UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

Simulation von flüssigen Zeichenmitteln

Bachelorarbeit

zur Erlangung des Grades Bachelor of Science (B.Sc.)
im Studiengang Computervisualistik

vorgelegt von

Heike Polders

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	M.Sc. Nils Höhner (Institut für Computervisualistik, AG Computergraphik)

Koblenz, im Februar 2019

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Abstract

Within the field of computer graphics non-photorealistic rendering is a focus of technical and scientific visualization and especially of the artistic field. Different art styles as well as the drawing materials and their properties present various challenges. One of these challenges is the simulation of drawing media, which uses fluid solvents.

This bachelor thesis will create an interactive drawing environment for one fluid based drawing media - watercolor paint. For this simulation a rough paper will be generated and the fluid simulation as well as the optical behaviour of the watercolor paint will be implemented.

Zusammenfassung

Im Bereich der Computergraphik bilden die Nicht-Photorealistischen Renderingverfahren einen Schwerpunkt in der technischen und wissenschaftlichen Visualisierung, vor allem aber in den künstlerischen Bereichen. Verschiedene Kunststile, sowie Zeichenmaterialien und ihre Eigenschaften stellen unterschiedliche Herausforderungen dar. Eine dieser Herausforderungen ist hierbei die Simulation flüssiger Zeichenmittel.

Diese Arbeit beschäftigt sich mit der Erstellung eines interaktiven Zeichensystems für ein flüssiges Zeichenmittel, der Aquarellmalerei. Für die Simulation wird eine raue Zeichengrundlage generiert, sowie die Fluid Simulation und das optische Farbverhalten der Aquarellmalerei implementiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die Computergraphik ist ein weitreichendes Teilgebiet der Informatik, welche längst nicht mehr nur auf Computern Anwendung findet. Durch die Möglichkeit computergenerierte Bilder photorealistisch zu rendern, wurden sie unter anderem auch in Film und Werbung etabliert. Jegliche Wesen aus Märchen- und Fantasiewelten können dargestellt werden, als wären sie ein wahrer Teil dieser Welt, wie beispielsweise der Drache Smaug aus der Filmreihe des Hobbits. Aber auch weniger abstrakte Situationen lassen sich naturgetreu wiedergeben. In der Automobilindustrie muss daher kein Fahrer mehr für die Werbung durch einen Regenwald, Wasserfall oder eine Wüste fahren, um kurz vor einer Klippe stilvoll zum Stehen zu kommen, da dies realistisch am Computer animiert werden kann. Allerdings ist die photorealistische Umsetzung nur ein Schwerpunkt der Computergraphik. Denn bei all ihrer Faszination lenkt ihre Detailfülle schnell von essentiellen Informationen ab. Bei einem photorealistisch gerenderten Auto ist es schwerer die Aufmerksamkeit auf spezielle Details zu lenken. Aus diesem Grund wurde auf Nicht-Photorealistische Renderingverfahren (NPR) zurückgegriffen, um simplere Darstellungsweisen zu entwickeln. Auch in der Architektur und Medizin ist die Vermittlung von Informationen anhand von stilisierten Graphiken wichtiger als eine realistische Abbildung, weshalb in technischen und medizinischen Handbüchern hauptsächlich Illustrationen verwendet werden.

Neben der wissenschaftlichen Visualisierung finden NPR vor allem im künstlerischen Bereich Anwendung. Die Kunst hat in der Geschichte des Menschen viele verschiedene Stilrichtungen hervorgebracht. Der Realismus und die Photographie sind nur zwei Beispiele aus einer reichen Auswahl. So anschaulich ein realistischer Drache auch sein mag, schränkt der Realismus doch die Darstellungsmöglichkeiten stark ein. Eine Stilrichtung als Handwerkszeug eines Künstlers verleiht dem geschaffenen Kunstwerk eine Bedeutung und Atmosphäre. Die NPR geben Künstlern die Möglichkeit, diese Stile auch am Computer umzusetzen.

1.1 Motivation

Für die Umsetzung verschiedener Kunststile mithilfe von NPR gibt es mehrere Möglichkeiten. Zum einen gibt es die Filter der Bildverarbeitung. Die Filter lassen einen Algorithmus über ein bereits existierendes Bild laufen und ändern dieses entsprechend der Parameter ab, um somit die Stilrichtungen zu imitieren. Dies ist eine gängige Methode, um mit Bildverarbeitungsprogrammen schnelle Ergebnisse zu erzielen.

Das Resultat dieser Methode ist jedoch lediglich eine Approximation eines realen Bildes. Physikalische Aspekte, wie z.B. das Verhalten von nassen Farben miteinander, gehen somit verloren.

Zum anderen ist es möglich ein interaktives Programm zu erstellen mit dem gezeichnet werden kann. Dieses Programm soll die physikalischen Eigenheiten der Materialien in seine Berechnungen integrieren und ein realitätsnahes Bild erschaffen. Zu den Herausforderungen dieser Vorgehensweise zählen die flüssigen Zeichenmittel, da ihre Pigmente nicht direkt auf das Papier übertragen werden, sondern mit Wasser oder anderen Lösungsmitteln über das Papier bewegt werden. Je nach Absorption und Verdunstung des Wassers sowie der Pigmentaufnahme des Papiers entstehen Farbeffekte, die typisch für das jeweilige Zeichenmittel sind. Das Project Wetbrush von Adobe Research, NVIDIA and CUDA ist ein Beispiel für ein solches Programm. [1] Bei diesem Projekt handelt es sich um die Umsetzung von Ölfarben in Echtzeit. Nicht nur die Farben und der Verlauf wurden simuliert, sondern auch die Konsistenz, Dicke und Tiefe der Farbe auf der Leinwand wurden berücksichtigt. Ein 3D-Drucker kann diese Informationen nutzen, um ein Bild zu drucken, dessen Oberfläche die gleichen Höhen und Tiefen aufweist, wie ein reales Ölgemälde.

Zu den Anfängen dieser NPR zählt eine Arbeit über computergenerierte Aquarelle von Curtis et al. [2] In dieser Arbeit von 1997 wird eine Flusssimulation für das Verhalten von Wassers auf dem Papier und ein ausgearbeitetes Farbmodell auf Basis von Kubelka-Munk vorgestellt. Aquarell ist eine bekannte Maltechnik, deren Kombination aus Wasser und Pigmenten einzigartige Muster und Formen bildet. Für Anfänger ist es schwierig vorherzusehen, wie das Wasser mit den Pigmenten und dem Papier interagiert. Einmal zu Papier gebracht, lässt sich ein Strich nicht mehr rückgängig machen. Die Umsetzung eines Aquarellzeichenprogramms könnte daher den Einstieg für Anfänger erleichtern, da Fehler auf dem Papier des Programms nicht absolut sein müssen.

Es ist daher das Ziel dieser Bachelorarbeit ein Zeichenprogramm auf Basis der Arbeit von Curtis et al. über computergenerierte Aquarelle zu implementieren. Es soll die Eigenschaften von Aquarellen in die Simulation einbeziehen, um größtmögliche Ähnlichkeit zu realen Effekten zu schaffen. Hierzu werden die Bilder des Programms mit realen Effekten einer selbstgefertigten Aquarellzeichnung verglichen. Zudem wird geprüft, ob mit der Weiterentwicklung der Hardware in den letzten 22 Jahren der Algorithmus schnell genug arbeitet, sodass das Zeichenprogramm als Echtzeitsystem eingestuft werden kann.

1.2 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in vier weitere Kapitel auf. Im zweiten Kapitel werden die Grundlagen der flüssigen Zeichenmittel aufgezeigt. Angesichts der Zielsetzung dieser Arbeit, werden die Grundlagen der Aquarelle vorgestellt. Es werden Materialeigenschaften aufgezeigt und Techniken der Aquarellmalerei gewählt, die sich auf die Materialeigenschaften beziehen sollen. Die Umsetzung der Simulation dieser Techniken wird daraufhin in der Arbeit der computergenerierten Aquarelle betrachtet. Für die Bewertung der Performance wird, nach der Vorstellung der Entwicklungsumgebung, noch der Begriff Echtzeitsystem definiert.

Anschließend wird in Kapitel 3 auf die Implementierung des Programms eingegangen. Es wird die Umsetzung von Benutzeroberfläche, Papier, Fluid Simulation und den Pigmenten betrachtet und ihr Zusammenhang aufgezeigt. Desweiteren werden Anpassungen an der Arbeit von Curtis et al. erläutert. Auf die Bewertungskriterien aus dem 2. Kapitel aufbauend, werden die Ergebnisse der Simulation in Kapitel 4 analysiert. Es wird ein optischer Vergleich zwischen Simulation und realen Aquarelltechniken und -bildern gezogen und überprüft, ob die Simulation sich als Echtzeitsystem qualifiziert. Im letzten Kapitel wird dann das Fazit zu der Simulation gezogen und ein Ausblick gegeben, der auf diesem System aufbauende Verbesserungsmöglichkeiten des Algorithmus zur Weiterentwicklung der Optik und Performance zeigt.

2 Grundlagen der flüssigen Zeichenmittel

Traditionell werden zum Erstellen eines Bildes drei unterschiedliche Materialien benötigt: der Zeichengrund, auf dem gearbeitet wird, die Zeichenmittel, mit denen gearbeitet wird und die Zeichengeräte, mit denen die Zeichenmittel aufgetragen werden.

Bei den Zeichenmitteln lassen sich trockene und flüssige Zeichenmittel unterscheiden. Im Gegensatz zu den trockenen Zeichenmitteln werden die Pigmente der flüssigen Zeichenmittel nicht direkt auf das Papier übertragen. Lösungsmittel wie Wasser oder Alkohol bewegen die Pigmente über das Papier, welches je nach Absorption und Verdunstung des Lösungsmittels sowie der Pigmentaufnahme des Papiers Farbeffekte entstehen lässt.

2.1 Die Aquarellmalerei

Die frühen Anfänge der Aquarellmalerei finden sich schon im alten Ägypten, wo Holzsarkophage bemalt und Totenbücher illustriert wurden. [3] Aber auch in Ostasien wurde mit Aquarell-ähnlichen Wasserfarben gearbeitet. Aquarelle und ihre Techniken, wie sie heute bekannt und charakterisiert sind, wurden unter anderem von Albrecht Dürer (1471 - 1528), James Mallord William Turner (1775 - 1851), John Constable (1776–1837) und David Cox (1783–1859) geprägt. In der Kunst wird zwischen Zeichnung und Malerei differenziert. Zeichnungen sind durch Striche und Linien vereinfachte Motive, während in der Malerei Farben flächenhaft aufgetragen werden. Da Aquarell sowohl für Zeichnungen als auch Malerei genutzt werden kann, werden die Begriffe in dieser Arbeit synonym genutzt. [3]

Abbildung 1: William Turner, Konstanz am Bodensee 1842 [4]

Der englische Begriff *watercolor* lässt sich mit Wasserfarben und Aquarell ins Deutsche übersetzen. Aquarelle werden durch einen dünnen, transparenten, wässrigen Farbauftrag definiert. [5] Hierbei wird Farbe durchscheinend (lasierend) von hell nach dunkel aufgetragen, da helle Farben dunkle nicht abdecken und Deckweiß nicht für Lichter verwendet wird. Für diese Art der Farbe ist das Auslassen von Farbe essentieller Bestandteil, um Licht und Tiefen zu kreieren, wie beispielhaft in Abbildung 1 zu sehen ist.

Um die Simulation von Aquarellen so real wie möglich erscheinen zu lassen, müssen neben den physikalischen Eigenschaften der Materialien auch die klassischen Techniken in Betracht gezogen werden, durch welche die typischen Aquarell-Effekte entstehen.

2.1.1 Materialien

Die Aquarellfarbe besteht aus Pigmenten, welche wiederum aus Partikeln bestehen und in einer Mischung aus Wasser, Bindemittel und Tensiden auf das Papier gebracht werden. [3, 6, 7]

Die Auswahl des *Papiers* wirkt sich erheblich auf das Ergebnis des Bildes aus. [5] Die Oberflächenstruktur reicht von feinkörnig bis markant grobkörnig. Die größeren Vertiefungen der grobkörnigen Papiere sammeln beispielsweise mehr Wasser, wodurch ein zu schnelles Abfließen des Wassers verhindert wird. Zudem erscheint die Farbe bei schnellen Pinselstrichen mit weniger Wasser nur auf den Höhen des Papieres, was zu einer bruchstückhaften Wirkung führt. [3, 5] Das Gewicht des Papiers hat Einfluss auf die Trocknung. Wiegt es unter 200 g/qm, kann das Papier wellen und resultiert in einer ungewollten, ungleichen Trocknung der Ränder. [5] Die Leimung des Papiers beeinflusst die Saugfähigkeit und somit, wie tief die Farbe in das Papier eindringt. Je höher die Saugfähigkeit, desto weniger Leuchtkraft hat die Farbe. Aquarellpapier wird typischerweise aus Leinen oder Baumwolle hergestellt, damit mehr Luft zwischen den einzelnen Fasern zirkuliert. [6]

Die *Pigmente* der Aquarelle haben je nach Ausgangsstoff und chemischer Zusammensetzung ihrer Partikel unterschiedliche Auswirkungen auf die Farbwirkung. Bei einer geringeren Pigmentdichte (*density*) bleiben die leichteren Pigmente länger in dem sich bewegenden Wasser und verteilen sich dadurch weiter auf dem Papier als schwerere Pigmente. [6] Mit der Färbekraft (*staining power*) wird die Fähigkeit der Pigmente beschrieben, an den Fasern des Papiers haften zu bleiben. Pigmente mit einer höheren Färbekraft setzen sich zu Beginn eines Pinselstriches daher vermehrt in die Fasern ab, wodurch die Menge der Pigmente in dem Pinselstrich reduziert wird. [3, 6] Zuletzt bestimmt die Körnigkeit (*granulation*) der Pigmente die

Farbintensivität. Feinere Körner sammeln sich in größeren Mengen in den Vertiefungen des groben Papiers, wodurch die Farbe intensiver erscheint. Bei gröberen Körnern sammeln sich weniger Pigmente und die Farben erscheinen verwaschener. [3, 6]

Das *Bindemittel* ermöglicht es den Pigmenten von dem Papier und vorigen Malschichten aufgenommen zu werden und diese einzufärben. Die *Tenside* helfen hingegen dem Wasser in das Papier einzudringen. [6, 7]

2.1.2 Techniken

Künstler müssen Papiere und Farben austesten, um herauszufinden mit welchen Materialien sie am Besten zurechtkommen, sie selbst können diese Eigenschaften nicht verändern. Durch das Wissen über das Verhalten der Materialien sowie ihre gelernten Techniken, können sie die Eigenschaften in ihrem Sinne nutzen.

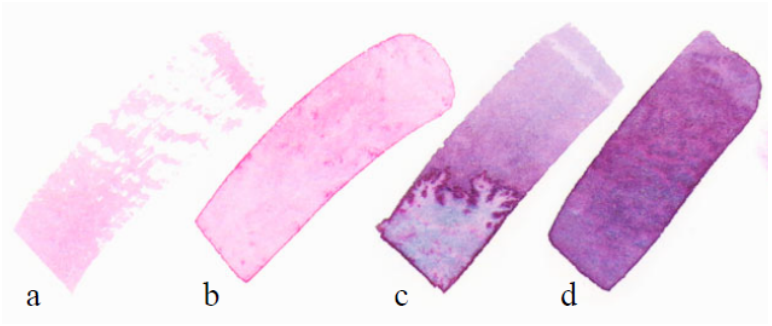


Abbildung 2: Effekte in realen Aquarellen von Curtis et al. Teil I: [2] (a) Trockener Pinsel, (b) Verdunkelte Kanten, (c) Rückläufe und (d) Granulation und Separation

Gekonnte Künstler können eine große Vielfalt an Effekten mit Aquarellen erzeugen. Die zwei grundlegenden Techniken sind *nass-in-nass* und *nass-auf-trocken*. Bei der ersten Technik wird das Papier vorher angefeuchtet, bei der zweiten Technik wird auf trockenem Papier gezeichnet. Durch das nasse Papier werden weitere Farbaufträge undefinierter, da die Farbe sich weiter ausbreiten kann als auf trockenem Papier. [5] Mithilfe der Basistechniken lassen sich eine Reihe von Effekten erzeugen, die in den Abbildungen 2 und 3 dargestellt sind.

- Trockener Pinsel (*dry-brush effects*) - Abbildung 2 (a): ein fast trockener Pinsel wird schnell und sanft über das Papier geführt. Somit werden nur die Höhen des groben Papiers mit Farbe benetzt, wodurch die Linie ungleichmäßige Bruchstücke erhält. [6] Diese Technik wurde, aufgrund des begrenzten Zeitrahmens dieser Arbeit, nicht genauer betrachtet.

- Verdunkelte Kanten (*edge darkening*) - Abbildung 2 (b): ein Pinselstrich auf trockenem Papier. Durch die Leimung des Papiers und die Wasserspannung breitet sich der Strich nicht weiter aus. Während die Farbe trocknet, bewegen sich die Pigmente aus dem Inneren stetig bis zur Strichkante, an der sie sich sammeln. Diese Sammelstellen erscheinen im getrockneten Strich dunkler und erzeugen somit dunklere Kanten. [5, 6]

- Rückläufe (*intentional backruns*) - Abbildung 2 (c): Wasser strömt zurück in einen noch feuchten Farbbereich. Bei einer Verwaschung oder Lavierung, bei der mit Wasser verdünnte Farbe auf das Papier aufgetragen wird, trocknet die Farbe ungleichmäßig. Dadurch bewegt das Wasser Pigmente in seine Bewegungsrichtung und kreiert komplexe Formen mit stark verdunkelten Kanten. [6]

- Granulation und Separation (*granulations and separation of pigments*) - Abbildung 2 (d): Die Granulation der Pigmente betont die Textur der Papieroberfläche. Durch die Sammlung von Pigmenten in den Vertiefungen des Papiers entstehen dort dunklere Stellen als an den Höhen. Bei der Separation handelt es sich um die Aufteilung von Farben, deren Pigmentdichten sich unterscheiden und eine Farbe sich so weiter verteilt als die andere. [6]

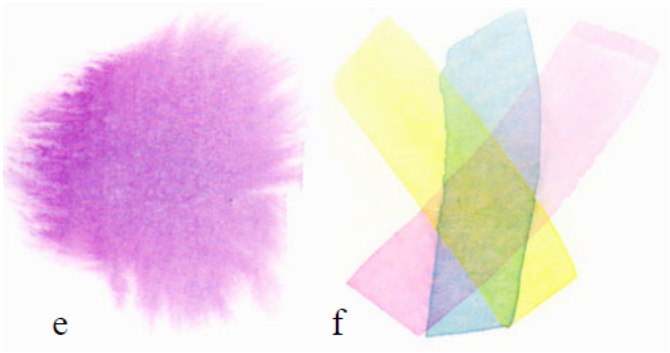


Abbildung 3: Effekte in realen Aquarellen von Curtis et al. Teil II: [2] (e) Flussmuster und (f) Lasieren

- Flussmuster (*flow patterns*) - Abbildung 3 (e): Bei nass-in-nass Zeichnungen kann ein Pinselstrich sich frei in alle Richtungen ausbreiten. Es entstehen hierbei sanft verzweigte Formen. [5]

- Lasieren (*glazing*) - Abbildung 3 (f): Es werden sehr dünne, blasse Lavierungen übereinander aufgetragen. Hierbei muss jede Schicht getrocknet sein, bevor die nächste hinzugefügt werden kann. Da jede Schicht für sich getrocknet ist, werden die Pigmente nicht miteinan-

der vermischt. So ergeben sich durch die Transparenz der Lavierungen optisch neue Farben. [5]

Diese Aquarelleffekte sind nur einige von vielen weiteren Techniken. [5] Die Arbeit von Curtis et al. beschränkt sich in der Umsetzung ihres Algorithmus auf die oben genannten Techniken, weshalb nur diese hier ausführlicher erwähnt sein sollen. [2]

2.2 Computergenerierte Aquarelle

Nachfolgend wird das Verfahren über computergenerierte Aquarelle vorgestellt. Dieses wurde 1997 in der Arbeit "Computer-Generated Watercolor" von Curtis et al. veröffentlicht. [2]

Es handelt sich bei den computergenerierten Aquarellen um eine empirisch-basierte Simulation. Ihr Ziel besteht darin, die zuvor aufgezählten Effekte so realistisch wie möglich wiederzugeben. Da das optische Verhalten des Ergebnisses wichtiger ist als physikalische Korrektheit, wurde die Simulation nicht strikt auf physikalisch-basierte Modelle begründet.

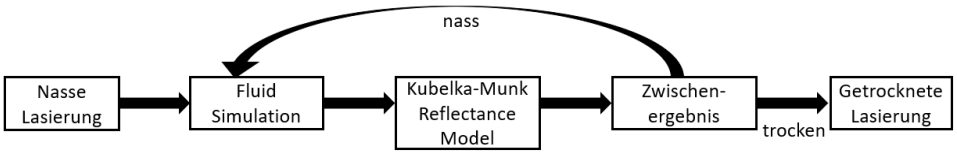


Abbildung 4: Simulationsablauf (abgeleitet aus [2])

Die Abbildung 4 beschreibt die grobe Struktur der Simulation. Ein Farbstrich wird auf das Papier aufgetragen. Jede Lasierung wird für sich in der *Fluid Simulation* berechnet, um den Fluss des Striches zu erhalten. In dieser Phase der Simulation werden die physikalischen Eigenschaften der Pigmente, des Papiers und des Wassers als Parameter zur Berechnung genutzt. Darüber hinaus wird eine "*wet-area Mask* " M als Maske mit booleschen Werten genutzt um festzuhalten, welche Bereiche des Papiers mit Wasser in Berührung gekommen sind (*true*), also nass sind, und welche trocken (*false*) sind.

Nach jedem Durchlauf der Fluid Simulation wird auf die Lasierung das *Kubelka-Munk Reflectance Model* angewandt, um die optische Zusammensetzung zu berechnen. Nach diesem Schritt erfolgt die visuelle Ausgabe auf dem Bildschirm. Falls auf dem Bild noch nasse Pixel vorhanden sind, wird der Durchlauf mit der Fluid Simulation erneut gestartet. Sind die Pixel trocken, ist der Durchlauf beendet.

2.2.1 Das Papier

Die Grundlage der Fluid Simulation ist die Generierung eines Papiers, auf dem gezeichnet werden kann. Das Papier kann, wie bereits erwähnt, unterschiedliche Auswirkungen auf den Fluss des Wassers und den Effekt der Farben haben. Es besteht daher die Möglichkeit, die Erstellung des Papiers parametergenau zu gestalten. Allerdings reicht für die Simulation bereits ein automatisch erstelltes Modell, welches alle nötigen Parameter ebenfalls automatisch berechnet. [2]

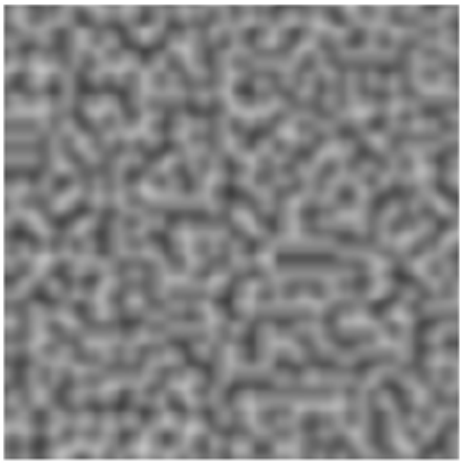


Abbildung 5: Ausgabe des Perlin Noise in Grauwerten

Hierfür wird die Papierhöhe h durch die Auswahl eines Pseudozufallsalgorithmus generiert. Ein möglicher Algorithmus ist der sogenannte Perlin Noise. Der Perlin Noise erstellt ein beliebig großes zweidimensionales Array und füllt dieses mit pseudozufälligen Zahlen. [8] Dieses Array wird zum einen als Textur angelegt, wodurch die Zahlen in Graustufenwerte zwischen 0 und 255 gespeichert werden, wie in Abbildung 5 zu sehen ist. Diese Textur wird dann vom Shader genutzt, um den Hintergrund zu beleuchten und die Struktur des Papiers zu Simulieren. Zum anderen wird das Höhen-Array h für weitere Funktionsberechnungen auf Werte zwischen 0 und 1 skaliert. Das Gefälle ∇h, berechnet aus dem Gradienten der Papierhöhe, wird genutzt, um die Flüssigkeitsgeschwindigkeiten u und v später zu beeinflussen.

Zuletzt muss aus dem Höhen-Array noch die Wasserkapazität c (1) berechnet werden. Diese soll später genutzt werden, um zu bestimmen, ob sich das Wasser auf dem Papier weiter ausbreitet.

$$c = h * (c_{max} - c_{min}) + c_{min} \tag{1}$$

2.2.2 Fluid Simulation

Die Fluid Simulation ist ein erheblicher Bestandteil der Simulation. Der Algorithmus wird auf einem Drei-Schichten Modell aufgebaut, welches das Verhalten von Wasser, Pigmenten und Papier in und zwischen diesen Schichten widerspiegelt (Abbildung 6). Im Folgenden werden alle Schichten genauer angeschaut und die Funktionen definiert, die das Verhalten umsetzen.

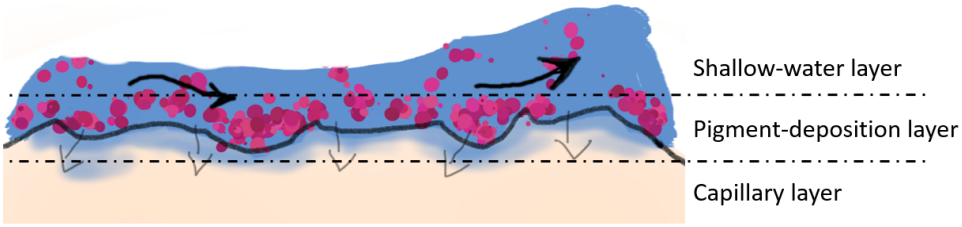


Abbildung 6: Drei-Schichten Modell

Shallow-water layer

In der oberen Schicht bewegen sich das Wasser und die Pigmente über das Papier. Um realitätsnahe Effekte zu erzielen, werden an das Verhalten des Wassers Bedingungen gestellt, die in der Arbeit der computergenerierten Aquarelle durch die Funktion *moveWater()* erfüllt werden sollen. Ihr grober Ablauf wird im Flussdiagramm der Abbildung 7 dargestellt. [2]

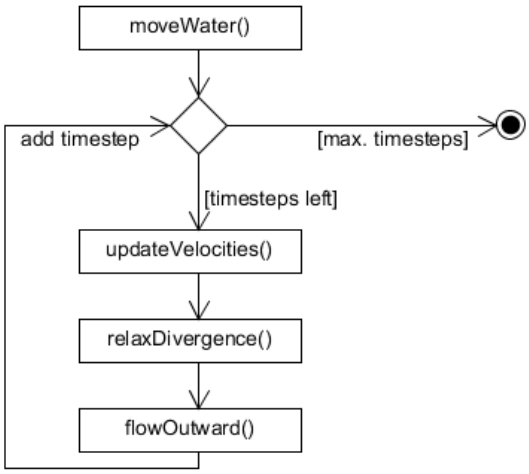


Abbildung 7: Flussdiagramm der Funktion moveWater (abgeleitet aus [2])

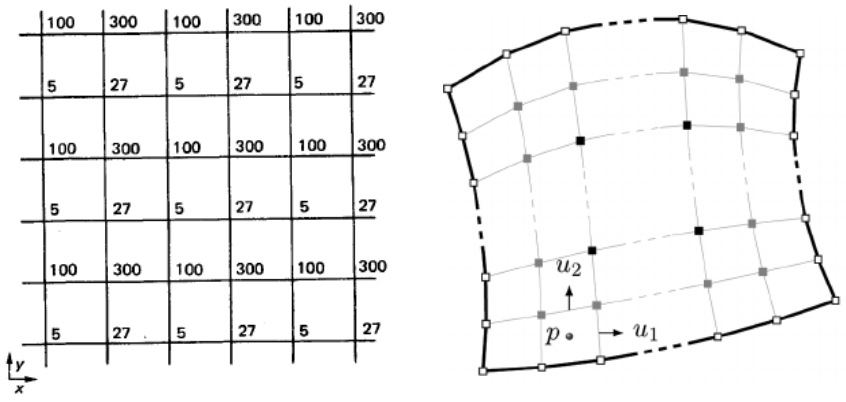
Die erste Funktion innerhalb von *moveWater() - updateVelocities()* - berechnet die Geschwindigkeit und die Bewegung des Wassers auf dem Papier. Zuerst wird das Gefälle des Papiers von der Geschwindigkeit u und v in x- und y-Richtung abgezogen. Somit beeinflusst die Textur des Papiers den Wasserfluss, da die höheren Stellen des Papiers die Geschwindigkeit und die Richtung des Flusses abändern. [2]

Danach wird die Bewegung des Wassers berechnet. Hierfür werden die Navier-Stokes-Gleichungen für inkompressible Fluide räumlich und zeitlich diskretisiert. Die Navier-Stokes-Gleichungen (2) und (3) sind partielle Differentialgleichungen und beschreiben das physikalische Momentum der Flüssigkeit unter Berücksichtigung der Geschwindigkeit u und v, des Wasserdrucks p , der Fließfähigkeit (*Viskosität*) μ und des Reibungswiderstands κ. [9] Die Einbindung der Fließfähigkeit und des Reibungswiderstandes reduziert die Flussgeschwindigkeit, wodurch die Wellenbildung auf dem Papier reduziert wird. [2]

$$\frac{\partial u}{\partial t} = -\left(\frac{\partial u^2}{\partial x^2} + \frac{\partial uv}{\partial y^2}\right) + \mu\nabla^2 u - \frac{\partial p}{\partial x} \tag{2}$$

$$\frac{\partial v}{\partial t} = -\left(\frac{\partial v^2}{\partial y^2} + \frac{\partial uv}{\partial x^2}\right) + \mu\nabla^2 v - \frac{\partial p}{\partial y} \tag{3}$$

Für die Speicherung der Parameter wird ein versetztes Gitter (*staggered grid* - Abbildung 8b) in Größe des Papiers verwendet. Hierbei werden die Geschwindigkeitswerte (u, v) versetzt an den Gitterwänden gespeichert, während der Druck p , die Pigmentkonzentration g^k und alle anderen Parameter in der Mitte der Gitterzellen gespeichert werden. [9]



(a) Checkboard pattern [10]

(b) Versetztes Gitter [11]

Abbildung 8: Checkboard pattern - Problem und Lösung

Der Grund für dieses Vorgehen liegt an einem Phänomen, dass sich das *checkerboard pattern* nennt (Abbildung 8a). Werden andere Gitterarten zum Speichern der Geschwindigkeiten genutzt, werden die Werte des Drucks wie bei einem Schachbrett angeordnet und der Druck ist somit nicht mehr realistisch. Durch die Nutzung des versetzten Gitters wird dieses Problem behoben. [10]

Für die versetzten Gitter werden die Indizes der Geschwindigkeiten u und v als Brüche dargestellt. Somit wird die Geschwindigkeit zwischen zwei Zellen $p_{i,j}$ und $p_{i+1,j}$ mit $u_{i+0.5,j}$ gekennzeichnet. Es wird zusätzlich noch eine *step size* $\Delta t < 1$ eingeführt, damit keine Zellen zwischen den Iterationsdurchläufen übersprungen werden. Die Navier-Stokes-Gleichungen (2) und (3) werden nun für alle Zellen des Gitters wie folgt diskretisiert: [2, 9]

for $all\ cells(i,j)$

$$A = u_{i,j}^2 - u_{i+1,j}^2 + (uv)_{i+0.5,j-0.5} - (uv)_{i+0.5,j+0.5}$$

$$B = (u_{i+1.5,j} + u_{i-0.5,j} + u_{i+0.5,j+1} + u_{i+0.5,j-1} - 4u_{i+0.5,j})$$

$$u_{i+0.5,j} = u_{i+0.5,j} + \Delta t\,(A - \mu B + p_{i,j} - p_{i+1,j} - \kappa u_{i+0.5,j})$$

$$A = v_{i,j}^2 - v_{i,j+1}^2 + (uv)_{i-0.5,j+0.5} - (uv)_{i+0.5,j+0.5}$$

$$B = (v_{i+1,j+0.5} + v_{i-1,j+0.5} + v_{i,j+1.5} + v_{i,j-0.5} - 4v_{i,j+0.5})$$

$$v_{i,j+0.5} = v_{i,j+0.5} + \Delta t\,(A - \mu B + p_{i,j} - p_{i,j+1} - \kappa v_{i,j+0.5}) \tag{4}$$

Diese Diskretisierung führt zu einer Konvergenz der Funktionen, womit die Geschwindigkeit des Wassers unter Berücksichtigung der physikalischen Aspekte aktualisiert wurde. [9]

Zum Schluss setzt die Funktion noch die Geschwindigkeiten der Ränder derjenigen Pixel auf 0, die an einen wet-area Maskenwert mit $M_{i,j} = false$ grenzen. Somit bleibt das Wasser innerhalb der wet-area Maske. [2]

Die zweite Funktion von *moveWater()* - *relaxDivergence()* - berechnet die Umverteilung von Wasser in die benachbarten Zellen, um die Divergenz der Geschwindigkeiten zu reduzieren. [9] Es werden daher abweichende Wassergeschwindigkeiten in u und v so lange angepasst, bis sie entweder unter einen selbst festgelegten Geschwindigkeitsgrenzwert τ oder eine festgelegte Anzahl an Umverteilungsdurchläufen N fallen. [2] Durch diese

Funktion wird sichergestellt, dass das Wasser sich nicht nur auf einer Stelle des Papiers festsetzt, sondern sich auch ausbreiten kann.

Die letzte Funktion von *moveWater() - flowOutward()* - reduziert den Wasserdruck *p* jeder Zelle proportional zum Abstand der Begrenzung der wet-area Maske. Um die Distanz zu berechnen, wird ein Gaußscher Weichzeichner mit einem quadratischen Kern auf die wet-area Maske angewandt. Die Werte der daraus resultierenden, weichgezeichneten Maske *M'* werden mit der wet-area Maske und einem vorher festgelegten Wert $0.01 \leq \eta \leq 0.05$ multipliziert und dadurch minimiert. Dieses Ergebnis wird dann vom Druck subtrahiert. [2]

Dies führt dazu, dass Zellen, die näher an der Grenze der wet-area Maske liegen, mehr Wasserdruck entzogen bekommen als Zellen, die weiter entfernt liegen. Der sinkende Druck führt dazu, dass viele Pigmente in den Pigment-deposition layer sinken und sich auf die Papieroberfläche absetzen. An diesen Stellen steigt daher die Pigmentmenge an, was zu einer Verdunklung der Farbe führt. Somit soll der Effekt des edge darkening erzielt werden. [2]

Nachdem das Wasser über das Papier bewegt wurde, müssen nun auch die Pigmente bewegt werden. Durch eine neue Funktion *movePigment()* werden die Pigmente von einer Zelle an ihre Nachbarn verteilt.

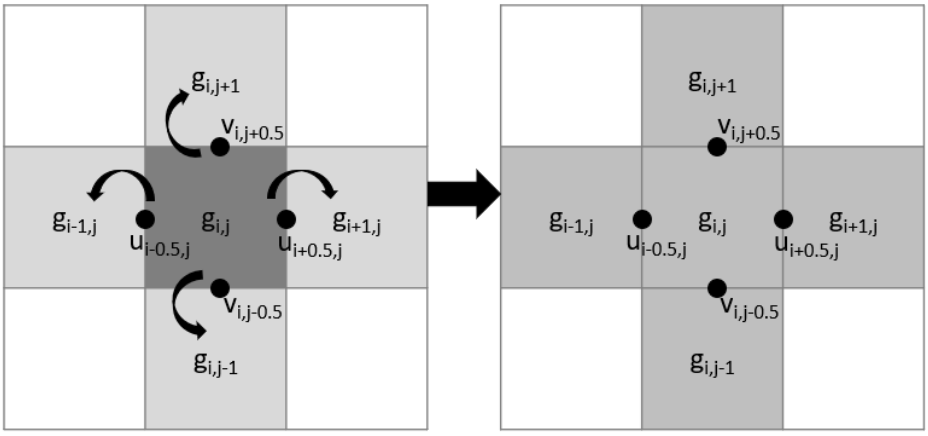


Abbildung 9: Schematische Darstellung des Pigmentflusses (abgeleitet aus [2])

Die Abbildung 9 stellt schematisch den Ablauf dieser Funktion dar. Für die Verteilung der Konzentration in die Nachbarzelle $g_{i+1,j}$ wird die Zelle $g_{i,j}$ mit der Geschwindigkeit in Richtung der Nachbarzelle $u_{i+0.5,j}$ multipliziert und zu der Pigmentkonzentration von $g_{i+1,j}$ addiert bzw. subtrahiert

im Falle von $g_{i-1,j}$ und $g_{i,j-1}$. Da die Nachbarzellen jedoch keine Konzentration verlieren dürfen, ist die Bedingung gesetzt, dass zwischen dem errechneten Produkt und der 0, die Größere der beiden Zahlen zu der Nachbarzelle addiert werden soll. [2]

Da in diesem Schritt Pigmente zur Simulation hinzugefügt wurden, muss zur Volumenerhaltung die gleiche Menge an Pigmenten entfernt werden. Daher werden alle eben berechneten Maxima addiert und von der Konzentration $g_{i,j}$ subtrahiert. Dadurch werden Pigmente aus der Zelle entfernt und Pigmente scheinen sich in die Nachbarzellen zu verteilen.

Pigment-deposition layer

In der mittleren Schicht können die Pigmente auf die Papieroberfläche sickern - Adsorption - und von dort auch wieder gelöst werden - Desorption - um weiter auf dem Papier bewegt zu werden. [2]

Für den Transfer der Pigmente sind drei Parameter von Bedeutung: die Dichte ρ, die Granularität ω in Abhängigkeit der Papierhöhe h und die Färbkraft γ der Pigmente. Diese Werte werden für jede Zelle berechnet und beeinflussen den Faktor, mit dem ein Pigment adsorbiert oder desorbiert wird. [2] Pigmente aus dem Shallow-water layer g^k ergeben im Zusammenspiel mit der Dichte und der Granularität die Konzentration δ_{down}, die adsorbiert werden soll. Pigmente aus dem Pigment-deposition layer d^k ergeben im Zusammenspiel mit der Dichte, der Granularität und Färbkraft hingegen die Konzentration δ_{up}, die desorbiert werden soll, siehe Abbildung 10. [2]

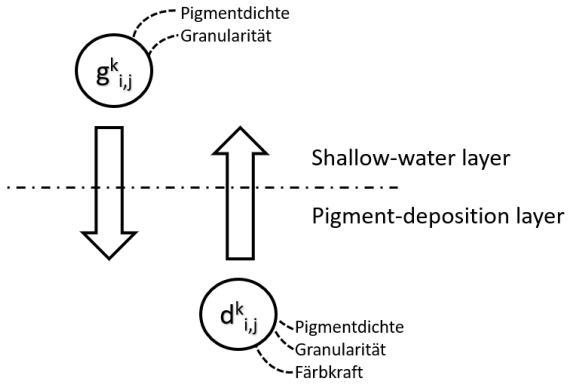


Abbildung 10: Schematische Darstellung der Adsorption und Desorption (abgeleitet aus [2])

Die neuen Konzentrationen der Pigmente ergeben sich aus ihrem ursprünglichen Wert, abzüglich der Konzentration, die in den jeweils anderen Layer wechselt und zuzüglich der Konzentration, die in den eigenen Layer wechselt. Um die Adsorptions- und Desorptionsraten gering zu halten, sollen δ_{down} und δ_{up} für den Fall, dass die Summe von $d^k + \delta_{down} > 1$ bzw. $g^k + \delta_{up} > 1$ ist, höchstens zwischen 0 und 1 liegen. [2]

Capillary layer

In der unteren Schicht wird das Wasser von dem Papier durch den Kapillareffekt absorbiert. Die letzte Funktion der Fluid Simulation berechnet daher den Wasserfluss im Papier.

Zuerst wird in allen Zellen, für die $M_{i,j} = true$ gilt, die Wassersaturation s angepasst. Diese Änderung wird in Abhängigkeit der Wasserkapazität c des Papiers und einer festgelegten Absorptionsrate α durchgeführt (5). [2]

$$s_{i,j} = s_{i,j} + max\left(0, min\left(\alpha, c_{i,j} - s_{i,j}\right)\right) \tag{5}$$

Danach wird die Saturation jeder Zelle im Zusammenhang mit jedem ihrer vier Nachbarn in x- und y-Richtung verglichen. Falls die Saturation der Zelle $s_{i,j}$ ein festgelegtes Minimum η überschreitet, kann das Wasser in eine benachbarte Zelle $s_{k,l}$ diffundieren. Die Wassersaturation dieser benachbarten Zelle muss ebenfalls ein festgelegtes, separates Minimum δ überschreiten, andernfalls kann kein Wasser in sie eindringen. Zuletzt muss die Wassersaturation der Zelle $s_{i,j}$ größer sein, als die ihres Nachbarn, ansonsten kann kein Wasser in diese Nachbarzelle eindringen. [2, 12]

Sind diese Bedingungen erfüllt, wird aus den Wassersaturationswerten von $s_{i,j}$ und seinen Nachbarn, sowie der Wasserkapazität der Nachbarn ein Saturationswert Δs ermittelt (6). Dieser Wert wird dann von der Zelle $s_{i,j}$ abgezogen (7), da das Wasser dort hinaus diffundiert, und den Nachbarzellen $s_{k,l}$ zu gleichen Teilen hinzugefügt wird (8). [2, 12]

$$\Delta s = max\left(0, min\left(s_{i,j} - s_{k,l}, c_{k,l} - s_{k,l}\right)/4\right) \tag{6}$$

$$s_{i,j} = s_{i,j} - \Delta s \tag{7}$$

$$s_{k,l} = s_{k,l} + \Delta s \tag{8}$$

Diese Simulation soll dazu führen, dass Wasser in feuchte Regionen des Papiers zurückfließen kann, da sich die Farbe so innerhalb eines Striches verteilt und dort trocknen kann. Dadurch sollen die Backruns der Aquarelle und der letzte somit vorgestellte Effekt aus Kapitel 2.1.2 simuliert werden. [2]

2.2.3 Kubelka-Munk Reflectance Model

Das *Kubelka-Munk Reflectance Model* (KM) ist ein Modell, welches das optische Farbverhalten von verschiedenen Lasierungen so realistisch wie möglich erscheinen lässt. [13, 14, 15]

Für die Umsetzung des Modells werden zwei Koeffizienten benötigt, welche zu den Eigenschaften des verwendeten Pigments gehören. Zum einen wird die Begrenzung der Absorption von Lichtenergie pro Dichteeinheit durch den Absorptionskoeffizient (*absorption coefficients*) *K* festgelegt. Zum anderen wird die Streuung der Lichtenergie pro Dichteeinheit durch den Streuungskoeffizienten (*scattering coefficients*) *S* gewählt. [15]

In der Implementation der computergenerierten Aquarelle werden die beiden Koeffizienten, im Gegensatz zum originalen KM, nicht experimentell bestimmt. [2] Über einem weißen und schwarzen Hintergrund kann das Erscheinungsbild eines Pigments eigenständig bestimmt werden, indem für den jeweiligen Hintergrund ein RGB-Wert gewählt wird. Diese RGB-Werte werden als R_w, für das Pigment auf einem weißen Hintergrund, und R_b, für das Pigment auf einem schwarzen Hintergrund, zur Berechnung der Variablen a und b in den Gleichungen (11) und (12) genutzt, um aus ihnen wiederum die Streuungs- und Absorptionskoeffizienten in den Gleichungen (9) und (10) zu bestimmen. [2]

$$S = \frac{1}{b} \cdot \coth^{-1}\left(\frac{b^2 - (a - R_w)(a - 1)}{b(1 - R_w)}\right) \tag{9}$$

$$K = S(a - 1) \tag{10}$$

$$a = \frac{1}{2}\left(R_w + \frac{R_b - R_w + 1}{R_b)}\right) \tag{11}$$

$$b = \sqrt{a^2 - 1} \tag{12}$$

Um eine Division durch 0 zu vermeiden, wird zusätzlich noch die Bedingung gestellt, dass $0 < R_b < R_w < 1$ gelten soll. [2] Durch diese Art der Pigmentbestimmung lassen sich beispielsweise transparente Farben erstellen. Sie erscheinen farbig auf weißem Untergrund und durchsichtig auf schwarzem Hintergrund. Opake Farben hingegen zeigen auf weißem und schwarzem Hintergrund die gleiche Farbe. Auch Interferenzfarben lassen sich so erstellen. Sie wirken transparent auf weißem und farbig auf schwarzem Hintergrund.

Durch die Benutzerauswahl der Pigmentwerte fallen so zusätzliche physikalische Formeln, wie z.B. die chemische und elektrostatische Interaktion zwischen Pigmenten und dem Farbmedium, zur Berechnung der Farbtypen weg. [2] Die Autoren der computergenerierten Aquarelle merken dazu an, dass die Ergebnisse des KM physikalisch nicht akkurat sind, jedoch eine möglichst plausible Approximation liefern.

Nach der Bestimmung von *S* und *K* berechnet das KM die optische Zusammensetzung übereinanderliegender Schichten. Zunächst wird die Dichte der Pigmentkonzentration aus dem Shallow-water und Pigment-deposition layer berechnet. Hierfür werden g^k und d^k addiert, um die relative Dichte x^k zu erhalten. [2] Danach wird pro Farbschicht aus *S* , *K* und x^k der Reflektionsgrad (*reflectance*) *R* (14) und die Lichtdurchlässigkeit (*transmittance*) *T* (15) mithilfe der Hilfsfunktion *c* (13) berechnet. [15]

$$c = a \sinh bSx + b \cosh bSx \tag{13}$$

$$R = \sinh bSx / c \tag{14}$$

$$T = b / c \tag{15}$$

Bei zwei übereinanderliegenden Farbschichten mit den Reflektionsgraden R_1 und R_2 sowie den Lichtdurchlässigkeiten T_1 und T_2, werden der Gesamtreflektionsgrad (16) und die Gesamtlichtdurchlässigkeit (17) errechnet. [15]

$$R = R_1 + \frac{T_1^2 R_2}{1 - R_1 R_2} \tag{16}$$

$$T = \frac{T_1 T_2}{1 - R_1 R_2} \tag{17}$$

Nach der Berechnung der Lichtdurchlässigkeit und des Reflektionsgrades können die Pigmente im Fragment Shader gerendert werden und erhalten dadurch ihre endgültige Farbe. Durch das Rendern im Fragment Shader wird das KM nicht in die Fluid Simulation eingebunden, sondern parallel berechnet.

2.3 Programmierumgebung - Qt

Für die Programmierung der graphischen Benutzeroberfläche oder kurz GUI, für die englische Bezeichnung *Graphical User Interface*, stehen verschiedene GUI-Toolkits zur Auswahl. Zu den plattformübergreifenden Toolkits gehören unter anderem Qt, GTK+, Tk, GLUI und FLTK. [16] Eine Bedingung an die Programmierumgebung ist, dass sie die Graphikbibliothek OpenGL unterstützt. OpenGL erlaubt es komplexe 3D-Szenen in Echtzeit zu rendern und ermöglicht es, Rechenaufwand von der CPU auf die GPU

zu verlagern. [17] Das Programm wird OpenGL für das Rendern des Papiers nutzen, um seine Struktur anhand des Höhen-Arrays h zu beleuchten. Darauf aufbauend werden die Pigmente durch das Kubelka-Munk Reflectance Model berechnet, als Textur gespeichert und ebenfalls gerendert. Da die Pigmente in eine Textur gespeichert werden, werden die Berechnungen auf der GPU ausgeführt und die CPU somit entlastet.

Die Qt Entwicklungsumgebung unterstützt OpenGL und ist sowohl für kommerzielle Programmierung mit proprietärer Lizenz als auch kostenfrei für die Open-Source-Programmierung verfügbar. [18] Es kann zudem aus mehreren Programmiersprachen wie C++, C, Java, Python oder auch Haskell ausgewählt werden.

2.4 Echtzeitsystem

Die Definition der Norm DIN 44300 (Informationsverarbeitung), Teil 9 (Verarbeitungsabläufe) definiert den Begriff Echtzeit wie folgt:

> "*Unter Echtzeit versteht man den Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen.*“ [19]

Ein Echtzeitsystem ist demnach ein System, welches keine Verzögerungen aufweist, die die Bedienbarkeit des Systems einschränken. Diese Verzögerungen lassen sich durch die Rate *FPS* (*frames per second*), mit der Bilder auf dem Bildschirm ausgegeben werden, ausdrücken.

Bei einer Rate von 1 FPS erscheint dem Benutzer ein System nicht sehr interaktiv. Dies ist ungefähr mit einem Daumenkino vergleichbar, welches die Abfolge eines laufenden Hundes zeigt. Wird nur ein Blatt pro Sekunde umgeblättert, scheint der Hund nur auf der Stelle zu stehen. Ab einer Rate von 6 FPS wird das Gefühl stärker, dass es sich tatsächlich um ein interaktives Programm handelt. [20]

Eine Bildaktualisierungsrate von 6 FPS wird daher im Rahmen dieser Arbeit als Bedingung gesetzt, damit die Simulation sich als Echtzeitsystem qualifiziert.

3 Implementierung

Aufbauend auf die Theorie aus Kapitel 2.2 wurde die Simulation implementiert. Zuerst wurde eine Benutzeroberfläche erstellt, auf der gezeichnet werden kann. Sie ermöglicht es außerdem flexible Variablen zu ändern und Farben auszuwählen. Für die Zeichengrundlage wurde daraufhin eine Funktion zur Papiergenerierung geschrieben, die die Berechnung des Höhenfelds aufruft und eine Textur an die Shader weitergibt.

Die Funktionen der Fluid Simulation zur Berechnung der Wasser- und Pigmentbewegung, der Adsorption und Desorption der Pigmente und des Kapillarflusses im Papier wurden auf dieser Basis implementiert. Zuletzt wurde noch eine Klasse für Pigmente erstellt, welche die Koeffizienten des KM berechnet und diese Werte an den Fragment Shader zum Rendern der Farben weitergibt.

3.1 Graphical User Interface

Damit der Benutzer mit dem Programm interagieren kann, wird ein User Interface erstellt. Das MainWindow (Abbildung 11) besteht hauptsächlich aus einem *OpenGLWidget*, welches in der Lage ist, OpenGL Graphiken zu rendern und darüber hinaus als Zeichengrund genutzt werden soll. Um zu überprüfen wie oft sich dieses Widget aktualisiert, wird die aktuelle Update Rate oberhalb in FPS ausgegeben.

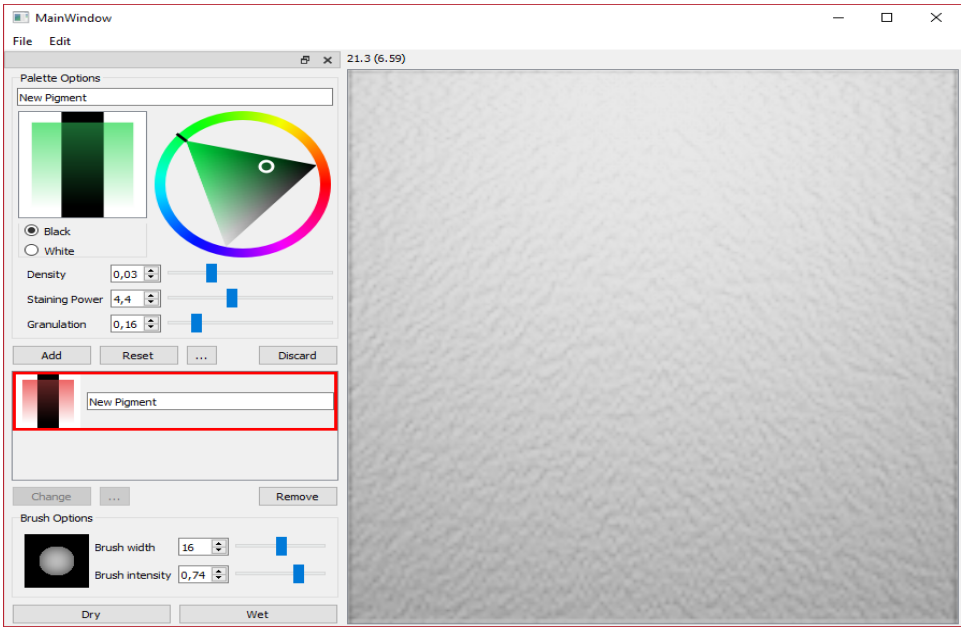


Abbildung 11: User Interface des Simulationsprogramms

Über die Schaltfläche *Edit* lässt sich eine *DockWidget Toolbar* öffnen und schließen, welche Optionen enthält, mit denen die Simulation beeinflusst werden kann. Zum einen kann die Farbpalette bearbeitet werden. Hierfür muss zuerst ein neues Pigment erstellt werden, dessen Farbe sich über ein *ColorWheel* auswählen lässt. [21] Für die Unterscheidung der Farben auf schwarzem und weißem Hintergrund legen die Buttons *Black* und *White* fest, für welchen Hintergrund die Farbe gilt. Diese Werte ergeben das R_w und R_b für die Farbberechnungen des KM. In einer kleinen Preview werden diese Farbwerte in einem Farbverlauf von opak bis transparent parallel zur Auswahl dargestellt.

Unterhalb des ColorWheels lassen sich durch Regler die Density ρ, die Staining Power ω und die Granulation γ der Pigmente auswählen. Auf Grundlage der computergenerierten Aquarelle wurden die Spannen von ρ auf 0.01 - 0.09, ω auf 1.0 - 9.9 und γ auf 0.01 - 0.99 gesetzt. Danach lässt sich das Pigment einer Liste hinzufügen, wodurch es immer wieder aufgerufen und genutzt werden kann. Es ist ebenfalls möglich, dieses gespeicherte Pigment im Nachhinein noch zu ändern (*Change*). Dabei ändert sich die Farbe eines bereits mit diesem Pigment gezeichneten Strichs simultan zu der Änderung im ColorWheel, wodurch nachträgliche Farbänderungen an einem gezeichneten Bild möglich sind.

Zum anderen können die Eigenschaften des Pinsels bearbeitet werden. Es lassen sich die Pinselgröße und -intensität ändern. Mit der Pinselintensität kann die Transparenz eines Pigments zusätzlich reguliert werden. Beide Änderungen werden simultan in einer kleinen Preview dargestellt.

Unten in der Toolbar sind noch zweit Buttons, welche die gesamte Oberfläche des Papiers trocknen (*Dry*) oder nass machen soll (*Wet*). Wet setzt die gesamte wet-area Maske $M = true$ und befüllt die gesamte Matrix p mit dem Wasserdruck. Dry hingegen setzt $M = false$ und $p = 0$.

Ebenfalls Teil der Toolbar ist ein *Debug Slider*, der zusätzlich ein- und ausgeblendet werden kann. Dieser Slider ermöglicht es, den Richtungseinfall und die Stärke des Lichts, mit dem die Szene im OpenGLWidget beleuchtet wird, während der Nutzung der Simulation zu ändern, wie in Abbildung 12 veranschaulicht.

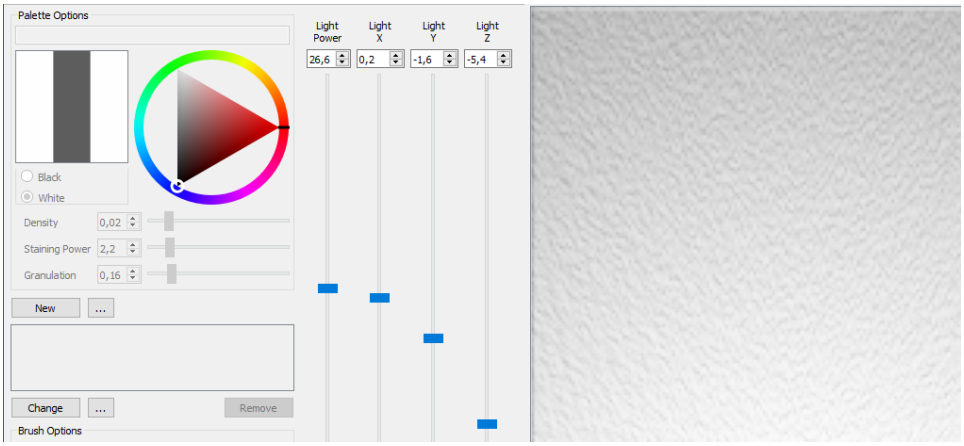


Abbildung 12: Toolbar mit Debug Slider und geänderten Lichteinstellungen

3.2 Generierung des Papiers

Das OpenGLWidget enthält neben den OpenGL Funktionen auch noch die Funktionen zum Zeichnen. Hier werden Pigmentfarbe und Pinselgröße gesetzt, die Aktivität des Eingabegerätes gespeichert und die Funktion zum Zeichnen in der Klasse *paper* aufgerufen. Für eine übersichtlichere Klassengestaltung wurde eine Hilfsklasse *paperentity* erstellt, die vom OpenGLWidget aufgerufen wird um die OpenGL Funktionen auszuführen und die Szene zu rendern. Die Grundlage für die Textur des Papiers, die die paperentity an die Shader weiterreicht, wird in der Klasse paper erstellt.

Zunächst wird eine Matrix mit 3D Vektoren für die Papierhöhe h, das *heightField*, erstellt und mit den Werten zwischen 0 - 255 aus einer Perlin Noise Berechnung gefüllt. [22] Der Algorithmus des Perlin Noise basiert auf einer verbesserten Version aus dem Jahr 2002, in welchem die Interpolationsdiskontinuität und die Gradientenberechnung ausgebessert wurden. [23] Der Perlin Noise erhält eine ganze Zahl als Argument, die einen Wert aus einem Array auswählt, die für die Berechnung des Perlin Noise genutzt wird. Wird eine Konstante als Argument gewählt, ergeben sich immer die gleichen Werte und daraus folgend auch immer die gleiche Textur. Damit unterschiedliche Texturen erzeugt werden können, wird die aktuelle Zeit des Systems in einen Integer umgewandelt und als Argument gewählt. Ist diese Zahl größer als das Array lang ist, wird das Array so oft dupliziert, bis

die Zahl erreicht werden kann. Nachdem das Heightfield erstellt ist, wird seine Matrix normalisiert, in der paperentity in eine Textur gespeichert und an den Fragment Shader weitergereicht.

Abbildung 13: Papier in der Simulation

Als Grundlage des Fragment Shaders dient der Source Code eines Normal Mapping Shaders, der durch das KM noch erweitert wird. Normal Mapping berechnet die Beleuchtung eines Fragments mit dem Wert seiner Normalen. Jedes Fragment besitzt unterschiedliche, zufällige Werte, wodurch sie unterschiedlich beleuchtet werden und der Effekt einer Höhenstruktur entsteht (Abbildung 13). [24]

Für die Berechnung der Wassergeschwindigkeit wird aus der Papierhöhe h noch das Gefälle, der Gradient ∇h, berechnet. Der Gradient ist ein Vektor der partiellen Ableitungen. Er gibt die Änderungsrate in alle Koordinatenachsen an und zeigt in die Richtung der stärksten Änderung. [25] Hierfür wird der nächste Höhenwert beispielsweise für $h_{i,j+1}$ vom vorhergehenden $h_{i,j-1}$ subtrahiert und durch 2 geteilt. Dies geschieht für alle Werte in x- und y-Richtung. Es muss allerdings eine Randbehandlung stattfinden, um eine Exception zu vermeiden. Hierzu wird beispielsweise an $h_{0,5}$ wie folgt verfahren: $(h_{1,5} - h_{0,5})\,/1$. Da $h_{0,5}$ keinen vorangehenden Höhenwert hat, wird der fehlende Wert durch den Wert an der aktuellen Position ersetzt.

Zum Schluss wird noch die Matrix c für die Wasserkapazität mit der Gleichung (2) erstellt und befüllt.

3.3 Aquarell Simulation

Nach der Generierung des Papiers folgt die Implementation der Zeichenfunktion *drawLine*. Sie erhält vom OpenGLWidget die Information über Start-, Endpunkt sowie den Verlauf einer Linie, die mit einem Eingabegerät gezeichnet wurde. Aus der normierten Subtraktion von End- und Startpunkt geteilt durch die Zeit, die zum Zeichnen der Linie benötigt wurde, wird die Geschwindigkeit errechnet und in die Matrizen u und v an den Positionen gespeichert, an denen die Linie gesetzt wurde.

Drawline wird ebenfalls die Matrizen für den Druck p, die wet-area Maske M und die Pigmentkonzentration g^k an diesen Positionen füllen. Die Höhe von p wird mit 0.05 definiert, ein Wert, welcher sich nach einigen Versuchen in der Simulation als sinnvoll ergeben hat. Wird das p höher gesetzt, entstehen die dark edges schneller, da mehr Wasser und Pigmente bewegt werden, das hat allerdings auch zur Folge, dass zwei überlappende Striche schneller und über eine weitere Fläche miteinander verlaufen. Die Pigmentkonzentration wird ebenfalls auf den Wert 0.05 gesetzt und M wird an den Positionen $true$.

Darauf aufbauend wird als nächstes die Funktion *updatePaper* implementiert, welche zuerst vier Funktionen für die Fluid Simulation und dann noch eine Funktion für die Farbdarstellung der Pigmente auf Basis des Pseudo Codes aus der Arbeit über computergenerierte Aquarelle von Curtis et al. aufrufen wird. [2]

```
proc updatePaper()
  for each time step do
      moveWater();
      movePigment();
      transferPigment();
      simulateCapillaryFlow();
      calculateTexture();
  end
end
```

3.3.1 Fluid Simulation

Im Konstruktor der Klasse paper wird die parallele Ausführung von updatePaper gestartet. Diese Funktion wird so lange aufgerufen, bis die Löschung des Objekts die Ausführung terminiert. Zuerst ruft updatePaper die vier Funktionen zur Berechnung des Drei-Schichten Modells auf und berechnet im Anschluss die Änderungen in der Textur, die an die Shader weitergereicht werden. Die Zeit, die diese Funktion für einen Durchlauf benötigt, wird gemessen und ebenfalls als FPS neben der Aktualisierungsrate

des OpenGLWidgets in Klammern ausgegeben.

moveWater()

MoveWater ruft zunächst *updateVelocities* auf, um die Matrizen der Geschwindigkeiten u und v zu aktualisieren. Dafür wird zunächst der Gradient ∇h von den Matrizen subtrahiert. Entsprechend der Diskretisierung der Navier-Stokes-Gleichungen aus Kapitel 2.2.2 Gleichung (4), werden die beiden Matrizen berechnet und ergeben die momentane Geschwindigkeit. Bei der Umsetzung des Codes mussten die zwei Zeilen zur Berechnung von $u_{i+0.5,j}$ und $v_{i,j+0.5}$ angepasst werden, da die Berechnungen ansonsten divergieren. $\Delta t\,(A - \mu B...)$ musste zu $\Delta t\,(A + \mu B...)$ abgeändert werden. Somit entstanden folgende neue Formeln für den Code:

$$u_{i+0.5,j} = u_{i+0.5,j} + \Delta t\,(A + \mu B + p_{i,j} - p_{i+1,j} - \kappa u_{i+0.5,j}) \tag{18}$$

$$v_{i,j+0.5} = v_{i,j+0.5} + \Delta t\,(A + \mu B + p_{i,j} - p_{i,j+1} - \kappa v_{i,j+0.5}) \tag{19}$$

Die Funktion *enforceBoundaryConditions* setzt daraufhin alle Geschwindigkeiten derjenigen Positionen in u und v auf 0, die an einen wet-area Maskenwert $M_{i,j} = false$ grenzen. Eine schematische Darstellung des Ergebnisses dieser Funktion wird in Abbildung 14 gezeigt. Somit wird der Wasserfluss auf die Positionen beschränkt, an denen $M_{i,j} = true$ gilt.

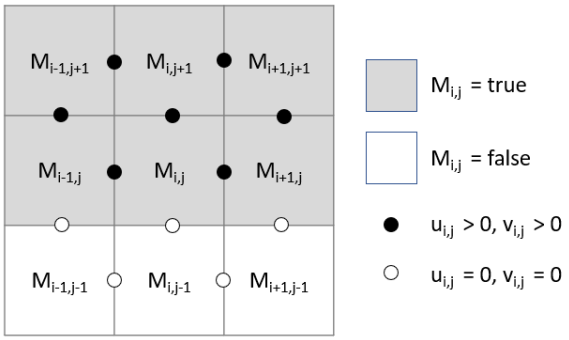


Abbildung 14: Schematische Darstellung der enforceBoundaryConditions (abgeleitet aus [2])

Danach folgt die Umverteilung des Wassers in die benachbarten Zellen durch *relaxDivergence*. Anhand der Vorgabe in der Arbeit von Curtis et al., wurde die maximale Anzahl an Umverteilungsdurchläufen auf $N = 50$ und die Toleranzgrenze der Geschwindigkeit auf $\tau = 0.1$ gesetzt. [2] Zuerst wird der Umverteilungswert δ bestimmt, der aus den vier Geschwindigkeitswerten, die an $p_{i,j}$ angrenzen, multipliziert mit dem Wert $\xi = 0.1$,

besteht. Der Wert von ξ wurde ebenfalls vorgegeben und sorgt dafür, dass δ minimiert wird und die Umverteilung in kleineren Schritten geschieht. [2]

Um der Logik der Umverteilung zu entsprechen, sollte δ von dem Druck der Zelle $p_{i,j}$ und den Geschwindigkeitsmatrizen u und v subtrahiert werden. Im Pseudo Code wurde sie jedoch addiert, der Druck also erhöht. Dies führte in der Ausführung der Simulation dazu, dass mehr und mehr Druck und damit Wasser und Pigmente zusätzlich erzeugt wurden. Um das gegebene Wasser- und Pigmentvolumen zu wahren, wurde ξ als ein Negativwert $\xi = -0.1$ deklariert. Somit wird δ nun von den Matrizen p, u und v subtrahiert.

Zuletzt wird mit *flowOutward* der Druck p an den Rändern der wet-area Maske M reduziert, um die darkedges zu erzeugen. Zunächst soll hierfür ein 2-Dimensionaler Gaußscher Weichzeichner mit einem 10x10 Kern auf M angewandt werden. Da es sich bei dem Gauß Filter um einen separierbaren Filter handelt und um die Zeit für die Berechnung der weichgezeichneten Maske M' zu reduzieren, werden anstelle eines 2-Dimensionalen Filters zwei 1-Dimensionale Filter nacheinander auf M angewandt. [26] Der Kern des Filters wurde auf 11x11 gesetzt, da der Mittelpunkt des Filters andernfalls nicht ganz in der Mitte war. Die darkedges waren daraufhin so verschoben, dass sie nur an den rechten und unteren Rändern der Striche zu sehen waren.

Nach der Berechnung von M', wird die Formel $\eta * \left(1 - M'_{i,j}\right) * M_{i,j}$ von jedem $p_{i,j}$ in p subtrahiert. Das η soll dabei zwischen 0.01 und 0.05 liegen. Der Unterschied liegt an der Deutlichkeit der darkedges. Abbildung 15 zeigt die in dem Programm erstellten darkedges und verdeutlicht den Unterschied in der Wahl des η.

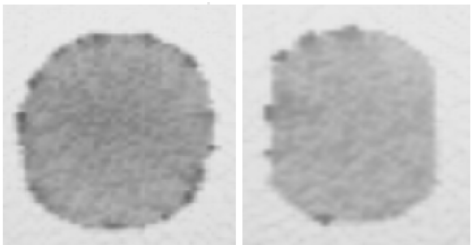


Abbildung 15: Links: $\eta = 0.01$; Rechts: $\eta = 0.05$

movePigment()

In *moveWater* wird für jede Zelle ihr Konzentrationswert $g^k_{i,j}$ mit der Geschwindigkeit u, v in Richtung der jeweiligen Nachbarzelle multipliziert. Dieser Wert wird dann zur Nachbarzelle addiert. Durch $max\left(0, u_{i+0.5,j} * g^k_{i,j}\right)$ ist die 0 der niedrigste Wert, der zu den Nachbarzellen addiert werden kann. Dadurch wird ihnen keine Konzentration entzogen. Zuletzt werden alle Werte, die den Nachbarzellen hinzugefügt wurden, von dem Konzentrationswert $g^k_{i,j}$ der Zelle subtrahiert (20).

$$g^k_{i,j} -= \left(max\left(0, u_{i+0.5,j} * g^k_{i,j}\right) + ... + max\left(0, -v_{i,j-0.5} * g^k_{i,j}\right)\right) \tag{20}$$

Somit bleibt das Konzentrationsvolumen erhalten und wird nicht reduziert, während die Pigmente über das Papier bewegt werden.

transferPigment()

Mit *transferPigment* wird als nächstes die Bewegung der Pigmente zwischen dem Shallow-water und dem Pigment-deposition layer berechnet. Dafür werden pro Pigment an den Stellen an denen $M_{i,j} = true$ gilt, die ad- und desorbierten Pigmentkonzentrationen berechnet. Die Menge der Konzentrationen δ_{down} und δ_{up}, welche die Layer wechseln, werden als double mithilfe des heightFields h, der Density ρ, der Staining Power ω und der Granulation γ in den Gleichungen (21) und (22) berechnet.

$$\delta_{down} = g^k_{i,j} * (1 - h_{i,j} * \gamma) * \rho; \tag{21}$$

$$\delta_{up} = g^k_{i,j} * (1 + (h_{i,j} - 1) * \gamma) * \rho/\rho; \tag{22}$$

Damit die Konzentrationen nicht zu schnell die Layer wechseln und die Simulation gleichmäßiger erscheint, werden zwei Bedingungen gesetzt. Falls $d^k_{i,j} + \delta_{down} > 1$, soll $\delta_{down} = max\left(0, 1 - d^k_{i,j}\right)$ gesetzt werden. Gleichfalls gilt für $g^k_{i,j} + \delta_{up} > 1$ die Umsetzung von $\delta_{down} = max\left(0, 1 - g^k_{i,j}\right)$. Sind δ_{down} und δ_{up} bestimmt, werden die adsorbierten Konzentrationen dem Pigment-deposition Layer hinzugefügt und die desorbierten Konzentrationen abgezogen, sodass $d^k_{i,j} += \delta_{down} - \delta_{up}$ berechnet wird. Das Gleiche gilt vice versa für $g^k_{i,j}$.

simulateCapillaryFlow()

Die Funktion *simulateCapillaryFlow* wurde gemäß der Beschreibung des Capillary layers in Kapitel 2.2.2 mit den Gleichungen (5) bis (8) implementiert. Die Werte der Konstanten für die Absorptionsrate α, sowie die Minima der

Wassersaturation ϵ und δ wurden in der Arbeit von Curtis et al. nicht vorgegeben. Da die Wassersaturation *s* höchstens 0 als niedrigsten Wert haben kann und den Wert der Wasserkapazität *c*, welche auf dem heightField basiert, als höchsten Wert besitzen kann, wurden ϵ und δ versuchsweise auf Werte zwischen 0.001 und 0.9 gesetzt. α, als Wert zur Steigerung der Saturation, wurde ebenfalls auf Werte innerhalb dieser Spanne gesetzt.

3.3.2 Pigmente

Für die Berechnungen des KM wurde eine neue Klasse *pigment* erstellt. Sie erhält die Informationen der Toolbar über die Farbwerte R_w und R_b, gespeicherte R_w und R_b Werte, sowie die Werte der Density, Staining Power und Granularity.

Die Klasse enthält Funktionen auf Basis der Gleichungen (9) bis (12) zur Berechnung des Absorptionskoeffizienten *K*, des Streuungskoeffizienten *S*, sowie den Hilfsvariablen a und b des KM. Sie enthält ebenfalls eine Funktion, die sicherstellt, dass $0 < R_b < R_w < 1$ eingehalten wird. Diese wirkt sich insofern auf das ColorWheel aus, als dass alle Farben, die die Bedingung nicht erfüllen, nicht ausgewählt werden können.

Für jedes Pigment, das in der Toolbar gespeichert wird, wird ein *pigmentLayer* erstellt. Der Layer bekommt dieses Pigment zugeordnet und erzeugt die zunächst leeren Matrizen für dessen Konzentrationen g^k und d^k, die in drawLine und der Fluid Simulation gefüllt werden. Die Pigmente werden einer Liste hinzugefügt, durch die sie und ihre pigmentLayer abgerufen werden können.

calculateTexture()

Die letzte Funktion in updatePaper - *calculateTexture* - ruft für jedes Pigment in der Pigmentliste vier Parameter ab und speichert sie in eine separate Liste *output*. Zum einen werden die Werte *S* , *a* und *b*, des Pigments der Liste hinzugefügt. Zum anderen wird der Liste ein 1-Dimensionales Array hinzugefügt, welches durch die Addition der Matrizen g^k und d^k und das anschließende spaltenweise Strecken in eine 1-Dimensionale Matrix befüllt wird und somit die relative Dichte x^k enthält. Nachdem die Parameter aller Pigmente der Liste output hinzugefügt wurden, wird diese als aktuelle Texturliste *currentTexture* gesetzt.

Die paperentity ruft bei jedem Renderdurchlauf die currentTexture ab und speichert die Werte *S* , *a* und *b* in eine Textur, während die Werte des 1-Dimensionalen Arrays in eine separate Textur *PigmentProperties* gespeichert werden. Daraufhin werden beide Texturen an die Shader weitergereicht.

```
// Material properties
    vec3 T = vec3(1, 1, 1);
    vec3 R = vec3(0, 0, 0);
    for (int i = 0; i < PigmentCount; i++) {
        vec3 a = texture(PigmentProperties, vec2(0.25, i)).rgb;
        vec3 b = texture(PigmentProperties, vec2(0.5, i)).rgb;
        vec3 S = texture(PigmentProperties, vec2(0.75, i)).rgb;

        float x = texture( DiffuseTextureSampler, vec3(texc.xy, i) ).r;
        vec3 c = a * sinh(b*S*x) + b * cosh(b*S*x);
        vec3 R2 = sinh(b*S*x) / c;
        vec3 T2 = b / c;

        vec3 R1 = R;
        vec3 T1 = T;
        R = R1 + (T1*T1*R2)/(vec3(1,1,1) - R1*R2);
        T = (T1 * T2) / (vec3(1,1,1) - R1*R2);
    }
    vec3 R2 = vec3(1,1,1);
    vec3 R1 = R;
    vec3 T1 = T;
    R = R1 + (T1*T1*R2)/(vec3(1,1,1) - R1*R2);

    vec3 MaterialDiffuseColor = R;
    vec3 MaterialAmbientColor = vec3(0.2,0.2,0.2) * MaterialDiffuseColor;
```

Abbildung 16: Ausschnitt des erweiterten Teils des Fragment Shaders

Zum Schluss wurde der Fragment Shader durch die Berechnungen der Gleichungen (13) bis (17) des KM erweitert. Zur Farbbestimmung der Pigmente in der MaterialDiffuseColor wird der Gesamtreflektionsgrad der Gleichung (16) genutzt. Die Umsetzung der Gleichungen im Shader ist in Abbildung 16 zu sehen.

4 Ergebnisse

Die Ergebnisse der Simulation lassen sich auf zwei Arten bewerten. Zum einen wird der optische Vergleich zwischen der realen Aquarellmalerei und der Simulation gezogen. Zum anderen wird anhand der Framerate bestimmt, ob es sich bei der Simulation um ein Echtzeitsystem handelt.

4.1 Optischer Vergleich

Das Ziel war es, die Techniken (b) bis (f) der Abbildungen 2 und 3 aus Kapitel 2.1.2 zu simulieren. Bei Betrachtung der Ergebnisse in Abbildung 17 lässt sich besonders an a) erkennen, dass die Effekte der darkedges und der Granulation erzielt wurden.

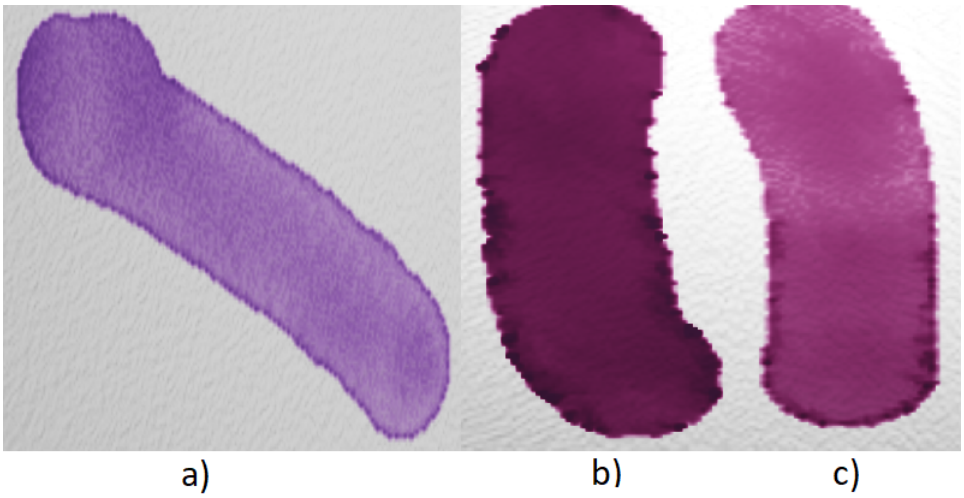
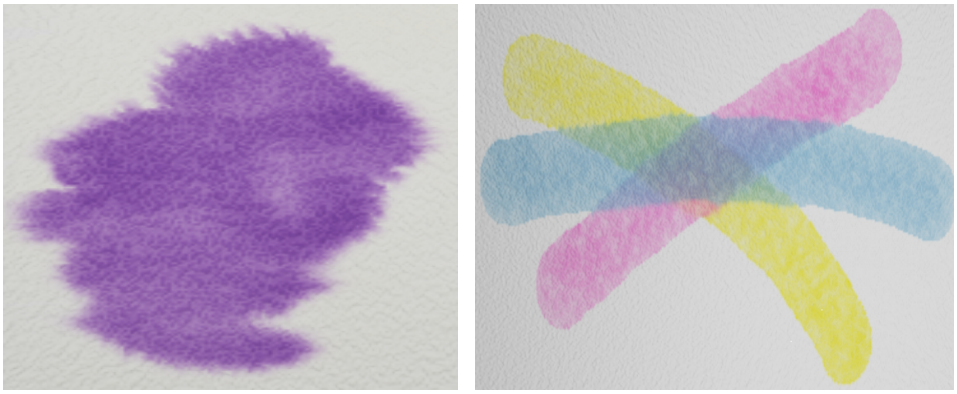


Abbildung 17: Aquarelleffekte der Simulation Teil I

Die Effekte erscheinen je nach Farbwahl, Pinselintensität und Staining Power mehr oder weniger intensiv. Die Striche b) und c) wurden beispielsweise mit demselben Pigment gezeichnet. Die Staining Power von b) war jedoch deutlich höher als die von c). Zu erwähnen ist auch, dass die Intensität der darkedges von der Framerate abhängt. Sinkt die Framerate unter 1 FPS, sind sie kaum zu sehen.

Die Flussmuster und die Lasierung sind ebenfalls erzeugt worden, wie in Abbildung 18a und 18b zu sehen ist. Die Flussmuster breiten sich etwas aus und bilden leichte Verzweigungen, ohne sich dabei über die gesamte Fläche zu verteilen. Die Lasierungen sehen gut aus und auch die Farben der Überlappungen sind vergleichbar mit den Farben aus Abbildung 3 (f). Allerdings bleiben die Linien der Überschneidungen nicht klar, sondern verlaufen nach einiger Zeit wie bei den Flussmustern.



(a) Flussmuster **(b)** Lasierung

Abbildung 18: Aquarelleffekte der Simulation Teil II

Dieser Effekt liegt daran, dass der Algorithmus zwar den Druck p reduziert, jedoch wird die wet-area Maske an den getrockneten Stellen nicht wieder auf $M_{i,j} = false$ gesetzt. Würden diese Werte in einer zusätzlichen Funktion nach einiger Zeit auf $false$ gesetzt oder auch der Dry Button des User Interfaces betätigt, der dies direkt erledigt, entstehen pixelige, dunkle Flecken auf der Linie. Abbildung 19 zeigt diese Trocknung in der Simulation.

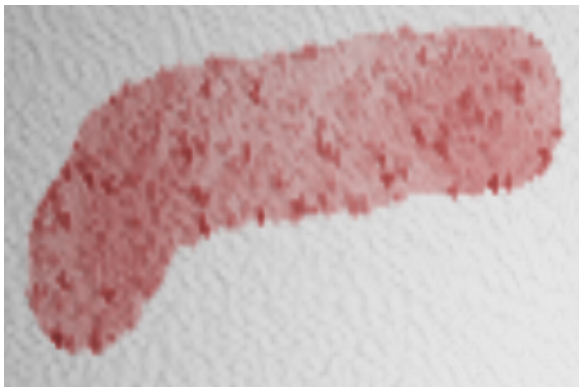


Abbildung 19: Effekt der Trocknung in der Simulation

Die Backruns ließen sich im Gegensatz zu den darkedges und der Granulation, trotz Anpassung der drei Konstanten α, ϵ und δ nicht rekonstruieren. Die Suche nach der Ursache hierfür blieb bislang ohne Erfolg.

Die Abbildungen 20 und 21 zeigen das Motiv einer Teekanne mit dem Logo der Computervisualistik, einmal real und einmal mit der Simulation gezeichnet. Da die Technik des drybrushes nicht in der Simulation enthalten ist, wird der Unterschied durch diesen Effekt an den Rändern ignoriert.

Abbildung 20: Von der Autorin angefertigtes, reales Bild einer Utah-Teekanne mit Logo der Computervisualistik

Abbildung 21: Von der Autorin angefertigtes, simuliertes Bild einer Utah-Teekanne mit Logo der Computervisualistik

Im direkten Vergleich derselben Elemente fällt auf, dass die Simulation nicht trocknet. Am Henkel und Boden der real gezeichneten Teekanne haben die nachträglichen Überlappungen der roten und blauen Farben die Umrisse nur minimal verwischt. In der Simulation hingegen wird diese Verwischung sehr deutlich. Jegliche, leichte Überschneidungen, wie am Ausguss zur gelben Farbe hin, führen auch 10 Minuten nach dem Zeichnen noch zu den Flusseffekten.

Das Papier der realen Zeichnung hat eine sehr prägnante und grobe Struktur, die Struktur des simulierten Blattes ist hingegen etwas feiner. Jedoch kann man ihre Struktur und den Einfluss auf die Bilder bei beiden gut erkennen.

Die Farben des simulierten Bildes erscheinen insgesamt sehr realitätsnah. Das wird an der Farbintensität und dem Farbverlauf der Schattierung am Boden und am Ausguss der simulierten Teekanne deutlich. Diese sehen der anderen Teekanne ziemlich ähnlich. Diese Anpassung der Farben an das Original konnte auch noch nach Vollendung der Zeichnung erfolgen, wodurch der Vergleich besser gelingt.

Wie die Abbildung 22 zeigt, erfüllt die Simulation auch die Anforderungen der drei Farbtypen. Opake Farben sind durchgehend, Interferenzfarben sind nur auf schwarzem Hintergrund und transparente Farben nur auf weißem Hintergrund farbig.

Abbildung 22: Farbtypen in der Simulation - Opak, Interferenz und Transparent

4.2 Performance

In Kapitel 2.4 wurde festgelegt, dass die Simulation mindestens eine Bildaktualisierungsrate von 6 FPS haben sollte, damit sie als Echtzeitsystem eingestuft werden darf. Die Simulation gibt zwei Aktualisierungsraten aus. Zum einen die Aktualisierungsrate des OpenGLWidgets und zum anderen die Rate, welche die Dauer eines Durchlaufs von updatePaper ausgibt.

Beide Raten sind abhängig von der Leistungsfähigkeit des Computersystems. Auf einem 64-Bit-Betriebssystem mit Windows 10, einem AMD Ryzen 5 1600 Six-Core Prozessor mit 3.20 GHz und einer NVIDIA GeForce GTX 970 Grafikkarte, beträgt die Aktualisierungsrate des OpenGLWidgets zwischen 21 und 23 FPS. Das OpenGLWidget ist also in der Lage die Szene oft genug zu rendern, sodass die Simulation als Echtzeitsystem eingestuft werden könnte.

Demgegenüber steht jedoch die Rate für die Durchlaufdauer von updatePaper. Diese hängt nicht nur vom System ab, sondern auch von der Größe des Papiers und somit der Größe der Matrizen. Je größer diese sind, desto mehr Rechenoperationen müssen ausgeführt werden und desto länger braucht die Funktion zur Ausführung. Bei einer Bildgröße von 150x150 Pixeln beträgt die Framerate zwischen 6 und 7.5 FPS. Wird das Bild auf 300x300 Pixel vergrößert, beträgt die Framerate mit 1.5 bis 1.7 FPS bereits deutlich weniger. Bei einer Bildgröße von 500x500 Pixel fällt die Framerate mit 0.5 bis 0.7 FPS sogar unter 1 FPS, womit ein Strich länger als 1 Sekunde braucht, um berechnet und ausgegeben zu werden.

Zum Vergleich wurde das Programm auf einem zweiten, leistungsschwächeren Computersystem getestet. Es handelt sich dabei um ein 64-Bit-Betriebssystem mit Windows 10, einem Intel(R)Core(TM) i5-5200U Dual-Core Prozessor mit 2.20 GHz, einer Intel(R) HD Graphics 5500 und einer zusätzlichen NVIDIA GeForce 920M Grafikkarte. Die Aktualisierungsrate des OpenGLWidgets beträgt hier immernoch zwischen 21 und 23 FPS. Die Durchlaufdauer von updatePaper hat sich hingegen verlangsamt. Die Frameraten belaufen sich für Bilder mit 150x150 Pixeln auf 4.3 bis 5.3 FPS, für Bilder mit 300x300 Pixeln auf 1.1 bis 1.3 FPS und für Bilder mit 500x500 Pixeln auf 0.2 bis 0.3 FPS.

Aufgrund der niedrigeren Durchlaufrate zur Berechnung der Simulation, die zusätzlich noch abhängig von dem Computersystem ist, ist das Programm nur eingeschränkt als Echtzeitsystem einzustufen.

5 Fazit

In dieser Bachelorarbeit wurde ein Zeichensystem mit dem Schwerpunkt auf der Aquarellmalerei entwickelt, welches die Eigenschaften dieses flüssigen Zeichenmittels simuliert. Die Basis für das Nicht-Photorealistische Renderingverfahren des Systems bildete eine Arbeit von Curtis et al. aus den Anfängen der Entwicklung dieser Verfahren.

Es wurden die Grundlagen für die Simulation aufgezeigt und aus ihnen die Bewertungskriterien abgeleitet. Im Zuge der Implementierung wurde ein interaktives System vorgestellt, das das Fluid Verhalten von Wasser und Pigmenten auf einer unebenen Oberfläche simuliert. Zudem ist es, durch Anpassung seines Fragment Shaders auf Basis des Kubelka-Munk Reflectance Models, in der Lage das optische Verhalten übereinanderliegender Farbschichten zu simulieren.

5.1 Ausblick

Das Zeichensystem ist an mehreren Stellen noch ausbaufähig. Es können noch realitätnähere optische Ergebnisse erzielt werden, wenn zum einen weitere Aquarelleffekte wie der drybrush integriert werden würden. Für diese Funktion sollte dann eine User definierbare Höhengrenze eingeführt werden, unter der die Pixel in einem gezeichneten Strich keine Pigmente oder Wasserdruck erhalten und für die die wet-area Maske an diesen Positionen $M = false$ gilt. Zum anderen können die Funktionen für die Backruns und die Trocknung überarbeitet und optimiert werden. Die Linien könnten dann ohne unnatürliche, dunkle Pixel trocknen und die ungewollten Flussmuster verhindert werden.

Die Performance in der Form der Aktualisierungsraten ist ebenfalls noch weiter ausbaubar. Eine Möglichkeit wäre es die Matrizen durch 1-Dimensionale Arrays zu ersetzen, da die Rechenkosten für den Zugriff auf dieses Array deutlich geringer sind als ein Zugriff auf eine Matrix. Desweiteren ist es möglich die Rechenleistung mithilfe von Multithreading auf verschiedene Threads aufzuteilen. Ohne Multithreading und trotz Auslagerung der Berechnung des Kubelka-Munk Verfahrens auf die GPU, ist eine der zwölf CPUs auf diesem Computersystem bei der Ausführung der Simulation zu 100% ausgelastet, während die Rechenkapazitäten der anderen teilweise noch zu 100% frei sind. Durch die Reduktion der Rechenkosten und die Quasi-Parallelisierung durch das Multithreading, könnte die Durchlaufgeschwindigkeit von updatePaper gesteigert werden, sodass auch große Bilder in Echtzeit gemalt und gerendert werden können.

Desweiteren wäre auch die Integration von Optionen für Zeichentabletts interessant. Qt ist in der Lage den Winkel des Stifts und den auf ihn ausgeübten Druck als Variablen zu nutzen. Diese Variablen könnten dann in der Simulation genutzt werden um die Pinselform und -intensität zu beeinflussen. Somit könnte der Nutzer den Stift wie einen echten Pinsel nutzen und die Simulation wirkt etwas wirklichkeitsgetreuer.

Literatur

[1] Adobe Research, NVIDIA and CUDA, Project Wetbrush, *https://blogs.nvidia.com/blog/2016/07/26/adobe-wetbrush*, Zugriff am 05.02.2019

[2] C. J. Curtis, S. E. Anderson, J. E. Seims, K. W. Fleischery, D. H. Salesin, *Computer-Generated Watercolor*, SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer graphics and Interactive Techniques, vol. 16, no. 3, S. 421-430, 1997.

[3] H. Düchting, *Grundlagen der künstlerischen Gestaltung*, Deubner Verlag für Kunst, Theorie und Praxis, 2008.

[4] J. M. W. Turner, Konstanz am Bodensee, 1842 *https://de.wikipedia.org/wiki/William_Turner*, Wikipedia, Zugriff am 05.02.2019

[5] G. Hillmayr, *Handbuch Aquarellmalerei*, Englisch Verlag, 1994.

[6] Ralph Mayer, *The Artist's Handbook of Materials and Techniques*, Penguin Books, 5 edition, 1991.

[7] Ray Smith. The Artist's Handbook. *The Artist's Handbook*, Alfred A. Knopf, 1987.

[8] K. Perlin, *An Image Synthesizer*, Computer Graphics (Proceedings of ACM SIGGRAPH 85), S. 287 - 296, 1985.

[9] N. Foster, D. Metaxas, *Realistic Animation of Liquids*, Graphical Models and Image Processing archive Volume 58 Issue 5, Pages 471-483, 1996.

[10] S. V. Patankar, *Numerical Heat Transfer and Fluid Flow*, Taylor and Francis, Kapitel 6, 1980.

[11] Staggered Grid, *https://www.researchgate.net/figure/Grid-layouts-a-staggered-grid-b-collocated-grid-at-vertex-c-collocated_fig1_260844310*, Zugriff am 05.02.2019

[12] D. Small, *Modeling Watercolor by Simulating Diffusion, Pigment, and Paper Fibers*, Proceedings of SPIE - The International Society for Optical Engineering 1995.

[13] P. Kubelka, F. Munk, *Ein Beitrag zur Optik der Farbanstriche*, Zeitschrift für technische Physik. 12, S. 593–601, 1931.

[14] P. Kubelka, *New contributions to the optics of intensely light-scattering materials. Part I.*, Journal of the Optical Society of America 38, S. 448–457, 1948.

[15] P. Kubelka, *New contributions to the optics of intensely light-scattering materials. Part II. Non-homogeneous layers*, Journal of the Optical Society of America 44, S. 330–334, 1954.

[16] Graphic User Interface FAQ, *https://docs.python.org/3/faq/gui.html*, Zugriff am 08.02.2019

[17] OpenGL FAQ, *https://www.khronos.org/opengl/wiki/FAQWhat_is_OpenGL.3F*, Zugriff am 08.02.2019

[18] Programmierumgebung Qt, *https://www.qt.io/*, Zugriff am 08.02.2019

[19] P. Scholz, *Softwareentwicklung Eingebetteter Systeme*, Physica-Verlag, 2006

[20] T. Akenine-Moller, E. Haines, *Real-Time Rendering*, A K Peters, 2002.

[21] Original Source Code des Color Wheels, *https://gitlab.com/mattia.basaglia/Qt-Color-Widgets*, Zugriff am 08.02.2019

[22] Original Source Code des Perlin Noise, *https://github.com/solprog/Perlin_noise/blob/master/PerlinNoise.cpp*, Zugriff am 08.02.2019

[23] K. Perlin, *Improving Noise*, ACM Transactions on Graphics (TOG) Volume 21 Issue 3, Pages 681-682, 2002.

[24] Original Source Code der Normal Map, *https://github.com/opengl-tutorials/ogl/tree/master/tutorial13_normal_mapping*, Zugriff am 08.02.2019

[25] Der Gradient, *https://www.massmatics.de/merkzettel/!201:Der_Gradient*, Zugriff am 09.02.2019

[26] H. Niemann, *Klassifikation von Mustern*, Springer, 1983.