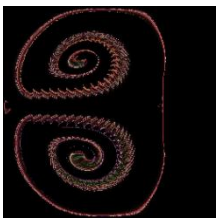
Tabelle 7-3: Ergebnisse der Nearest Neighbour Komponente.

5	10	25	50	100
				
				
				

In dem Schachbrettmuster ist besonders gut zu erkennen, dass die einzelnen Zellen nach einigen Schritten ihre Form verlieren und gestreckt werden. Bei der Berechnung des Vektorfelds box3.dat ist zu beobachten, dass ab 50 Schritten die braune Farbe das Ergebnis dominiert.

7.2.1.1.2 Differenzbilder

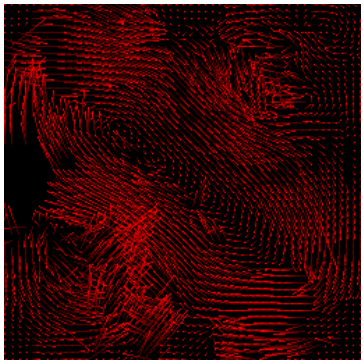
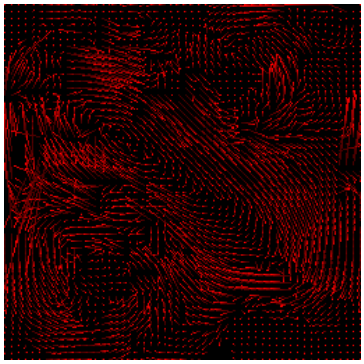
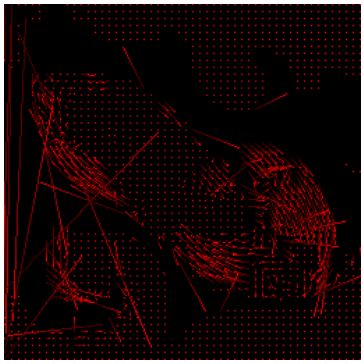
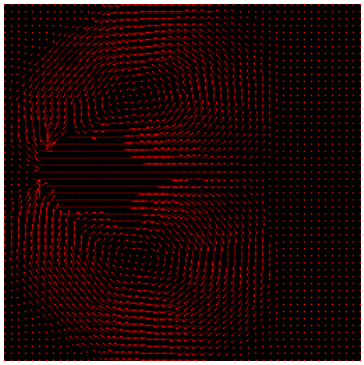
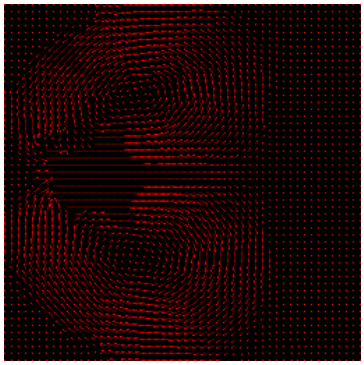
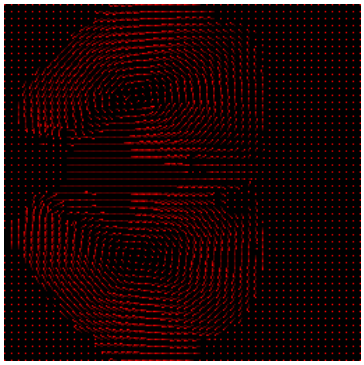
Tabelle 7-4: Differenzbilder der Nearest Neighbour Komponente zu den Referenzbildern.

5	10	25	50	100
				
				

Bei diesen Ergebnissen ist zu erkennen, dass in dieser Komponente die Fehler mit steigender Schrittzahl zunehmen.

7.2.1.1.3 Optischer Fluss

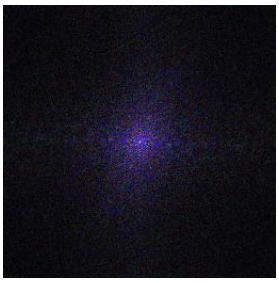
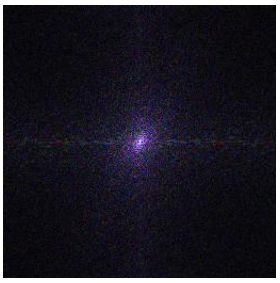
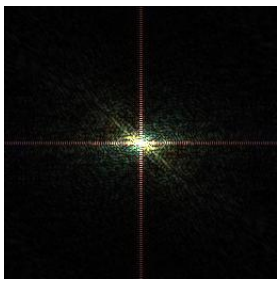
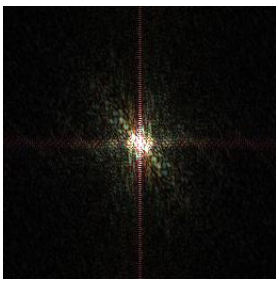
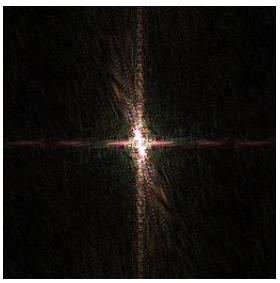
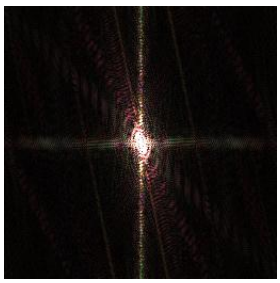
Tabelle 7-5: Optischer Fluss der Nearest Neighbour Komponente.

0-1	25-26	100-101
		
		

Bei dieser Komponente ist der optische Fluss erstaunlicherweise recht gut abgebildet. Besonders im zweiten Vektorfeld ist der Fluss sehr gut rekonstruierbar. Im Vektorfeld box3.dat ist der Fluss allerdings bei hohen Schrittzahlen nicht mehr rekonstruierbar. Dies ist auf die einfarbigen Bereiche des advektierten Bildes, welche das Bild in diesem Schritt dominieren, zurückzuführen.

7.2.1.1.4 Spektrum

Tabelle 7-6: Spektrum der Nearest Neighbour Komponente.

0	5	25	100
			
			

Das Spektrum ist relativ starken Veränderungen unterworfen. Besonders beim zweiten Vektorfeld ist gut zu erkennen, dass das Spektrum sich von einer grünen Wolke in einen rötlichen Stern verwandelt.

Das Verfahren erhält also nicht das Spektrum.

7.2.1.1.5 Performance

Tabelle 7-7: Performance der Nearest Neighbour Komponente.

Gesamtzeit:	1,05
Durchschnittszeit pro Schritt:	0,0105
Kürzeste Zeit eines Schrittes:	0,0102
Längste Zeit eines Schrittes:	0,0136
FPS:	95,26

Diese Komponente ist sehr schnell in der Berechnung. Sie ist selbst als reine CPU-Implementierung vollständig echtzeitfähig.

7.2.1.1.6 Bewertung

Dieses Verfahren erzeugt, wie erwartet, relativ starke Fehler. Diese Fehler sind anhand von Differenzbildern zu dem Referenzverfahren sehr deutlich geworden. Auch das eigentliche Ergebnis des Verfahrens ist optisch nicht ausreichend genau. Schon nach 25 Schritten kann man das Schachbrettmuster nicht mehr überall als solches erkennen.

Überraschenderweise ist es relativ gut möglich den optischen Fluss aus dem berechneten Ergebnis zu bestimmen. Der optische Fluss wird also erhalten.

Da in diesem Verfahren keine Berechnungen zum Interpolieren der Daten getätigt werden, ist die Performance des Verfahrens, wie erwartet, sehr gut.

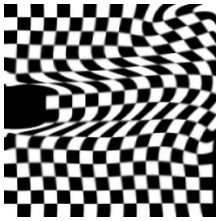
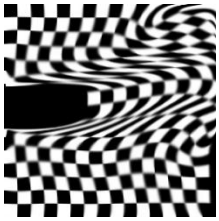
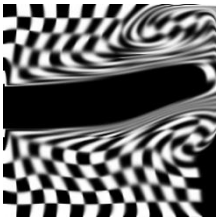
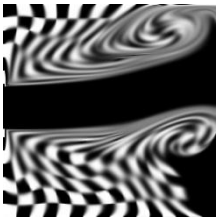
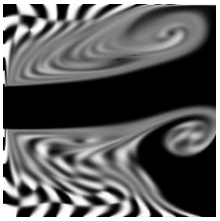
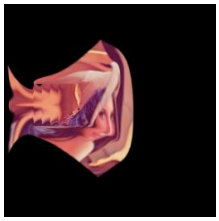
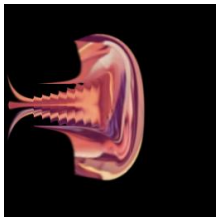
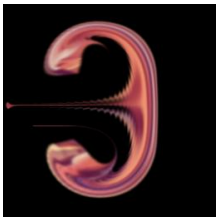
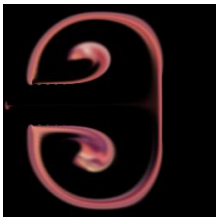
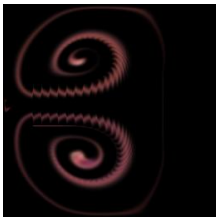
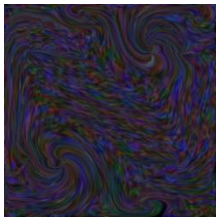
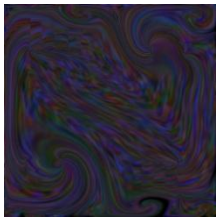
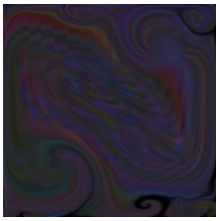
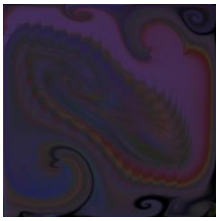
Diese Komponente sollte wegen der schlechten Ergebnisse nicht eingesetzt werden. Lediglich bei untergeordneten, performancekritischen Teilbereichen eines Verfahrens sollte diese Komponente in Betracht gezogen werden.

7.2.1.2 Bilineare Interpolation

Diese Komponente ist in 6.1.1.2 beschrieben. Hier wird eine bilineare Interpolation verwendet um die Daten auszulesen.

7.2.1.2.1 Ergebnisse

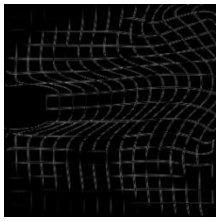
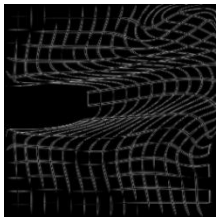
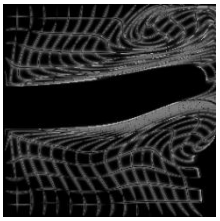
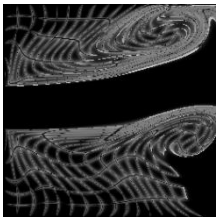
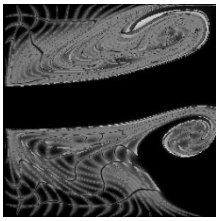
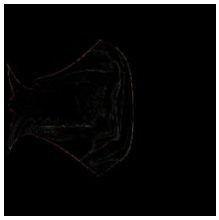
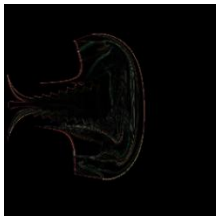
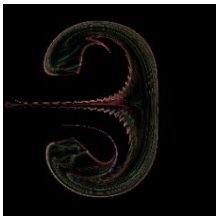
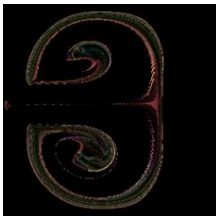
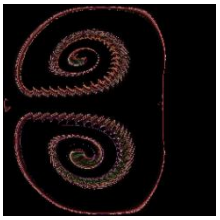
Tabelle 7-8: Ergebnisse der bilinearen Interpolation.

5	10	25	50	100
				
				
				

Besonders bei hohen Schrittzahlen ist zu erkennen, dass die Ergebnisse unscharf werden. Dies ist besonders gut im Schachbrettmuster zu sehen. Beim dritten Vektorfeld tendiert das Ergebnis sehr stark zu der Durchschnittsfarbe des Quellbildes. Dieses Verhalten ist auf die bilineare Interpolation zurückzuführen, welche die Farbwerte zu ungenau ausliest.

7.2.1.2.2 Differenzbilder

Tabelle 7-9: Differenzbilder der bilinearen Interpolation zu den Referenzbildern.

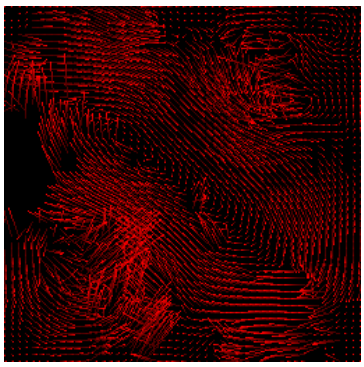
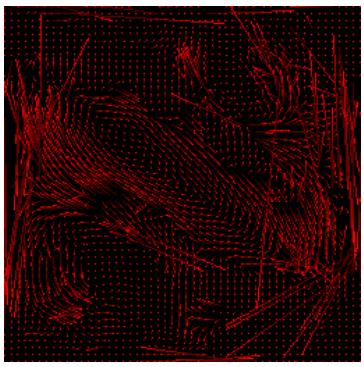
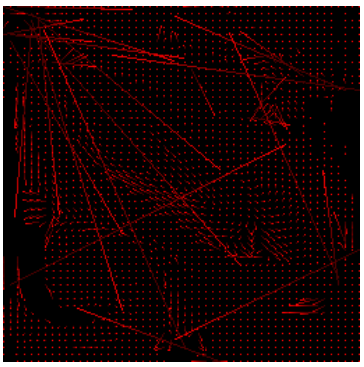
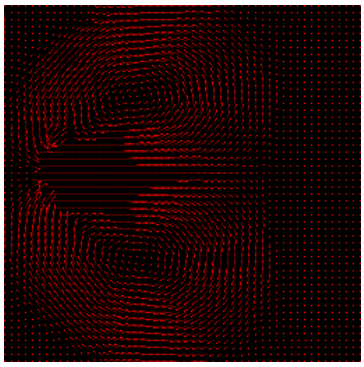
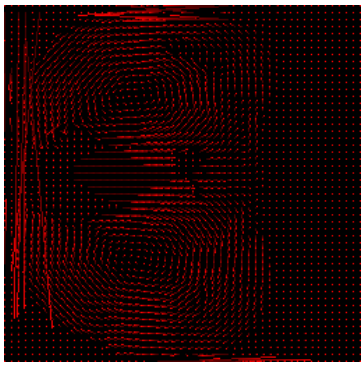
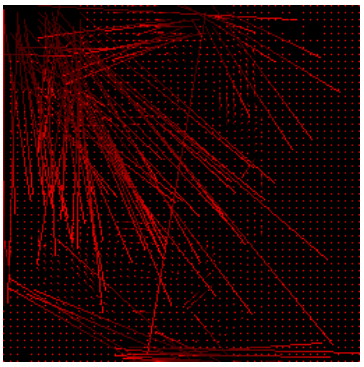
5	10	25	50	100
				
				

Diese Komponente erzeugt noch immer starke Abweichungen zu den Referenzbildern. Allerdings sind diese Abweichungen nicht so stark wie bei der Nearest Neighbour Komponente (Tabelle 7-4). Die Abweichungen verstärken sich bei hohen Schrittzahlen, da das Ergebnis dieser Komponente hier

gegen den Mittelwert der Farben aus dem Quellbild tendiert. Diese Abweichungen sind im Wesentlichen auf die Schärfe der Referenzbilder zurückzuführen, welche hier nicht gegeben ist.

7.2.1.2.3 Optischer Fluss

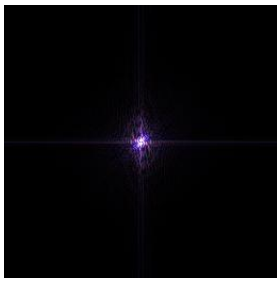
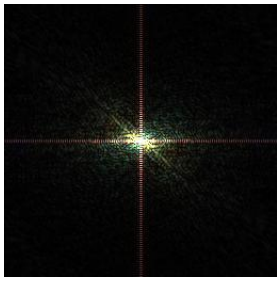
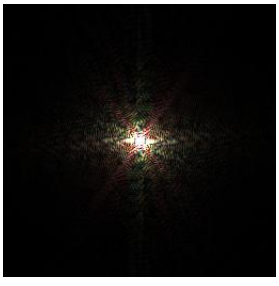
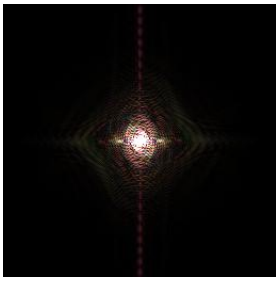
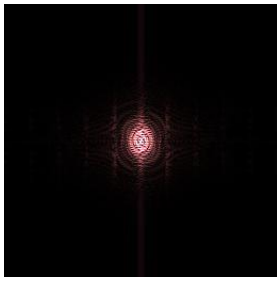
Tabelle 7-10: Optischer Fluss der bilinearen Interpolation.

0-1	25-26	100-101
		
		

Bei niedrigeren Schrittzahlen lässt sich der optische Fluss noch gut berechnen, was allerdings bei höheren Schrittzahlen nicht mehr zutrifft. Die Konstanz des optischen Flusses ist also nicht gegeben.

7.2.1.2.4 Spektrum

Tabelle 7-11: Spektrum der bilinearen Interpolation.

0	5	25	100
			
			

Das Spektrum wird in dieser Komponente nicht erhalten. Dies ist daran zu erkennen, dass das Spektrum starken Veränderungen unterworfen ist. Je höher die Schrittzahl ist, desto stärker wird das Spektrum auf einen Punkt konzentriert. Dies kann auf die Dominanz dieser Farbe in dem berechneten Bild zurückgeführt werden.

7.2.1.2.5 Performance

Tabelle 7-12: Performance der bilinearen Interpolation.

Gesamtzeit:	1,89
Durchschnittszeit pro Schritt:	0,0189
Kürzeste Zeit eines Schrittes:	0,0159
Längste Zeit eines Schrittes:	0,024
FPS:	53,02

Diese Komponente lässt sich sehr schnell berechnen. Mit 53,02 FPS ist die Echtzeitfähigkeit auch auf der CPU vollständig gegeben.

7.2.1.2.6 Bewertung

Diese Komponente liefert sehr schnell ein Ergebnis. Bei niedrigen Schrittzahlen ist dieses Ergebnis sehr gut. Allerdings nimmt die Qualität des Ergebnisses mit steigender Schrittzahl recht schnell ab.

Der optische Fluss und das Spektrum sind in dieser Komponente nicht sehr gut enthalten. Speziell der optische Fluss ist bei höheren Schrittzahlen nicht mehr erkennbar. Dies liegt daran, dass das Ergebnis dieser Komponente stark verschmiert und zum Mittelwert der Farben des Originalbildes tendiert.

Im Vergleich zur Nearest Neighbour Komponente ist jedoch eine deutliche Verbesserung des Ergebnisses zu beobachten.

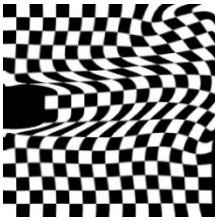
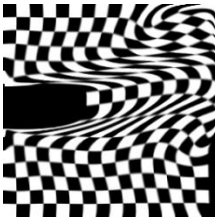
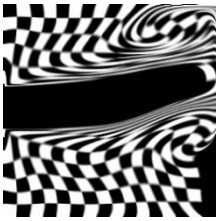
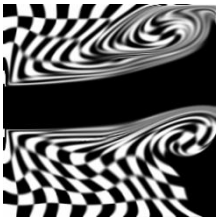
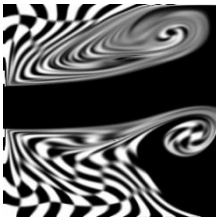
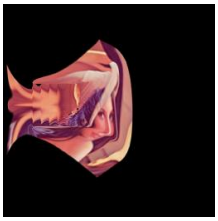
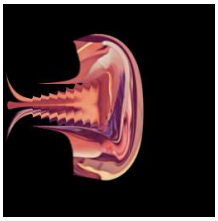
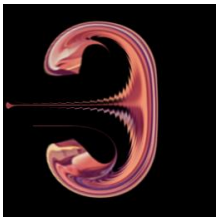
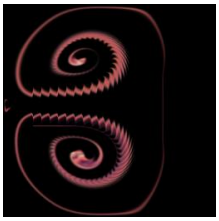
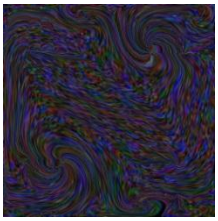
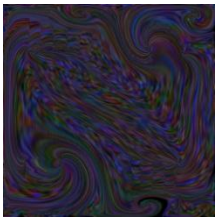
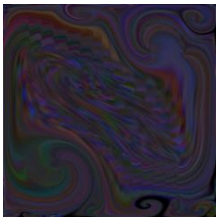
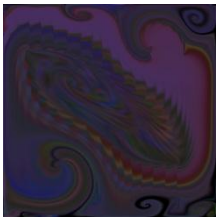
Diese Komponente ist ein guter Allroundansatz bei niedrigen Schrittzahlen. Es sollte jedoch darauf geachtet werden, dass diese Komponente nur bei niedrigen Schrittzahlen ein gutes Ergebnis liefert.

7.2.1.3 Interpolation mit Polynomen dritten Grades (Polynom 3)

Diese Komponente verwendet Polynome dritten Grades um die Interpolation der Daten durchzuführen. Sie ist in Kapitel 6.1.1.3 beschrieben.

7.2.1.3.1 Ergebnisse

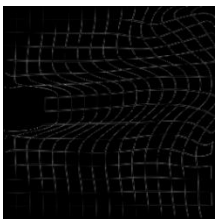
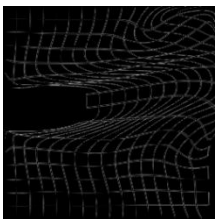
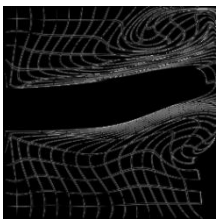
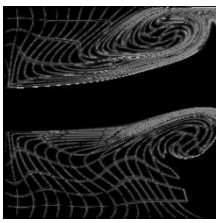
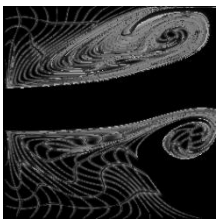
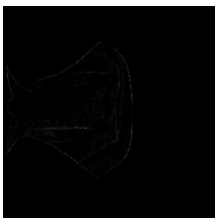
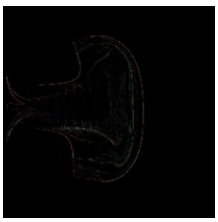
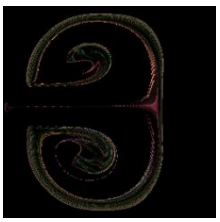
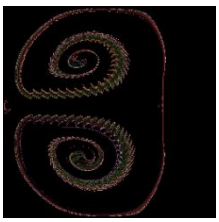
Tabelle 7-13: Ergebnisse der Interpolation mit Polynomen dritten Grades.

5	10	25	50	100
				
				
				

Bei dieser Komponente sind auch in den Ergebnissen mit hohen Schrittzahlen noch Details erkennbar. Die Ergebnisse werden nicht so stark unscharf wie bei der bilinearen Interpolation (Tabelle 7-8). Allerdings dominiert wie bei den anderen Komponenten eine Farbe das Ergebnis des dritten Vektorfeldes.

7.2.1.3.2 Differenzbilder

Tabelle 7-14: Differenzbilder der Interpolation mit Polynomen dritten Grades zu den Referenzbildern.

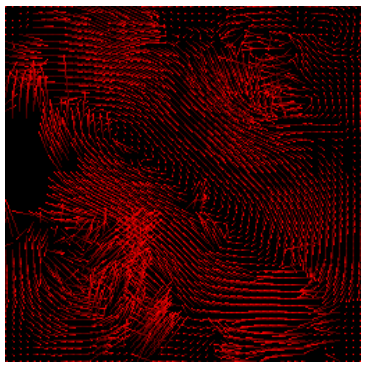
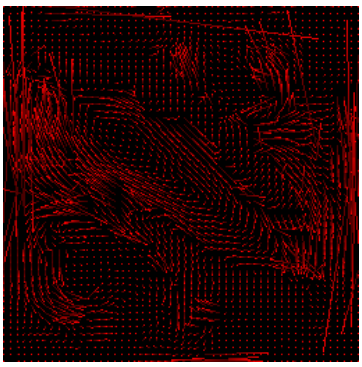
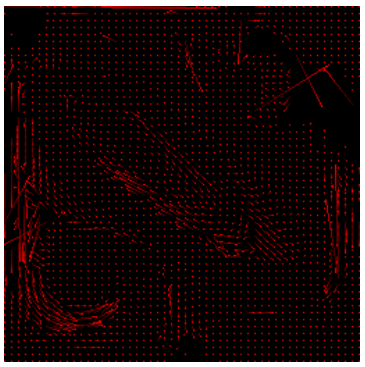
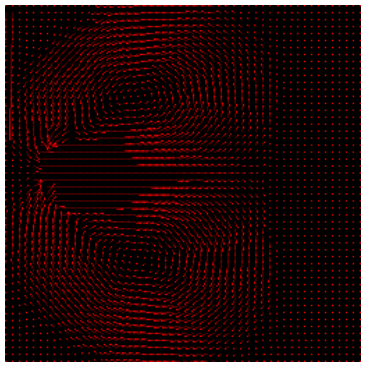
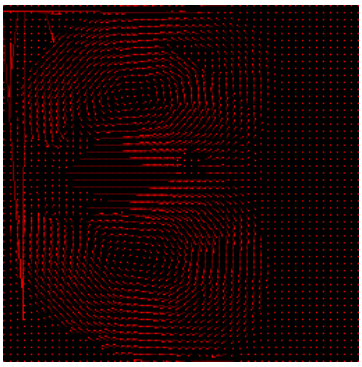
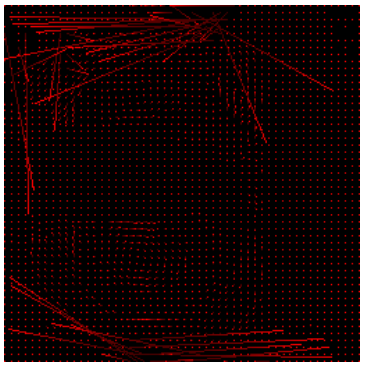
5	10	25	50	100
				
				

Bei den Differenzbildern merkt man, dass die Differenz nicht so stark ausfällt wie bei den anderen Komponenten. Dies ist besonders gut an dem Schachbrettmuster zu erkennen. Selbst bei hohen

Schrittzahlen sind hier nur die Kanten der Zellen markiert. Der mittlere Teil der Zellen ist hingegen korrekt.

7.2.1.3.3 Optischer Fluss

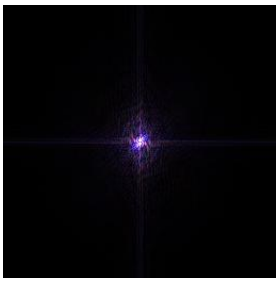
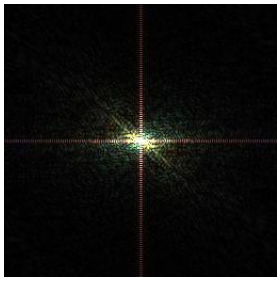
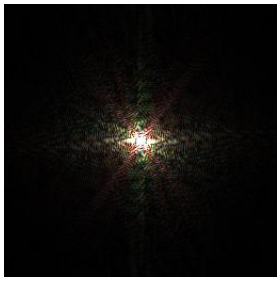
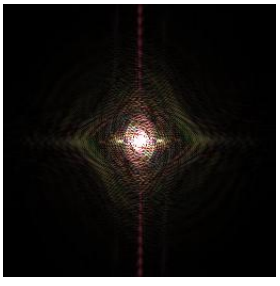
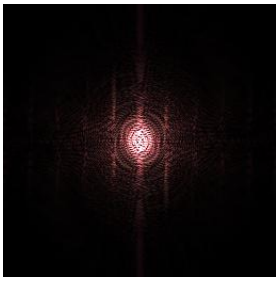
Tabelle 7-15: Optischer Fluss der Interpolation mit Polynomen dritten Grades.

0-1	25-26	100-101
		
		

Im Gegensatz zu den anderen Komponenten ist der optische Fluss in dieser Komponente auch bei hohen Schrittzahlen noch erkennbar. Dann ist er jedoch nur relativ schwach ausgeprägt.

7.2.1.3.4 Spektrum

Tabelle 7-16: Spektrum der Interpolation mit Polynomen dritten Grades.

0	5	25	100
			
			

Auch bei dieser Komponente ist das Spektrum nicht konstant erhalten. Speziell bei dem zweiten Vektorfeld ist zu erkennen, dass sich das Spektrum rötlich verfärbt.

Das Spektrum im ersten Vektorfeld wird, wie bei den anderen Komponenten, auf den Mittelpunkt konzentriert.

7.2.1.3.5 Performance

Tabelle 7-17: Performance der Interpolation mit Polynomen dritten Grades.

Gesamtzeit:	11,42
Durchschnittszeit pro Schritt:	0,1142
Kürzeste Zeit eines Schrittes:	0,0863
Längste Zeit eines Schrittes:	0,1603
FPS:	8,76

Diese Komponente benötigt deutlich mehr Rechenaufwand als die bilineare Interpolation (Tabelle 7-12). Dies liegt hauptsächlich daran, dass diese Komponente die Quelldaten an $4 * 4 = 16$ statt nur an vier Stellen auswertet. Auf der CPU ist diese Komponente nicht mehr echtzeitfähig.

7.2.1.3.6 Bewertung

Diese Komponente liefert ein gutes Ergebnis, obwohl das Spektrum nicht erhalten wird. Es ist eine deutliche Verbesserung gegenüber der bilinearen Interpolation festzustellen. Im Gegensatz zur bilinearen Interpolation ist bei hohen Schrittzahlen das Ergebnis noch gut und detailliert. Allerdings ist die Performance zu bemängeln.

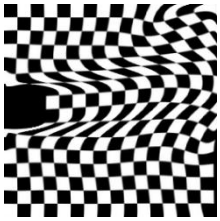
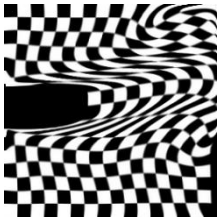
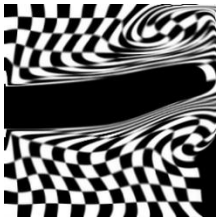
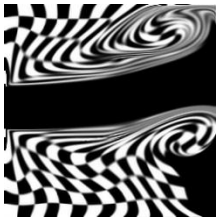
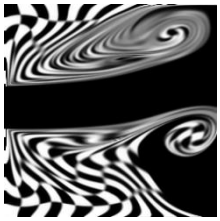
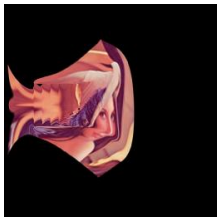
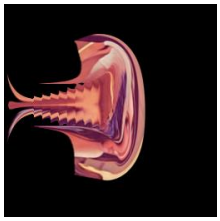
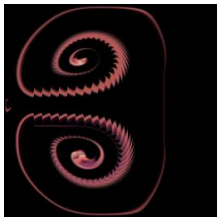
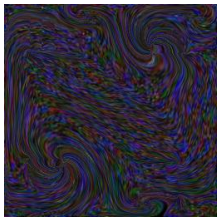
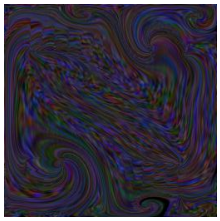
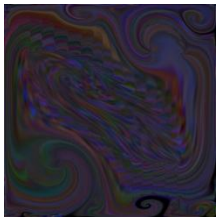
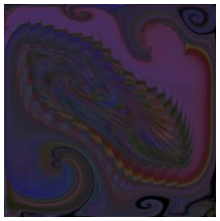
Wenn die Performance nicht ausschlaggebend ist, sollte diese Komponente der bilinearen Interpolation vorgezogen werden, da die Ergebnisse sowohl bei hohen als auch bei niedrigen Schrittzahlen besser sind.

7.2.1.4 Interpolation mit Polynomen fünften Grades (Polynom 5)

Diese Komponente ist in Kapitel 6.1.1.4 beschrieben. Hier werden Polynome fünften Grades eingesetzt um die Interpolation durchzuführen.

7.2.1.4.1 Ergebnisse

Tabelle 7-18: Ergebnisse der Interpolation mit Polynomen fünften Grades.

5	10	25	50	100
				
				
				

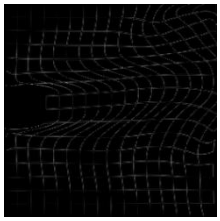
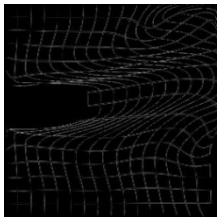
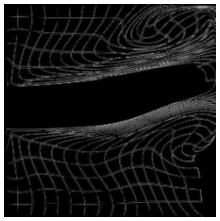
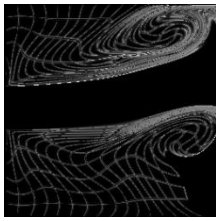
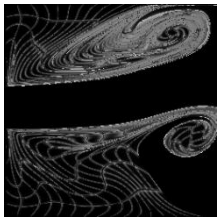
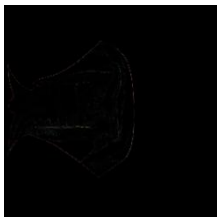
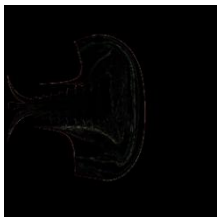
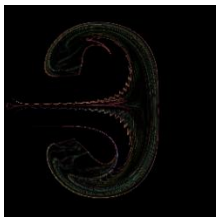
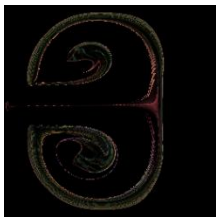
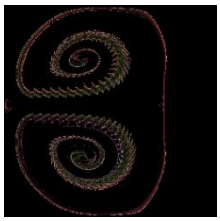
Die Ergebnisse sind noch etwas besser als bei der Interpolation mit Polynomen dritten Grades (Tabelle 7-13). Dies ist auch erwartet worden, da bei dieser Komponente lediglich der Grad des Polynoms erhöht wurde. Das Problem, dass im dritten Vektorfeld eine Farbe das Bild dominiert, ist auch hier vorhanden.

Bei hohen Schrittzahlen sind in dieser Komponente leichte Artefakte erkennbar. Diese Artefakte kommen von dem Polynom fünften Grades. Sie entstehen an Stellen, an denen die Limitierung der Daten nicht ausreicht um die Daten ausreichend zu korrigieren. Durch den iterativen Ansatz werden diese Artefakte bei hohen Schrittzahlen sichtbar. In Abbildung 7-7 ist ein solches Artefakt zu sehen.

Abbildung 7-7: Artefakte, welche durch das Polynom fünften Grades erstellt werden. Diese sind nur bei sehr großen Schrittzahlen erkennbar. Als Vektorfeld wird high_viscosity.dat verwendet und 250 Schritte berechnet.

7.2.1.4.2 Differenzbilder

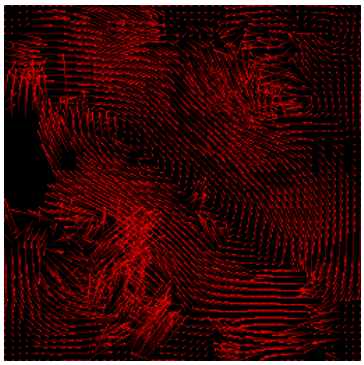
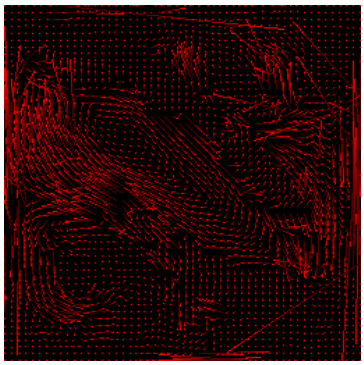
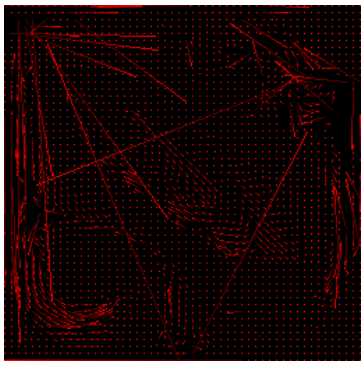
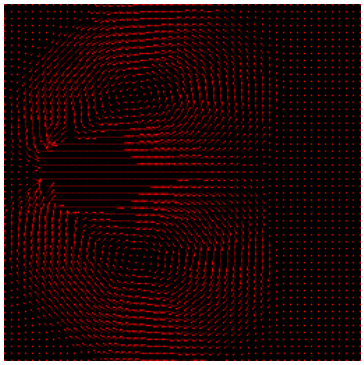
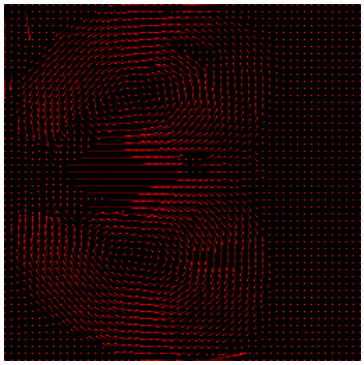
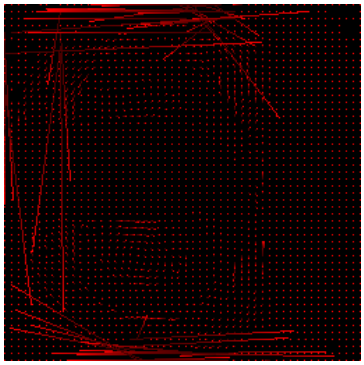
Tabelle 7-19: Differenzbilder der Interpolation mit Polynomen fünften Grades zu den Referenzbildern.

5	10	25	50	100
				
				

Diese Differenzbilder sind noch etwas blasser als die Differenzbilder der Interpolation mit Polynomen dritten Grades (Tabelle 7-14). Dies bedeutet, dass die Ergebnisse dieser Komponente genauer an den Referenzbildern liegen. Im Vergleich zu den Differenzbildern der Polynom 3 Komponente (Tabelle 7-14) ist feststellbar, dass die Kanten zwischen zwei Zellen beim Schachbrettmuster dünner werden. Diese Komponente berechnet also schärfere Bilder als die Polynom 3 Komponente.

7.2.1.4.3 Optischer Fluss

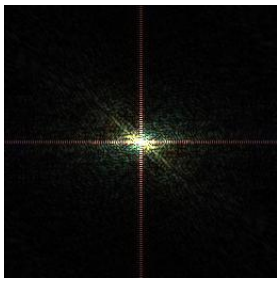
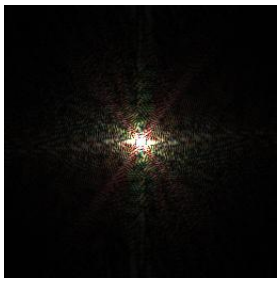
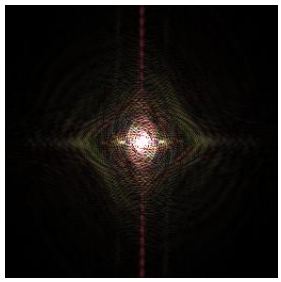
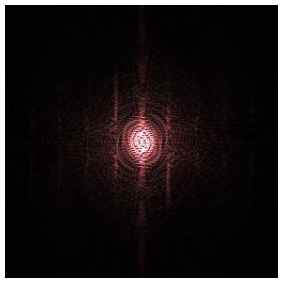
Tabelle 7-20: Optischer Fluss der Interpolation mit Polynomen fünften Grades.

0-1	25-26	100-101
		
		

In dieser Komponente ist der optische Fluss bei niedrigen Schrittzahlen gut sichtbar. Bei hohen Schrittzahlen ist der Fluss jedoch nur noch an einigen wenigen Stellen erkennbar. Dies ist ähnlich zu der Polynom 3 (Tabelle 7-15) Komponente.

7.2.1.4.4 Spektrum

Tabelle 7-21: Spektrum der Interpolation mit Polynomen fünften Grades.

0	5	25	100
			
			

Das Spektrum ist dem Spektrum der Interpolation mit Polynomen dritten Grades (Tabelle 7-16) sehr ähnlich. Das erste Spektrum wird mit steigender Schrittzahl im Zentrum konzentriert, während das zweite Spektrum sich rötlich verfärbt.

7.2.1.4.5 Performance

Tabelle 7-22: Performance der Interpolation mit Polynomen fünften Grades.

Gesamtzeit:	24,24
Durchschnittszeit pro Schritt:	0,2424
Kürzeste Zeit eines Schrittes:	0,2107
Längste Zeit eines Schrittes:	0,285
FPS:	4,12

Die Performance dieser Komponente ist erwartungsgemäß schlecht. Dies liegt hauptsächlich daran, dass hier die Daten an $6 * 6 = 36$ Stellen ausgewertet werden. Diese Komponente ist auf der CPU nicht mehr echtzeitfähig.

7.2.1.4.6 Bewertung

Diese Komponente liefert geringfügig bessere Ergebnisse als die Interpolation mit Polynomen dritten Grades. Allerdings ist der Rechenaufwand ungefähr doppelt so hoch wie der Rechenaufwand bei der Interpolation mit Polynomen dritten Grades. Die minimal besseren Ergebnisse rechtfertigen den erhöhten Rechenaufwand jedoch nicht. Außerdem entstehen bei dieser Interpolation bei hohen Schrittzahlen Artefakte in dem Ergebnis. Aus diesem Grund sollte die Interpolation mit Polynomen dritten Grades vorgezogen werden.

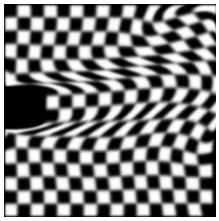
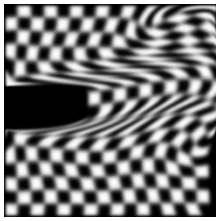
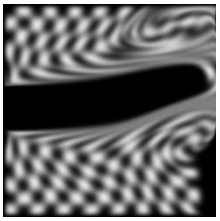
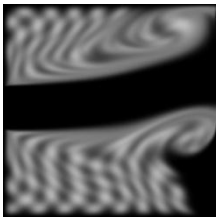
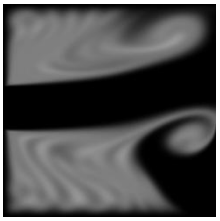
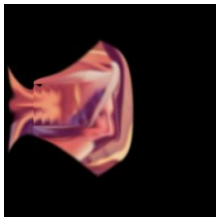
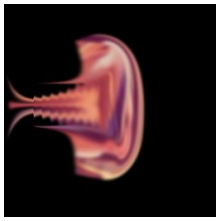
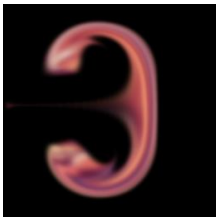
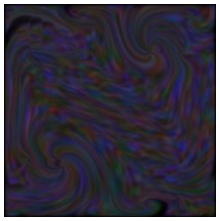
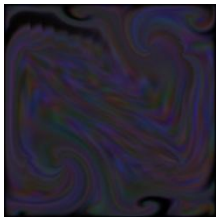
Der optische Fluss und das Spektrum ist vergleichbar mit dem bei der Interpolation mit Polynomen dritten Grades. Beide werden nicht erhalten.

7.2.1.5 Approximation mit B-Splines dritten Grades

Bei dieser Komponente werden B-Splines dritten Grades eingesetzt um die Daten zu approximieren. Diese Komponente ist in Kapitel 6.1.1.5 beschrieben.

7.2.1.5.1 Ergebnisse

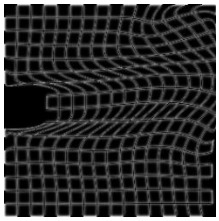
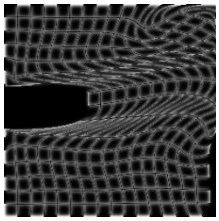
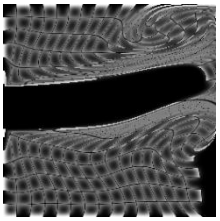
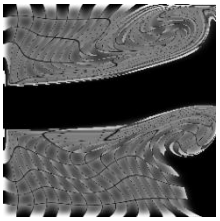
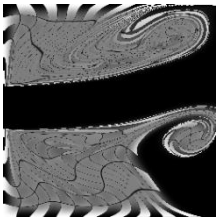
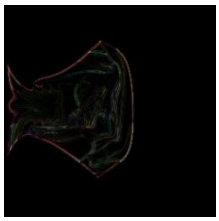
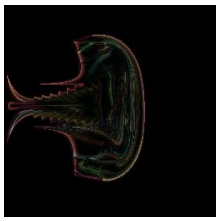
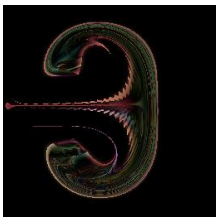
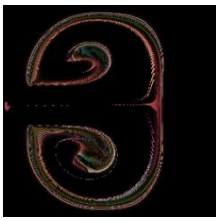
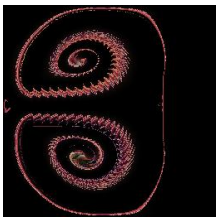
Tabelle 7-23: Ergebnisse der Approximation mit B-Splines dritten Grades.

5	10	25	50	100
				
				
				

In diesen Ergebnissen ist deutlich zu erkennen, dass die Bilder mit steigender Schrittzahl an Schärfe verlieren. Dies wird durch die Approximation der Datenpunkte verursacht. Bei der Approximation werden die beiden nächstliegenden Punkte nicht genau getroffen, sondern beeinflussen nur den B-Spline. Aus diesem Grund entsteht auch der schwarze Rand. Bei dieser Datenquelle wird ein Pixel außerhalb des Bildes als transparent angenommen. Da die nächstliegenden Punkte die Kurve nur beeinflussen wird der transparente Randpixel langsam in das Ergebnis hineingezogen, wodurch der schwarze Hintergrund sichtbar wird.

7.2.1.5.2 Differenzbilder

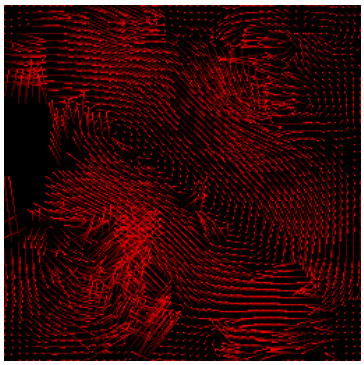
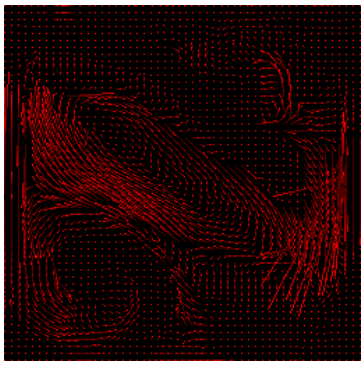
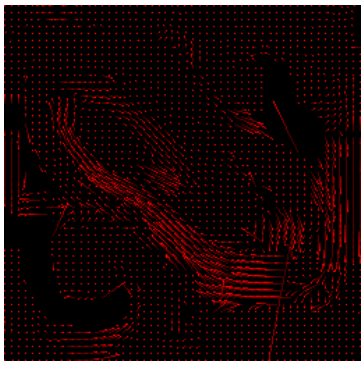
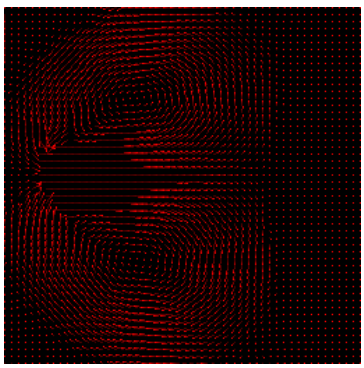
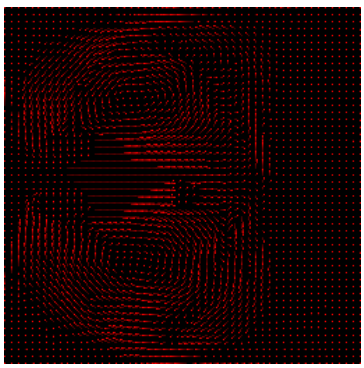
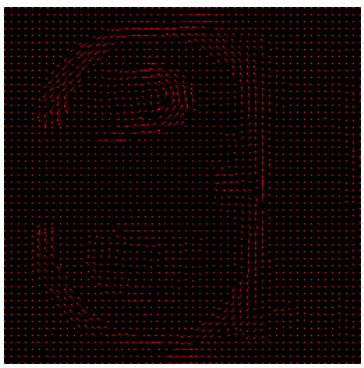
Tabelle 7-24: Differenzbilder der Approximation mit B-Splines dritten Grades zu den Referenzbildern.

5	10	25	50	100
				
				

Bei diesen Differenzbildern sind große Unterschiede zu erkennen. Besonders beim Schachbrettmuster sind diese an den Kanten der einzelnen Zellen sehr gut zu erkennen. An den Rändern der Zellen sind die Abweichungen besonders stark. Da in dieser Komponente der Rand des Bildes zur Hintergrundfarbe tendiert entsteht eine große Differenz zu dem Referenzbild, welches das Schachbrettmuster bis zum Rand aufweist. Bei hohen Schrittzahlen ist die Differenz sogar innerhalb der Zellen des Schachbrettmusters sichtbar.

7.2.1.5.3 Optischer Fluss

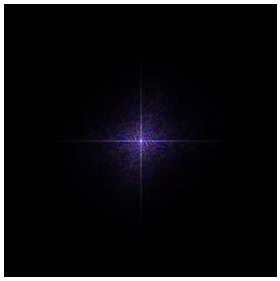
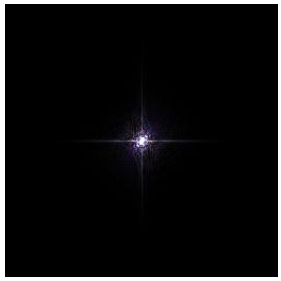
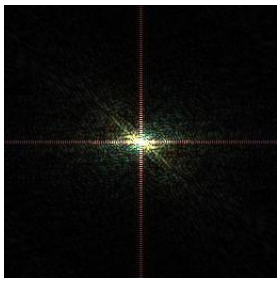
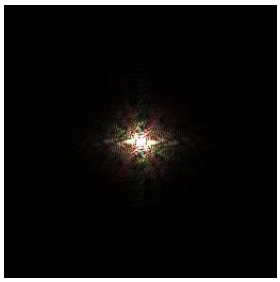
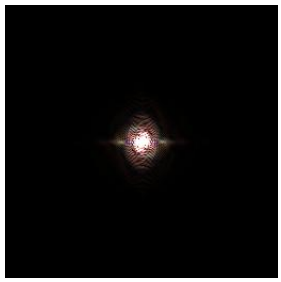
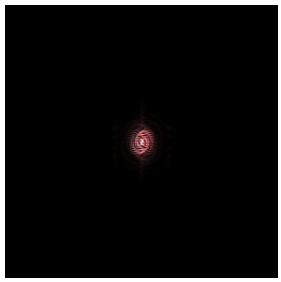
Tabelle 7-25: Optischer Fluss der Approximation mit B-Splines dritten Grades.

0-1	25-26	100-101
		
		

Der optische Fluss ist in dieser Komponente gut zu erkennen. Selbst bei hohen Schrittzahlen ist noch ein guter Fluss erkennbar, auch wenn er nicht mehr so stark ausgeprägt ist.

7.2.1.5.4 Spektrum

Tabelle 7-26: Spektrum der Approximation mit B-Splines dritten Grades.

0	5	25	100
			
			

Das Spektrum wird mit steigender Schrittzahl sehr stark auf einen Punkt konzentriert. Im Vergleich zur Interpolation mit Polynomen dritten (Tabelle 7-16) oder fünften Grades (Tabelle 7-21) wird das Spektrum in dieser Komponente wesentlich schneller auf einen Punkt konzentriert und wird somit auch nicht erhalten.

7.2.1.5.5 Performance

Tabelle 7-27: Performance der Approximation mit B-Splines dritten Grades.

Gesamtzeit:	16,09
Durchschnittszeit pro Schritt:	0,1609
Kürzeste Zeit eines Schrittes:	0,1429
Längste Zeit eines Schrittes:	0,1825
FPS:	6,21

In dieser Komponente werden die Quelldaten an $4 * 4 = 16$ Stellen ausgewertet. Dies ist vergleichbar mit der Interpolation mit Polynomen dritten Grades (Tabelle 7-17), welche auch 16 Stellen auswerten. Im Vergleich dazu benötigt diese Komponente jedoch die doppelte Zeit zur Berechnung.

Mit sechs FPS ist diese Komponente jedoch nicht mehr echtzeitfähig.

7.2.1.5.6 Bewertung

Wie erwartet, wird das Ergebnis in dieser Komponente sehr schnell unscharf. Auch das Spektrum wird mit steigender Schrittzahl sehr schnell auf einen Punkt konzentriert. Lediglich der optische Fluss ist hier gut erhalten.

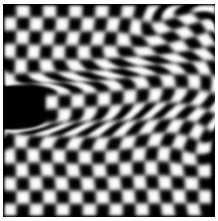
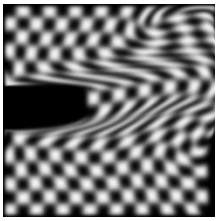
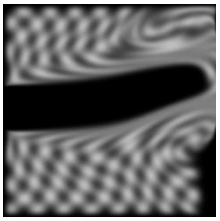
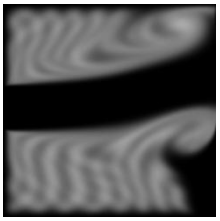
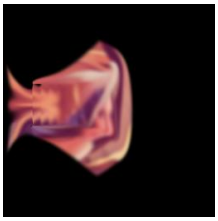
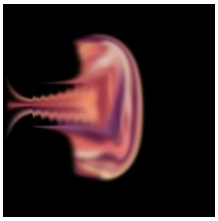
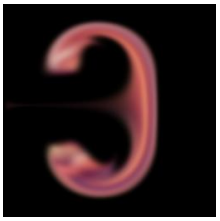
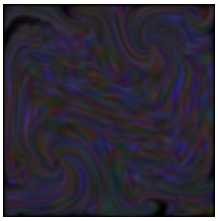
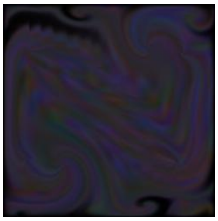
Im Vergleich zu der Interpolation mit Polynomen dritten Grades ist diese Komponente unterlegen. Die Ergebnisse sind schlechter, obwohl etwa doppelt so viel Rechenzeit benötigt wird. Daher sollte diese Komponente nicht eingesetzt werden.

7.2.1.6 Approximation mit B-Splines fünften Grades

Diese Komponente setzt B-Splines fünften Grades zur Approximation der Quelldaten ein. Sie ist in Kapitel 6.1.1.6 beschrieben.

7.2.1.6.1 Ergebnisse

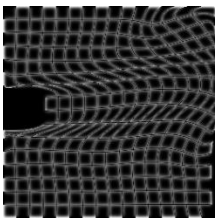
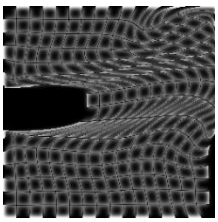
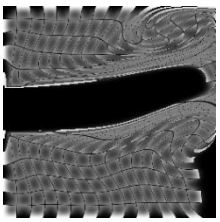
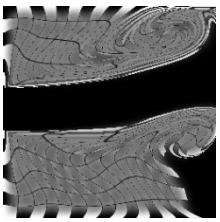
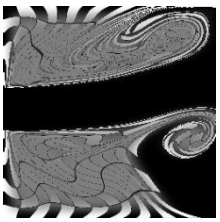
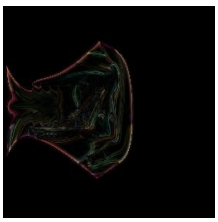
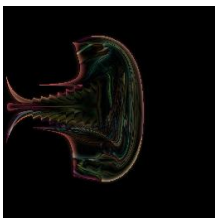
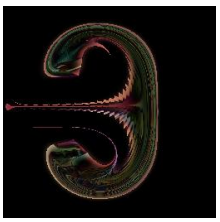
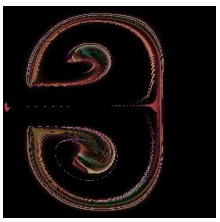
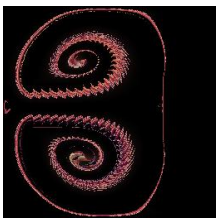
Tabelle 7-28: Ergebnisse der Approximation mit B-Splines fünften Grades.

5	10	25	50	100
				
				
				

Die Ergebnisse dieser Komponente sind mit denen der Approximation mit B-Splines dritten Grades (Tabelle 7-23) vergleichbar. Sie werden lediglich schneller unscharf. Auch hier findet ein Import der Randpixel in das Bild statt.

7.2.1.6.2 Differenzbilder

Tabelle 7-29: Differenzbilder der Approximation mit B-Splines fünften Grades zu den Referenzbildern.

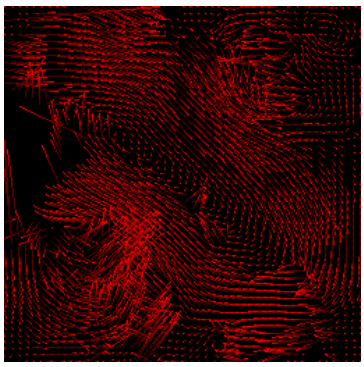
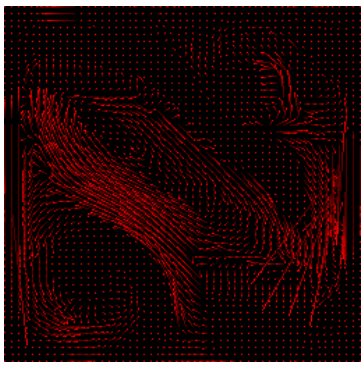
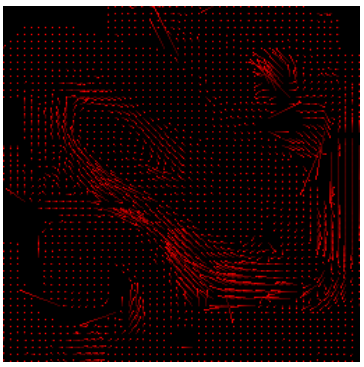
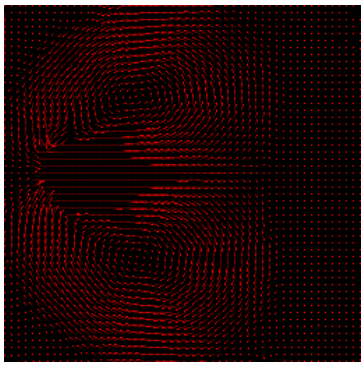
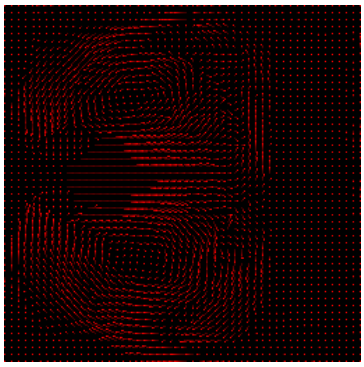
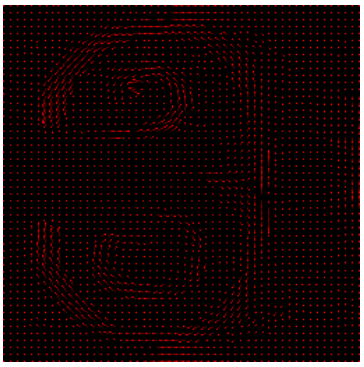
5	10	25	50	100
				
				

Auch die Differenzbilder sind mit der Approximation mit B-Splines dritten Grades (Tabelle 7-24) vergleichbar. Die Differenz ist hier etwas stärker da die Approximation eine größere Fläche verwendet, was in einer stärkeren Unschärfe der Ergebnisse resultiert. Wie bei der B-Spline 3

Komponente (Tabelle 7-24) ist auch hier bei hohen Schrittzahlen die Differenz innerhalb der Zellen des Schachbrettmusters sichtbar.

7.2.1.6.3 Optischer Fluss

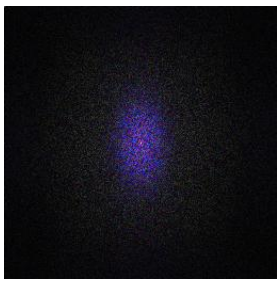
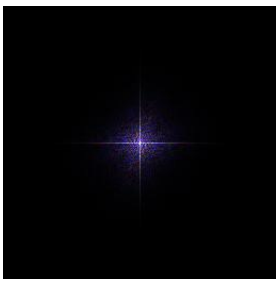
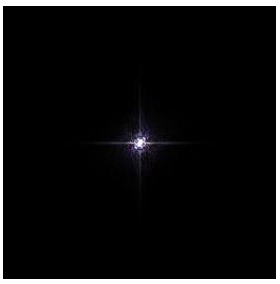
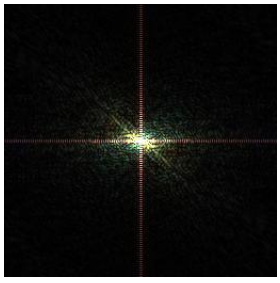
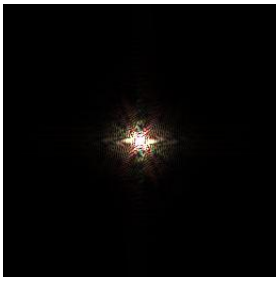
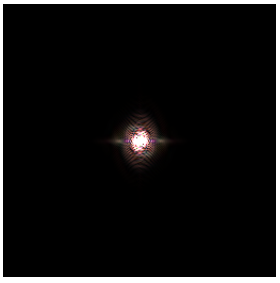
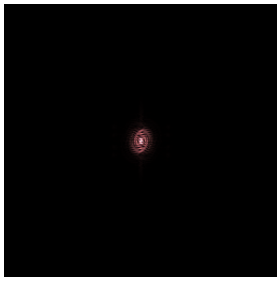
Tabelle 7-30: Optischer Fluss der Approximation mit B-Splines fünften Grades.

0-1	25-26	100-101
		
		

Der optische Fluss ist in dieser Komponente sehr gut erhalten. Selbst bei hohen Schrittzahlen ist der optische Fluss noch gut erkennbar. Allerdings ist er wie bei der B-Spline 3 Komponente (Tabelle 7-25) nicht mehr so stark ausgeprägt.

7.2.1.6.4 Spektrum

Tabelle 7-31: Spektrum der Approximation mit B-Splines fünften Grades.

0	5	25	100
			
			

Wie erwartet wird das Spektrum schneller als bei der Approximation mit B-Splines dritten Grades (Tabelle 7-26) auf einen Punkt konzentriert. Dies liegt an dem größeren Einzugsbereich der B-Splines. Das Spektrum wird also nicht erhalten.

7.2.1.6.5 Performance

Tabelle 7-32: Performance der Approximation mit B-Splines fünften Grades.

Gesamtzeit:	96,35
Durchschnittszeit pro Schritt:	0,9635
Kürzeste Zeit eines Schrittes:	0,9026
Längste Zeit eines Schrittes:	1,0576
FPS:	1,04

Die Performance dieser Komponente ist wie erwartet sehr niedrig. Im direkten Vergleich zur Interpolation mit Polynomen fünften Grades (Tabelle 7-22), welche die Quelldaten auch an 36 Stellen auswertet, benötigt diese Komponente vier Mal mehr Berechnungszeit und kommt nur auf ein FPS.

Diese Komponente ist die langsamste der hier analysierten Komponenten.

7.2.1.6.6 Bewertung

Dieses Verfahren ist mit der Approximation mit B-Splines dritten Grades vergleichbar. Die Ergebnisse werden etwas schneller unscharf und das Spektrum wird etwas schneller auf einen Punkt konzentriert. Einzig der optische Fluss ist gut erhalten, obwohl diese Komponente wesentlich mehr Rechenaufwand benötigt.

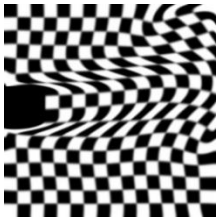
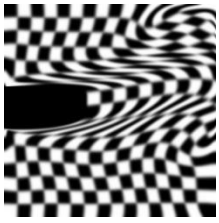
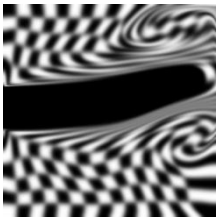
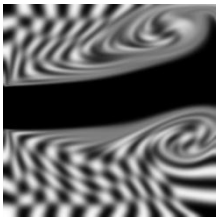
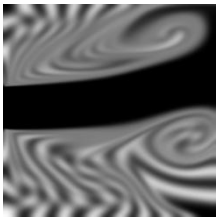
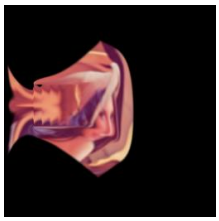
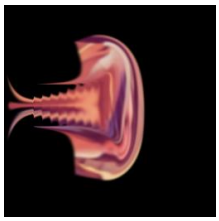
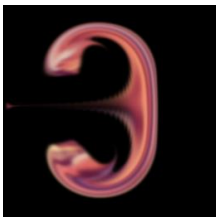
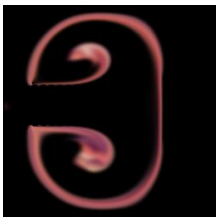
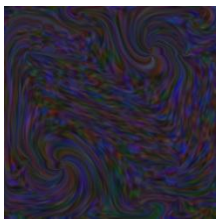
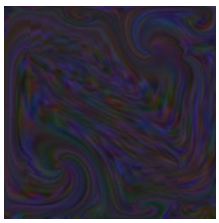
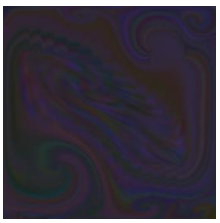
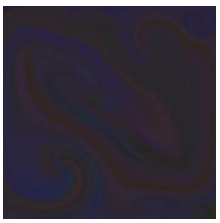
Wie auch die Approximation mit B-Splines dritten Grades sollte diese Komponente nicht eingesetzt werden.

7.2.1.7 Kernel

Diese Komponente verwendet einen Kernel um die Daten zu interpolieren und ist in 6.1.1.7 beschrieben. Der Radius des Kernels ist bei dieser Analyse $r = 2$.

7.2.1.7.1 Ergebnisse

Tabelle 7-33: Ergebnisse der Kernel Komponente.

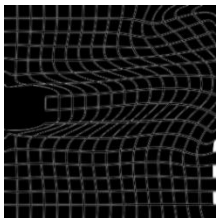
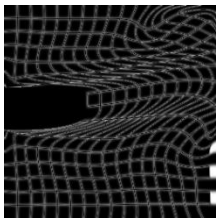
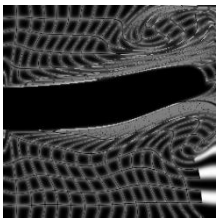
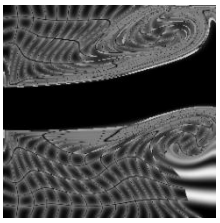
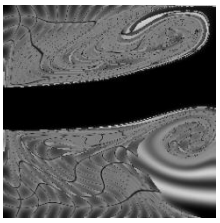
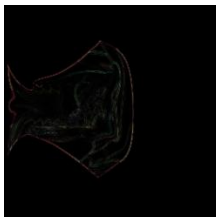
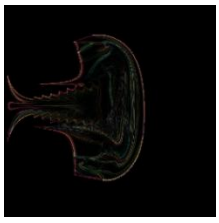
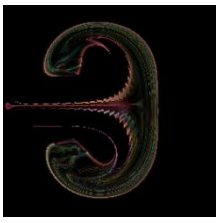
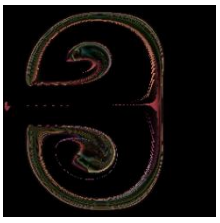
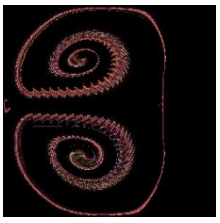
5	10	25	50	100
				
				
				

Die Ergebnisse sind vergleichbar mit den Ergebnissen der bilinearen Interpolation (Tabelle 7-8). Allerdings sind bei diesem Kernel, der den Radius $r = 2$ hat, die Ergebnisse bei höheren Schrittzahlen noch stärker verwischt. Die Ergebnisse verlieren schneller ihre Schärfe. Dies ist besonders gut an der Advektion des Schachbretts und des dritten Vektorfeldes zu sehen.

Im Gegensatz zur bilinearen Interpolation entsteht bei dieser Komponente auch eine Unschärfe an Stellen, an denen sich das Vektorfeld und damit die einzelnen Pixel nur sehr langsam oder gar nicht bewegen. Dies ist auch auf den Kernel zurückzuführen, welcher hier die Ursache für die Unschärfe ist. Zur Behebung müsste ein adaptiver Kernel eingesetzt werden. Also ein Kernel, der seinen Radius je nach Situation anpasst.

7.2.1.7.2 Differenzbilder

Tabelle 7-34: Differenzbilder der Kernel Komponente zu den Referenzbildern.

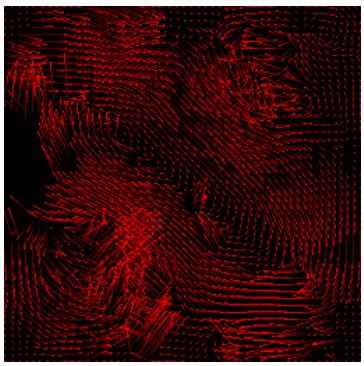
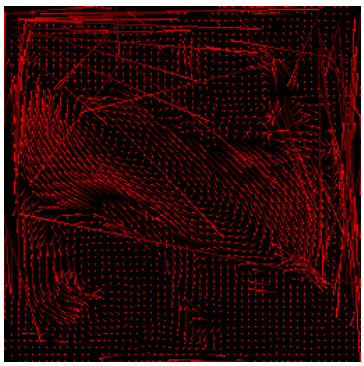
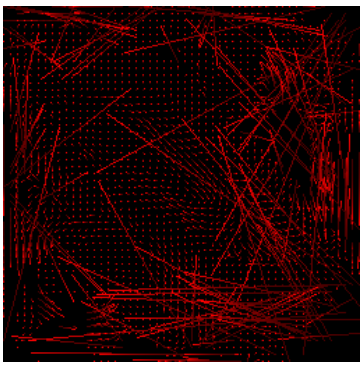
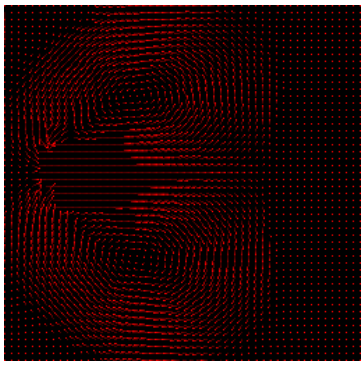
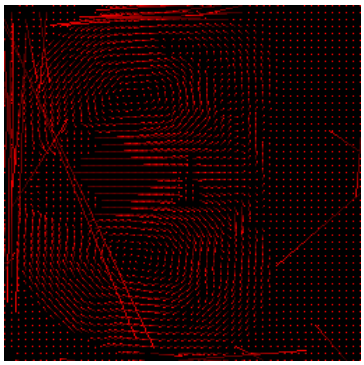
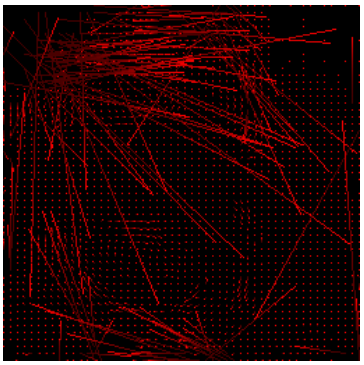
5	10	25	50	100
				
				

Die Ergebnisse der Differenzbilder sind mit denen der bilinearen Interpolation (Tabelle 7-9) vergleichbar. Besonders bei hohen Schrittzahlen ist die Differenz zu den Originalbildern groß. Bei hohen Schrittzahlen ist im Schachbrettmuster zu erkennen, dass die Differenz auch innerhalb der Zellen sichtbar ist. Dies ist auf die steigende Unschärfe dieses Verfahrens zurückzuführen.

Bei der Advektion des Schachbretts ist zu sehen, dass die Advektion an Schärfe verliert. Besonders die Kanten zwischen zwei Kacheln sind hervorgehoben. Dies ist darauf zurückzuführen, dass das Referenzverfahren die Kanten scharf abbildet, diese Komponente die Kanten jedoch verschmiert. Dieser Effekt verstärkt sich mit steigender Schrittzahl.

7.2.1.7.3 Optischer Fluss

Tabelle 7-35: Optischer Fluss der Kernel Komponente.

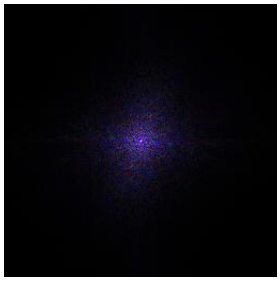
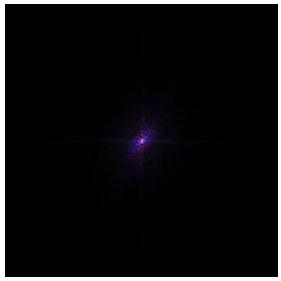
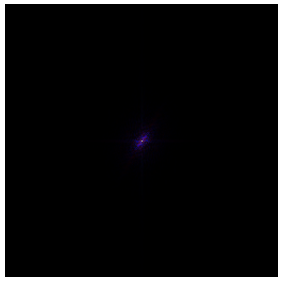
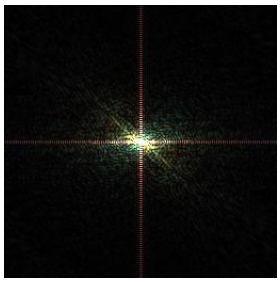
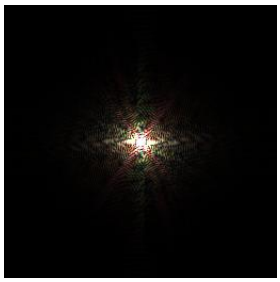
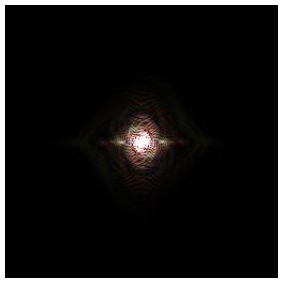
0-1	25-26	100-101
		
		

Der optische Fluss kann in dieser Komponente nur bei sehr kleinen Schrittzahlen korrekt ausgelesen werden.

Auch dieses Verhalten ist mit der bilinearen Interpolation (Tabelle 7-10) vergleichbar. Somit ist auch hier die Konstanz des optischen Flusses nicht gegeben.

7.2.1.7.4 Spektrum

Tabelle 7-36: Spektrum der Kernel Komponente.

0	5	25	100
			
			

Wie bei der bilinearen Interpolation (Tabelle 7-11) wird das Spektrum nicht erhalten. In dieser Komponente konzentriert sich das Spektrum allerdings schneller auf einen Punkt.

7.2.1.7.5 Performance

Tabelle 7-37: Performance der Kernel Komponente.

Gesamtzeit:	8,11
Durchschnittszeit pro Schritt:	0,0811
Kürzeste Zeit eines Schrittes:	0,0698
Längste Zeit eines Schrittes:	0,0934
FPS:	12,34

Die Performance dieser Komponente ist nicht gut. Mit der reinen CPU-Implementierung ist die Echtzeitfähigkeit nicht mehr gegeben.

7.2.1.7.6 Bewertung

Diese Komponente ist unter Qualitätsgesichtspunkten mit der bilinearen Interpolation vergleichbar. Die Ergebnisse sind allerdings etwas unschärfer und das Spektrum wird schneller auf einen Punkt konzentriert.

Von Seiten der Performance ist diese Komponente allerdings deutlich schlechter. Der Performancetest hat ergeben, dass gut die vierfache Zeit benötigt wird um 100 Schritte zu berechnen.

Diese Komponente besitzt allerdings den Vorteil, dass der Radius einstellbar ist, was eine gewisse Flexibilität erlaubt. Hierdurch kann die Komponente bis zu einem gewissen Grad auf die gewünschte Qualitätsstufe eingestellt werden.

Trotz der Einstellungsmöglichkeit der Komponente sollte die bilineare Komponente vorgezogen werden, da die Qualität der Ergebnisse vergleichbar ist, unter Performancegesichtspunkten jedoch deutlich besser ist.

7.2.1.8 Zusammenfassung

Abschließend kann man sagen, dass die Komponenten, welche einen Mittelwert aus einer Fläche aus dem Quellbild berechnen das Ergebnis sehr schnell unscharf werden lassen. Diese Komponenten sind die B-Spline 3 und B-Spline 5 Komponente und die Kernel Komponente. Diese sollten nicht eingesetzt werden.

Die Nearest Komponente berechnet auch kein gutes Ergebnis. Hier werden die einzelnen Details je nach Vektorfeld gestreckt und gezerrt. Diese Komponente sollte auch nicht eingesetzt werden.

Die bilineare Interpolation liefert dagegen schon ein brauchbares Ergebnis. Bei dieser Komponente werden die Daten zwar unscharf, sind aber noch brauchbar. Sie sollte in Situationen eingesetzt werden, wenn die Qualität der Daten nicht entscheidend ist. Beispielsweise beim Interpolieren des Vektorfeldes.

Die besten Ergebnisse liefern die Komponenten, welche Polynome zur Interpolation einsetzen. Je höher der Grad des verwendeten Polynoms hierbei ist, desto genauer ist das Ergebnis. Diese Komponente sollte bei ausreichender Rechenzeit eingesetzt werden.

7.2.2 Vektorfeldintegration

Diese Komponente bezieht sich auf die Integration der Vektorfelddaten. Das Interpolieren der Daten wird mit Hilfe einer anderen Komponente erledigt.

Für die Analyse dieser Komponente wird die Pfaddarstellung verwendet. Da die Komponente nicht eigenständig analysiert werden kann, wird ein Verfahren benötigt, das diese Komponente verwendet. Die eigentliche Komponente wird anschließend in diesem Verfahren ausgetauscht. Auf diese Weise erhält man vergleichbare Werte.

Das Vektorfeld ist in dieser Analyse ein Kreis, der gegen den Uhrzeigersinn dreht. Bei diesem Vektorfeld weiß man, dass bei einer idealen Integration der Pfad einen Kreis darstellt. Über die Abweichung von dieser Kreisbahn kann man feststellen wie gut ein Verfahren das Vektorfeld integriert.

Als Verfahren wird hier ein Backward-Verfahren eingesetzt, das die Daten aus dem Vektorfeld und aus dem Datenfeld mittels der bilinearen Interpolation auswertet.

7.2.2.1 Euler'sche Methode

Diese Komponente wird in Kapitel 6.1.2.1 beschrieben.

7.2.2.1.1 Pfaddarstellung

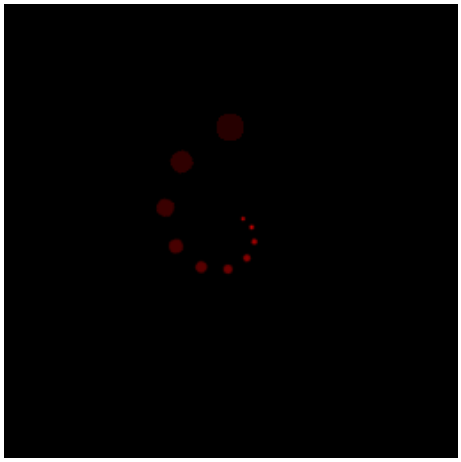


Abbildung 7-8: Pfaddarstellung der Euler'schen Methode.

Wie in Abbildung 7-8 zu sehen ist, verläuft der Pfad in einer Spirale auf den Mittelpunkt des Kreises zu. Dies ist eine erhebliche Differenz zu dem eigentlichen Vektorfeld, welches einen Kreis darstellt.

Dieser Fehler entsteht dadurch, dass die neue Position mittels des Weg-Zeit Gesetzes berechnet wird. Da nur eine einmalige Auswertung des Vektorfeldes stattfindet ist es dem Verfahren nicht möglich der Kreisbahn zu folgen. Dies resultiert in der sich verengenden Spirale.

7.2.2.1.2 Performance

Tabelle 7-38: Performance der Euler'schen Methode.

Gesamtzeit:	2,75
Durchschnittszeit pro Schritt:	0,0275
Kürzeste Zeit eines Schrittes:	0,0251
Längste Zeit eines Schrittes:	0,0323
FPS:	36,32

Diese Komponente ist trotz der reinen CPU-Implementierung echtzeitfähig.

7.2.2.1.3 Bewertung

Diese Methode kann dem Vektorfeld nur schlecht folgen. Allerdings muss bei dieser Methode das Vektorfeld nur an einer Stelle ausgewertet werden, was zu einer ordentlichen Performance beiträgt.

Bei der Reversibilitätsbetrachtung wird diese Komponente Probleme verursachen. Bei einem Vektorfeld kann nicht davon ausgegangen werden, dass mithilfe der negativen Geschwindigkeit einer berechneten Position wieder genau die Ausgangsposition erreicht wird. Dies ist nur bei Vektorfeldern, die keine räumlichen Änderungen enthalten, der Fall. Deshalb ist diese Methode für die Reversibilitätsberechnung nicht geeignet.

Diese Komponente sollte nur bei einfachen Vektorfeldern, die keine größeren Krümmungen aufweisen, eingesetzt werden. Bei Vektorfeldern mit großen räumlichen Änderungen liefern andere Methoden wesentlich bessere Ergebnisse.

7.2.2.2 Leap-Frog Verfahren

Das Leap-Frog Verfahren zur Berechnung der Position wird in Kapitel 6.1.2.2 beschrieben. Wie der Name schon sagt, wird in dieser Komponente das Leap-Frog Verfahren eingesetzt.

7.2.2.2.1 Pfaddarstellung

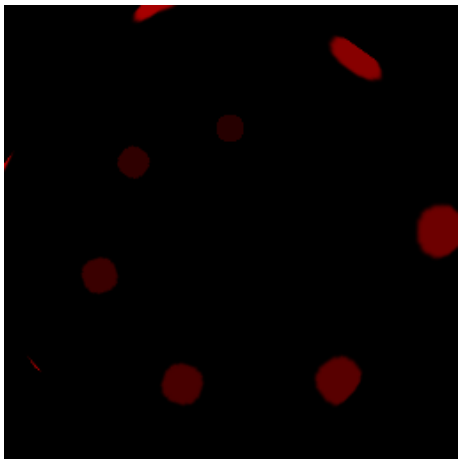


Abbildung 7-9: Pfaddarstellung der Interpretation mithilfe des Leap-Frog Verfahrens.

Der Einsatz des Leap-Frog Verfahrens liefert im Gegensatz zu den bisherigen Verfahren keine nach innen gerichtete Spirale. Wie in Abbildung 7-9 zu sehen ist, ist die Spirale, welche hierdurch entsteht, nach außen gerichtet.

7.2.2.2.2 Performance

Tabelle 7-39: Performance des Leap-Frog Verfahrens.

Gesamtzeit:	3,64
Durchschnittszeit pro Schritt:	0,0364
Kürzeste Zeit eines Schrittes:	0,0327
Längste Zeit eines Schrittes:	0,0478
FPS:	27,47

Diese Komponente ist auch in der CPU-Implementierung echtzeitfähig.

7.2.2.2.3 Bewertung

Im Gegensatz zur Euler'schen Methode ist die Spirale, die durch die Pfaddarstellung entsteht nach außen gerichtet. Das Ergebnis dieser Methode ist allerdings genauso schlecht wie die Euler'sche Methode.

Diese Komponente sollte nur eingesetzt werden, wenn aus Performancegründen keine andere Komponente verwendet werden kann.

7.2.2.3 Klassisches Runge-Kutta Verfahren (RK 4)

Diese Komponente verwendet das klassische Runge-Kutta Verfahren um die Position zu berechnen und ist in Kapitel 6.1.2.3 beschrieben.

7.2.2.3.1 Pfaddarstellung

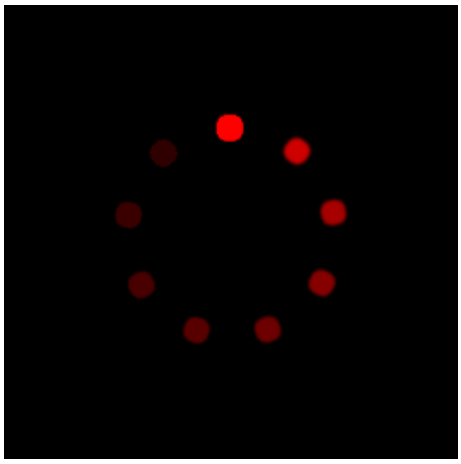


Abbildung 7-10: Pfaddarstellung dieser Komponente.

Diese Komponente liefert ein sehr gutes Ergebnis bei der Integration des Vektorfeldes. Wie in Abbildung 7-10 zu sehen ist, liegen der erste und letzte Punkt des Pfades nahezu perfekt aufeinander. Die Positionen werden hier sehr genau berechnet, obwohl das Vektorfeld in diesem Verfahren nur an vier Stellen ausgewertet werden muss.

7.2.2.3.2 Performance

Tabelle 7-40: Performance des klassischen Runge-Kutta Verfahrens.

Gesamtzeit:	6,71
Durchschnittszeit pro Schritt:	0,0671
Kürzeste Zeit eines Schrittes:	0,0477
Längste Zeit eines Schrittes:	0,0896
FPS:	14,90

Diese Komponente benötigt die meiste Rechenzeit der hier vorgestellten Komponenten. Allerdings liefert diese auch das beste Ergebnis.

Diese Komponente ist nur bedingt auf der CPU echtzeitfähig.

7.2.2.3.3 Bewertung

Diese Komponente liefert ein sehr gutes Ergebnis. Allerdings ist dieses Ergebnis noch immer nicht ideal, da auch diese Komponente nur eine Näherung darstellt. An Stellen, an denen das Vektorfeld sehr große räumliche Änderungen aufweist, wie zum Beispiel im Mittelpunkt eines Strudels, ist auch dieses Verfahren nicht in der Lage diesen ausreichend genau zu folgen. Diese Komponente ist allerdings die Beste der hier vorgestellten Komponenten.

Diese Komponente sollte, wenn möglich, eingesetzt werden.

7.2.2.4 Zusammenfassung

Von den drei hier vorgestellten Methoden zur Integration des Vektorfeldes ist das klassische Runge-Kutta Verfahren am Besten in der Lage räumlichen Änderungen im Vektorfeld zu folgen. Diese Methode sollte eingesetzt werden.

Die Euler'sche Methode ist hingegen nicht sehr gut geeignet um Krümmungen im Vektorfeld zu folgen. Allerdings ist diese schnell zu berechnen.

Das Leap-Frog Verfahren liefert hingegen in etwa gleich schlechte Ergebnisse wie die Euler'sche Methode, benötigt aber mehr Rechenzeit. Aus diesem Grund sollte dieses nicht eingesetzt werden.

7.3 Verfahren

In diesem Kapitel werden die einzelnen Verfahren analysiert. Hierbei wird eine Unterteilung der Verfahren in Backward- und Forward-Verfahren vorgenommen.

Die Ergebnisse der Verfahren basieren auf den Quellbildern aus Abbildung 7-1 und Abbildung 7-3 und sind auf den Vektorfeldern high_viscosity.dat, box1.dat und box3.dat berechnet. Die Vektorfelder sind in Abbildung 4-1 rechts oben, links oben und rechts unten zu sehen. Die Ergebnisse sind für die Schritte 5, 10, 25, 50 und 100 berechnet worden.

Für die einzelnen Verfahren sind Differenzbilder berechnet worden. Die Verfahren werden auch auf die Reversibilität, den optischen Fluss und des Spektrums hin analysiert. Zusätzlich wird die Performance der Verfahren mit der integrierten Messeinheit berechnet.

Die Ergebnisse des ersten und zweiten Vektorfeldes werden eingesetzt um die Differenzbilder zu berechnen. Außerdem werden die Ergebnisse des zweiten Vektorfeldes bei der Berechnung der Reversibilität und bei der Berechnung des Spektrums verwendet. Jeweils das zweite Vektorfeld verwendet diese Ergebnisse. Die Ergebnisse des dritten Vektorfeldes werden bei der Berechnung des optischen Flusses im ersten Vektorfeld verwendet.

7.3.1 Backward-Verfahren

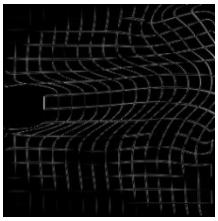
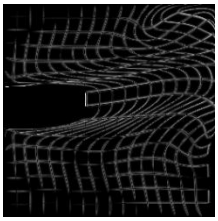
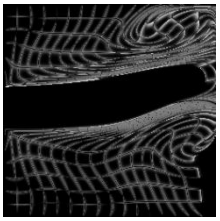
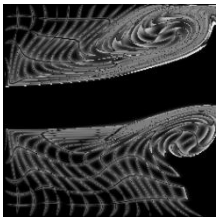
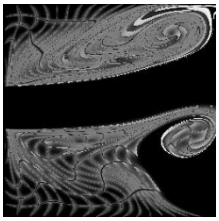
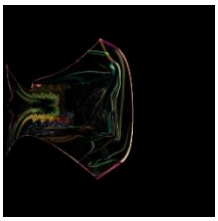
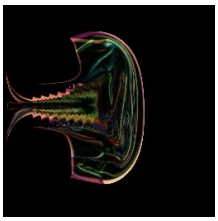
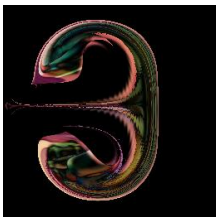
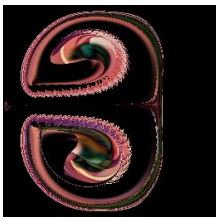
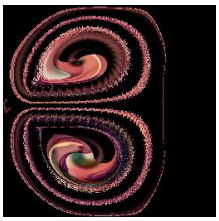
Die Backward-Verfahren beinhalten zwei Kategorien von Verfahren. Einerseits gibt es die Verfahren, die ein eigenes System verwenden um die Pixelfarbe zu berechnen. Andererseits gibt es die Verfahren, die sich aus den Komponenten zusammensetzen und kein eigenständiges Verfahren verwenden.

7.3.1.1 Semi-Lagrange

In diesem Verfahren werden die bilineare Interpolation zum Interpolieren der Daten und die Euler'sche Methode zum Berechnen der Position verwendet. Das Semi-Lagrange Verfahren ist in Kapitel 6.2.2.3 beschrieben. Die Analyse dieses Verfahrens ist identisch mit der Analyse der bilinearen Interpolation zum Auswerten der Daten, welche in Kapitel 0 durchgeführt wurde. Dies betrifft die Ergebnisse, die Differenzbilder, den optischen Fluss, das Spektrum und die Performance.

7.3.1.1.1 Differenzbilder

Tabelle 7-41: Differenzbilder des Semi-Lagrange Verfahrens zu den Referenzbildern.

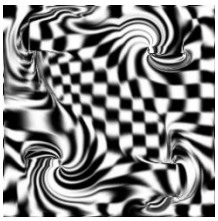
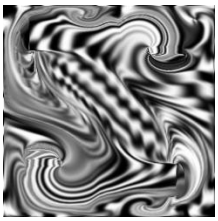
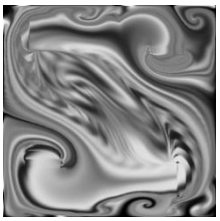
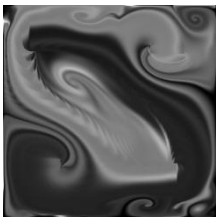
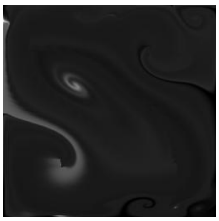
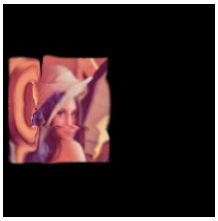
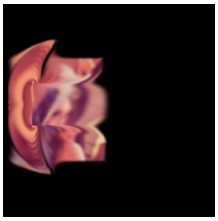
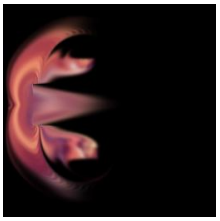
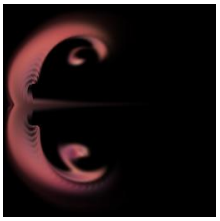
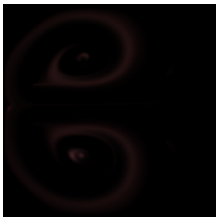
5	10	25	50	100
				
				

Anhand dieser Differenzbilder ist sehr gut zu erkennen, dass die Differenz besonders an Stellen mit starker räumlicher Änderung im Vektorfeld sehr groß ist. Sie steigt, wie bei allen iterativen Verfahren, mit der Schrittzahl.

Die große Differenz ist auch auf die unterschiedlich berechneten Positionen in diesem und dem Referenzverfahren zurückzuführen. Besonders im zweiten Vektorfeld ist dies deutlich sichtbar.

7.3.1.1.2 Reversibilität

Tabelle 7-42: Ergebnisse der Reversibilitätsberechnung des Semi-Lagrange Verfahrens.

5	10	25	50	100
				
				

Die Reversibilität dieser Komponente ist nicht sehr gut. Bereits ab dem zehnten Schritt ist eine Reversibilität nicht mehr ausreichend gegeben. Dies liegt daran, dass die Euler'sche Methode zur Integration des Vektorfeldes eingesetzt wird. Diese Methode kann dem Vektorfeld nicht ausreichend genau folgen, was sich in den falschen Positionen der Pixel wiederspiegelt.

7.3.1.1.3 Bewertung

Die Bewertung dieses Verfahrens ist größtenteils identisch mit der Bewertung der bilinearen Interpolation in Kapitel 7.2.1.2.6.

Dieses Verfahren liefert einen schnellen und einfach zu implementierenden Ansatz um eine einfache Advektion durchzuführen. Die Ergebnisse sind allerdings auch nicht sehr gut.

Dieses Verfahren sollte nur eingesetzt werden wenn die Performance kein anderes Verfahren erlaubt, oder die Ergebnisse nur eine untergeordnete Rolle spielen. Es muss darauf geachtet werden, dass dieses Verfahren bei hohen Schrittzahlen ein schlechtes Ergebnis liefert. Es sollte daher nur in Situationen mit wenigen Schritten eingesetzt werden.

7.3.1.2 BFECC-Original

Das BFECC-Original Verfahren ist in 6.2.2.4 beschrieben. Im Folgenden werden die Analyseergebnisse des BFECC-Original Verfahren mit allen Modifikationen dargestellt.

7.3.1.2.1 Ergebnisse

Tabelle 7-43: Ergebnisse des BFECC-Original Verfahrens.

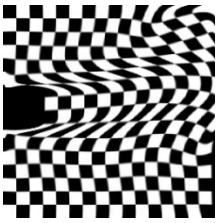
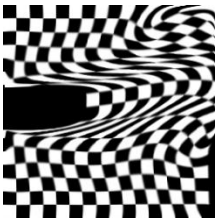
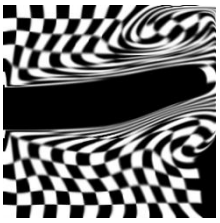
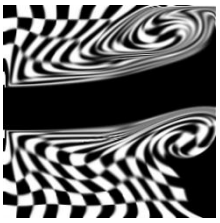
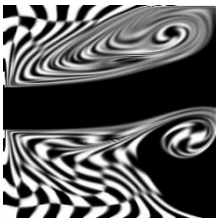
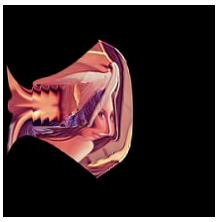
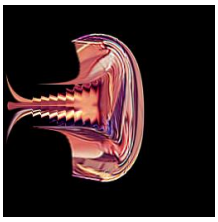
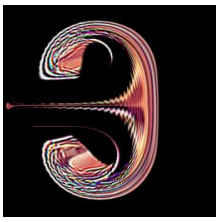
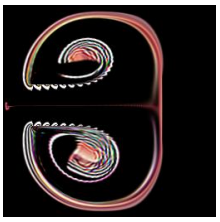
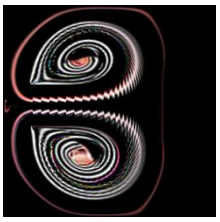
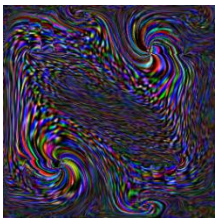
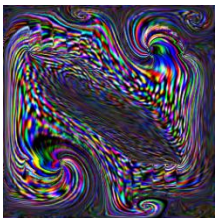
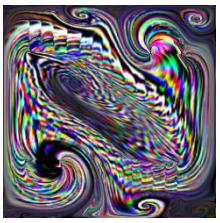
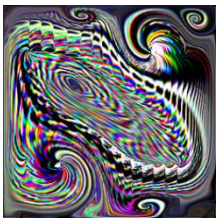
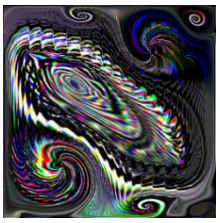
5	10	25	50	100
				
				
				

Tabelle 7-44: Ergebnisse des BFECC-Original + Polynom 3 Verfahrens.

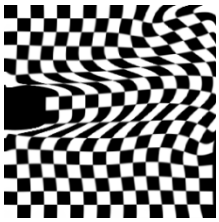
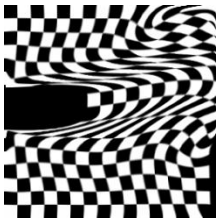
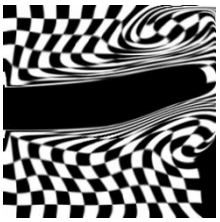
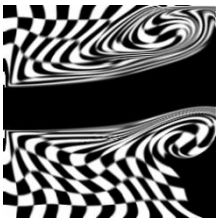
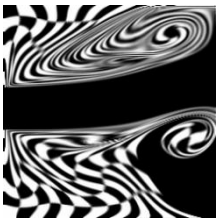
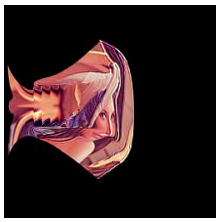
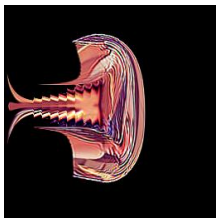
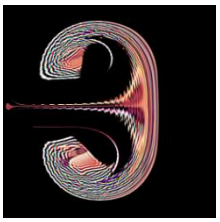
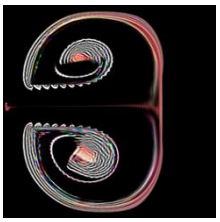
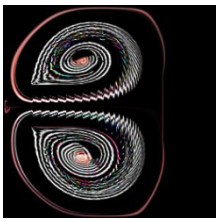
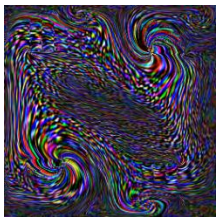
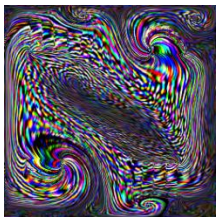
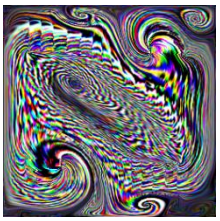
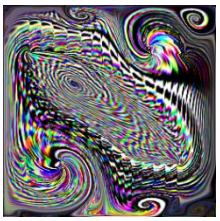
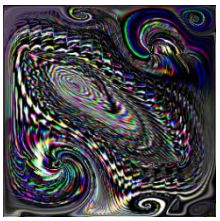
5	10	25	50	100
				
				
				

Tabelle 7-45: Ergebnisse des BFECC-Original + Polynom 5 Verfahrens.

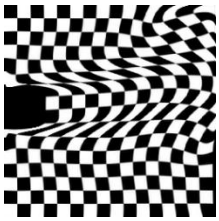
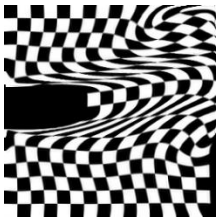
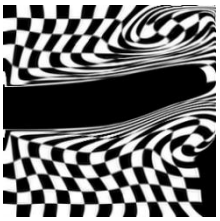
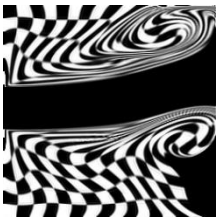
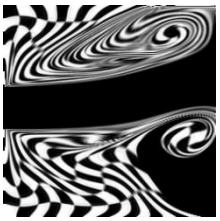
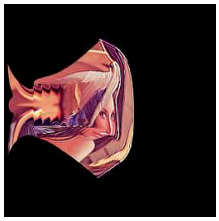
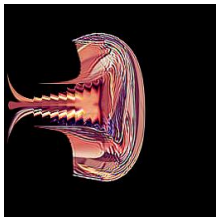
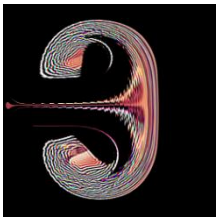
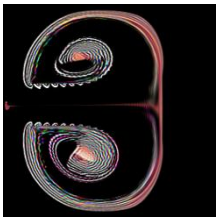
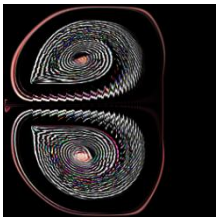
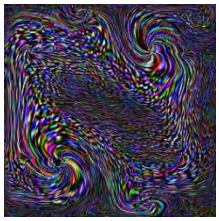
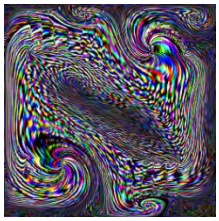
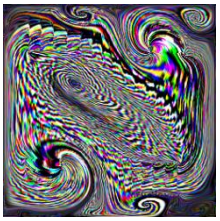
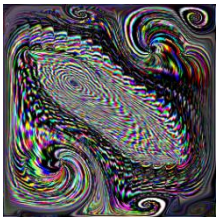
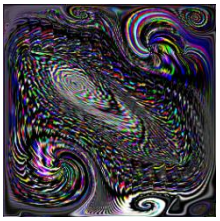
5	10	25	50	100
				
				
				

Tabelle 7-46: Ergebnisse des BFECC-Original + Polynom 3 Last Verfahrens.

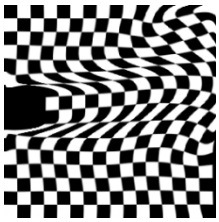
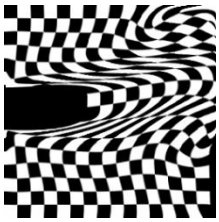
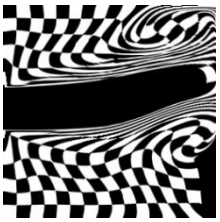
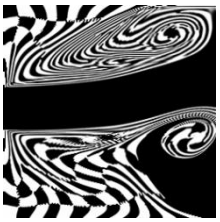
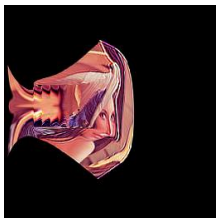
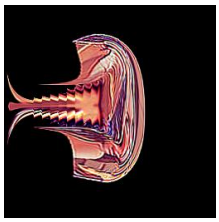
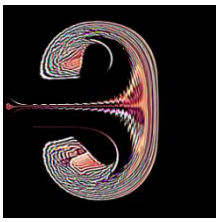
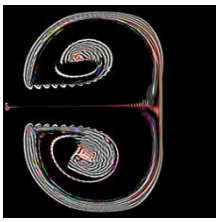
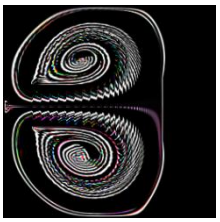
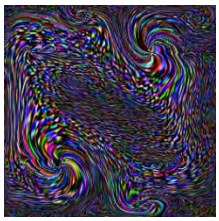
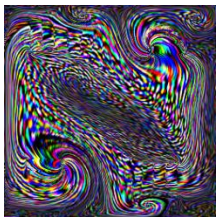
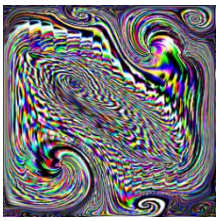
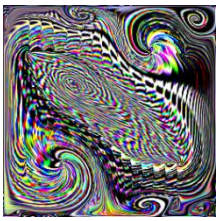
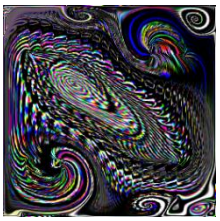
5	10	25	50	100
				
				
				

Tabelle 7-47: Ergebnisse des BFECC-Original + Polynom 5 Last Verfahrens.

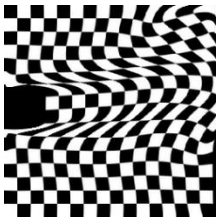
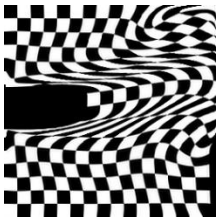
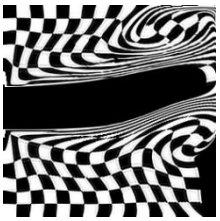
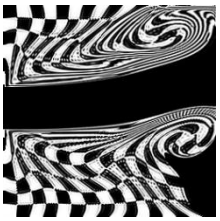
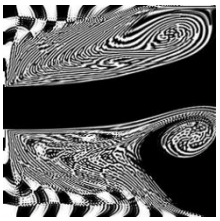
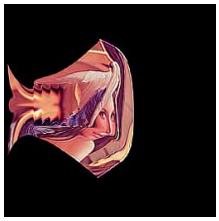
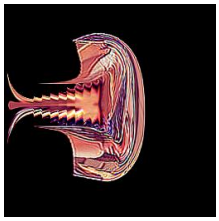
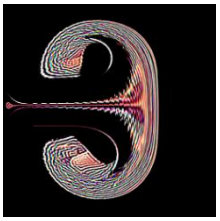
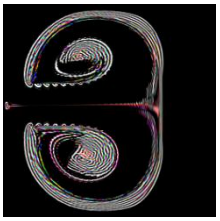
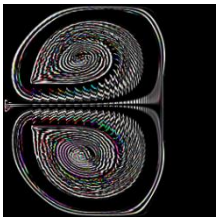
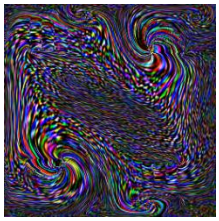
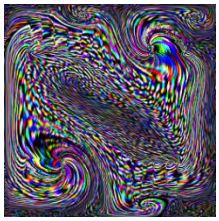
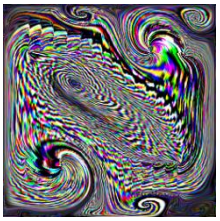
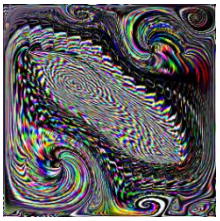
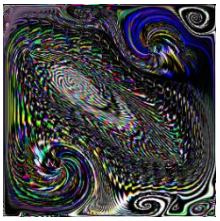
5	10	25	50	100
				
				
				

Tabelle 7-48: Ergebnisse des BFECC-Original + RK 4 Verfahrens.

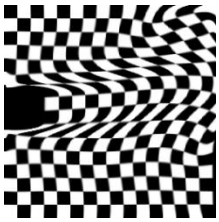
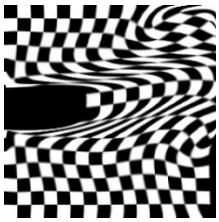
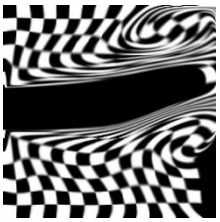
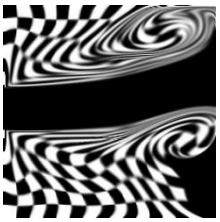
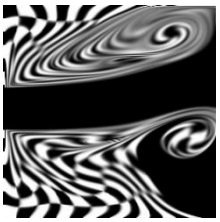
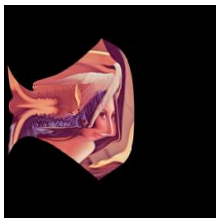
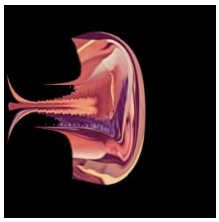
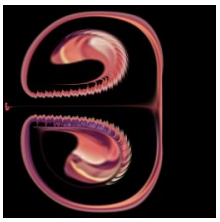
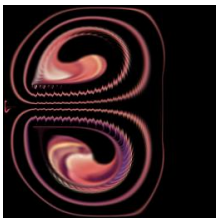
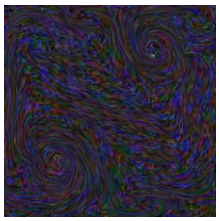
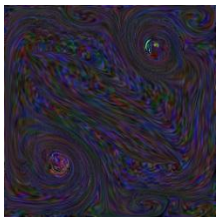
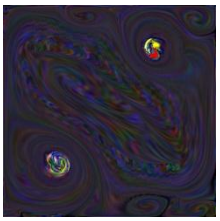
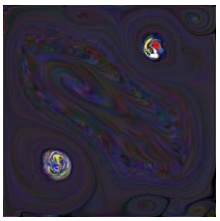
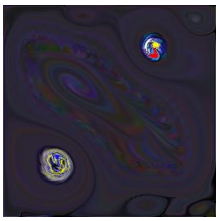
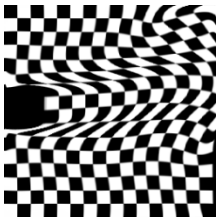
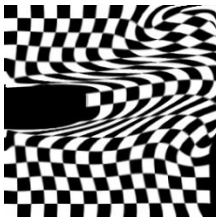
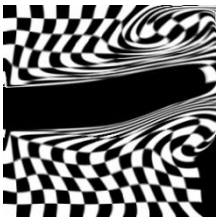
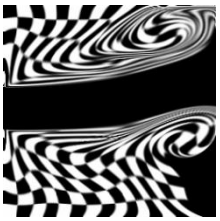
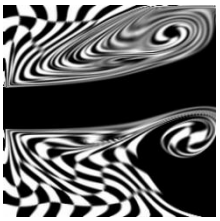
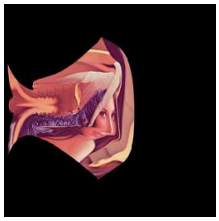
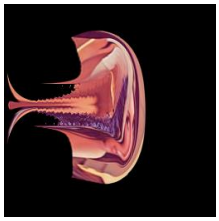
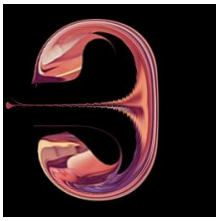
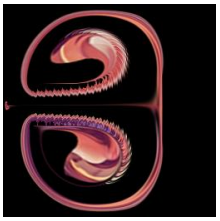
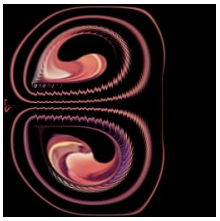
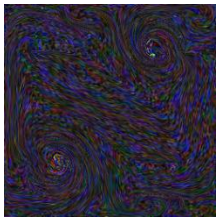
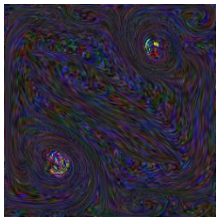
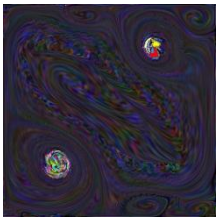
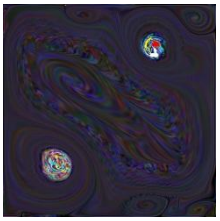
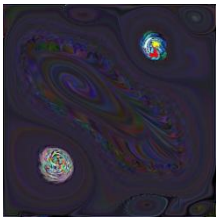
5	10	25	50	100
				
				
				

Tabelle 7-49: Ergebnisse des BFECC-Original + Polynom 3 + RK 4 + Polynom 3 Verfahrens.

5	10	25	50	100
				
				
				

Die Ergebnisse bei diesem Verfahren und seinen Modifikationen sind relativ gut. Allein im BFECC-Original Verfahren (Tabelle 7-43) ist schon eine deutliche Verbesserung gegenüber dem Semi-Lagrange Verfahren (Tabelle 7-8) zu erkennen. Allerdings ist auch hier mit der Schrittzahl eine immer stärker werdende Unschärfe erkennbar. Besonders in den Ergebnissen des zweiten und dritten Vektorfeldes ist ein Verlust der Farbkonstanz zu erkennen. Hier tendieren die Farben in Richtung der absoluten Farbanteile.

Wird ein höherer Grad des Polynoms zum Interpolieren der Daten verwendet steigt auch die Schärfe der Ergebnisse (Tabelle 7-44 und Tabelle 7-45). Allerdings wird durch den höheren Grad der Verlust der Farbkonstanz nicht behoben.

Bei der Polynom 5 Modifikation (Tabelle 7-45) sind innerhalb der einzelnen Zellen des Schachbrettmusters Artefakte zu erkennen (siehe Abbildung 7-11). Diese Artefakte werden durch das Polynom fünften Grades verursacht. Der Korrekturschritt verstärkt diese Artefakte zusätzlich, so dass sie hier sichtbar werden. Die Limitierung der Daten ist hier nicht in der Lage diese Artefakte komplett abzufangen.

Abbildung 7-11: Artefakte bei der Polynom 5 Modifikation. Dies ist eine Vergrößerung des Ergebnisses des high_viscosity.dat Vektorfeldes nach 50 Schritten. Eine Stelle mit Artefakten ist markiert.

Die Modifikationen, die nur im letzten Schritt Polynome höheren Grades einsetzen (Tabelle 7-46 und Tabelle 7-47), erzielen hingegen keine Verbesserung des Ergebnisses. Aufgrund des nur bilinear ausgeführten Korrekturschrittes kombiniert mit der Interpolation mit einem höheren Polynomgrad entstehen neue Fehler und die Originaldaten sind nicht mehr zu erkennen. Besonders bei der Polynom 5 Last Modifikation (Tabelle 7-47) ist gut zu erkennen, dass die Artefakte stark verstärkt und ausgeweitet werden, was zu einer hohen Verpixelung führt.

In der RK 4 Modifikation ist das Ergebnis (Tabelle 7-48) wie erwartet wieder unschärfer. Dies liegt daran, dass hier wieder der bilineare Ansatz verwendet wird um die Daten zu interpolieren. Allerdings wird in dieser Modifikation der Fehler des Verlusts der Farbkonstanz behoben. Dies liegt daran, dass in dem Korrekturschritt ein Forward- und ein Backward-Schritt verwendet wird. Diese Schritte sollten im Idealfall aufeinander fallen. Bei der Euler‘schen Methode ist an Stellen mit starker Krümmung im Vektorfeld die Differenz der Positionen und damit die der Farbinformation zu groß. Mit dem klassischen Runge-Kutta Verfahren wird dieser Krümmung besser gefolgt, was eine niedrigere Differenz zur Folge hat. Dies verringert wiederum den Fehler im Korrekturschritt, woraus der Erhalt der Farbkonstanz resultiert.

Die Polynom 3 + RK 4 + Polynom 3 Modifikation (Tabelle 7-49) ist die Kombination der RK 4 Methode mit einem Polynom höheren Grades zur Interpolation der Daten. Die hier erstellten Ergebnisse (Tabelle 7-49) vereinen eine gute Schärfe bei hohen Schrittzahlen mit dem Erhalt der Farbkonstanz.

Der Verlust der Farbkonstanz an den Punkten bei den RK 4 Modifikationen (Tabelle 7-48 und Tabelle 7-49) ist vermutlich darauf zurückzuführen, dass an diesen Stellen auch das klassische Runge-Kutta Verfahren nicht in der Lage ist der Krümmung des Vektorfeldes ausreichend genau zu folgen. Eine Verringerung der Integrationsschrittweite h würde hier Abhilfe schaffen. Allerdings würde dies auch einen größeren Rechenaufwand zur Herstellung der gleichen Integrationsweite bedeuten, da mehr Schritte berechnet werden müssten.

7.3.1.2.2 Differenzbilder

Tabelle 7-50: Differenzbilder des BFECC-Original Verfahrens zu den Referenzbildern.

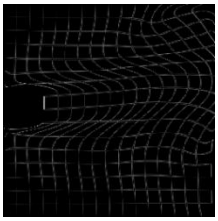
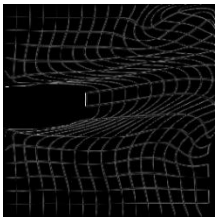
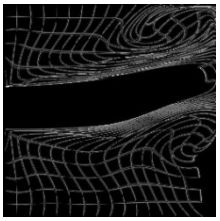
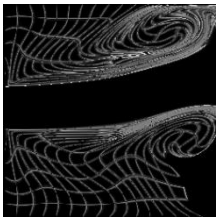
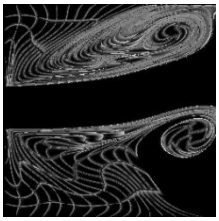
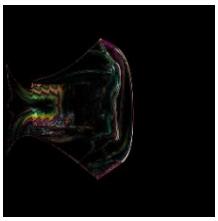
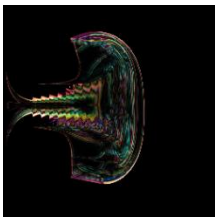
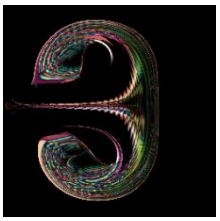
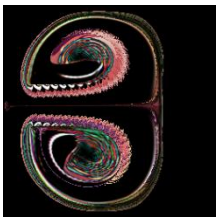
5	10	25	50	100
				
				

Tabelle 7-51: Differenzbilder des BFECC-Original + Polynom 3 Verfahrens zu den Referenzbildern.

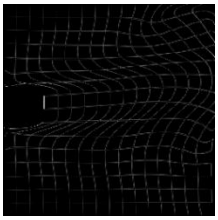
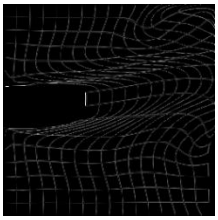
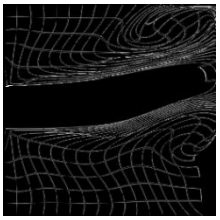
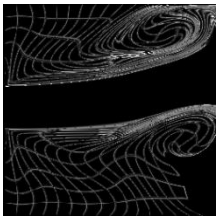
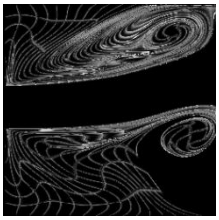
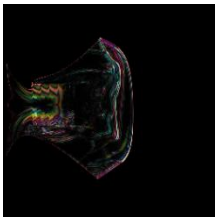
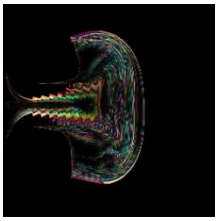
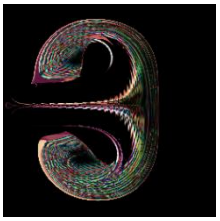
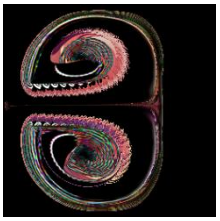
5	10	25	50	100
				
				

Tabelle 7-52: Differenzbilder des BFECC-Original + Polynom 5 Verfahrens zu den Referenzbildern.

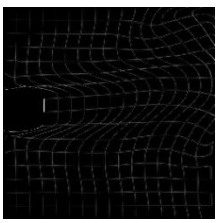
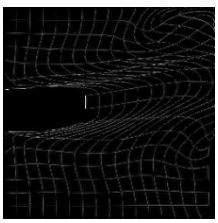
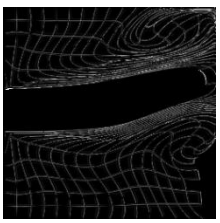
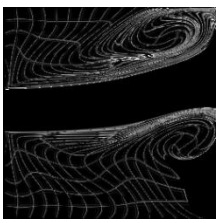
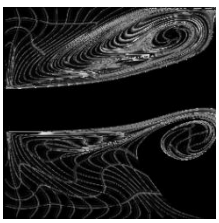
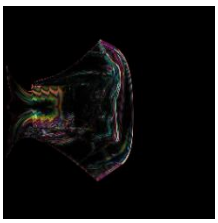
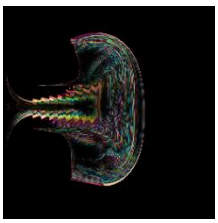
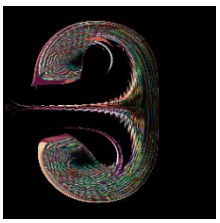
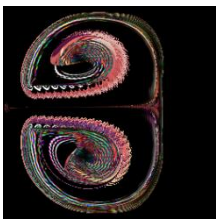
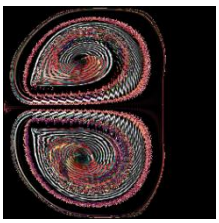
5	10	25	50	100
				
				

Tabelle 7-53: Differenzbilder des BFECC-Original + Polynom 3 Last Verfahrens zu den Referenzbildern.

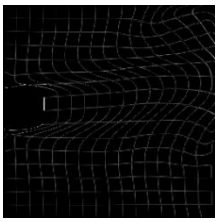
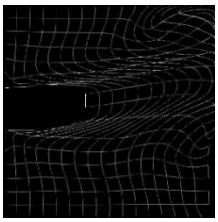
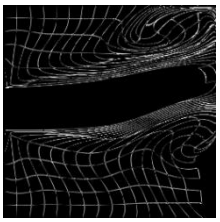
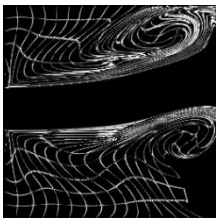
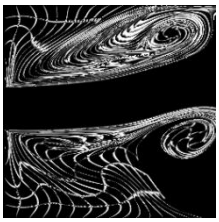
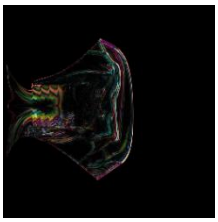
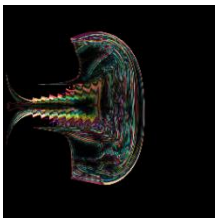
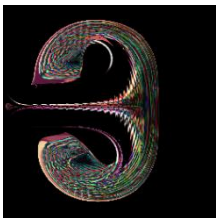
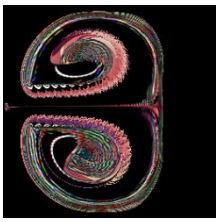
5	10	25	50	100
				
				

Tabelle 7-54: Differenzbilder des BFECC-Original + Polynom 5 Last Verfahrens zu den Referenzbildern.

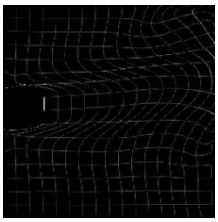
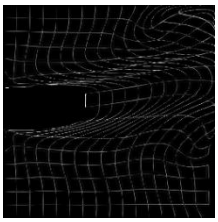
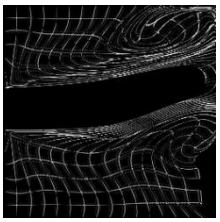
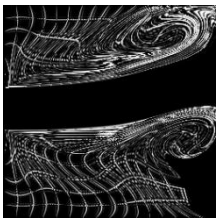
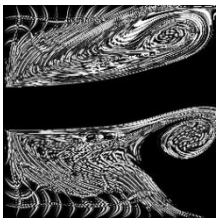
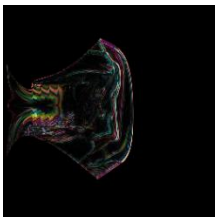
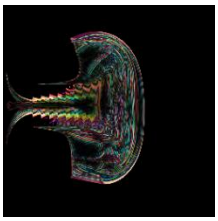
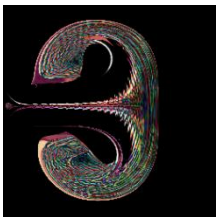
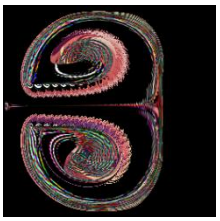
5	10	25	50	100
				
				

Tabelle 7-55: Differenzbilder des BFECC-Original + RK 4 Verfahrens zu den Referenzbildern.

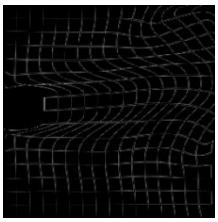
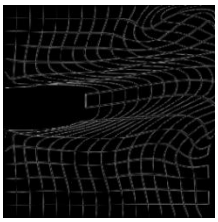
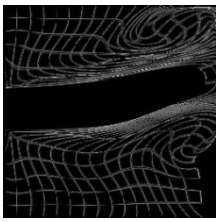
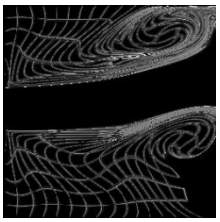
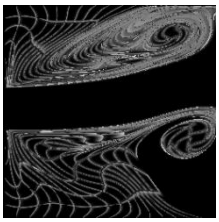
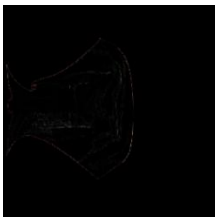
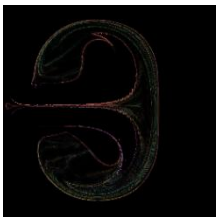
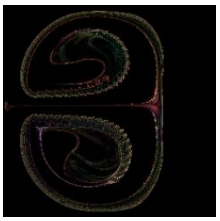
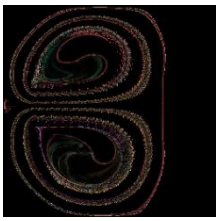
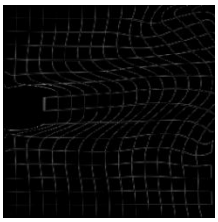
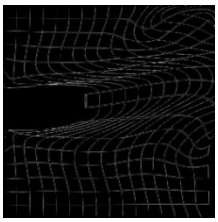
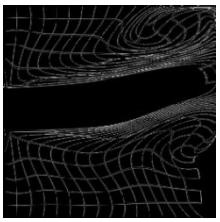
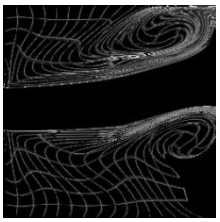
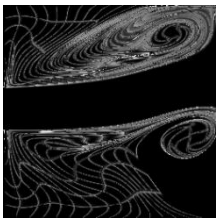
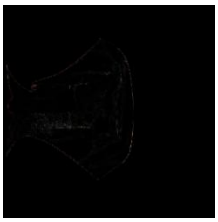
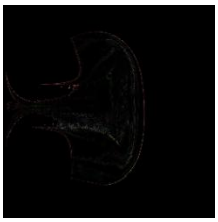
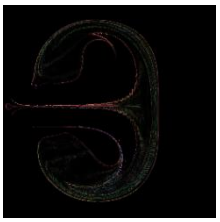
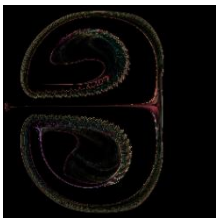
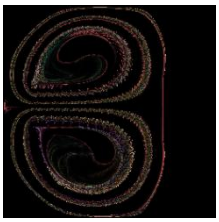
5	10	25	50	100
				
				

Tabelle 7-56: Differenzbilder des BFECC-Original + Polynom 3 + RK 4 + Polynom 3 Verfahrens zu den Referenzbildern.

5	10	25	50	100
				
				

Die Differenz dieses Verfahrens zu den Referenzbildern ist bei den ersten drei Modifikationen (Tabelle 7-50, Tabelle 7-51 und Tabelle 7-52) in etwa gleich. Bei dem zweiten Vektorfeld ist die Differenz aufgrund der mangelnden Farbkonstanz höher.

Bei den Modifikationen, welche die Polynome höheren Grades nur im letzten Schritt einsetzen (Tabelle 7-53 und Tabelle 7-54), ist die Differenz allerdings größer. Dies liegt daran, dass diese Modifikationen das Muster in dem Ergebnis auflösen und zu einer starken Verpixelung führen.

Bei den beiden Modifikationen, die das klassische Runge-Kutta Verfahren im Korrekturschritt einsetzen (Tabelle 7-55 und Tabelle 7-56), ist die Differenz zu den Referenzbildern deutlich niedriger, als bei den anderen Modifikationen. Das liegt daran, dass die Referenzbilder auch mithilfe des klassischen Runge-Kutta Verfahrens berechnet wurden.

Der zusätzliche Einsatz von Polynomen höheren Grades zur Interpolation der Daten (Tabelle 7-56) reduziert die Differenz zusätzlich.

7.3.1.2.3 Reversibilität

Tabelle 7-57: Ergebnisse der Reversibilitätsberechnung des BFECC-Original Verfahrens.

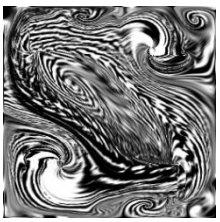
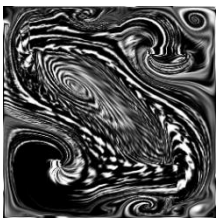
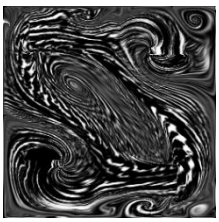
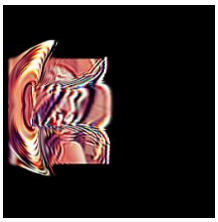
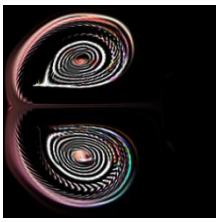
5	10	25	50	100
				
				

Tabelle 7-58: Ergebnisse der Reversibilitätsberechnung des BFECC-Original + Polynom 3 Verfahrens.

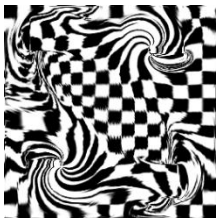
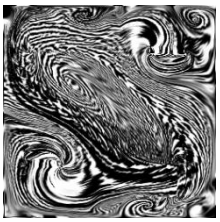
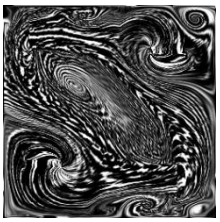
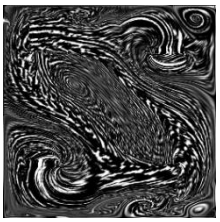
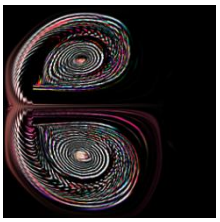
5	10	25	50	100
				
				

Tabelle 7-59: Ergebnisse der Reversibilitätsberechnung des BFECC-Original + Polynom 5 Verfahrens.

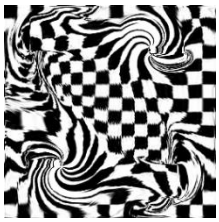
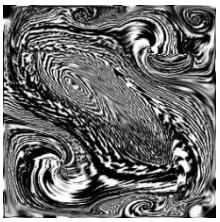
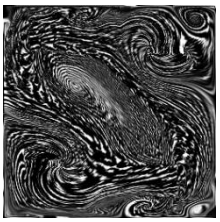
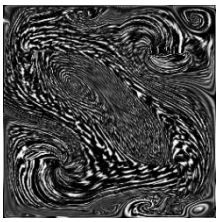
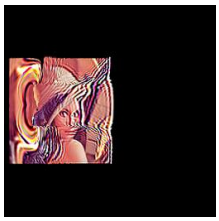
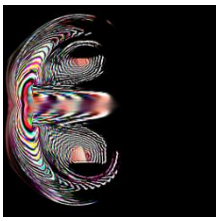
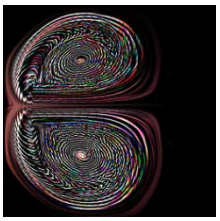
5	10	25	50	100
				
				

Tabelle 7-60: Ergebnisse der Reversibilitätsberechnung des BFECC-Original + Polynom 3 Last Verfahrens.

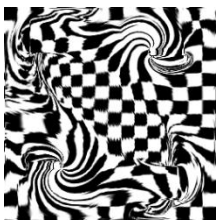
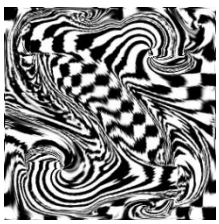
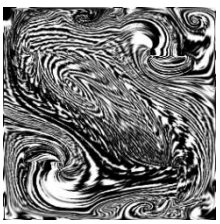
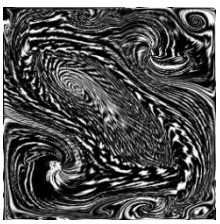
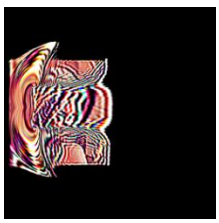
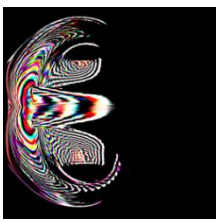
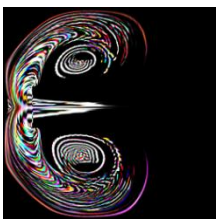
5	10	25	50	100
				
				

Tabelle 7-61: Ergebnisse der Reversibilitätsberechnung des BFECC-Original + Polynom 5 Last Verfahrens.

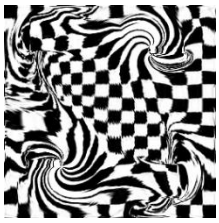
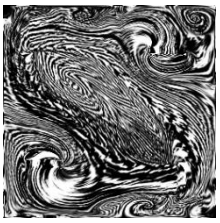
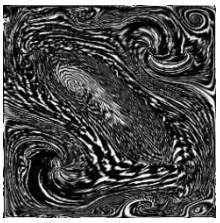
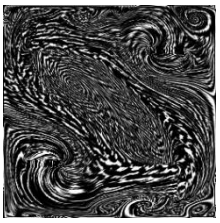
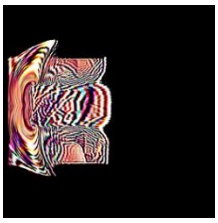
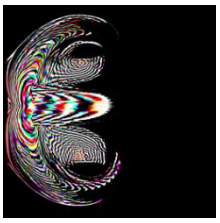
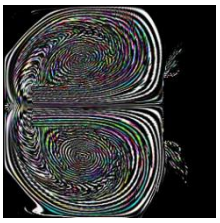
5	10	25	50	100
				
				

Tabelle 7-62: Ergebnisse der Reversibilitätsberechnung des BFECC-Original + RK 4 Verfahrens.

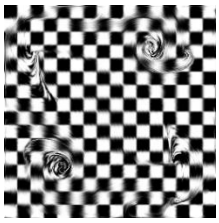
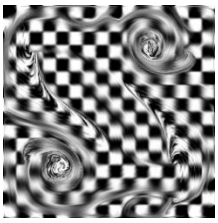
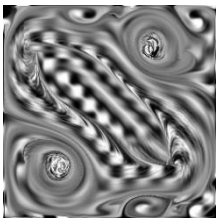
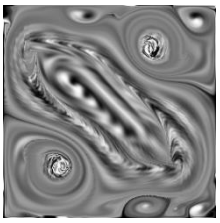
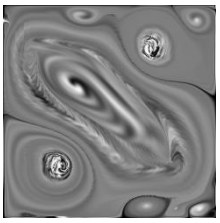
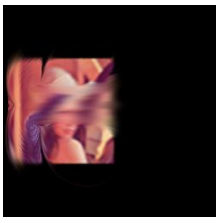
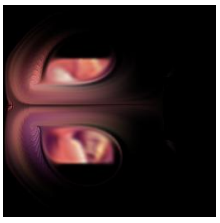
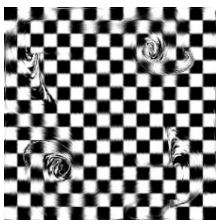
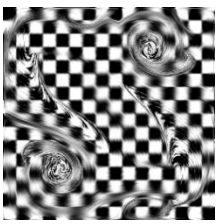
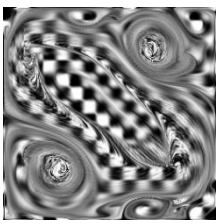
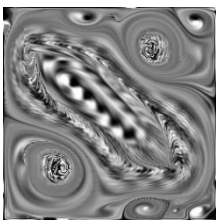
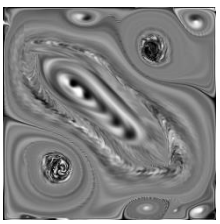
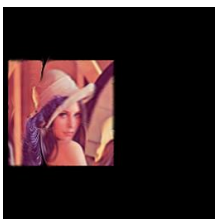
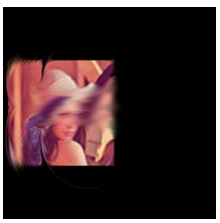
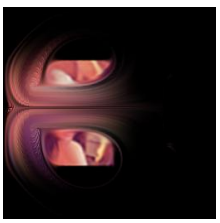
5	10	25	50	100
				
				

Tabelle 7-63: Ergebnisse der Reversibilitätsberechnung des BFECC-Original + Polynom 3 + RK 4 + Polynom 3 Verfahrens.

5	10	25	50	100
				
				

Bei den ersten fünf Methoden (Tabelle 7-57, Tabelle 7-58, Tabelle 7-59, Tabelle 7-60 und Tabelle 7-61) ist die Position der Pixel nicht ausreichend genau berechnet worden. Aus diesem Grund ist die ursprüngliche Form des Bildes nicht wiederherstellbar. Auch die Farbwiederherstellung ist bei diesen Modifikationen nur für sehr kleine Schrittzahlen noch ausreichend gegeben.

Bei den Modifikationen, die das klassische Runge-Kutta Verfahren einsetzen (Tabelle 7-62 und Tabelle 7-63), wird hingegen die Position der einzelnen Pixel auch bei hohen Schrittzahlen gut berechnet. Die Farbwiederherstellung der einzelnen Pixel funktioniert an Stellen, die keiner starken räumlichen Änderung des Vektorfeldes unterworfen sind, sehr gut. Stellen, die starken räumlichen Änderungen ausgesetzt sind, haben hierbei allerdings Probleme. Dies ist besonders gut in der Augenpartie des Bildes des zweiten Vektorfeldes zu beobachten. Diese Partie wird ab mittleren Schrittzahlen verwischt. Das ist das Resultat davon, dass die entsprechenden Pixel den Farbwert aus einer Kombination anderer Pixel erhalten haben. Bei der Rückrechnung kann dieser Farbwert nicht mehr ausreichend genau aufgeteilt werden um den Originalfarbwert wieder herzustellen.

Der Unterschied zwischen dem bilinearen Ansatz und einem Polynom höheren Grades zur Interpolation der Daten ist lediglich in der etwas höheren Schärfe der Ergebnisse bei dem Polynom höheren Grades zu bemerken.

7.3.1.2.4 Optischer Fluss

Tabelle 7-64: Optischer Fluss des BFECC-Original Verfahrens.

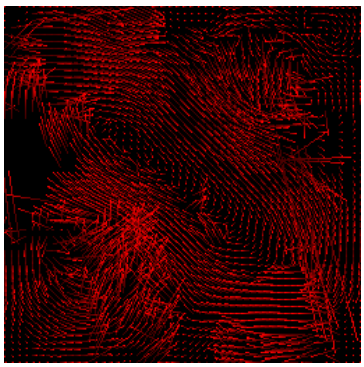
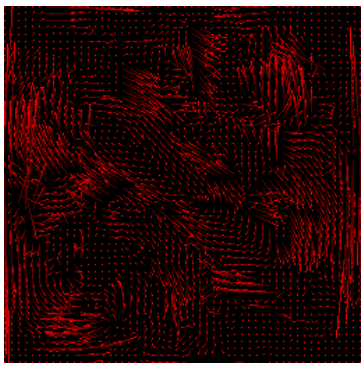
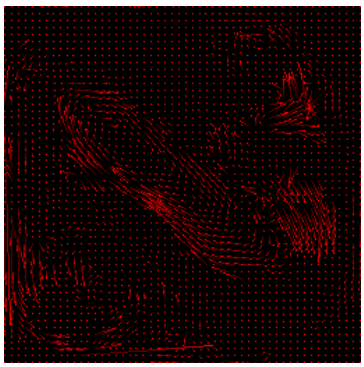
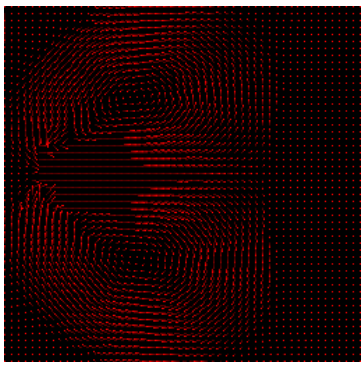
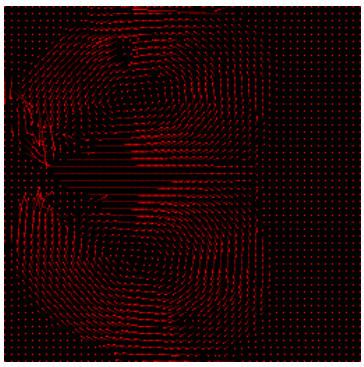
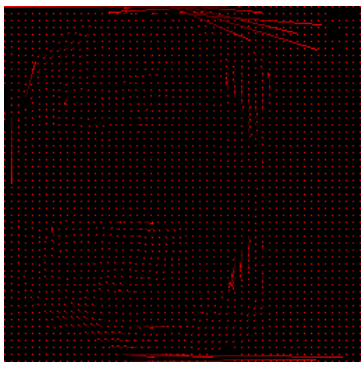
0-1	25-26	100-101
		
		

Tabelle 7-65: Optischer Fluss des BFECC-Original + Polynom 3 Verfahrens.

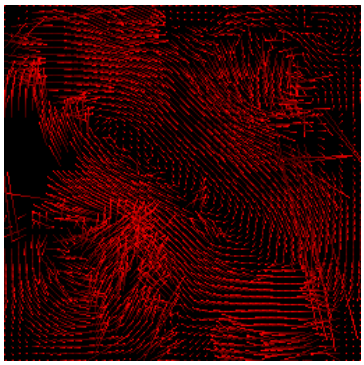
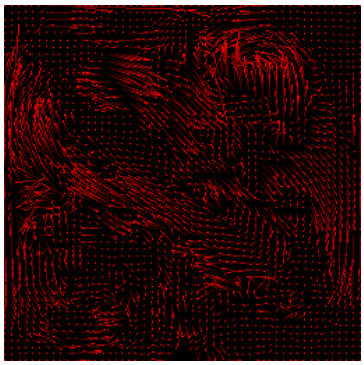
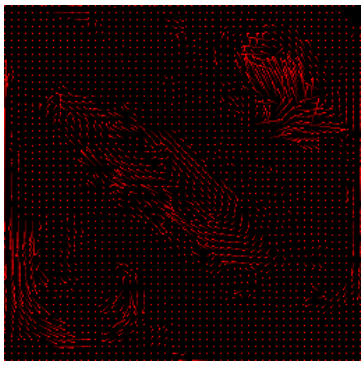
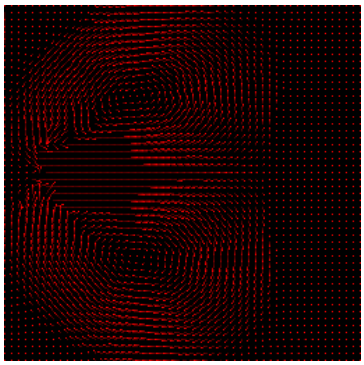
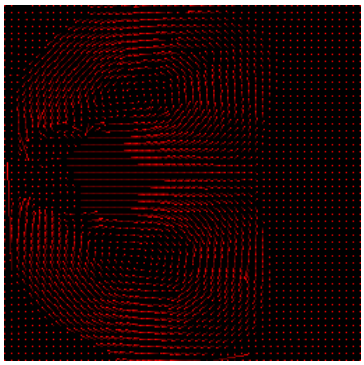
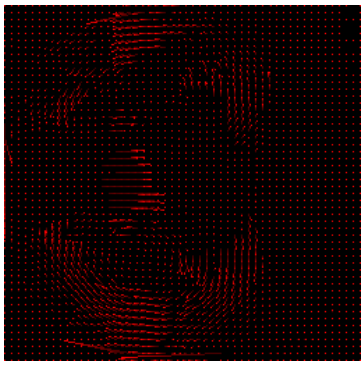
0-1	25-26	100-101
		
		

Tabelle 7-66: Optischer Fluss des BFECC-Original + Polynom 5 Verfahrens.

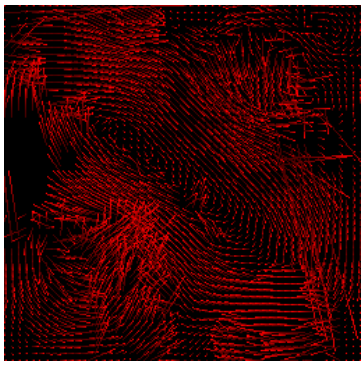
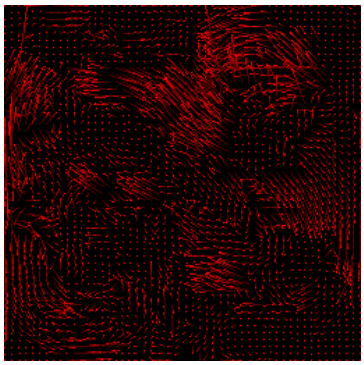
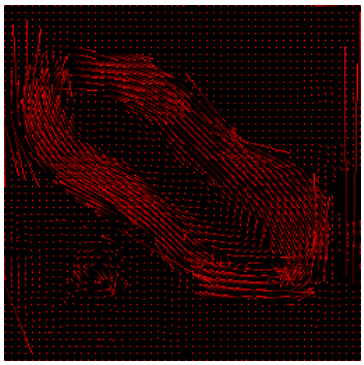
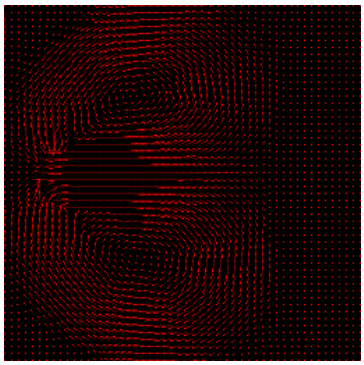
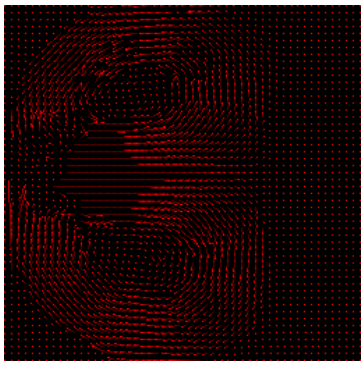
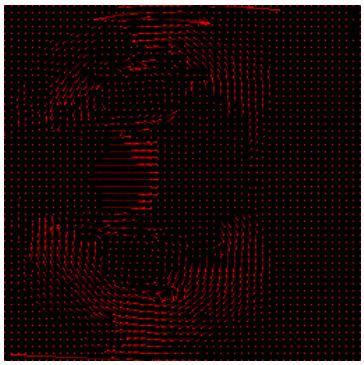
0-1	25-26	100-101
		
		

Tabelle 7-67: Optischer Fluss des BFECC-Original + Polynom 3 Last Verfahrens.

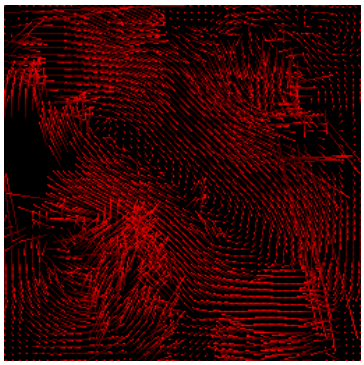
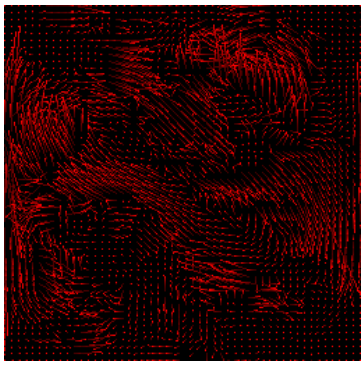
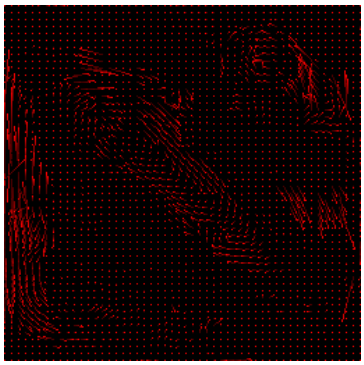
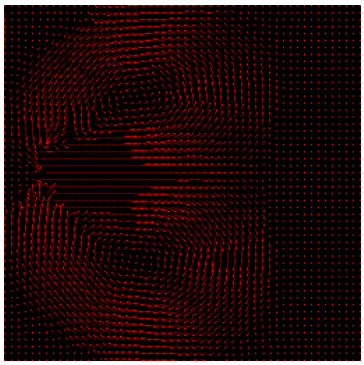
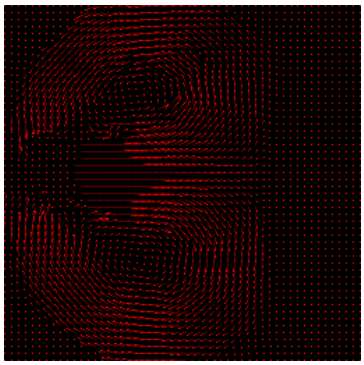
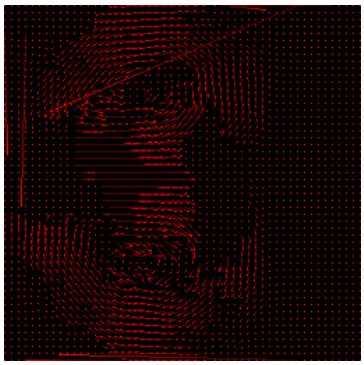
0-1	25-26	100-101
		
		

Tabelle 7-68: Optischer Fluss des BFECC-Original + Polynom 5 Last Verfahrens.

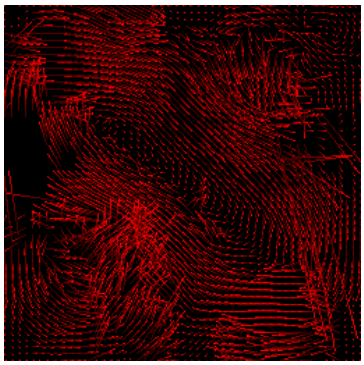
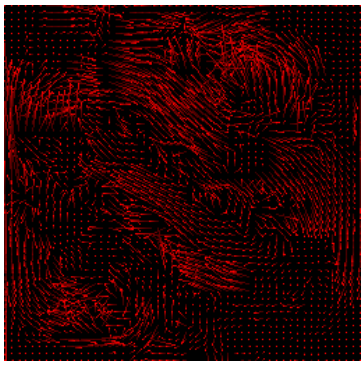
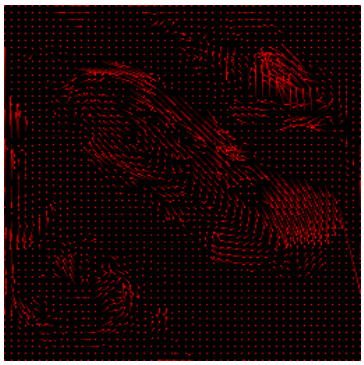
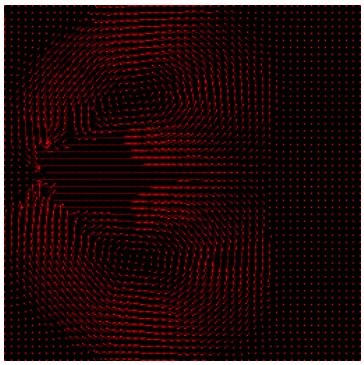
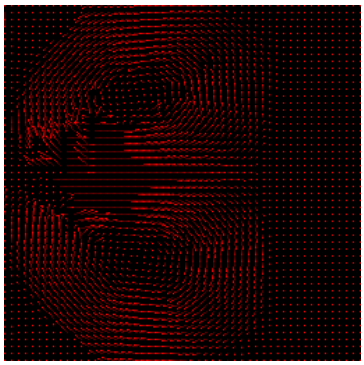
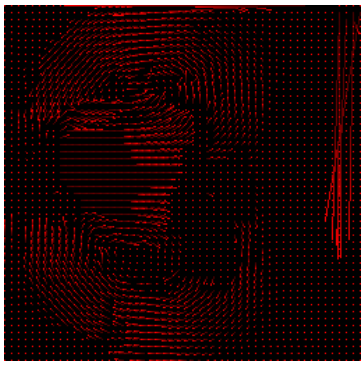
0-1	25-26	100-101
		
		

Tabelle 7-69: Optischer Fluss des BFECC-Original + RK 4 Verfahrens.

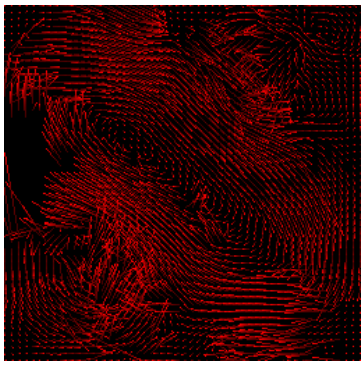
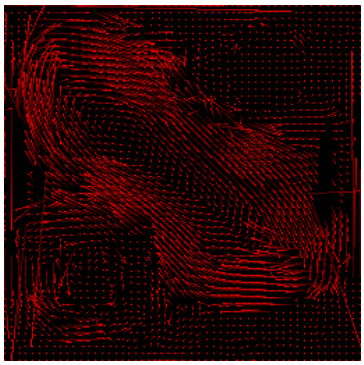
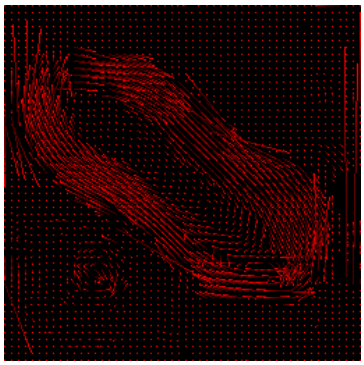
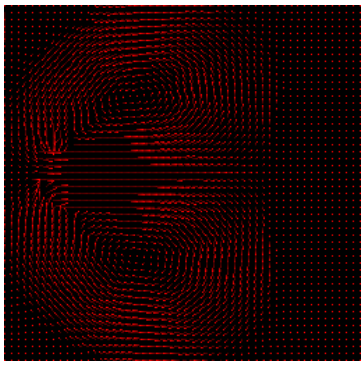
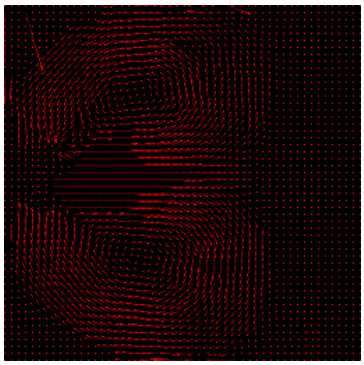
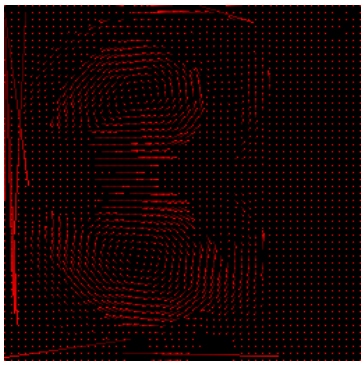
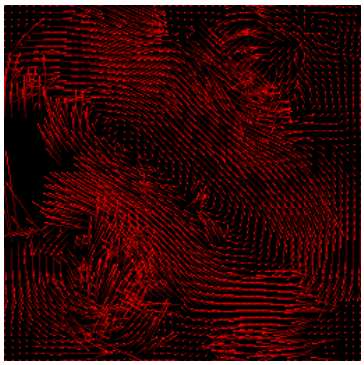
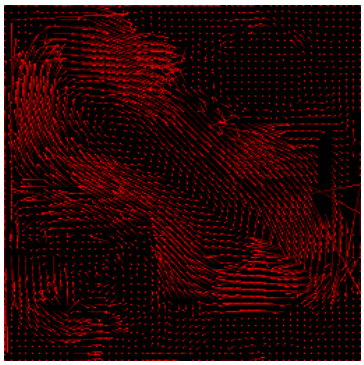
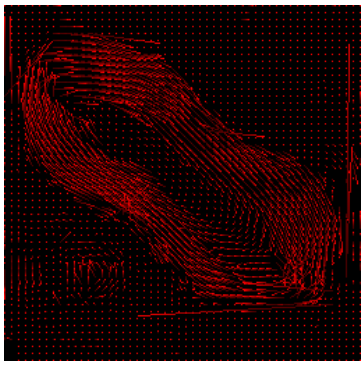
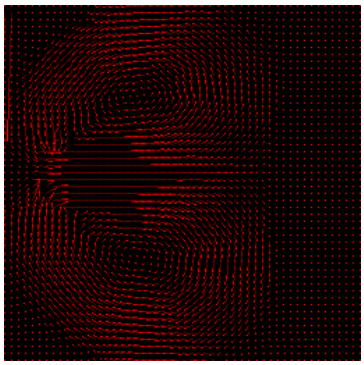
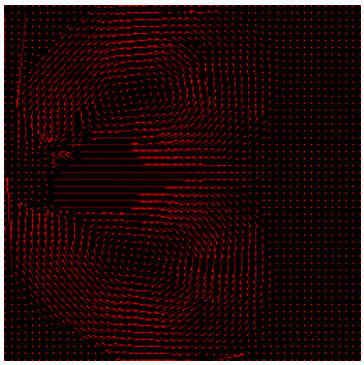
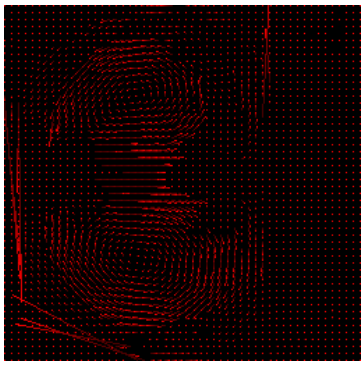
0-1	25-26	100-101
		
		

Tabelle 7-70: Optischer Fluss des BFECC-Original + Polynom 3 + RK 4 + Polynom 3 Verfahrens.

0-1	25-26	100-101
		
		

Der optische Fluss ist bei den Methoden ohne das klassische Runge-Kutta Verfahren (Tabelle 7-64, Tabelle 7-65, Tabelle 7-66, Tabelle 7-67 und Tabelle 7-68) nur bei niedrigeren Schrittzahlen ausreichend wiederzuerkennen. Je höher der Grad des Polynoms des verwendeten Verfahrens, desto

besser ist der optische Fluss erkennbar. Hierbei entsteht kein Unterschied, ob der höhere Grad nur bei dem letzten Schritt des Verfahrens angewandt wird oder schon beim Korrekturschritt.

Bei den Methoden, die das klassische Runge-Kutta Verfahren einsetzen (Tabelle 7-69 und Tabelle 7-70), ist der optische Fluss hingegen auch bei hohen Schrittzahlen gut zu erkennen.

7.3.1.2.5 Spektrum

Tabelle 7-71: Spektrum des BFECC-Original Verfahrens.

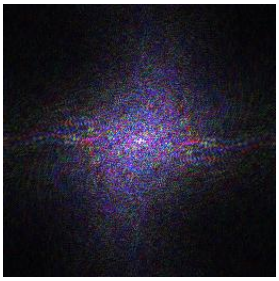
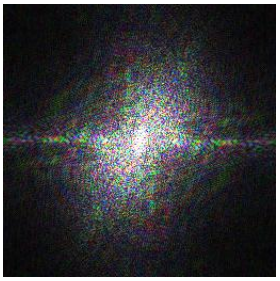
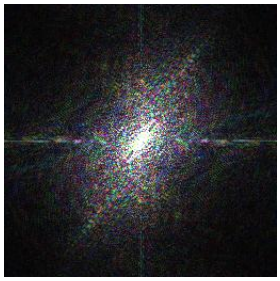
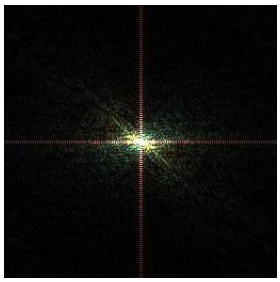
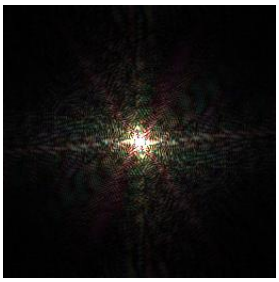
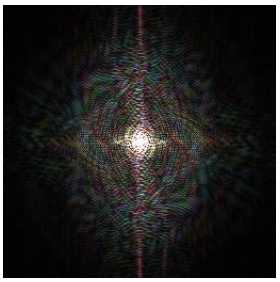
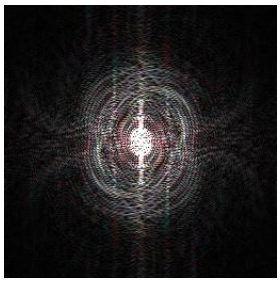
0	5	25	100
			
			

Tabelle 7-72: Spektrum des BFECC-Original + Polynom 3 Verfahrens.

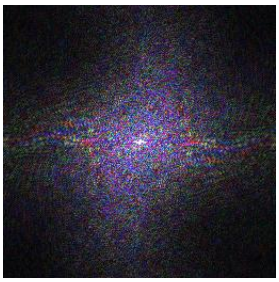
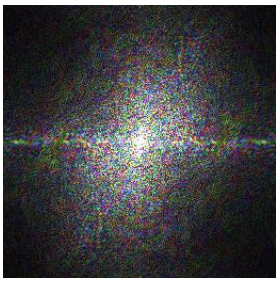
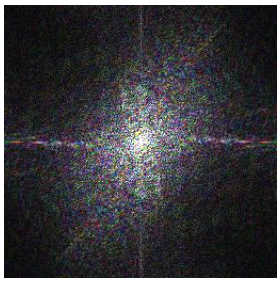
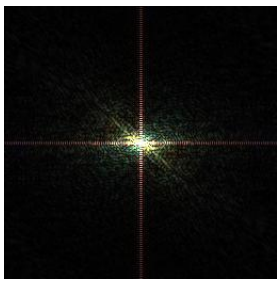
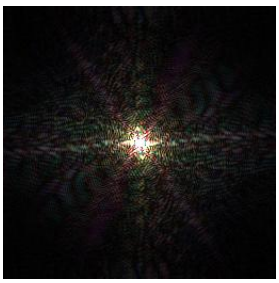
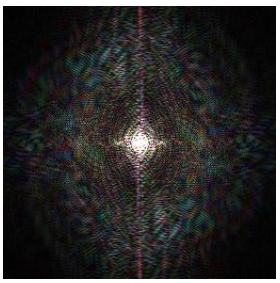
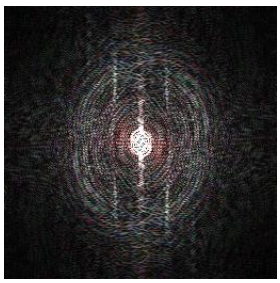
0	5	25	100
			
			

Tabelle 7-73: Spektrum des BFECC-Original + Polynom 5 Verfahrens.

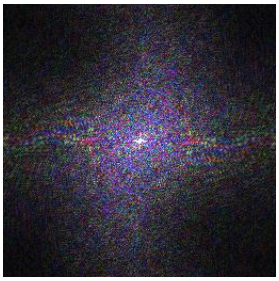
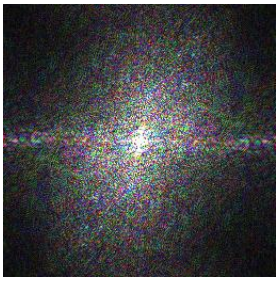
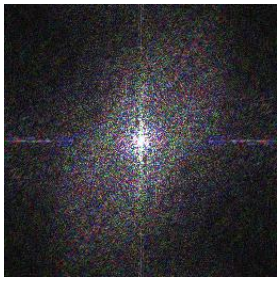
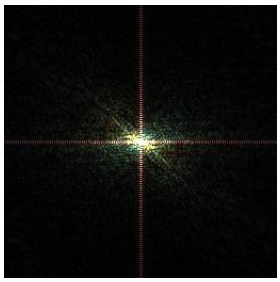
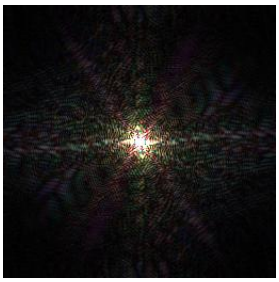
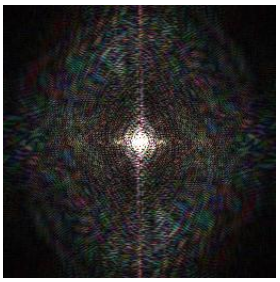
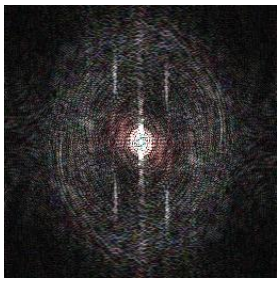
0	5	25	100
			
			

Tabelle 7-74: Spektrum des BFECC-Original + Polynom 3 Last Verfahrens.

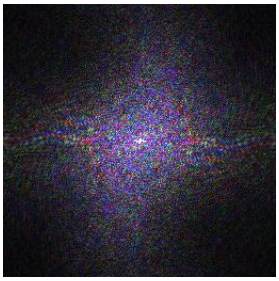
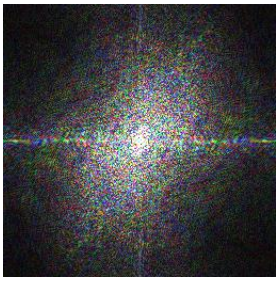
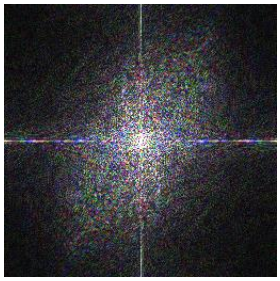
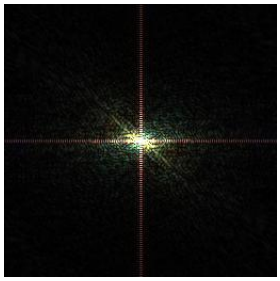
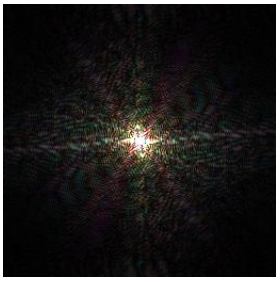
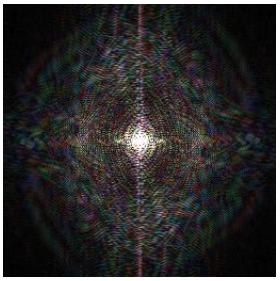
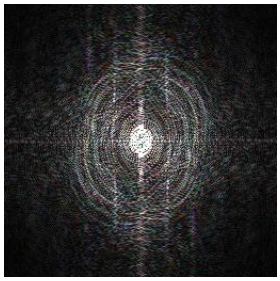
0	5	25	100
			
			

Tabelle 7-75: Spektrum des BFECC-Original + Polynom 5 Last Verfahrens.

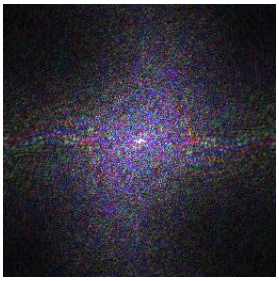
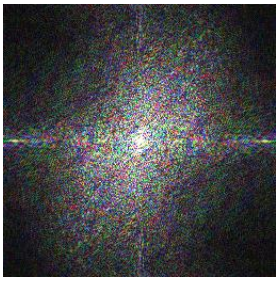
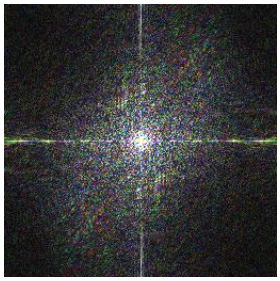
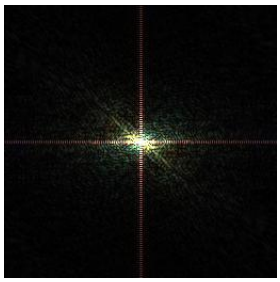
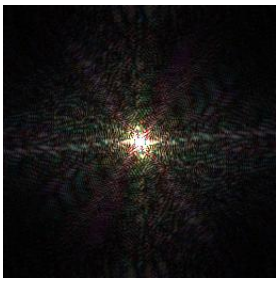
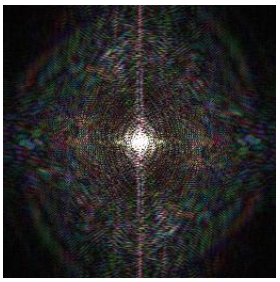
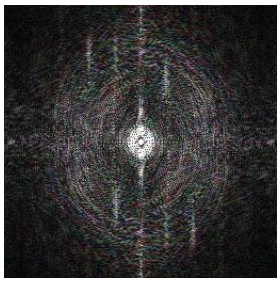
0	5	25	100
			
			

Tabelle 7-76: Spektrum des BFECC-Original + RK 4 Verfahrens.

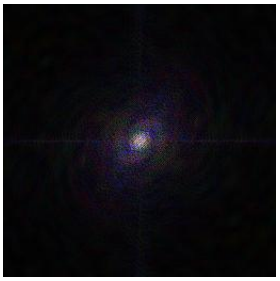
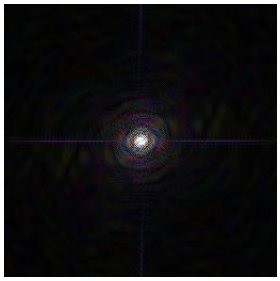
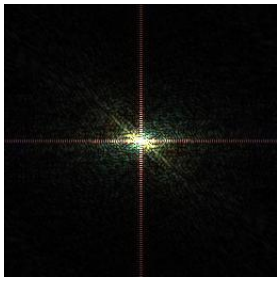
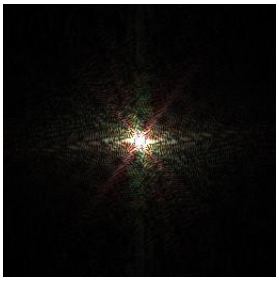
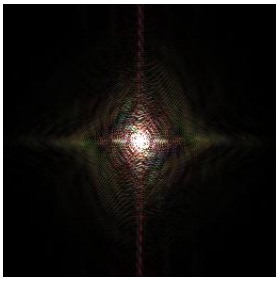
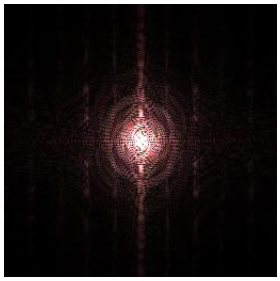
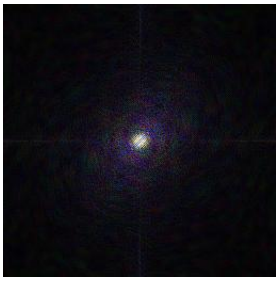
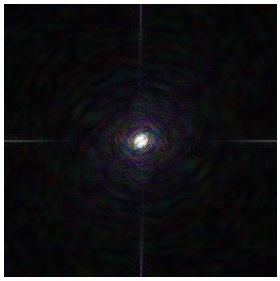
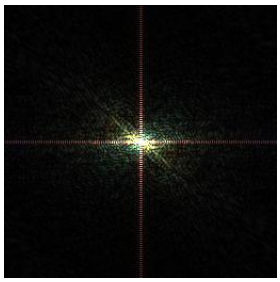
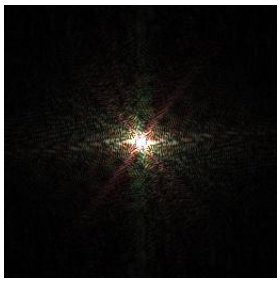
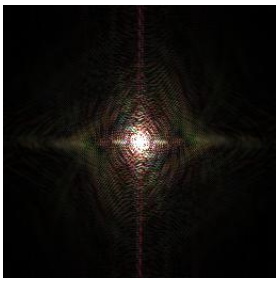
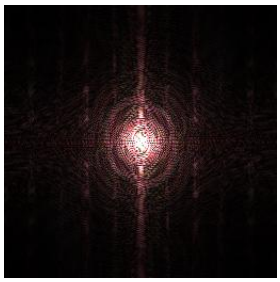
0	5	25	100
			
			

Tabelle 7-77: Spektrum des BFECC-Original + Polynom 3 + RK 4 + Polynom 3 Verfahrens.

0	5	25	100
			
			

Das Spektrum wird in den Modifikationen, welche das klassische Runge-Kutta Verfahren nicht einsetzen (Tabelle 7-71, Tabelle 7-72, Tabelle 7-73, Tabelle 7-74 und Tabelle 7-75), stark ausgedehnt.

Bei den Modifikationen, welche das klassische Runge-Kutta Verfahren einsetzen (Tabelle 7-76 und Tabelle 7-77), wird das Spektrum hingegen auf einen Punkt konzentriert. Das zweite Vektorfeld tendiert zudem noch ins Rötliche.

Bei der Modifikation, welche das klassische Runge-Kutta Verfahren und ein Polynom höheren Grades zum Interpolieren der Daten verwendet (Tabelle 7-77), wird das Spektrum nicht so stark auf den Punkt konzentriert wie bei der bilinearen Interpolation (Tabelle 7-76).

7.3.1.2.6 Performance

Tabelle 7-78: Performance des BFECC-Original (Links), des BFECC-Original + Polynom 3 (Mitte) und des BFECC-Original + Polynom 5 (Rechts) Verfahrens.

Gesamtzeit:	7,11
Durchschnittszeit pro Schritt:	0,0711
Kürzeste Zeit eines Schrittes:	0,0637
Längste Zeit eines Schrittes:	0,0831
FPS:	14,07

Gesamtzeit:	44,13
Durchschnittszeit pro Schritt:	0,4413
Kürzeste Zeit eines Schrittes:	0,2968
Längste Zeit eines Schrittes:	0,433
FPS:	2,27

Gesamtzeit:	79,66
Durchschnittszeit pro Schritt:	0,7966
Kürzeste Zeit eines Schrittes:	0,6975
Längste Zeit eines Schrittes:	0,8705
FPS:	1,26

Tabelle 7-79: Performance des BFECC-Original + Polynom 3 Last (Links), des BFECC-Original + Polynom 5 Last (Mitte) und des BFECC-Original + RK 4 (Rechts) Verfahrens.

Gesamtzeit:	19,54
Durchschnittszeit pro Schritt:	0,1954
Kürzeste Zeit eines Schrittes:	0,1506
Längste Zeit eines Schrittes:	0,2381
FPS:	5,12

Gesamtzeit:	32,78
Durchschnittszeit pro Schritt:	0,3278
Kürzeste Zeit eines Schrittes:	0,2996
Längste Zeit eines Schrittes:	0,356
FPS:	3,05

Gesamtzeit:	14,14
Durchschnittszeit pro Schritt:	0,1414
Kürzeste Zeit eines Schrittes:	0,1297
Längste Zeit eines Schrittes:	0,1708
FPS:	7,07

Tabelle 7-80: Performance des BFECC-Original + Polynom 3 + RK 4 + Polynom 3 Verfahrens.

Gesamtzeit:	89,47
Durchschnittszeit pro Schritt:	0,8947
Kürzeste Zeit eines Schrittes:	0,8103
Längste Zeit eines Schrittes:	1,1625
FPS:	1,12

Die Performance des BFECC-Original Verfahren ist gerade noch im akzeptablen Bereich. Bei den eingesetzten Modifikationen bemerkt man, dass die Performance mit dem Grad des Polynoms abnimmt, was auch zu erwarten ist. Die beiden Modifikationen, die das Polynom höheren Grades nur auf den letzten Schritt anwenden, verbessern die Performance im Gegensatz zu den Modifikationen, die das Polynom auch beim Korrekturschritt anwenden, zwar etwas, allerdings nicht in dem Maße um wieder in den echtzeitfähigen Bereich zu kommen.

Die Modifikation, die das klassische Runge-Kutta Verfahren einsetzt, ist von der Performance her gesehen ebenfalls nicht echtzeitfähig. Wird die Modifikation mit einem Polynom höheren Grades kombiniert, so fällt die Performance deutlich.

Auf der CPU erreicht keine dieser Modifikationen die erforderlichen 24 FPS um echtzeitfähig zu sein.

7.3.1.2.7 Bewertung

Dieses Verfahren ist eine gute Verbesserung zum Semi-Lagrange Verfahren. Allerdings ist die Performance auf der CPU relativ schlecht. Die Modifikationen, die Polynome höheren Grades einsetzen, erreichen eine zusätzliche Verbesserung der Qualität. Diese sind allerdings mit entsprechenden Kosten bei der Performance verbunden. Die Modifikation, das Polynom höheren Grades nur im letzten Schritt anzuwenden, ergab hingegen keine Verbesserung, sondern eine Verschlechterung der Ergebnisse.

Setzt man das klassische Runge-Kutta Verfahren zur Integration der Vektorfelddaten ein wird allerdings eine deutliche Verbesserung gegenüber der Euler'schen Methode erreicht. Diese sollte nach Möglichkeit immer verwendet werden.

Die Kombination aus Polynom höheren Grades zur Interpolation der Daten und klassischem Runge-Kutta Verfahren zur Berechnung der Position ergibt das beste Ergebnis dieses Verfahrens. Allerdings ist diese Kombination sehr rechenintensiv.

7.3.1.3 BFECC

Das BFECC Verfahren ist in 6.2.2.5 beschrieben. Es wurde das BFECC Verfahren mit allen Modifikationen untersucht.

7.3.1.3.1 Ergebnisse

Tabelle 7-81: Ergebnisse des BFECC Verfahrens.

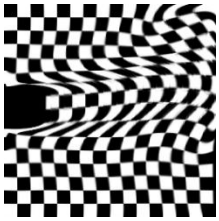
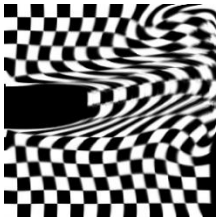
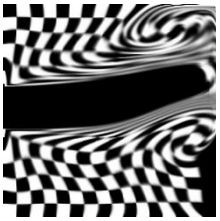
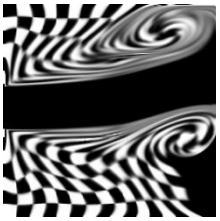
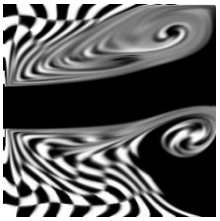
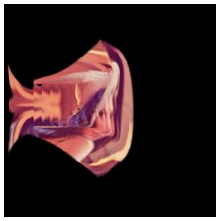
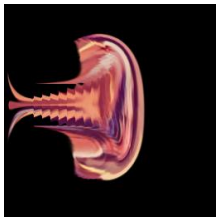
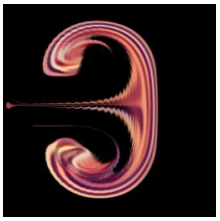
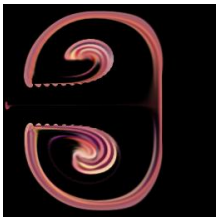
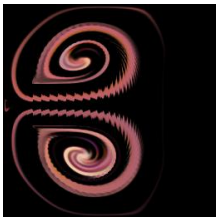
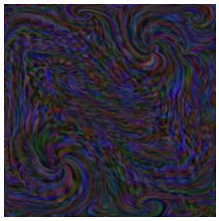
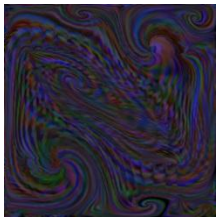
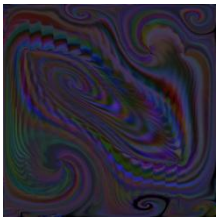
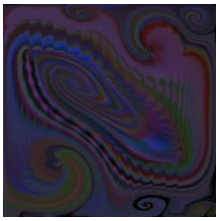
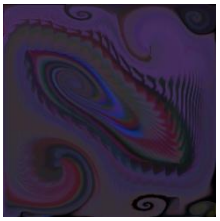
5	10	25	50	100
				
				
				

Tabelle 7-82: Ergebnisse des BFECC + Polynom 3 Verfahrens.

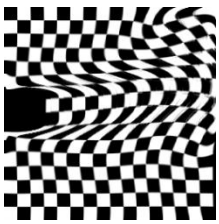
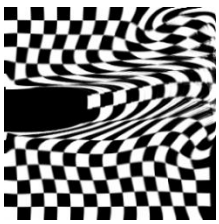
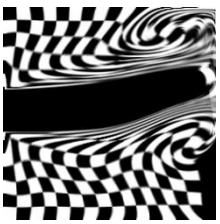
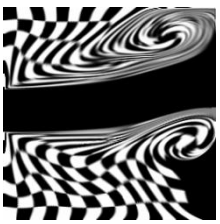
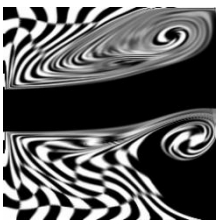
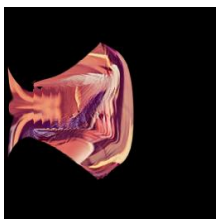
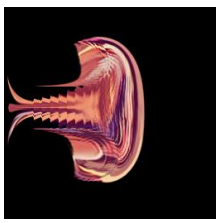
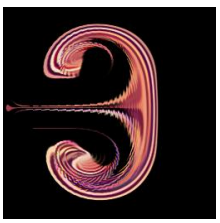
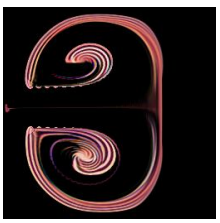
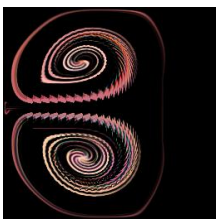
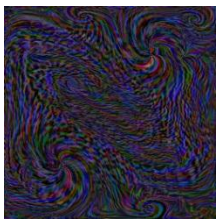
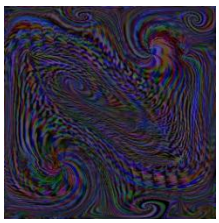
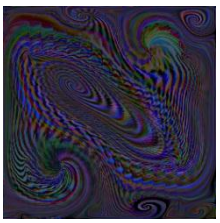
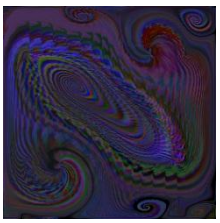
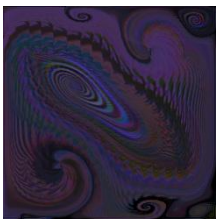
5	10	25	50	100
				
				
				

Tabelle 7-83: Ergebnisse des BFECC + Polynom 5 Verfahrens.

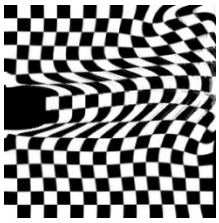
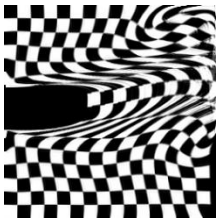
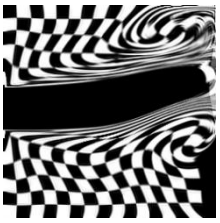
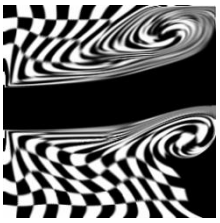
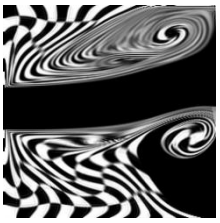
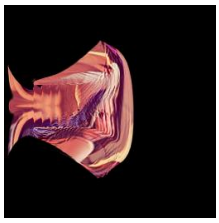
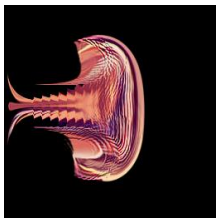
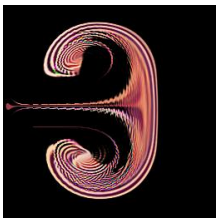
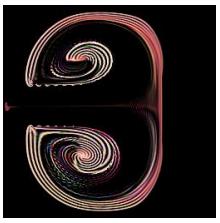
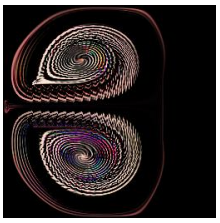
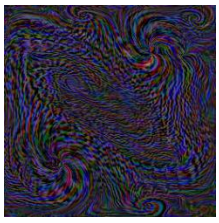
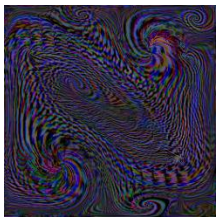
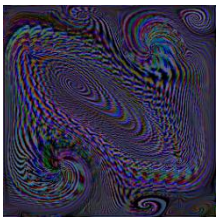
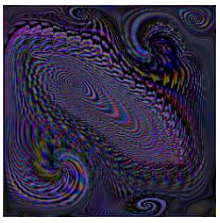
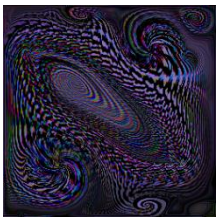
5	10	25	50	100
				
				
				

Tabelle 7-84: Ergebnisse des BFECC + RK 4 Verfahrens.

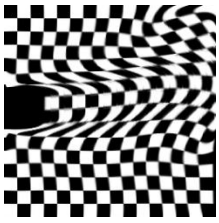
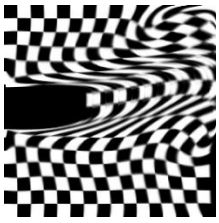
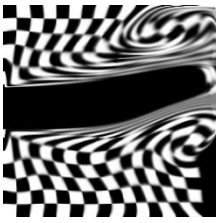
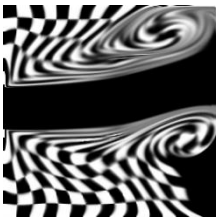
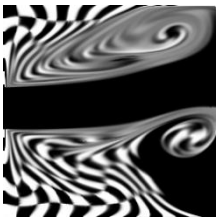
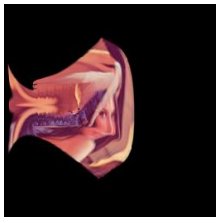
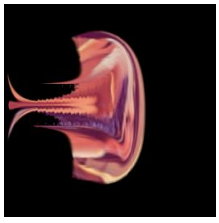
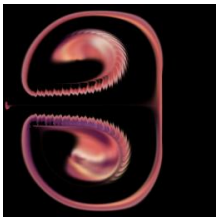
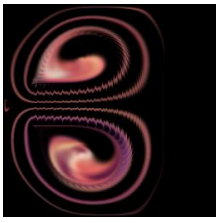
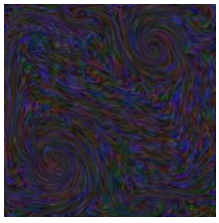
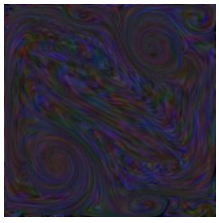
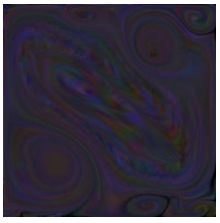
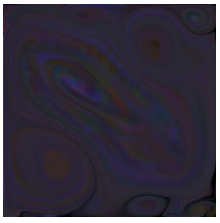
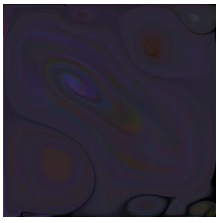
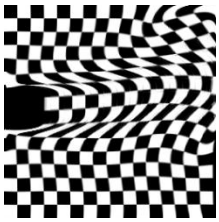
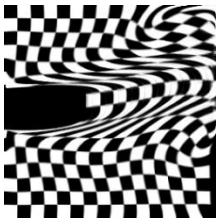
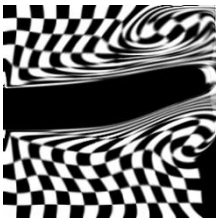
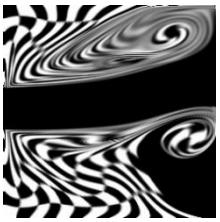
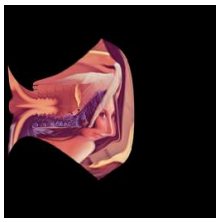
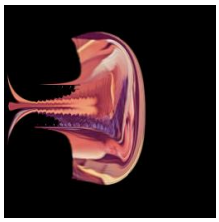
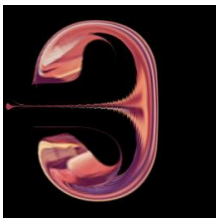
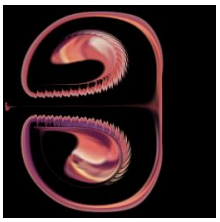
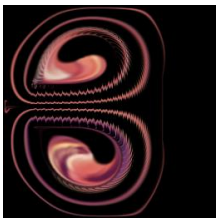
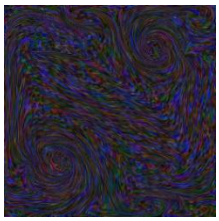
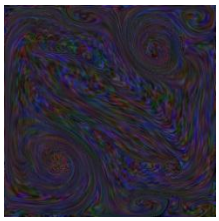
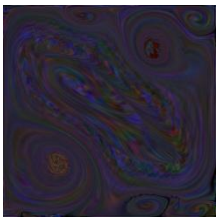
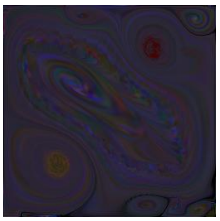
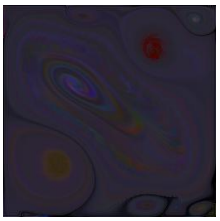
5	10	25	50	100
				
				
				

Tabelle 7-85: Ergebnisse des BFECC + Polynom 3 + RK 4 + Polynom 3 Verfahrens.

5	10	25	50	100
				
				
				

Die Ergebnisse dieses Verfahrens sind mit denen des Original BFECC Verfahrens vergleichbar. Dieses Verfahren produziert allerdings unschärfere Ergebnisse, da der dritte Schritt wegfällt und der erste Schritt nur korrigiert wird. Im Gegensatz zum Original BFECC Verfahren ist die Farbkonstanz auch ohne den Einsatz des klassischen Runge-Kutta Verfahrens gegeben. Dies ist in den Ergebnissen in Tabelle 7-81, Tabelle 7-82 und Tabelle 7-83 zu beobachten.

Allerdings entstehen in diesem Verfahren neue Artefakte (siehe Abbildung 7-12). Diese Artefakte sind wahrscheinlich darauf zurückzuführen, dass der dritte Schritt fehlt und nur der erste Schritt korrigiert wird. Da die Korrektur direkt auf dem Ergebnis ausgeführt wird werden die Artefakte erzeugt. Im Gegensatz dazu wird im Original BFECC Verfahren die Korrektur dazu verwendet um die Quelldaten des Schrittes zu verbessern und diese werden anschließend interpoliert. Dadurch können die Artefakte nicht auftreten. Diese Artefakte werden durch den Einsatz eines Limiters reduziert, aber nicht komplett unterdrückt.

Abbildung 7-12: Artefakte im BFECC Verfahren. Dieses Bild wird mit high_viscosity.dat als Vektorfeld nach 10 Schritten erzeugt.

Wird statt des bilinearen Ansatzes (Tabelle 7-81) ein Polynom höheren Grades (Tabelle 7-82 und Tabelle 7-83) verwendet, wird die Genauigkeit der Bilder erhöht. Dies ist besonders bei dem Schachbrettmuster erkennbar.

Bei den Modifikationen, welche das klassische Runge-Kutta Verfahren einsetzen (Tabelle 7-84 und Tabelle 7-85), ist wie bei dem Original BFECC Verfahren eine Qualitätssteigerung der Ergebnisse feststellbar. Auch die Farbkonstanz wird in diesen Modifikationen bewahrt.

Die Kombination des klassischen Runge-Kutta Verfahrens mit einem Polynom höheren Grades zum Interpolieren der Daten ergibt nochmal eine Qualitätsverbesserung.

Die Punkte, an denen die Farbkonstanz bei den RK 4 Modifikationen verloren geht, haben wahrscheinlich die gleiche Ursache wie beim Original BFECC Verfahren. Das klassische Runge-Kutta Verfahren kann dem Vektorfeld an diesen Stellen nicht ausreichend genau folgen.

7.3.1.3.2 Differenzbilder

Tabelle 7-86: Differenzbilder des BFECC Verfahrens zu den Referenzbildern.

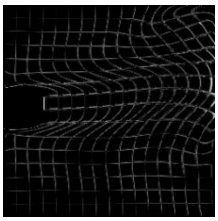
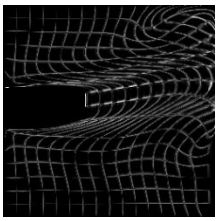
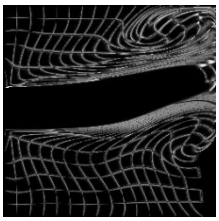
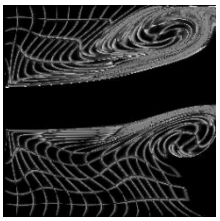
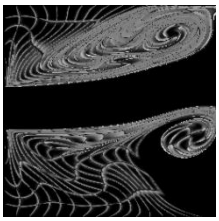
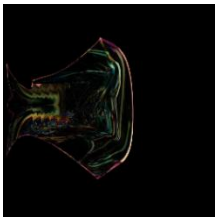
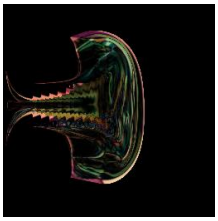
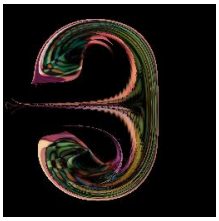
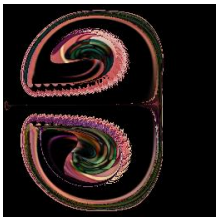
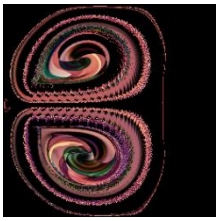
5	10	25	50	100
				
				

Tabelle 7-87: Differenzbilder des BFECC + Polynom 3 Verfahrens zu den Referenzbildern.

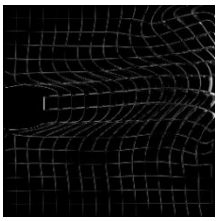
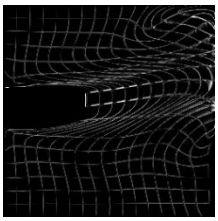
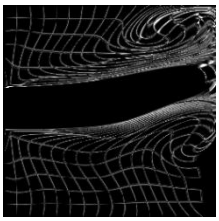
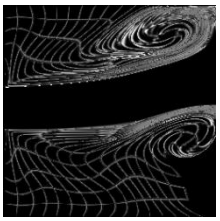
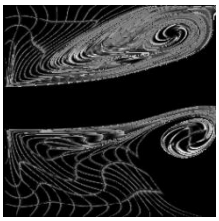
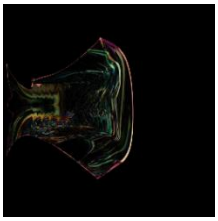
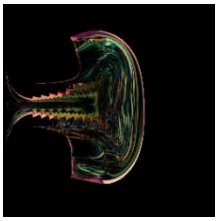
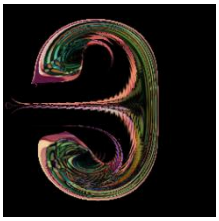
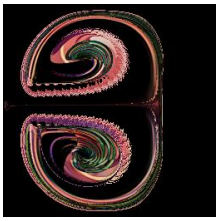
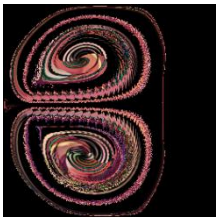
5	10	25	50	100
				
				

Tabelle 7-88: Differenzbilder des BFECC + Polynom 5 Verfahrens zu den Referenzbildern.

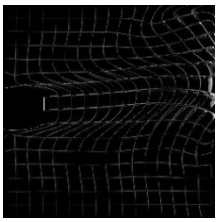
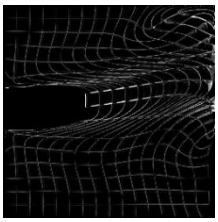
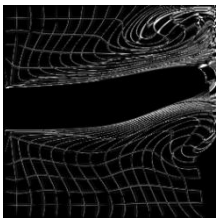
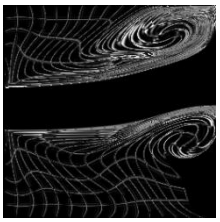
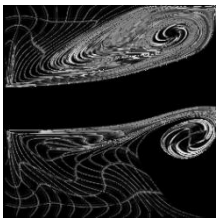
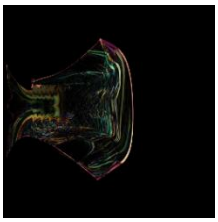
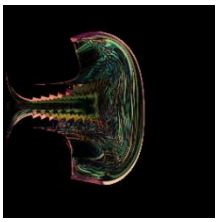
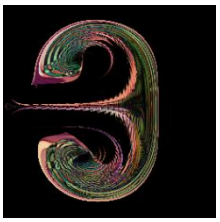
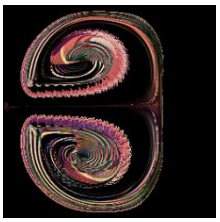
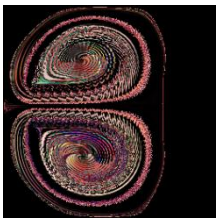
5	10	25	50	100
				
				

Tabelle 7-89: Differenzbilder des BFECC + RK 4 Verfahrens zu den Referenzbildern.

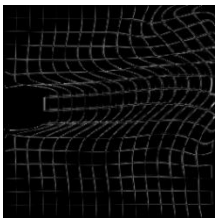
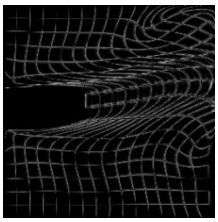
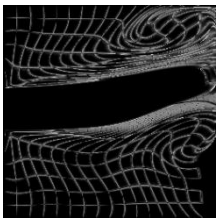
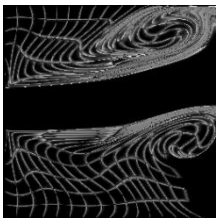
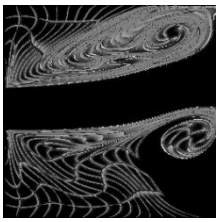
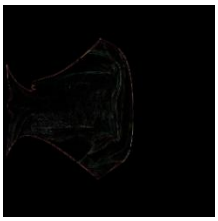
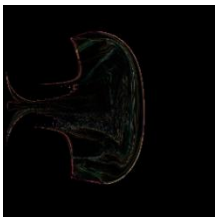
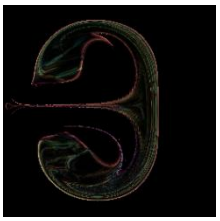
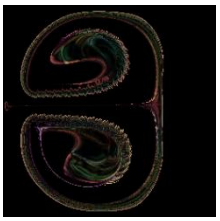
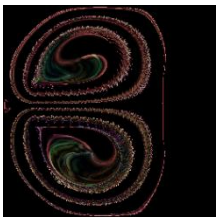
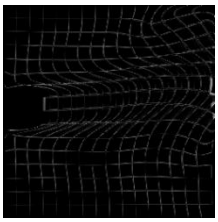
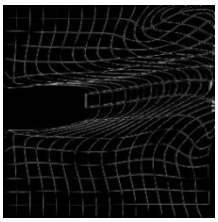
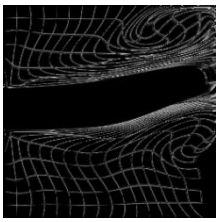
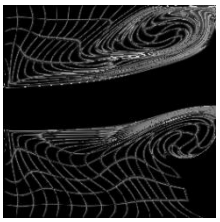
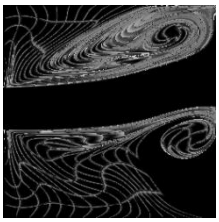
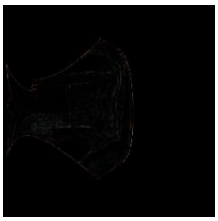
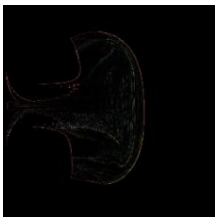
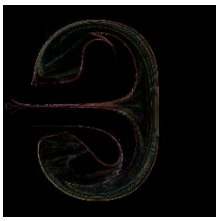
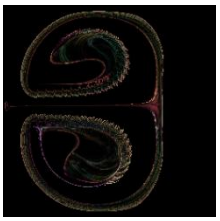
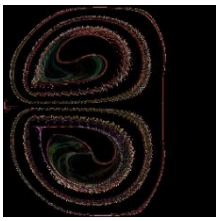
5	10	25	50	100
				
				

Tabelle 7-90: Differenzbilder des BFECC + Polynom 3 + RK 4 + Polynom 3 Verfahrens zu den Referenzbildern.

5	10	25	50	100
				
				

Die Differenzbilder sind vergleichbar mit denen aus dem Original BFECC Verfahren. Ohne das klassische Runge-Kutta Verfahren ist die Differenz aufgrund der falschen Positionsberechnung groß. Mit dem Runge-Kutta Verfahren ist die Differenz gering.

7.3.1.3.3 Reversibilität

Tabelle 7-91: Ergebnisse der Reversibilitätsberechnung des BFECC Verfahrens.

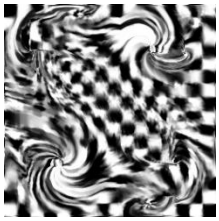
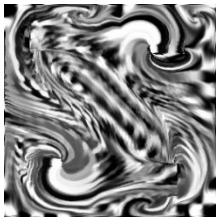
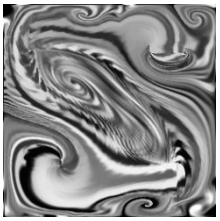
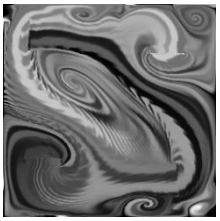
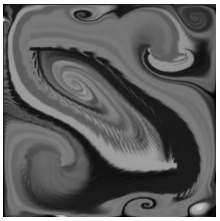
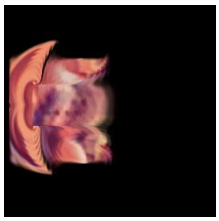
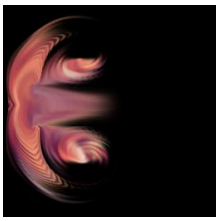
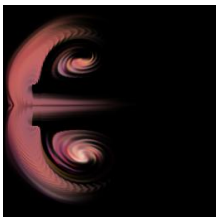
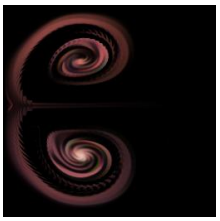
5	10	25	50	100
				
				

Tabelle 7-92: Ergebnisse der Reversibilitätsberechnung des BFECC + Polynom 3 Verfahrens.

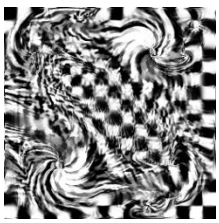
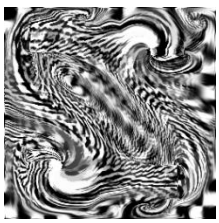
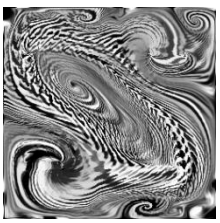
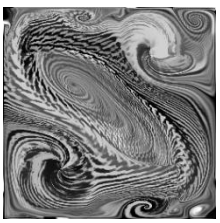
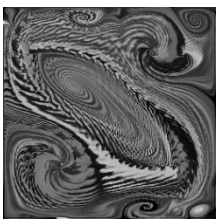
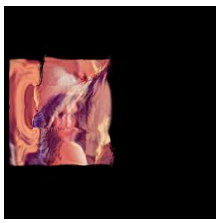
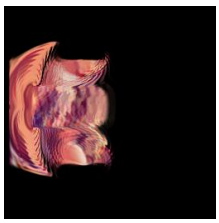
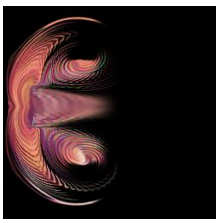
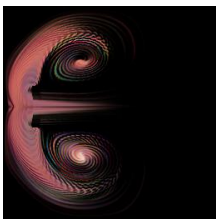
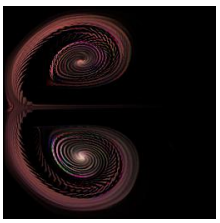
5	10	25	50	100
				
				

Tabelle 7-93: Ergebnisse der Reversibilitätsberechnung des BFECC + Polynom 5 Verfahrens.

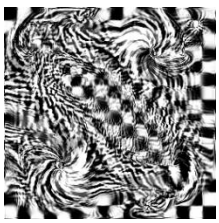
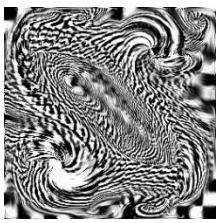
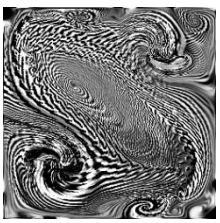
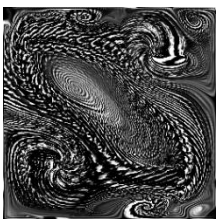
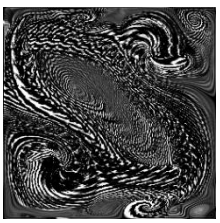
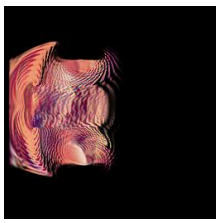
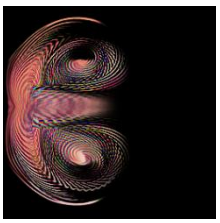
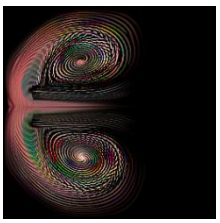
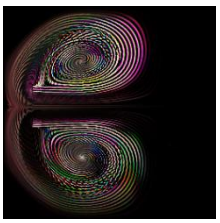
5	10	25	50	100
				
				

Tabelle 7-94: Ergebnisse der Reversibilitätsberechnung des BFECC + RK 4 Verfahrens.

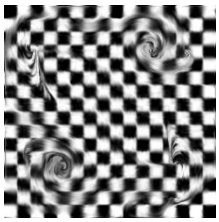
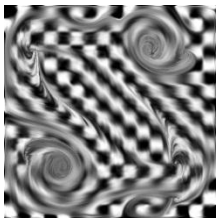
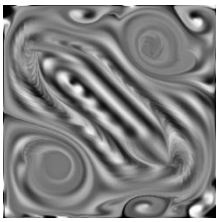
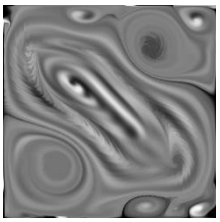
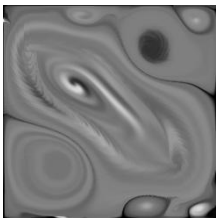
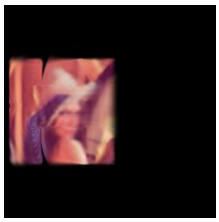
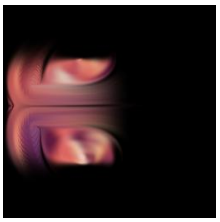
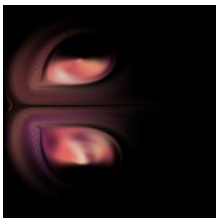
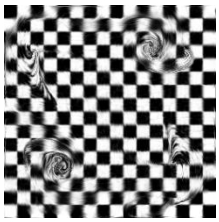
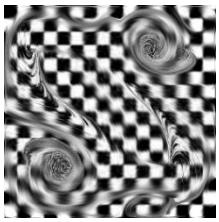
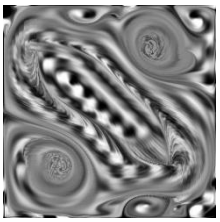
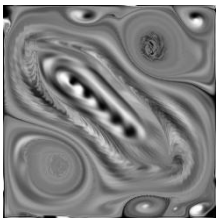
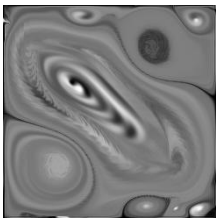
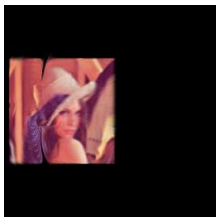
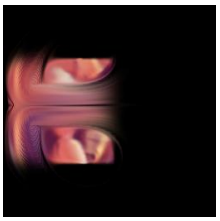
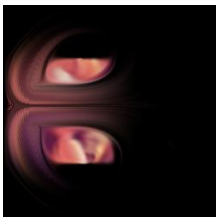
5	10	25	50	100
				
				

Tabelle 7-95: Ergebnisse der Reversibilitätsberechnung des BFECC + Polynom 3 + RK 4 + Polynom 3 Verfahrens.

5	10	25	50	100
				
				

Die Reversibilität der Modifikationen, die das klassische Runge-Kutta Verfahren nicht einsetzen (Tabelle 7-91, Tabelle 7-92 und Tabelle 7-93), ist sehr schlecht, da die Euler'sche Methode nicht dazu geeignet ist der Krümmung des Vektorfeldes ausreichend genau zu folgen.

Bei den Modifikation mit dem klassischen Runge-Kutta Verfahren (Tabelle 7-94 und Tabelle 7-95) ist die Reversibilität besser. Im direkten Vergleich der Reversibilität dieser beiden Modifikationen (Tabelle 7-94 und Tabelle 7-95) mit den entsprechenden Modifikationen im Original BFECC Verfahren (Tabelle 7-62 und Tabelle 7-63) ist allerdings feststellbar, dass diese hier eine größere Unschärfe aufweisen als die im Original BFECC Verfahren.

Der direkte Vergleich der beiden Modifikationen (Tabelle 7-94 und Tabelle 7-95) ergibt, dass die Modifikation, welche Polynome höheren Grades zur Interpolation der Daten einsetzt, ein besseres Ergebnis liefert.

7.3.1.3.4 Optischer Fluss

Tabelle 7-96: Optischer Fluss des BFECC Verfahrens.

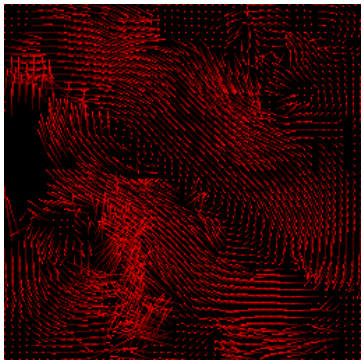
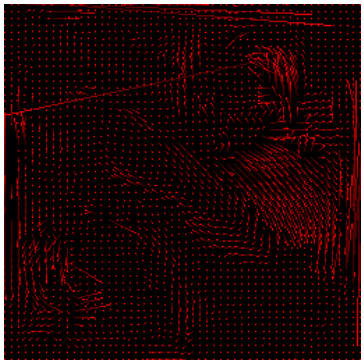
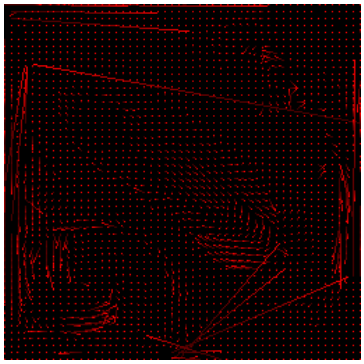
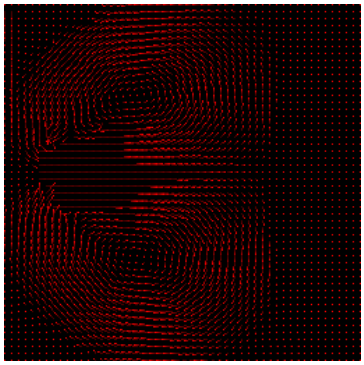
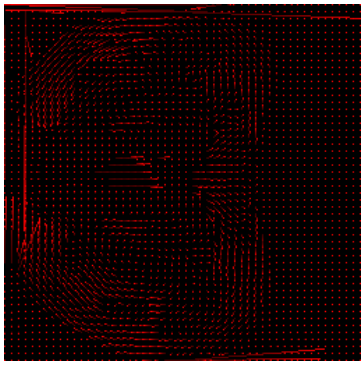
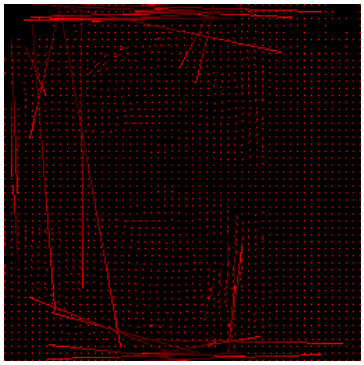
0-1	25-26	100-101
		
		

Tabelle 7-97: Optischer Fluss des BFECC + Polynom 3 Verfahrens.

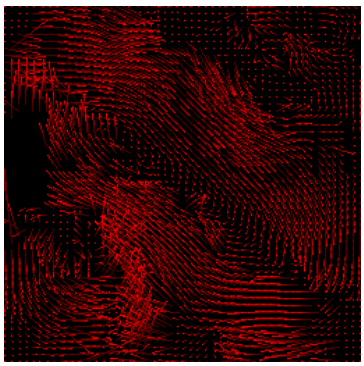
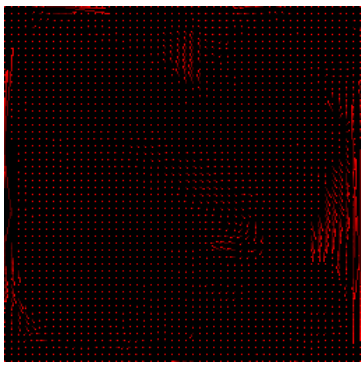
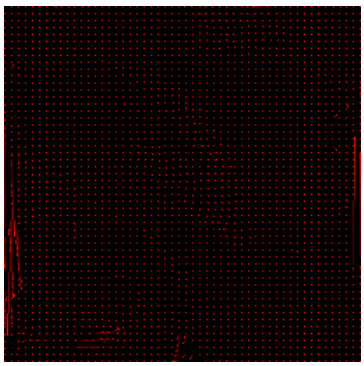
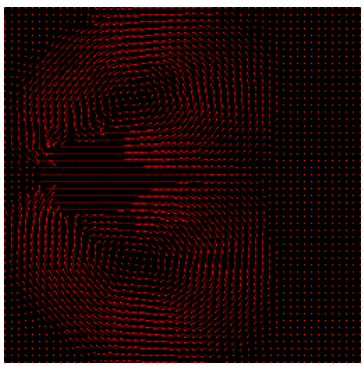
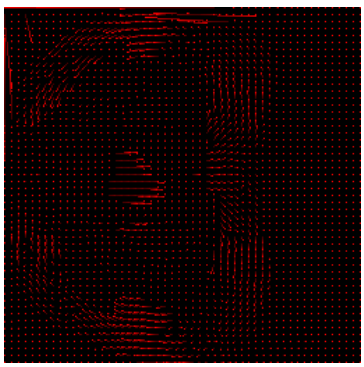
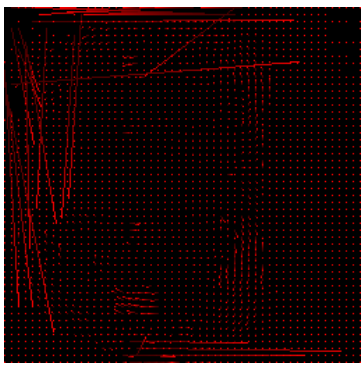
0-1	25-26	100-101
		
		

Tabelle 7-98: Optischer Fluss des BFECC + Polynom 5 Verfahrens.

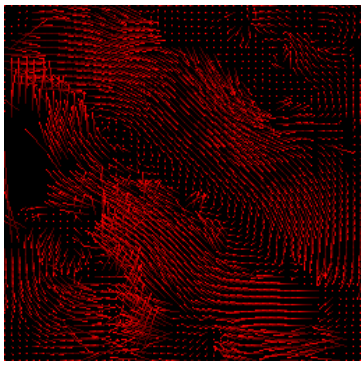
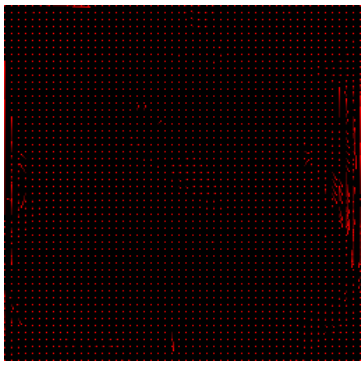
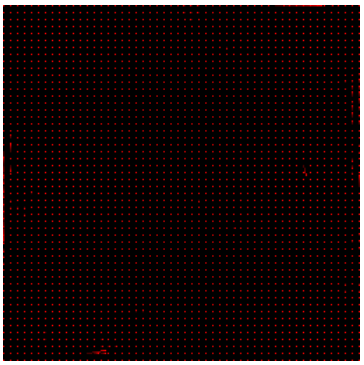
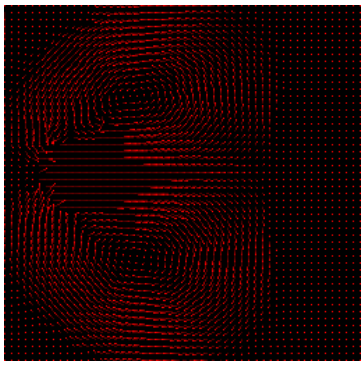
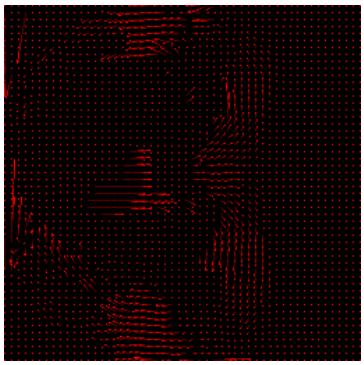
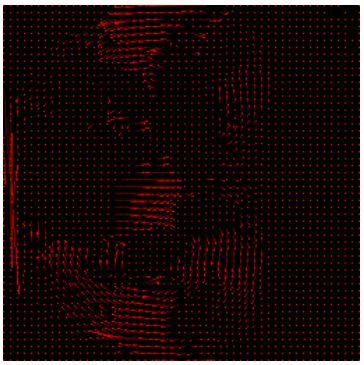
0-1	25-26	100-101
		
		

Tabelle 7-99: Optischer Fluss des BFECC + RK 4 Verfahrens.

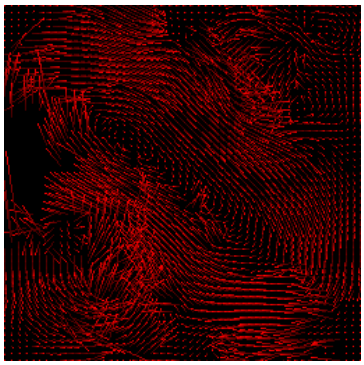
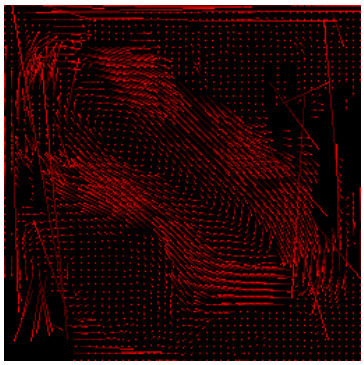
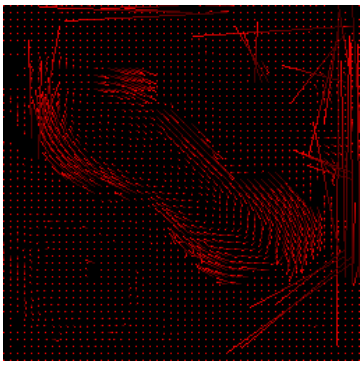
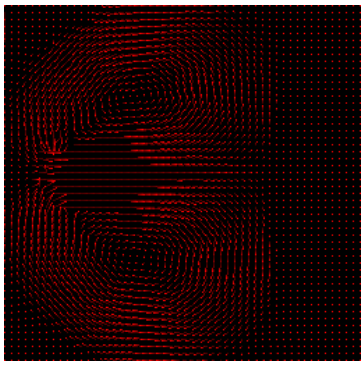
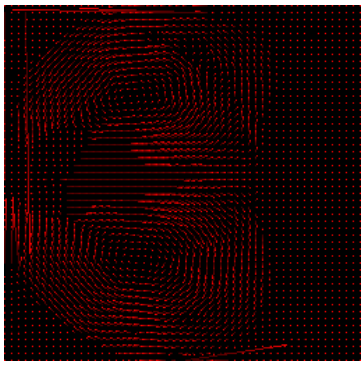
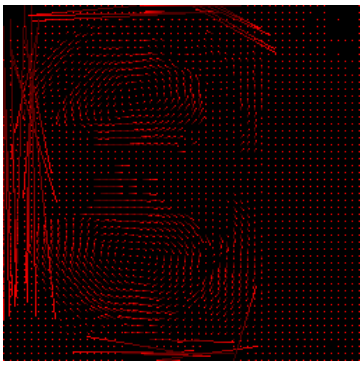
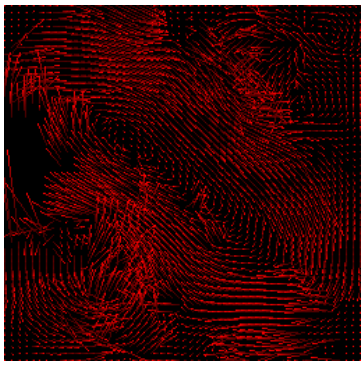
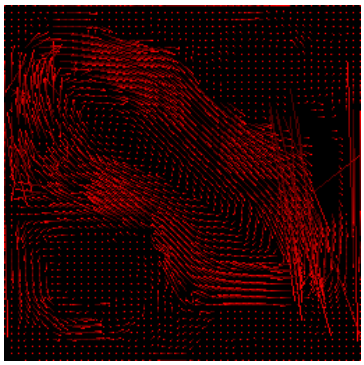
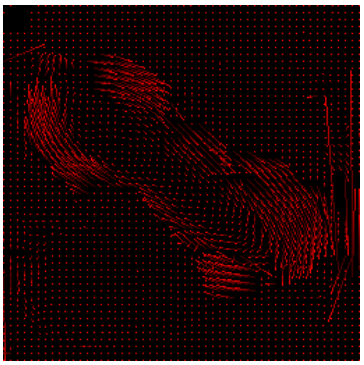
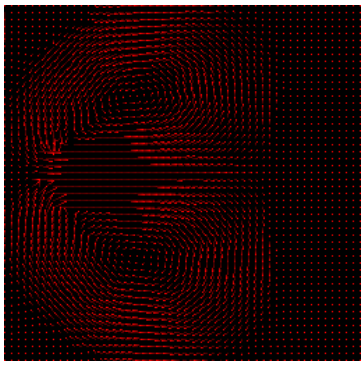
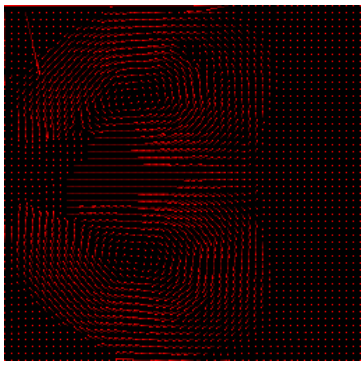
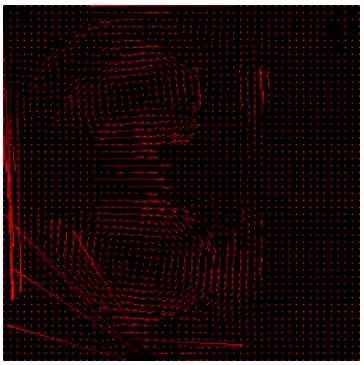
0-1	25-26	100-101
		
		

Tabelle 7-100: Optischer Fluss des BFECC + Polynom 3 + RK 4 + Polynom 3 Verfahrens.

0-1	25-26	100-101
		
		

Bei hohen Schrittzahlen ist der optische Fluss bei keiner der Modifikationen gut erhalten. Bei niedrigeren Schrittzahlen ist dieser allerdings überall noch ausreichend erkennbar. Modifikationen, die das klassische Runge-Kutta Verfahren einsetzen (Tabelle 7-94 und Tabelle 7-95), lassen den optische Fluss auch noch bei mittleren Schrittzahlen erkennen.

Im Vergleich zum Original BFECC Verfahren ist der optische Fluss bei allen Modifikationen nicht gut erhalten.

7.3.1.3.5 Spektrum

Tabelle 7-101: Spektrum des BFECC Verfahrens.

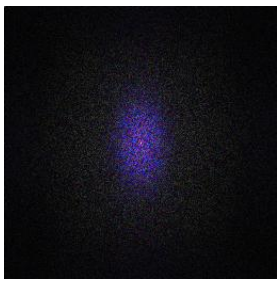
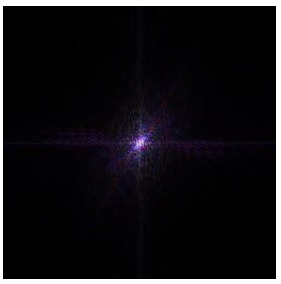
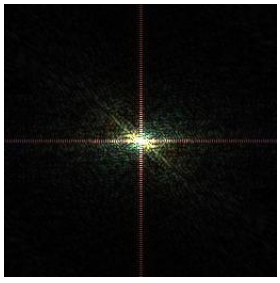
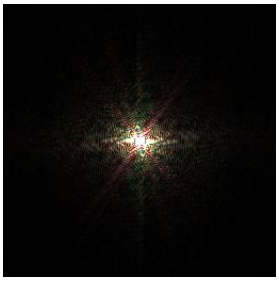
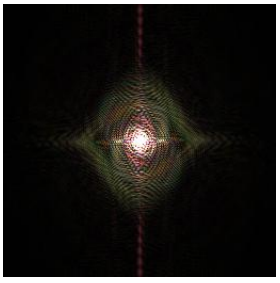
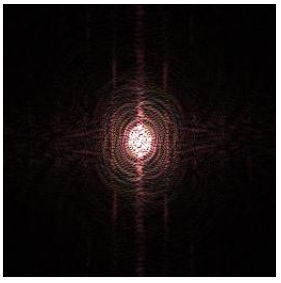
0	5	25	100
			
			

Tabelle 7-102: Spektrum des BFECC + Polynom 3 Verfahrens.

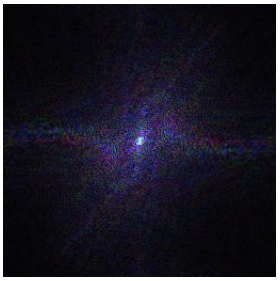
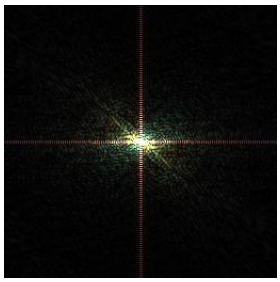
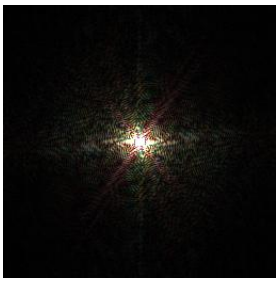
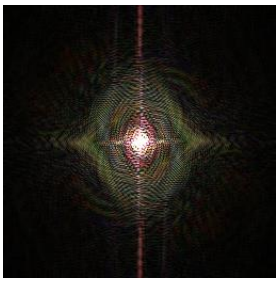
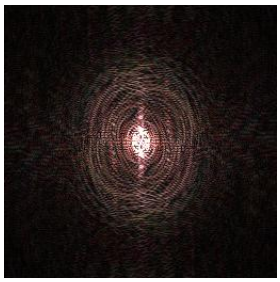
0	5	25	100
			
			

Tabelle 7-103: Spektrum des BFECC + Polynom 5 Verfahrens.

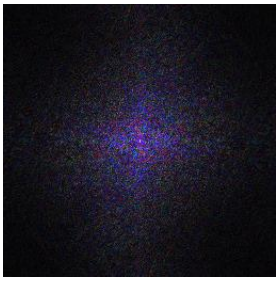
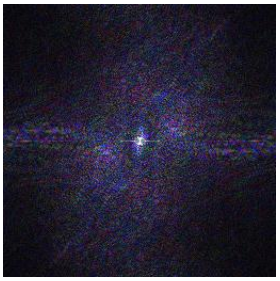
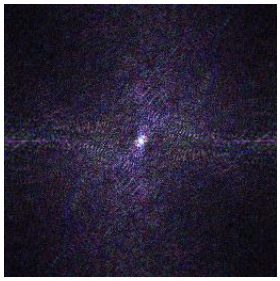
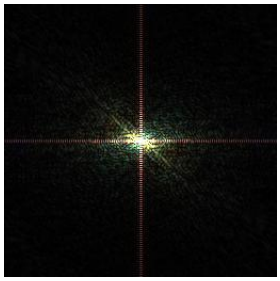
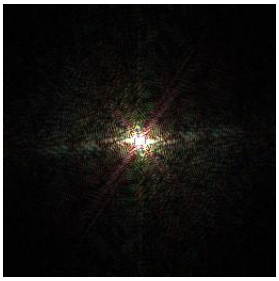
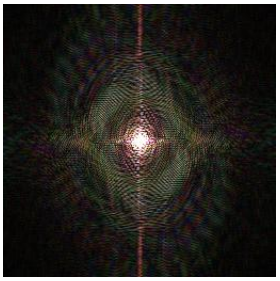
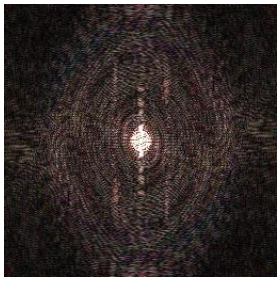
0	5	25	100
			
			

Tabelle 7-104: Spektrum des BFECC + RK 4 Verfahrens.

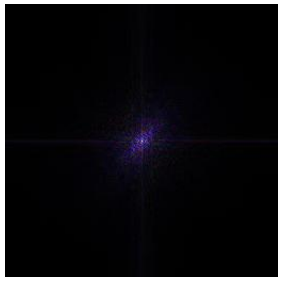
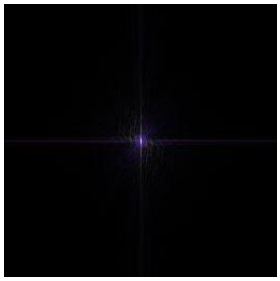
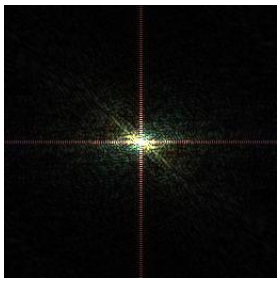
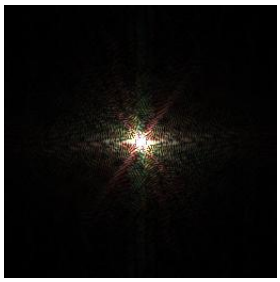
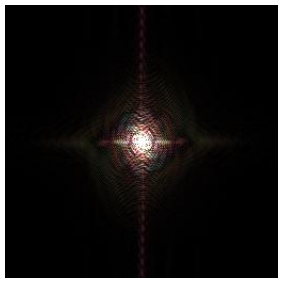
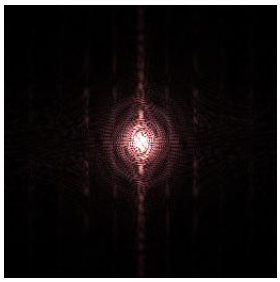
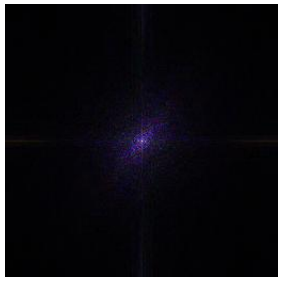
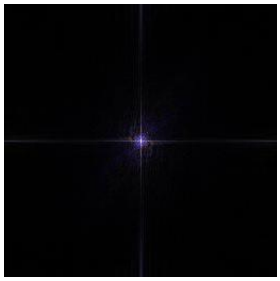
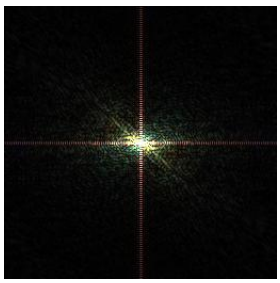
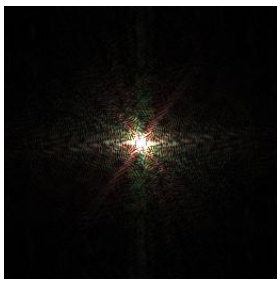
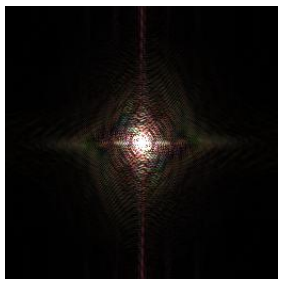
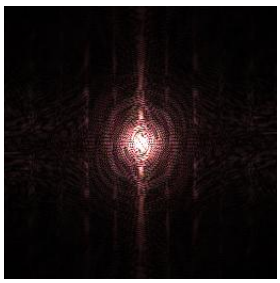
0	5	25	100
			
			

Tabelle 7-105: Spektrum des BFECC + Polynom 3 + RK 4 + Polynom 3 Verfahrens.

0	5	25	100
			
			

Das Spektrum dieses Verfahrens ist von der Qualität mit dem Spektrum des Original BFECC Verfahrens vergleichbar. Bei den Modifikationen ohne das klassische Runge-Kutta Verfahren dehnt sich das Spektrum mit der Schrittzahl aus. Bei den Modifikationen mit dem klassischen Runge-Kutta Verfahren wird es auf einen Punkt konzentriert.

Bei der BFECC + Polynom 5 Modifikation entsteht neben dem Punkt auch eine Wolke um diesen Punkt.

7.3.1.3.6 Performance

Tabelle 7-106: Performance des BFECC (Links), des BFECC + Polynom 3 (Mitte) und des BFECC + Polynom 5 (Rechts) Verfahrens.

Gesamtzeit:	7,18
Durchschnittszeit pro Schritt:	0,0718
Kürzeste Zeit eines Schrittes:	0,0622
Längste Zeit eines Schrittes:	0,0919
FPS:	13,93

Gesamtzeit:	32,2
Durchschnittszeit pro Schritt:	0,322
Kürzeste Zeit eines Schrittes:	0,2182
Längste Zeit eines Schrittes:	0,6822
FPS:	3,11

Gesamtzeit:	59,25
Durchschnittszeit pro Schritt:	0,5925
Kürzeste Zeit eines Schrittes:	0,5023
Längste Zeit eines Schrittes:	0,962
FPS:	1,69

Tabelle 7-107: Performance des BFECC + RK 4 (Links) und des BFECC + Polynom 3 + RK 4 + Polynom 3 (Rechts) Verfahrens.

Gesamtzeit:	11,7
Durchschnittszeit pro Schritt:	0,117
Kürzeste Zeit eines Schrittes:	0,1056
Längste Zeit eines Schrittes:	0,1565
FPS:	8,55

Gesamtzeit:	67,43
Durchschnittszeit pro Schritt:	0,6743
Kürzeste Zeit eines Schrittes:	0,5593
Längste Zeit eines Schrittes:	0,9715
FPS:	1,48

Die Performance dieses Verfahrens ist bei allen Modifikationen bei einer Berechnung auf der CPU nicht im echtzeitfähigen Bereich.

Interessant ist der direkte Vergleich des BFECC Verfahrens (Tabelle 7-106 Links) mit dem Original BFECC Verfahren (Tabelle 7-78 Links). Das Ergebnis zeigt, dass das Original schneller berechnet werden kann als diese Vereinfachung. Bei den Modifikationen (Tabelle 7-106 und Tabelle 7-107) sind diese im direkten Vergleich mit der entsprechenden Modifikation aus dem Original BFECC Verfahren (Tabelle 7-78, Tabelle 7-79 und Tabelle 7-80) immer ein wenig schneller.

7.3.1.3.7 Bewertung

Dieses Verfahren liefert ein gutes Ergebnis. Allerdings ist im direkten Vergleich zum Original BFECC Verfahren das Ergebnis geringfügig schlechter. Beide Verfahren benötigen ungefähr die gleiche Rechenzeit, wenn auch dieses Verfahren ein wenig schneller als das Original BFECC Verfahren ist.

Bei diesem Verfahren wird weder der optische Fluss noch das Spektrum erhalten.

Von den betrachteten Modifikationen erzeugt die Modifikation BFECC + Polynom 3 + Runge-Kutta 4 + Polynom 3 das beste Ergebnis. Diese Modifikation benötigt allerdings auch sehr viel Rechenzeit.

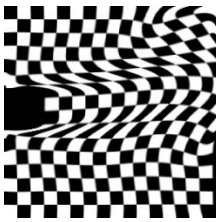
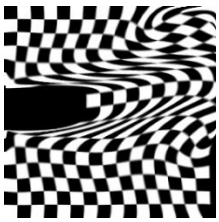
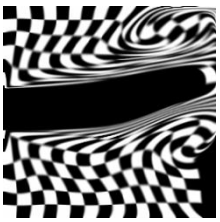
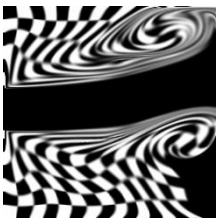
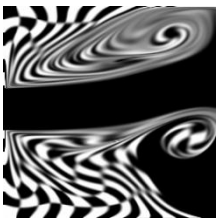
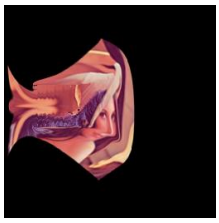
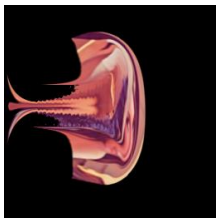
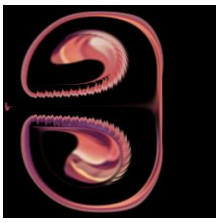
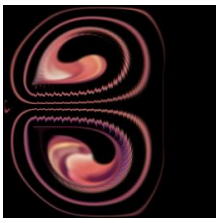
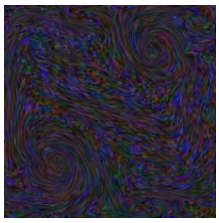
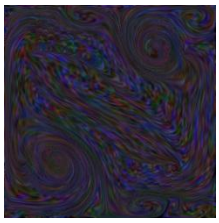
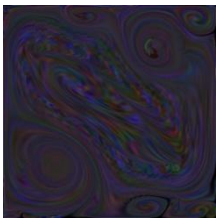
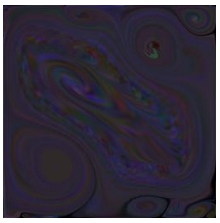
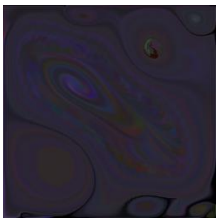
Die direkte Entscheidung zwischen dem Original BFECC und dem BFECC Verfahren geht zugunsten des Original BFECC Verfahrens aus. Das Ergebnis ist besser und die Rechenzeit nur minimal höher. Außerdem treten im Original BFECC Verfahren keine Artefakte auf, die durch das Verfahren bedingt sind.

7.3.1.4 Bilinear + RK 4 + Polynom 3

Dieses Verfahren ist in Kapitel 6.2.2.6 beschrieben. Hier wird die Komponente eingesetzt, die das Vektorfeld bilinear interpoliert. Die Position wird mittels des klassischen Runge-Kutta Verfahrens berechnet. Die Daten werden mithilfe der Polynom 3 Komponente interpoliert.

7.3.1.4.1 Ergebnisse

Tabelle 7-108: Ergebnisse des Bilinear + RK 4 + Polynom 3 Verfahrens.

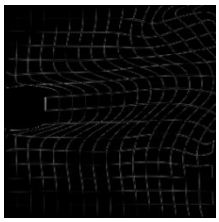
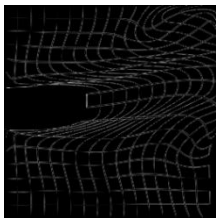
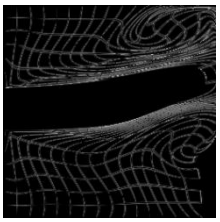
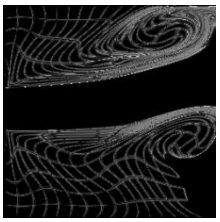
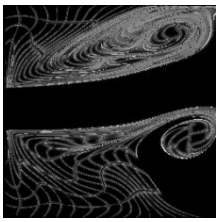
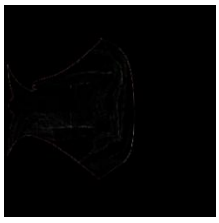
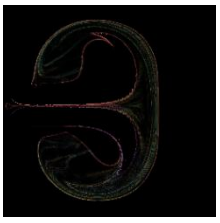
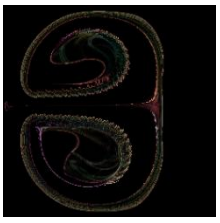
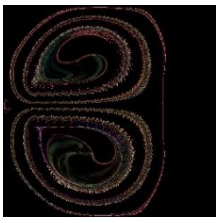
5	10	25	50	100
				
				
				

Da bei diesem Verfahren das klassische Runge-Kutta Verfahren eingesetzt wird, sind die Ergebnisse im Vergleich zum Semi-Lagrange Verfahren (Tabelle 7-8) viel präziser. Dies liegt auch daran, dass zum Interpolieren der Daten die Polynom 3 Komponente zum Einsatz kommt.

Allerdings sind bei hohen Schrittzahlen weiterhin unscharfe Stellen erkennbar.

7.3.1.4.2 Differenzbilder

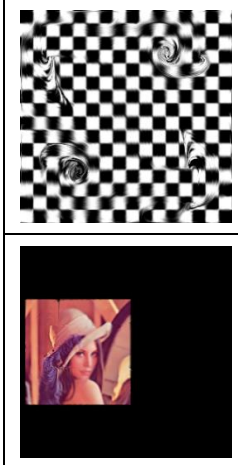
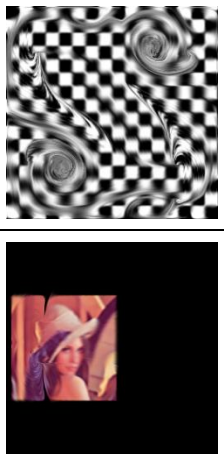
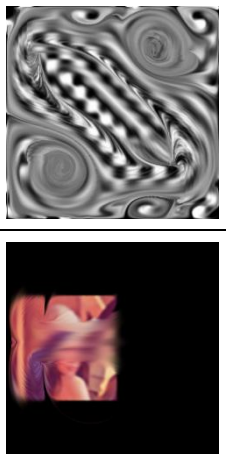
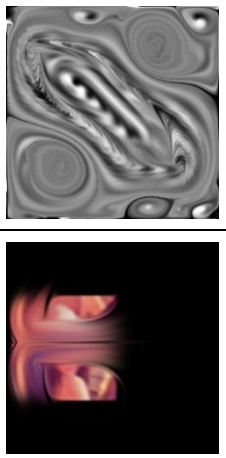
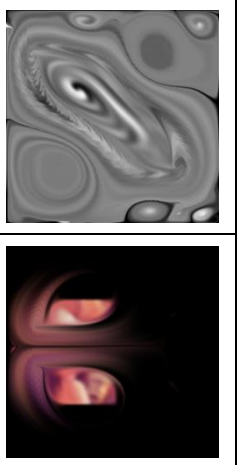
Tabelle 7-109: Differenzbilder des Bilinear + RK 4 + Polynom 3 Verfahrens zu den Referenzbildern.

5	10	25	50	100
				
				

Die Differenzbilder weisen nur geringfügige Unterschiede auf. Lediglich bei hohen Schrittzahlen wird die Differenz, aufgrund des aufsummierten Fehlers durch die iterativen Verfahren, größer.

7.3.1.4.3 Reversibilität

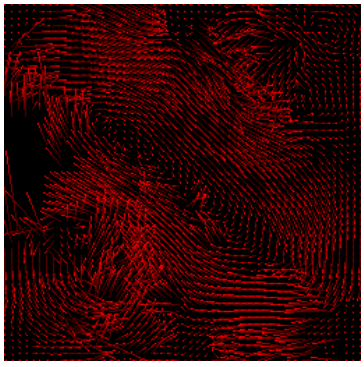
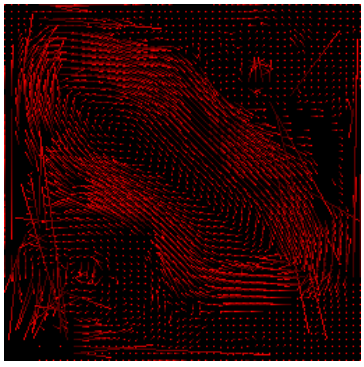
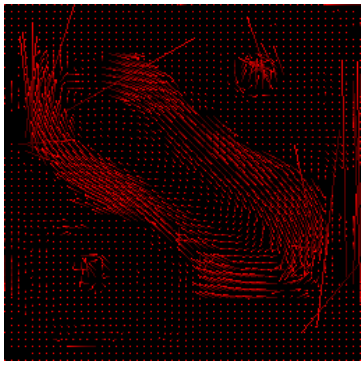
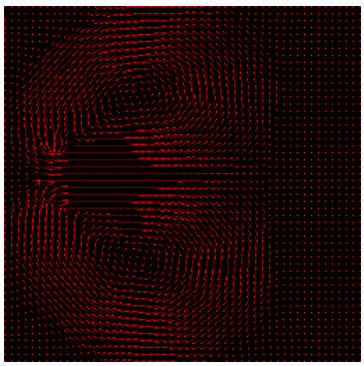
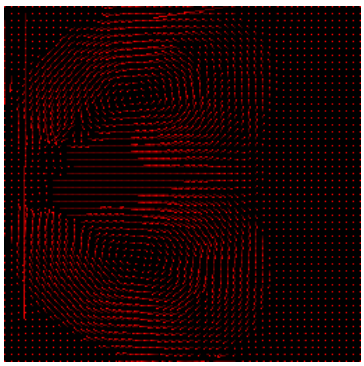
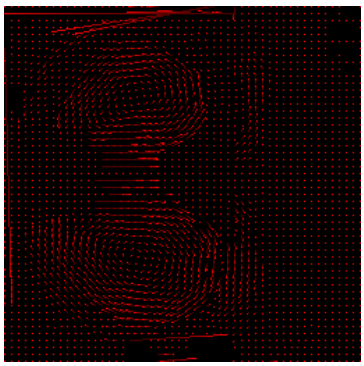
Tabelle 7-110: Ergebnisse der Reversibilitätsberechnung des Bilinear + RK 4 + Polynom 3 Verfahrens.

5	10	25	50	100
				

Dieses Verfahren besitzt durch die Verwendung des klassischen Runge-Kutta Verfahrens eine akzeptable Reversibilität. Im Vergleich zum BFECC Verfahren ist das Ergebnis allerdings schlechter, da es eine größere Unschärfe aufweist.

7.3.1.4.4 Optischer Fluss

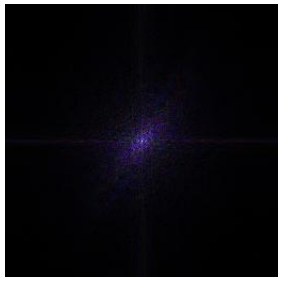
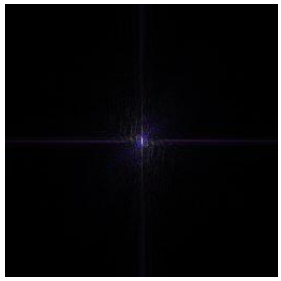
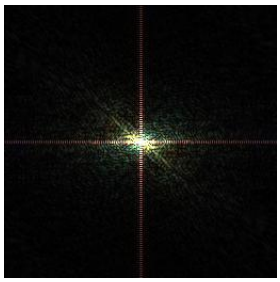
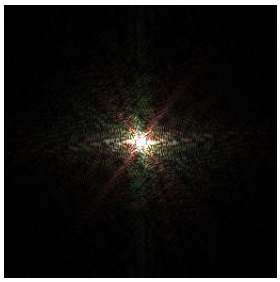
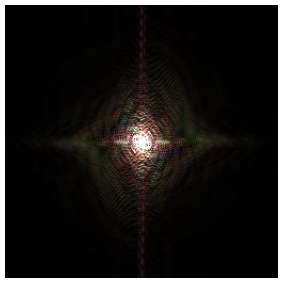
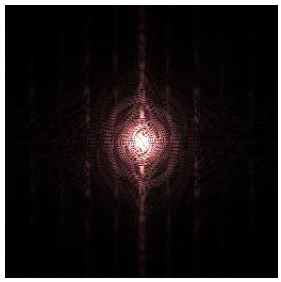
Tabelle 7-111: Optischer Fluss des Bilinear + RK 4 + Polynom 3 Verfahrens.

0-1	25-26	100-101
		
		

Bei diesem Verfahren ist der optische Fluss bei mittleren Schrittzahlen gut erkennbar. Auch bei hohen Schrittzahlen ist dieser noch zu sehen. Allerdings nimmt die Qualität des optischen Flusses ab.

7.3.1.4.5 Spektrum

Tabelle 7-112: Spektrum des Bilinear + RK 4 + Polynom 3 Verfahrens.

0	5	25	100
			
			

Das Spektrum wird bei diesem Verfahren wieder auf einen Punkt konzentriert. Diese Konzentration ist im ersten Vektorfeld stärker, im zweiten Vektorfeld allerdings nicht so stark wie bei dem Semi-Lagrange Verfahren (Tabelle 7-11).

Bei dem zweiten Vektorfeld findet auch hier eine Verschiebung des Spektrums ins Rötliche statt.

7.3.1.4.6 Performance

Tabelle 7-113: Performance des Bilinear + RK 4 + Polynom 3 Verfahrens.

Gesamtzeit:	14,8
Durchschnittszeit pro Schritt:	0,148
Kürzeste Zeit eines Schrittes:	0,1087
Längste Zeit eines Schrittes:	0,2016
FPS:	6,76

Die Performance des Verfahrens ist bei der CPU-Implementierung relativ niedrig. Dies liegt daran, dass das Daten- und Vektorfeld an $4 * 4 + 4 * 4 = 32$ Stellen ausgewertet werden muss.

7.3.1.4.7 Bewertung

Die Ergebnisse dieses Verfahrens sind mit denen des Original BFECC + RK 4 Verfahrens vergleichbar. Beide Verfahren haben an Stellen, an denen das Vektorfeld starke räumliche Änderungen aufweist, unscharfe Stellen. Beide Verfahren benötigen auch ungefähr die gleiche Rechenzeit um die Ergebnisse herzustellen.

Dieses Verfahren ist nicht in der Lage das Spektrum zu erhalten. Der optische Fluss wird allerdings relativ gut erhalten.

Dieses Verfahren stellt eine gute Alternative zum Original BFECC + RK 4 Verfahren dar. Es wird allerdings keine Fehlerkorrektur wie beim Original BFECC Verfahren verwendet. Dafür werden die Daten mit einem höheren Polynomgrad interpoliert.

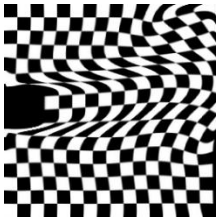
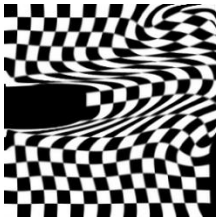
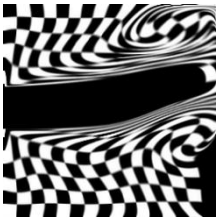
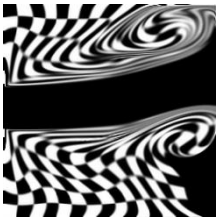
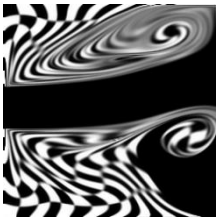
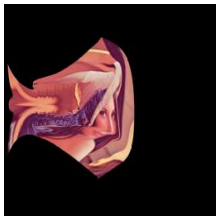
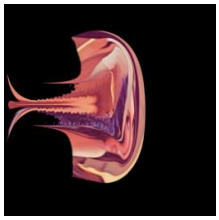
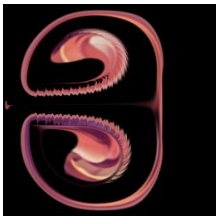
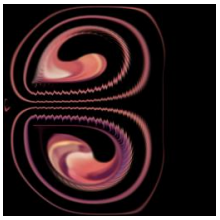
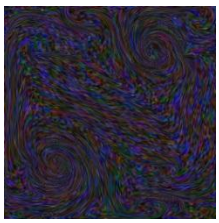
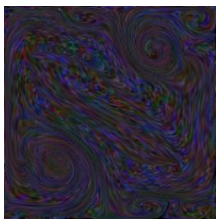
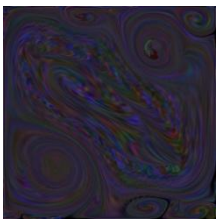
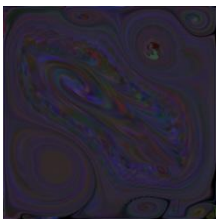
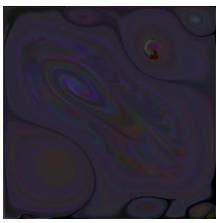
7.3.1.5 Bilinear + RK 4 + Polynom 5

In diesem Verfahren wird die Interpolation des Datenfeldes mittels der Polynom 5 Komponente durchgeführt. Die Interpolation der Geschwindigkeitsvektoren aus dem Vektorfeld erfolgt wieder mithilfe der bilinearen Komponente. Die Position wird mithilfe des klassischen Runge-Kutta Verfahrens berechnet.

Dieses Verfahren wird in Kapitel 6.2.2.7 beschrieben.

7.3.1.5.1 Ergebnisse

Tabelle 7-114: Ergebnisse des Bilinear + RK 4 + Polynom 5 Verfahrens.

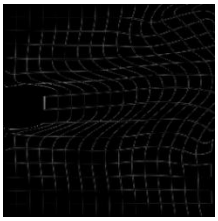
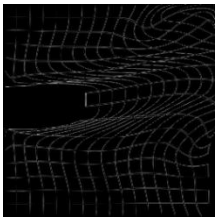
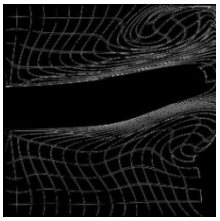
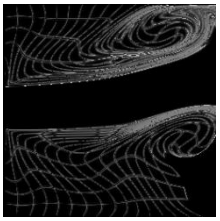
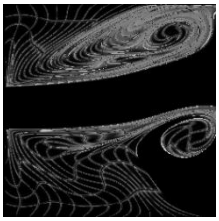
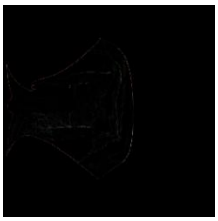
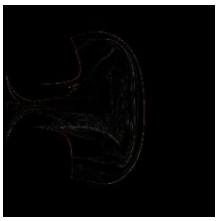
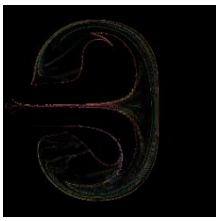
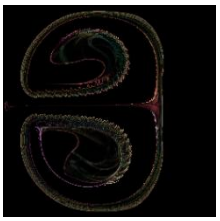
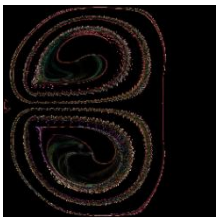
5	10	25	50	100
				
				
				

Im Vergleich zum Bilinear + RK 4 + Polynom 3 Verfahren (Tabelle 7-108) sind die Ergebnisse in diesem Verfahren deutlich schärfer. Besonders an Stellen, an denen das Vektorfeld eine große räumliche Änderung aufweist, ist dies klar zu erkennen.

Allerdings ist immer noch eine Unschärfe des Ergebnisses an Stellen mit starker räumlicher Änderung erkennbar. Dies ist besonders beim Ergebnis des dritten Vektorfeldes zu beobachten. Hier ist ein Verlust an Details bei den Bildern mit hoher Schrittzahl zu erkennen.

7.3.1.5.2 Differenzbilder

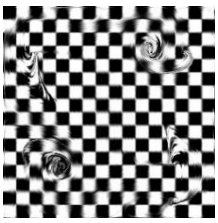
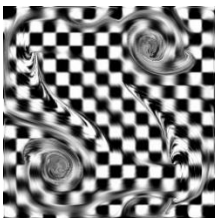
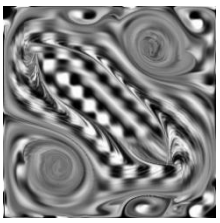
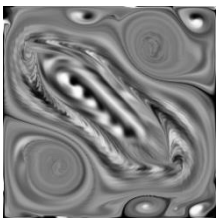
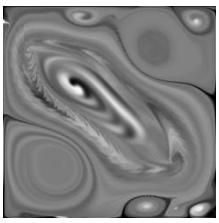
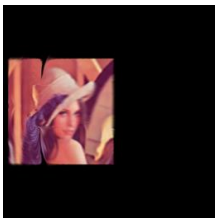
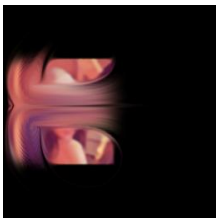
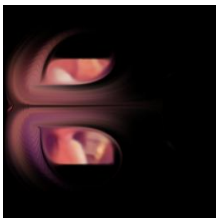
Tabelle 7-115: Differenzbilder des Bilinear + RK 4 + Polynom 5 Verfahrens zu den Referenzbildern.

5	10	25	50	100
				
				

Wie erwartet weisen die Differenzbilder wieder nur sehr geringe Unterschiede auf. Lediglich bei hohen Schrittzahlen wird die Differenz wieder, aufgrund des aufsummierten Fehlers durch das iterative Verfahren, größer.

7.3.1.5.3 Reversibilität

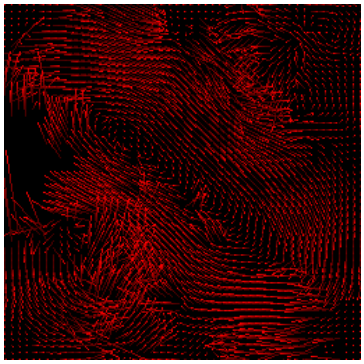
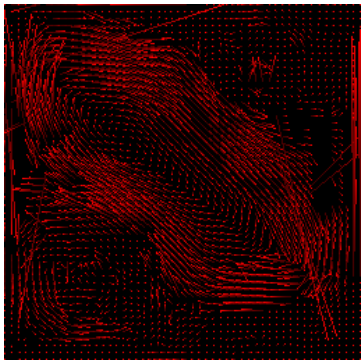
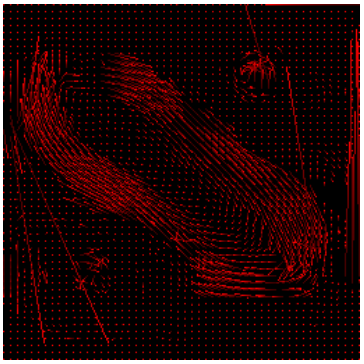
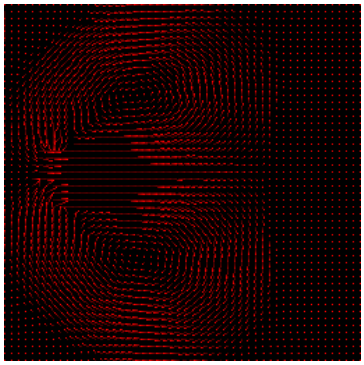
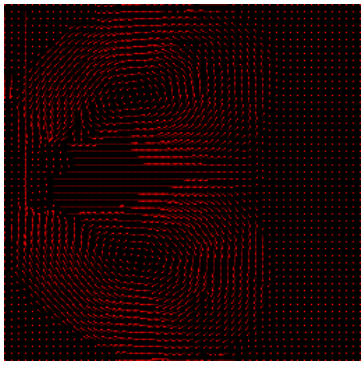
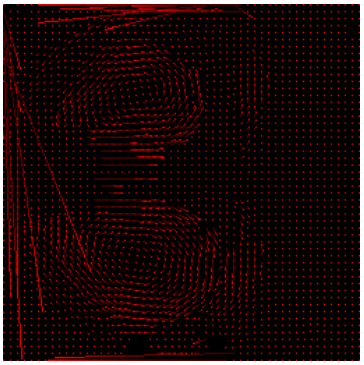
Tabelle 7-116: Ergebnisse der Reversibilitätsberechnung des Bilinear + RK 4 + Polynom 5 Verfahrens.

5	10	25	50	100
				
				

Die Reversibilität ist in diesem Verfahren besser als beim Bilinear + RK 4 + Polynom 3 Verfahren (Tabelle 7-110). Die errechneten Bilder weisen eine höhere Schärfe auf. Dies ist auf den höheren Polynomgrad der Interpolation des Datenfeldes zurückzuführen.

7.3.1.5.4 Optischer Fluss

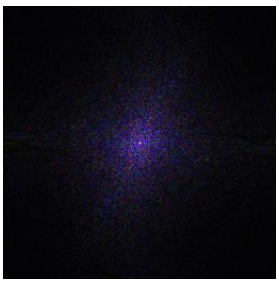
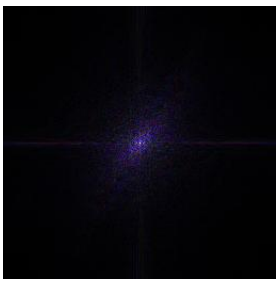
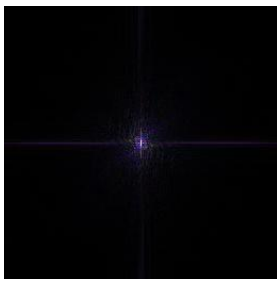
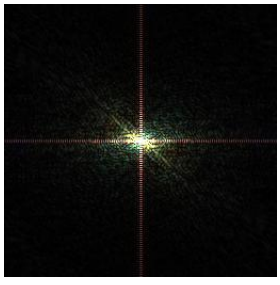
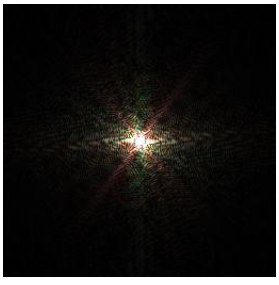
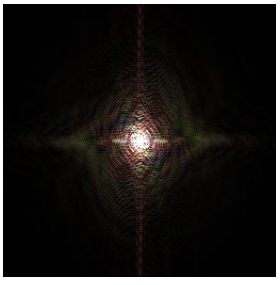
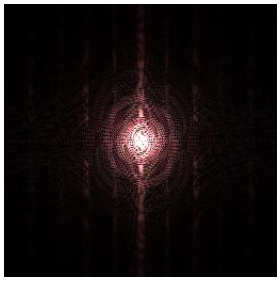
Tabelle 7-117: Optischer Fluss des Bilinear + RK 4 + Polynom 5 Verfahrens.

0-1	25-26	100-101
		
		

Wie beim Bilinear + RK 4 + Polynom 3 Verfahren (Tabelle 7-111) ist der optische Fluss auch bei hohen Schrittzahlen noch erkennbar. Allerdings nimmt auch hier die Qualität des berechneten Flusses ab.

7.3.1.5.5 Spektrum

Tabelle 7-118: Spektrum des Bilinear + RK 4 + Polynom 5 Verfahrens.

0	5	25	100
			
			

Wie beim Bilinear + RK 4 + Polynom 3 Verfahren (Tabelle 7-112) konzentriert sich das Spektrum des ersten Vektorfeldes stark auf einen Punkt. Diese Konzentration ist wie erwartet etwas stärker als bei dem Bilinear + RK 4 + Polynom 3 Verfahren. Das Spektrum des zweiten Vektorfeldes verschiebt sich wie gehabt ins Rötliche.

7.3.1.5.6 Performance

Tabelle 7-119: Performance des Bilinear + RK 4 + Polynom 5 Verfahrens.

Gesamtzeit:	25,03
Durchschnittszeit pro Schritt:	0,2503
Kürzeste Zeit eines Schrittes:	0,2373
Längste Zeit eines Schrittes:	0,2665
FPS:	3,9

Wie erwartet ist die Performance schlechter als beim Bilinear + RK 4 + Polynom 3 Verfahren (Tabelle 7-113). Hier müssen $6 * 6 + 4 * 4 = 52$ Stellen im Daten- und Vektorfeld ausgewertet werden.

7.3.1.5.7 Bewertung

Dieses Verfahren erstellt ein besseres Ergebnis als das Bilinear + RK 4 + Polynom 3 Verfahren. Allerdings sinkt die Performance in diesem Verfahren ein wenig, da mehr Stellen im Datenfeld ausgewertet werden müssen. Das Spektrum und der optische Fluss verhalten sich wie beim Bilinear + RK 4 + Polynom 3 Verfahren.

7.3.2 Forward-Verfahren

Forward-Verfahren sind die Verfahren, welche versuchen die Information eines Pixels vorwärts zu transportieren und anschließend diese Information auf die einzelnen Pixel aufzuteilen.

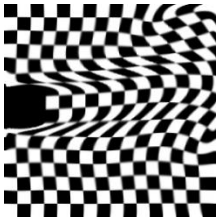
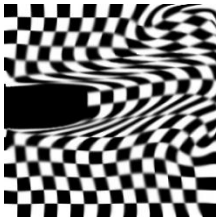
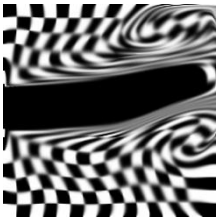
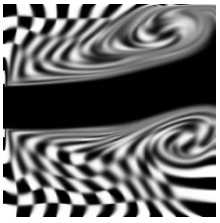
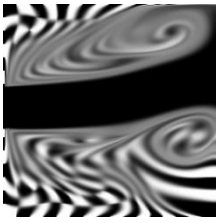
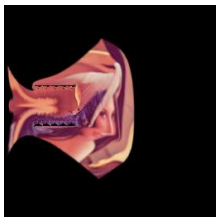
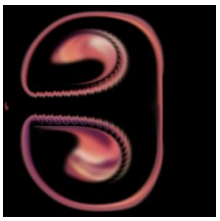
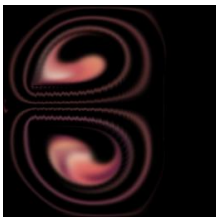
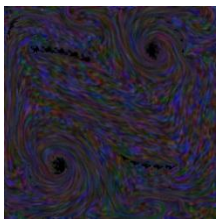
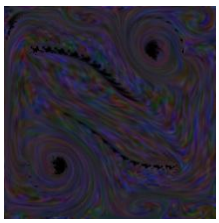
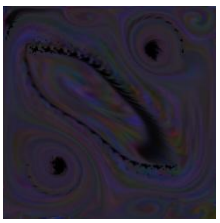
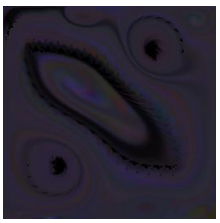
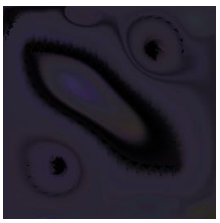
7.3.2.1 Forward RK 4

Das Forward RK 4 Verfahren integriert das Vektorfeld mithilfe des klassischen Runge-Kutta Verfahrens für jeden Pixel. Der Farbwert des Pixels wird entsprechend der neuen Position auf die einzelnen Pixel aufgeteilt.

Eine genaue Beschreibung dieses Verfahrens ist in Kapitel 6.2.3.1 zu finden.

7.3.2.1.1 Ergebnisse

Tabelle 7-120: Ergebnisse des Forward RK 4 Verfahrens.

5	10	25	50	100
				
				
				

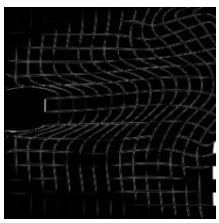
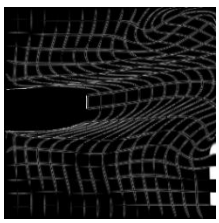
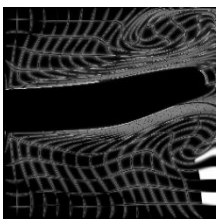
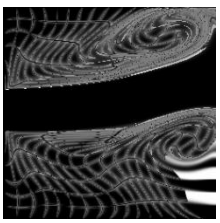
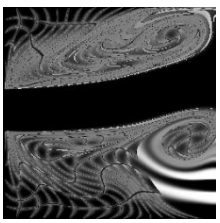
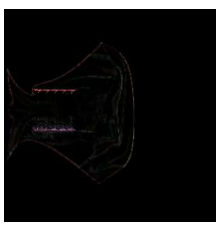
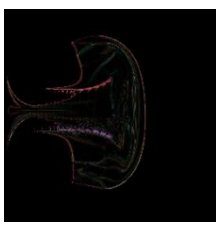
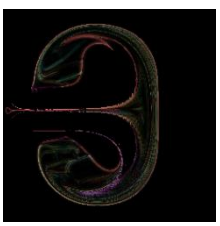
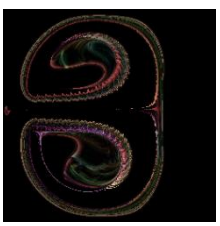
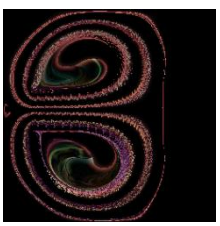
Das Forward RK 4 Verfahren funktioniert bei manchen Vektorfeldern recht gut, bei anderen allerdings nur unzureichend.

Nur beim ersten Vektorfeld erzeugt dieses Verfahren relativ gute Ergebnisse. Hierbei ist allerdings anzumerken, dass bei diesem Verfahren die Details relativ stark verwischt werden. Dieses Ergebnis ist mit dem Semi-Lagrange Verfahren (Tabelle 7-8) vergleichbar.

Bei den anderen beiden Vektorfeldern ist das Ergebnis nicht sehr gut. Besonders beim dritten Vektorfeld entstehen schon nach fünf Schritten Löcher. Diese sind hier schwarz, da der Hintergrund schwarz ist. Bei hohen Schrittzahlen sind bei diesen beiden Vektorfeldern keine Details mehr erkennbar und die Löcher haben sich ausgedehnt.

7.3.2.1.2 Differenzbilder

Tabelle 7-121: Differenzbilder des Forward RK 4 Verfahrens zu den Referenzbildern.

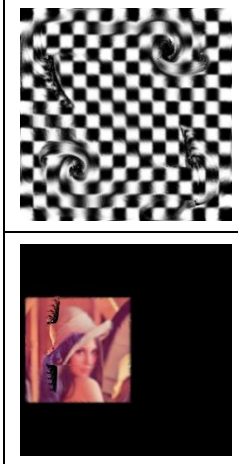
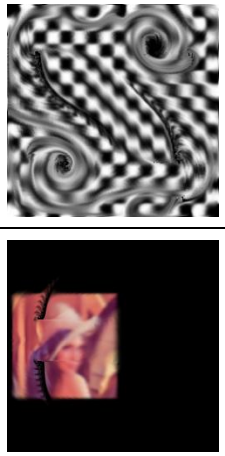
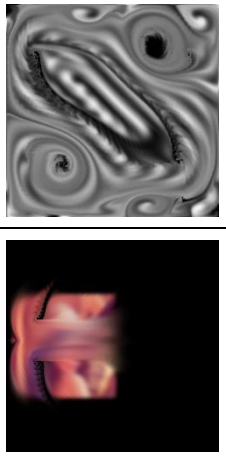
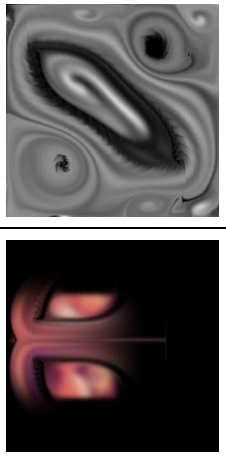
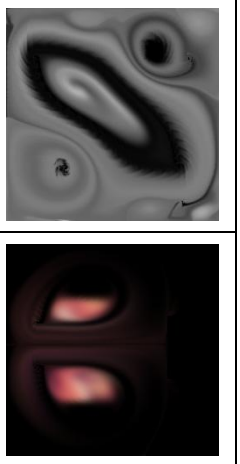
5	10	25	50	100
				
				

Die Differenzen sind bei niedrigen Schrittzahlen noch ausreichend gering. Sie treten allerdings mit steigender Schrittzahl deutlicher zum Vorschein.

Im ersten Vektorfeld ist bei hohen Schrittzahlen im rechten unteren Eck zu erkennen, dass das Schachbrettmuster hier fälschlicherweise bis zum Rand gezogen ist.

7.3.2.1.3 Reversibilität

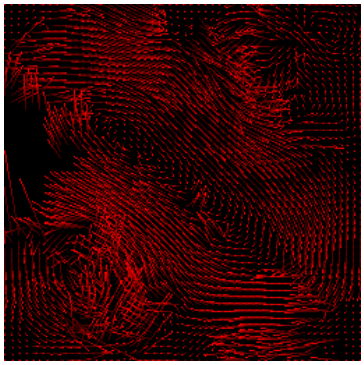
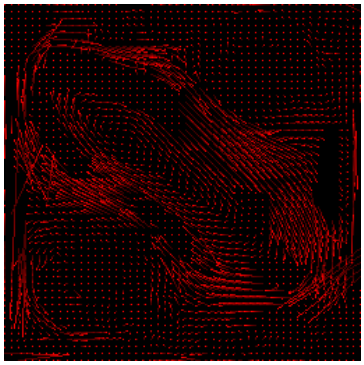
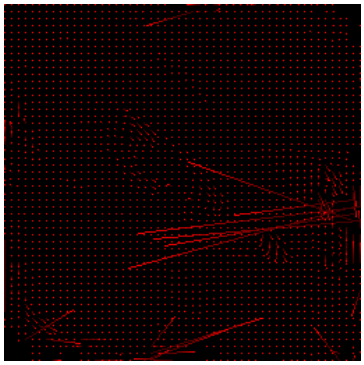
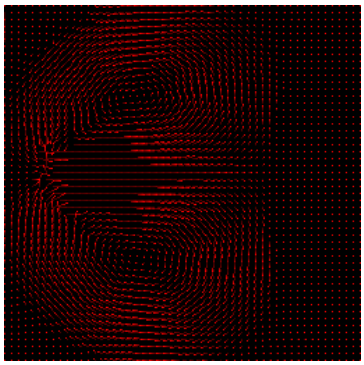
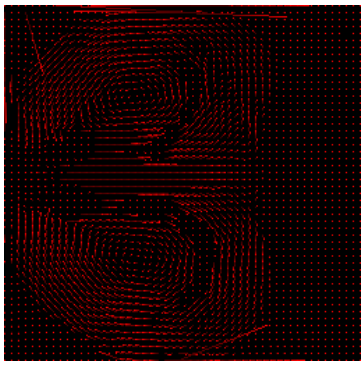
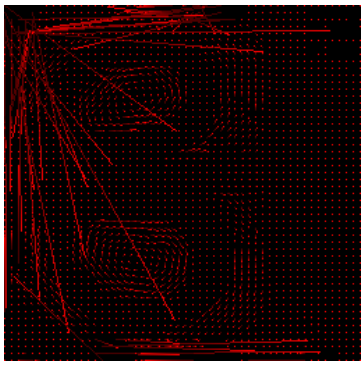
Tabelle 7-122: Ergebnisse der Reversibilitätsberechnung des Forward RK 4 Verfahrens.

5	10	25	50	100
				

Die Reversibilität ist in diesem Verfahren nicht sehr gut. Die Unschärfe in den einzelnen Bildern ist deutlich erkennbar. Bei hohen Schrittzahlen ist das Originalbild nicht mehr erkennbar.

7.3.2.1.4 Optischer Fluss

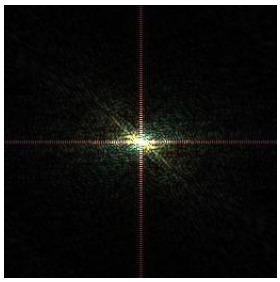
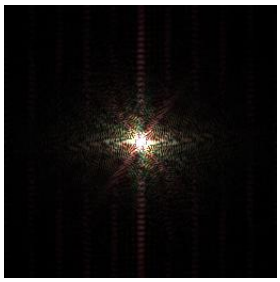
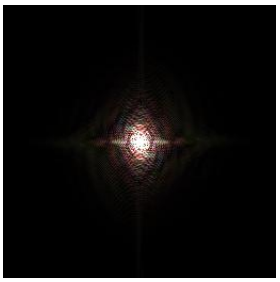
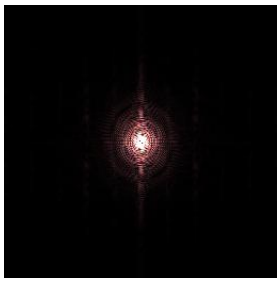
Tabelle 7-123: Optischer Fluss des Forward RK 4 Verfahrens.

0-1	25-26	100-101
		
		

Der optische Fluss ist nicht erhalten. Bei hohen Schrittzahlen ist der Fluss nur noch an wenigen Stellen erkennbar.

7.3.2.1.5 Spektrum

Tabelle 7-124: Spektrum des Forward RK 4 Verfahrens.

0	5	25	100
			
			

Das Spektrum ist in diesem Verfahren nicht erhalten und wird wieder auf einen Punkt konzentriert. Im zweiten Vektorfeld verschiebt sich das Spektrum wieder ins Rötliche.

7.3.2.1.6 Performance

Tabelle 7-125: Performance des Forward RK 4 Verfahrens.

Gesamtzeit:	4,04
Durchschnittszeit pro Schritt:	0,0404
Kürzeste Zeit eines Schrittes:	0,0385
Längste Zeit eines Schrittes:	0,0446
FPS:	24,75

Die Performance dieses Verfahrens ist gut. Es ist auf der CPU echtzeitfähig.

7.3.2.1.7 Bewertung

Die Ergebnisse dieses Verfahrens sind bei den Bildern, bei denen es funktioniert, mit dem Semi-Lagrange Verfahren vergleichbar. Der Verlust an Details in diesem Verfahren entsteht einerseits durch den iterativen Ansatz andererseits durch die geschätzte Aufteilung der Farbwerte auf die Nachbarpixel.

Allerdings gibt es auch Vektorfelder, bei denen dieses Verfahren versagt und ein sehr schlechtes Ergebnis erzeugt. Bei solchen Vektorfeldern entstehen Löcher im Ergebnisbild.

Dieses Verfahren stellt eine Alternative zu den einfachen Backward-Verfahren dar. Da es schnell berechnet werden kann sollte es eingesetzt werden wenn ein Forward-Verfahren benötigt wird und die anderen Verfahren eine zu schlechte Performance haben.

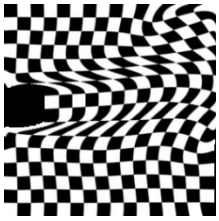
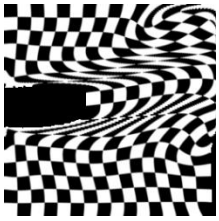
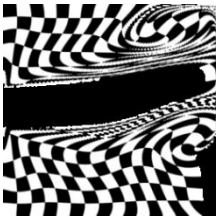
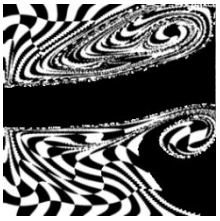
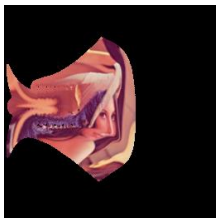
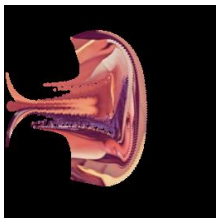
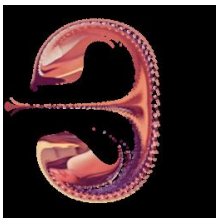
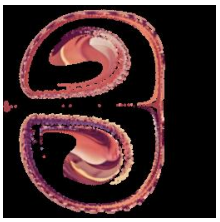
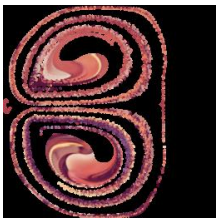
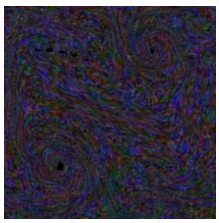
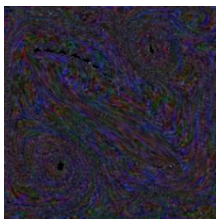
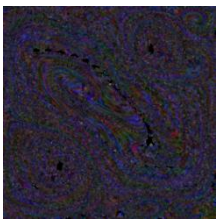
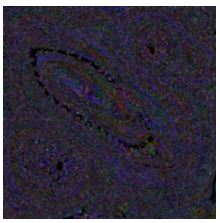
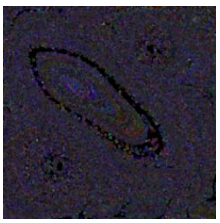
7.3.2.2 Forward RK 4 Particle Kernel

Das Forward RK 4 Particle Kernel Verfahren betrachtet die einzelnen Pixel der Quelldaten als Partikel. Um aus dieser Partikelwolke, die jeden Schritt advektiert wird, ein Bild zu erzeugen wird ein Kernel eingesetzt.

Dieses Verfahren ist in Kapitel 6.2.3.2 beschrieben.

7.3.2.2.1 Ergebnisse

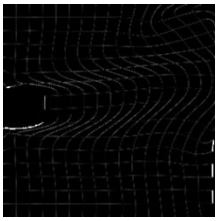
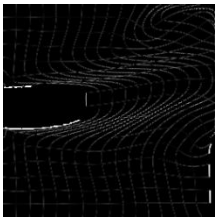
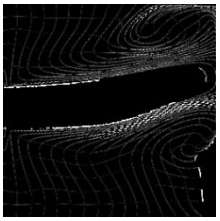
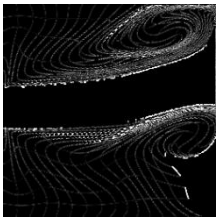
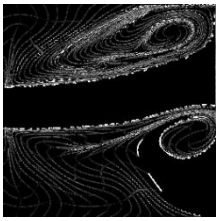
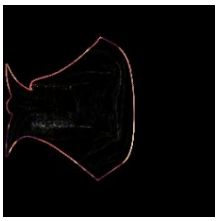
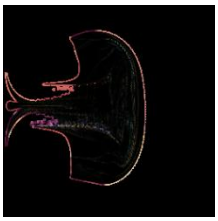
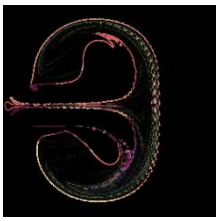
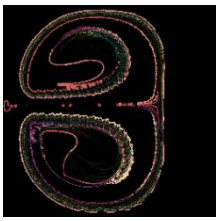
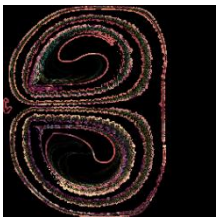
Tabelle 7-126: Ergebnisse des Forward RK 4 Particle Kernel Verfahrens.

5	10	25	50	100
				
				
				

Bei diesem Verfahren bleibt der Detailierungsgrad erhalten. Allerdings entsteht eine Verpixelung des Ergebnisses. Der Kernel wirkt der Verpixelung zwar entgegen indem er den Farbwert eines Partikels nicht nur auf die direkten Nachbarpixel verteilt, kann aber den Effekt nicht komplett unterbinden, da die Abstände zwischen zwei Partikeln unterschiedlich groß sind. Ein zu großer Kernel würde den Detailgrad des Ergebnisses reduzieren. Ideal wäre ein adaptiver Kernel, der seinen Radius aufgrund der Partikeldichte anpasst.

7.3.2.2.2 Differenzbilder

Tabelle 7-127: Differenzbilder des Forward RK 4 Particle Kernel Verfahrens zu den Referenzbildern.

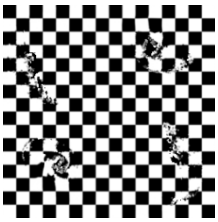
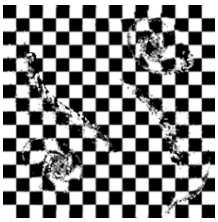
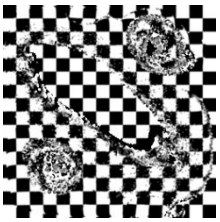
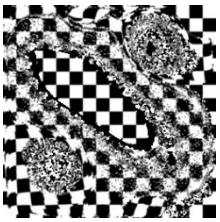
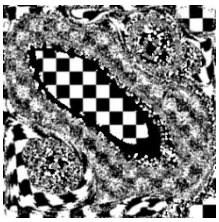
5	10	25	50	100
				
				

Die Differenz ist im ersten Vektorfeld an den Rändern der einzelnen Zellen sichtbar. Dies ist auf den verwendeten Kernel zurückzuführen. Im Vergleich zu den anderen Verfahren ist die Differenz allerdings in einem akzeptablen Maß.

Beim zweiten Vektorfeld ist die Differenz vor allem bei hohen Schrittzahlen ziemlich groß.

7.3.2.2.3 Reversibilität

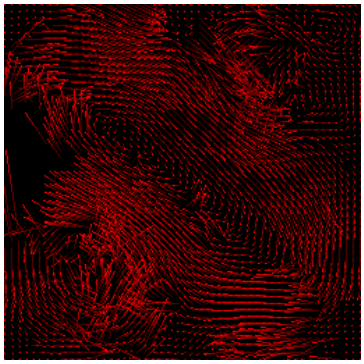
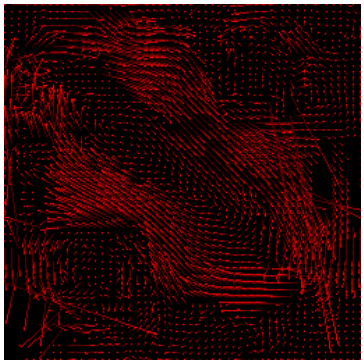
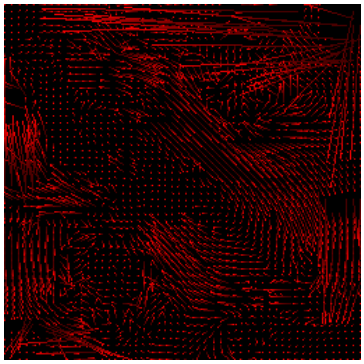
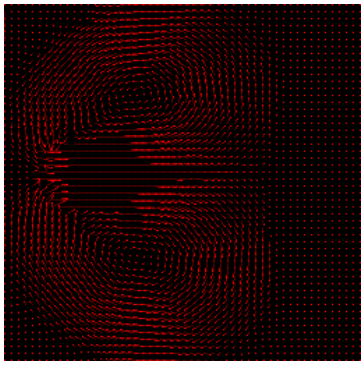
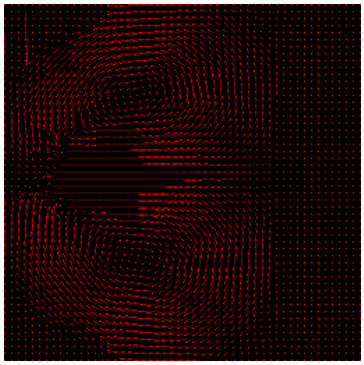
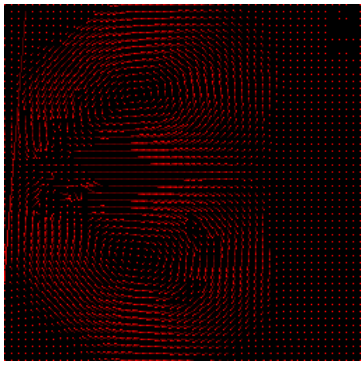
Tabelle 7-128: Ergebnisse der Reversibilitätsberechnung des Forward RK 4 Particle Kernel Verfahrens.

5	10	25	50	100
				
				

Die Ergebnisse sind aufgrund der nicht veränderten Farbinformationen sehr gut. Lediglich die Position der Partikel kann an manchen Stellen nicht genau genug berechnet werden. Dies ist bei höheren Schrittzahlen gut zu beobachten.

7.3.2.2.4 Optischer Fluss

Tabelle 7-129: Optischer Fluss des Forward RK 4 Particle Kernel Verfahrens.

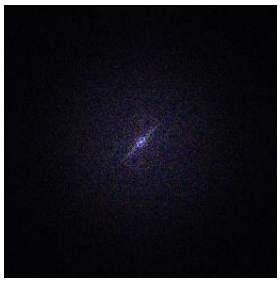
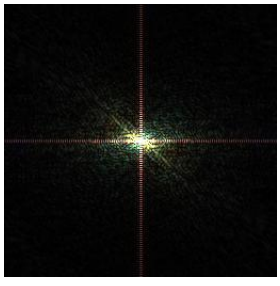
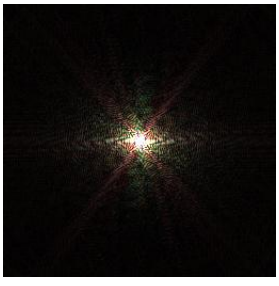
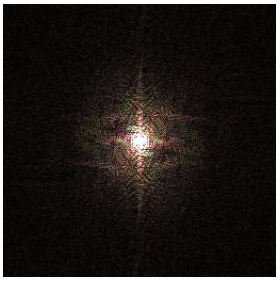
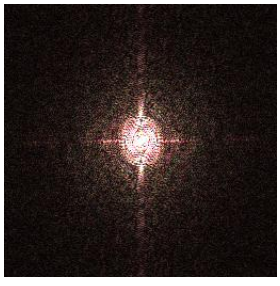
0-1	25-26	100-101
		
		

Der optische Fluss ist auch bei hohen Schrittzahlen noch gut erkennbar. Im ersten Vektorfeld entstehen zwar einige nicht berechenbare Stellen, aber im zweiten Vektorfeld ist der Fluss in allen Schritten gut berechenbar.

Der optische Fluss bleibt bei diesem Verfahren gut erhalten.

7.3.2.2.5 Spektrum

Tabelle 7-130: Spektrum des Forward RK 4 Particle Kernel Verfahrens.

0	5	25	100
			
			

Im ersten Vektorfeld wird das Spektrum langsam auf einen Punkt konzentriert.

Im zweiten Vektorfeld verändert sich das Spektrum hingegen stark.

7.3.2.2.6 Performance

Tabelle 7-131: Performance des Forward RK 4 Particle Kernel Verfahrens.

Gesamtzeit:	9,67
Durchschnittszeit pro Schritt:	0,0967
Kürzeste Zeit eines Schrittes:	0,0895
Längste Zeit eines Schrittes:	0,1124
FPS:	10,29

Die Performance ist in diesem Verfahren zwar nicht mehr echtzeitfähig, aber noch vergleichsweise gut. Die meiste Rechenzeit wird beim Aufteilen der Farbwerte mit dem Kernel benötigt, da der Kernel gegen viele Pixel getestet werden muss.

7.3.2.2.7 Bewertung

Die Ergebnisse dieses Verfahrens sind relativ gut. Allerdings kann es vorkommen, dass einzelne Pixel keinen Farbwert zugewiesen bekommen und somit Löcher entstehen. Dies ist durch die an manchen Stellen zu geringe Partikeldichte, wohingegen andere Stellen eine zu hohe Dichte aufweisen.

Dieses Verfahren besitzt jedoch den Vorteil, dass der Radius des Kernels eingestellt werden kann. Dabei muss ein gutes Mittelmaß zwischen hohem Detailreichtum und dem Schließen von Löchern gefunden werden. Ideal wäre hier ein adaptiver Kernel der den Radius aufgrund der Partikeldichte einstellt.

Im Vergleich zum Forward RK 4 Verfahren ist dieses Verfahren eine deutliche Verbesserung aufgrund der besseren Qualität der Ergebnisse, da kein iterativer Ansatz verwendet wird. Allerdings wird mehr Rechenzeit benötigt.

7.3.2.3 Forward Yu

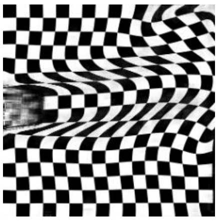
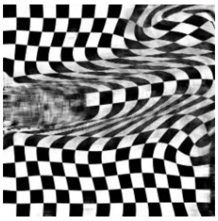
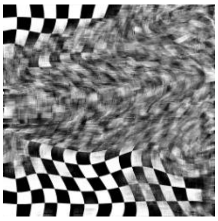
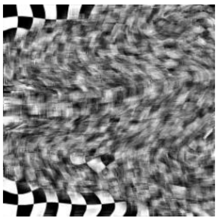
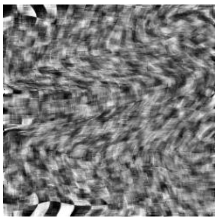
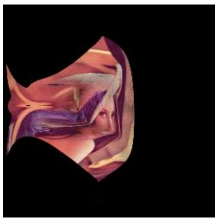
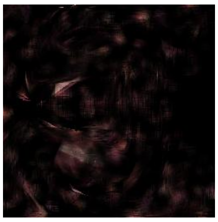
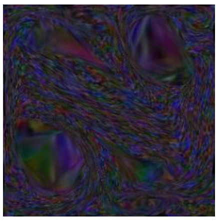
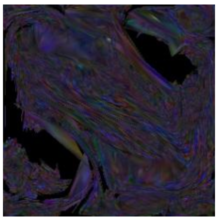
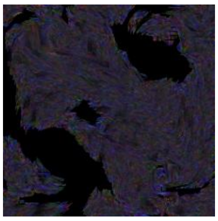
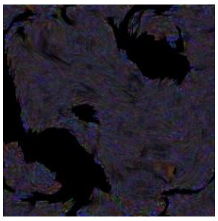
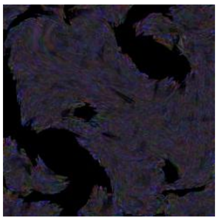
Das Forward Yu Verfahren verwendet einen komplett eigenständigen Ansatz um die Advektion durchzuführen. Dieser ist in Kapitel 6.2.3.3 beschrieben.

Dieses Verfahren erzeugt ein neues Bild, welches auf dem Muster der Originaldaten basiert. Die Ergebnisse sind also nicht mit denen der anderen Verfahren vergleichbar.

Für das Rendering der Gitterzellen wird hier ein einfacher Softwarerenderer verwendet.

7.3.2.3.1 Ergebnisse

Tabelle 7-132: Ergebnisse des Forward Yu Verfahrens.

5	10	25	50	100
				
				
				
				

Bei den ersten drei Vektorfeldern ist zu erkennen, dass dieses Verfahren ein komplett eigenständiges Bild erzeugt. Da bei diesen Vergleichsbildern der Wrap der Quelldaten ausgeschaltet ist, entstehen auch schwarze Bereiche. Dies passiert immer dann, wenn das Gitter eines Punktes außerhalb der Bildgrenzen liegt.

Das vierte Vektorfeld ist das Feld bifurcation.dat, welches in Abbildung 4-1 links unten zu sehen ist. Als Quelldaten wurde das Bild in Abbildung 7-13 verwendet.

Abbildung 7-13: Quelldaten für das vierte Vektorfeld.

Wie in diesem Vektorfeld zu sehen ist, ist das Ergebnis auch bei hohen Schrittzahlen noch relativ gut erhalten. Dies liegt daran, dass dieses Verfahren speziell hierfür entwickelt wurde.

7.3.2.3.2 Differenzbilder

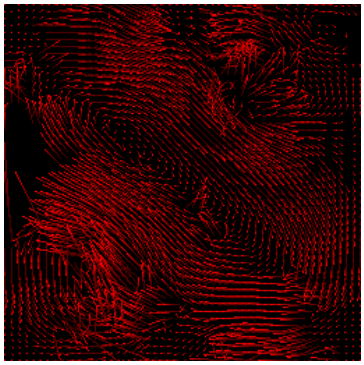
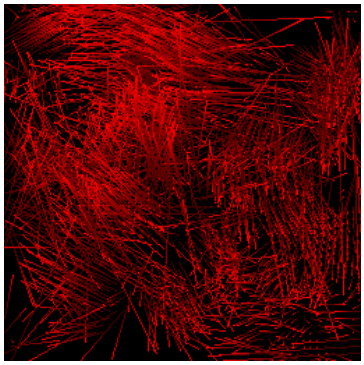
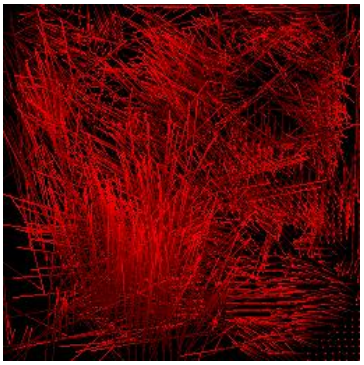
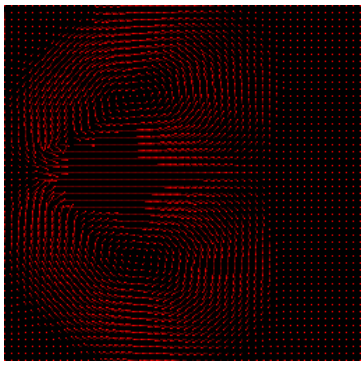
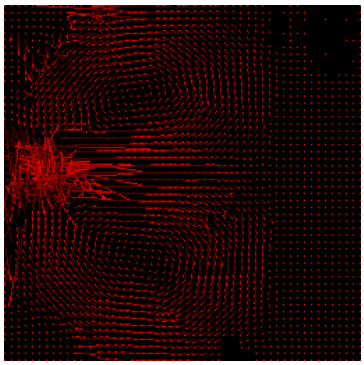
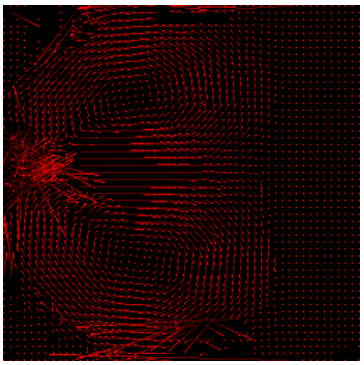
Da neue Partikel in diesem Verfahren eine zufällige Position im Quellbild bekommen sind die Ergebnisse nicht mit denen aus anderen Verfahren vergleichbar. Daher sind diese Differenzbilder nicht aussagekräftig.

7.3.2.3.3 Reversibilität

Aus dem gleichen Grund wie bei den Differenzbildern ist die Reversibilität in diesem Verfahren nicht berechenbar.

7.3.2.3.4 Optischer Fluss

Tabelle 7-133: Optischer Fluss des Forward Yu Verfahrens.

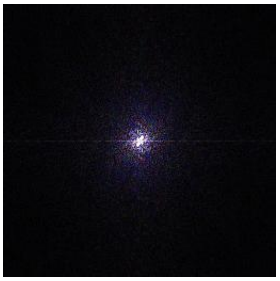
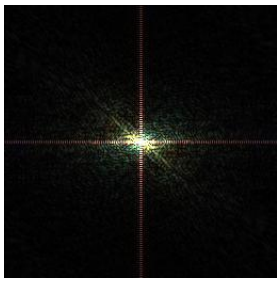
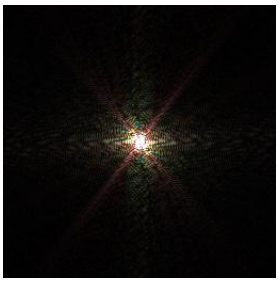
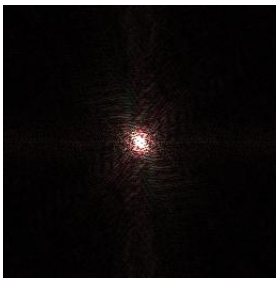
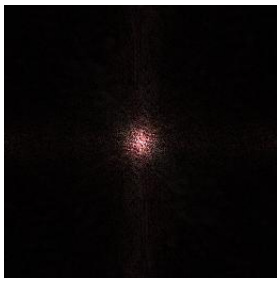
0-1	25-26	100-101
		
		

Der optische Fluss dieses Verfahrens ist auch nach hohen Schrittzahlen noch vorhanden. Das liegt daran, dass die Bewegung der einzelnen Partikel solange durchgeführt wird, bis ein Partikel entfernt wird.

Im ersten Vektorfeld ist allerdings die Berechnung des optischen Flusses bei hohen Schrittzahlen gescheitert, da sich die einzelnen Stellen zu ähnlich sehen.

7.3.2.3.5 Spektrum

Tabelle 7-134: Spektrum des Forward Yu Verfahrens.

0	5	25	100
			
			

Das Spektrum ist in diesem Verfahren konstant, nachdem es sich eingependelt hat. Ab Schritt 25 war dies hier der Fall.

7.3.2.3.6 Performance

Tabelle 7-135: Performance des Forward Yu Verfahrens.

Gesamtzeit:	35,73
Durchschnittszeit pro Schritt:	0,3573
Kürzeste Zeit eines Schrittes:	0,275
Längste Zeit eines Schrittes:	0,4739
FPS:	2,8

Dieses Verfahren ist nicht auf der CPU echtzeitfähig. Allerdings verwendet dieses Verfahren einen Softwarerenderer um die Gitter zu rendern. Dieser Renderer benötigt einen Großteil der Rechenzeit.

7.3.2.3.7 Bewertung

Dieses Verfahren wurde entwickelt, um eine kontinuierliche Textur zu erzeugen. Aus diesem Grund ist ein direkter Vergleich mit anderen Verfahren nicht möglich. Diese Verfahren erzeugen keine kontinuierliche Textur, sondern advektieren die vorhandene Daten.

Dieses Verfahren basiert auf einem normalen Rendervorgang, der hier in einer Softwarelösung implementiert wurde. Daher ist dieses Verfahren auf der CPU sehr rechenintensiv.

8 Fazit

Einige der hier vorgestellten und analysierten Verfahren führen die Advektion sehr gut aus. Die besten Ergebnisse liefert hierbei das Original BFECC Verfahren, welches Polynome dritten Grades einsetzt um die Daten auf den Feldern zu interpolieren und das klassische Runge-Kutta Verfahren zur Berechnung der Position verwendet.

Je höher der Polynomgrad bei der Interpolation der Daten auf den Feldern gewählt wird umso besser ist das Ergebnis der Advektion. Dies gilt allgemein für die Backward-Verfahren. Es muss allerdings darauf geachtet werden, dass die Performance mit dem Polynomgrad auch stark sinkt.

Der Einsatz eines Kernels oder der Einsatz von B-Splines dritten oder fünften Grades zur Approximation der Daten verschlechtert hierbei die Advektion. Dies ist darauf zurückzuführen, dass diese Methoden die Daten über eine Fläche mitteln und die gegebenen Stützstellen des B-Splines von dem B-Spline nicht getroffen werden.

Um die Position aus den Geschwindigkeitsvektoren des Vektorfeldes zu berechnen sollte das klassische Runge-Kutta Verfahren eingesetzt werden. Dieses Verfahren ist als einziges in der Lage Krümmungen im Vektorfeld ausreichend genau zu folgen. Die anderen Methoden erzeugen hier Ungenauigkeiten, die sich in niedrigerer Qualität des Ergebnisses wiederspiegeln. Die Reversibilität ist auch nur mit diesem Verfahren ausreichend genau gegeben.

Das BFECC Verfahren versucht mit einer Fehlerkorrektur das Ergebnis der Advektion weiter zu verbessern. Dies gelingt auch sehr gut. Es ist eine deutliche Verbesserung des Ergebnisses sichtbar. Allerdings geht das wieder zu Lasten der Performance.

Zu den Verfahren kann allgemein gesagt werden, dass jene, welche einen iterativen Ansatz verwenden, Probleme mit sich aufsummierenden Fehlern bekommen. Dieser Effekt kann nur dadurch verhindert werden, dass keine iterativen Verfahren verwendet werden. Dies wird bei den Forward RK 4 Particle Kernel und dem Forward Yu Verfahren versucht.

Bei dem Forward RK 4 Particle Kernel Verfahren entsteht allerdings bei manchen Vektorfeldern das Problem, dass nicht mehr das gesamte Ergebnis mit einer ausreichenden Dichte an Partikeln gefüllt wird. Es entstehen also Löcher in dem advektierten Bild. Dies wird in dem Forward Yu Verfahren gelöst, indem an entsprechenden Stellen neue Partikel erzeugt werden. Für diese Partikel ist allerdings der Ursprung nicht mehr vorhanden. Aus diesem Grund wird die Ursprungsposition des Partikels zufällig gewählt. Das erzeugt zwar eine geschlossene Advektion der Quelldaten, aber das Ergebnis ist keine vollständige Advektion der Quelldaten mehr.

Das Spektrum wird in den meisten Verfahren nicht gut erhalten. Lediglich beim Forward Yu Verfahren ist das Spektrum konstant nachdem es sich eingependelt hat. Der Erhalt des optischen Flusses hängt hingegen stark von dem verwendeten Vektorfeld ab. Der Fluss ist bei mehreren Verfahren ausreichend genau wiederherstellbar.

Abschließend kann gesagt werden, dass ein möglichst hoher Polynomgrad in Verbindung mit dem Original BFECC Verfahren und dem klassischen Runge-Kutta Verfahren verwendet werden sollte um die komplette Advektion der Daten durchzuführen. Wird jedoch keine komplette Advektion, sondern nur ein Fluss auf dem Vektorfeld benötigt, sollte das Forward Yu Verfahren verwendet werden, da es keinen iterativen Ansatz verwendet.

9 Literaturverzeichnis

Runge-Kutta. (18. 9 2011). Von http://einstein.drexel.edu/courses/Comp_Phys/Integrators/rk4.html abgerufen

Bridson, R., & Müller-Fischer, M. (2007). Fluid Simulation. *SIGGRAPH.*

ByungMoon, K., Yingjie, L., Ignacio, L., & Jarek, R. (2005). FlowFixer: Using BFECC for Fluid Simulation. *Eurographics Workshop on Natural Phenomena.*

Dunbar, D., & Humphreys, G. (2006). A Spatial Data Structure for Fast Poisson-Disk Sample Generation. *ACM Trans. Graph, vol. 25, no. 3*, (S. 503–508).

Enright, D., Marschner, S., & Fedkiw, R. (2002). Animation and Rendering of Complex Water Surfaces. *SIGGRAPH.* ACM.

Fedkiw, R., Stam, J., & Jensen, H. W. (2001). Visual Simulation of Smoke. *SIGGRAPH* (S. 23-30). ACM.

Foster, N., & Fedkiw, R. (2001). Practical animation of liquids. *SIGGRAPH* (S. 15-22). ACM.

Foster, N., & Metaxas, D. (1996). Realistic Animation of Liquids. *Graphical Models and Image Processing*, (S. 471-483).

Hairer, E., Nørsett, S. P., & Wanner, G. (2009). *Solving Ordinary Differential Equations I: Nonstiff Problems.* Berlin Heidelberg: Springer.

Microsoft. (August 2009). Windows DirectX Graphics Documentation.

Qizhi, Y., Neyret, F., Bruneton, E., & Holzschuch, N. (2010). *Lagrangian Texture Advection: Preserving both Spectrum and Velocity Field.* IEEE Transactions on Visualization and Computer Graphics.

Reeves, W. T. (1983). Particle Systems A Technique for Modeling a Class of Fuzzy Objects. *ACM Trans. Graph Vol. 2*, (S. 91–108).

Selle, A., Fedkiw, R., ByungMoon, K., Yingjie, L., & Jarek, R. (June 2008). An Unconditionally Stable MacCormack Method. *Journal of Scientific Computing Volume 35 Issue 2-3.*

Song, O.-Y., Shin, H., & Ko, H.-S. (2005). Stable But Nondissipative Water. *ACM Transactions on Graphics*, (S. 81-97).

Sorkine, O., Cohen-Or, D., Goldenthal, R., & Lischinski, D. (2002). Bounded-distortion piecewise mesh parameterization. *VIS'02: Visualization*, (S. 355–362).

Speith, R. (2006). Vorlesung Numerische Hydrodynamik Kapitel 7. Universität Tübingen.

Stam, J. (1999). Stable fluids. *Proc. SIGGRAPH*, (S. 121–128).

Takahashi, T., Fujii, H., Kunimatsu, A., Hiwada, K., Saito, T., Tanaka, K., et al. (2003). Realistic Animation of Fluid with Splash and Foam. *EUROGRAPHICS Vol 22.*