UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

Stilisierung von Renderings mit Hilfe von handgemalten Vorlagen

Masterarbeit

zur Erlangung des Grades Master of Science (M.Sc.)
im Studiengang Computervisualistik

vorgelegt von

Raphael Heidrich

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergrafik)
Zweitgutachter:	M.Sc. Bastian Krayer (Institut für Computervisualistik, AG Computergrafik)

Koblenz, im Oktober 2017

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.	X	☐

Koblenz, 20.10.2017 RHeidrich

(Ort, Datum) (Unterschrift)

Abstract

One of the greatest goals in computer graphics is the aesthetic representation of objects. In addition to conventional methods, another field focuses on non-photorealistic renderings. The so-called *example-based rendering* is an area where users can transfer their art style to a pre-computed 3D rendering, using a hand-painted template. There are some algorithms that already provide impressive results, but their problem is that most of these procedures count as offline methods and are not able to produce results in real-time. For this reason, this work shows a method that satisfies this condition. In addition, the influence of the run-time reduction on the results is investigated. Requirements are defined, to which the method and its results are examined. Other methods in this field are referenced and compared with their results.

Zusammenfassung

Eins der größten Ziele der Computergrafik ist die ästhetische Darstellung von Objekten. Neben herkömmlichen Verfahren existiert ein weiteres Feld, welches sich mit nicht-photorealistischen Renderings beschäftigt. Das *Example-Based Rendering* ist ein Gebiet, bei dem Benutzer ihren Kunststil, mit Hilfe einer handgemalten Vorlage, auf ein vorberechnetes 3D-Rendering übertragen können. Es existieren einige Algorithmen die bereits beeindruckende Ergebnisse liefern. Das Problem ist, dass die meisten Verfahren aus diesem Bereich zu den Offline-Verfahren zählen und nicht in der Lage sind Ergebnisse in Echtzeit zu produzieren. Aus diesem Grund zeigt diese Arbeit ein Verfahren, dass diese Bedingung erfüllt. Darüber hinaus wird untersucht, welchen Einfluss die Laufzeitminimierung auf die Resultate hat. Es sind Anforderungen definiert, auf die das Verfahren und dessen Ergebnisse überprüft werden. Dabei wird Bezug zu anderen Verfahren aus diesem Gebiet genommen und mit deren Resultaten verglichen.

Inhaltsverzeichnis

1 Einleitung

Seit jeher ist es das Ziel der Computergrafik, Bilder zu generieren. Neben der realistischen und physikalisch-plausiblen Darstellung von Objekten in der traditionellen Computergrafik existiert ein weiteres Feld. Es beschäftigt sich in erster Linie mit der ästhetischen, illustrativen oder künstlerischen Darstellung von Objekten. Das *Non-Photorealistic Rendering* (kurz *NPR*)[34] ermöglicht das Rendern von Bildern, welche eine Vielzahl von Stilen widerspiegelt. Neben der rein künstlerischen Darstellung gibt es Techniken des *NPR*, welche zur Visualisierung in der Medizin[25], der Architektur[21] und noch vielen weiteren Bereichen dienen. Viele Videospiele und Filme leben von einem bestimmten Grafikstil[28], der mit Hilfe von *NPR* erschaffen werden kann. Damit ist man in der Lage, das Augenmerk auf Details zu lenken, welche beim realistischen Rendering untergehen, oder sogar versteckte Strukturen zur Veranschaulichung hervorzuheben. Ein Vor- und Nachteil dieser Techniken ist, dass man darauf verzichten kann komplexe Beleuchtungsszenarien zu berechnen. Dies bedeutet zwar oft eine schnellere Rechenzeit, jedoch gibt es viele Fälle, in denen es das Gegenteil bewirkt. Viele Visualisierungen (beispielsweise die 3D-Darstellung eines CT-Scans[22]) bedürfen eher höherer Laufzeit, da deren Berechnung über die klassischen Verfahren des Echtzeitrenderings hinausgehen. Eine Unterklasse des *NPR* beschäftigt sich damit, herkömmliche Bilder (Fotos oder 3D-Renderings) in einem bestimmten Kunststil nachzubilden. Dabei wird eine Kunstvorlage verwendet, um deren Stil auf das Bild zu übertragen. Das Ziel dieser Techniken ist es, Bilder darzustellen, als wären sie gemalt oder gezeichnet worden. Sie lassen sich unter dem Begriff *Example-Based Rendering* (*EBR*) zusammenfassen.
Diese Arbeit umfasst aktuelle Techniken aus dem Bereich des *EBR*. Es werden verbreitete Verfahren erläutert und es wird auf deren Stärken und Schwächen eingegangen. Des Weiteren werden die Grundlagen erklärt, welche das maßgebliche Vorgehen verständlich machen. Dazu gehören ebenso Konzepte, welche im Hauptteil verwendet werden. Der Hauptteil behandelt die Forschungsfrage dieser Arbeit. Zu untersuchen sind die Ergebnisse eines *EBR*-Algorithmus unter Echtzeitbedingungen. Ebenso werden Maßnahmen zur Laufzeitbeschleunigung sowie deren Auswirkungen auf die visuellen Ergebnisse erläutert. Auf genauere Details zur Implementierung wird anschließend eingegangen. Hier werden Schwierigkeiten und Vorzüge der verwendeten Programmierschnittstellen aufgezählt. Im Anschluss findet man Ergebnisse zum implementierten Verfahren. Diese werden anhand von Anforderungen an das Verfahren untersucht. Dabei wird sowohl auf die Ergebnisse einer naiven Implementierung als auch auf eine optimierte Version eingegangen. Zuletzt wird ein Fazit zu dieser Arbeit, der Thematik sowie dem *State of the Art* aufgeführt.

1.1 Anforderungen

Neben der reinen visuellen Qualität von Bildern, die ein Algorithmus erzeugt, gibt es weitere Faktoren, die eine wichtige Rolle spielen. In dieser Arbeit wird die visuelle Qualität daran gemessen, wie gut die Stilisierung eines Bildes ist. Das bedeutet, je mehr es nach einer Zeichnung oder einem Gemälde aussieht, desto höher ist die visuelle Qualität. Zweifelsfrei gibt es bereits Verfahren, die unter dem Einsatz komplexer Implementierungen beeindruckende Ergebnisse liefern. Bei vielen von ihnen[18][5][11][19][20] handelt es sich um iterative Annäherungen an ein Ergebnis. Eine höhere Zahl an Iterationen bedeutet dabei oftmals eine höhere beziehungsweise längere Laufzeit. Für die Filmindustrie ist eine hohe Laufzeit durchaus zu verkraften, denn die Erstellung der Bilder findet nicht parallel zu deren Darstellung statt und weist eine hohe Qualität auf.
Im Bereich des Echtzeitrenderings gibt es ebenfalls Verfahren[31][25][24], die ein gewisses Maß an Stilisierung aufweisen. Dennoch ist deren Qualität nicht mit den Offline-Verfahren des *EBR* vergleichbar. Ein paar von diesen werden in späteren Kapiteln noch genannt. Die meisten *NPR*- und *EBR*-Verfahren, welche unter Echtzeitbedingungen laufen, können nur einen bestimmten Stil abbilden.
Mit diesen Einschränkungen im Hinterkopf wird nun erneut Bezug auf die Forschungsfrage dieser Arbeit genommen. Es wird untersucht, wie sich ein *EBR*-Verfahren unter Echtzeitbedingungen verhält. Der Hauptteil kommt darauf zurück und geht auf die Problematik ein, die mit einigen Anforderungen einhergeht. Daher werden folgende Anforderungen gestellt, die während der Implementierungsphase beachtet werden.

1. **Echtzeitfähigkeit**
   Eine Stilisierung von Echtzeitrenderings bedeutet in dieser Arbeit, dass die Stilisierung ebenfalls in Echtzeit erfolgt. Dabei wird gemessen, wie lange der Prozess dauert und ob es sich im Rahmen der Echtzeit bewegt.

2. **Visuelle Qualität**
   Unabhängig davon, wie die Vorlage aussieht, kann es zu visuellen Problemen kommen. Bestimmte Artefakte können auftauchen und den Gesamteindruck des Resultats beeinflussen. Die visuelle Qualität beschreibt unter anderem, wie gut das Verfahren in der Lage ist, eine Stilisierung wie gemalt oder gezeichnet aussehen zu lassen. Nebenbei sollen die Ergebnisse an einem verwandten Verfahren gemessen werden.

3. **Beleuchtung**
   Für die plausible Darstellung einer 3D-Szene kann die Benutzung eines Beleuchtungsmodells von Vorteil sein. Korrekte Beleuchtung kann den räumlichen Eindruck verstärken und

die Oberflächenbeschaffenheit eines Objektes verdeutlichen. Andere Verfahren[11] bauen darauf, die Beleuchtung einer Szene mit einzubeziehen und damit die Stilisierung zu unterstützen. Dabei wird Wert auf die korrekte Darstellung von Beleuchtungseffekten wie direktes Licht, Glanzpunkte und Schatten gelegt.

4. **Anpassungsfähigkeit**
   Bei einem Stilisierungsverfahren führt jede Vorlage zu einem anderen Ergebnis. Dabei können für unterschiedliche Vorlagen auch unterschiedliche Effekte im Resultat auftauchen[18]. Es ist zu untersuchen, wie sich die Struktur des Ergebnisses zur Struktur der Vorlage verhält.

Hierbei wird versucht ein Kompromiss zu finden, welcher alle Anforderungen zu einem Mindestmaß erfüllt. Natürlich kann es dazu führen, dass sich diese anti-proportional zueinander verhalten. Es ergibt beispielsweise Sinn, dass niedrigere Laufzeiten zu Einschnitten in der visuellen Qualität führen können. Außerdem ist zu beachten, dass diese Anforderungen nach Relevanz bezüglich der Forschungsfrage sortiert sind. Eine echtzeitfähige Implementierung ist in diesem Fall das Maß aller Dinge. Die Rate in der Bilder auf dem Bildschirm angezeigt werden wird in Frames pro Sekunde (fps) gemessen. Bei $1 - 15fps$ kann man bereits von Interaktivität[2] sprechen, allerdings ist das Ziel dieser Arbeit Echtzeit. Somit wird hier alles was schneller als $15fps$ ist als Echtzeit bezeichnet.

2 Verwandte Arbeiten

Wie bereits erwähnt, gibt es eine Vielzahl an Stilisierungsverfahren, sowohl im Offline- als auch im Echtzeitrendering. Hier werden einige von beiden Bereichen vorgestellt. Darüber hinaus gibt es allerdings noch verwandte Gebiete, die indirekt damit zu tun haben. Diese unterscheiden sich im Anwendungszweck, auch wenn sie ähnliche Techniken verwenden.

2.1 *Lit Sphere*

Ein sehr etabliertes Verfahren wurde von *Sloan et al.*[31] vorgestellt. Die sogenannte *Lit Sphere* ist ein *EBR*-Algorithmus, der sowohl sehr leichtgewichtig als auch anpassungsfähig bezüglich der Stilvorlagen ist. Ein gerendertes Modell wird nicht direkt beleuchtet, sondern über eine Stilvorlage texturiert. Diese Vorlage stellt eine beleuchtete Kugel dar, wodurch sich auch der Name erklären lässt. Diese Kugel kann gezeichnet, gemalt, fotografiert oder gerendert sein. Betrachtet man einen Punkt auf der Oberfläche des Modells, kann man anhand der Normale einen korrespondierenden Punkt auf der Kugeloberfläche finden, welcher die gleiche Normale hat. Dieser

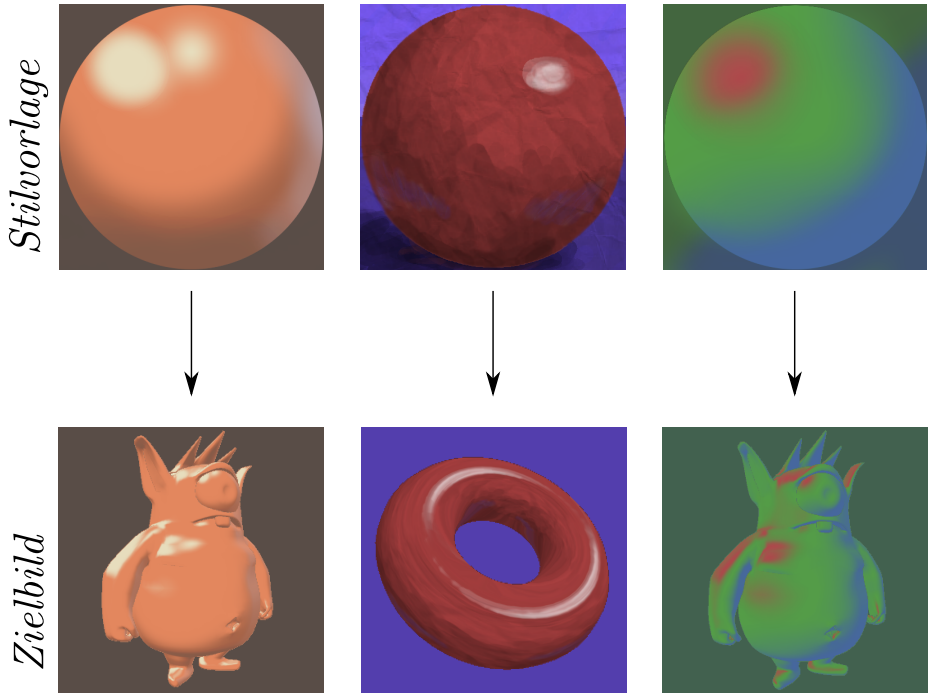


Abbildung 1: Beispiele für die Stilisierung mit Hilfe des *Lit Sphere*-Verfahrens. Dabei wird die gemalte Vorlage auf ein Vordergrund Objekt übertragen. *Eigene Darstellung auf Grundlage von Sloan et al.[31]*

Punkt wird von der Kugel auf das Modell projiziert und somit texturiert. Inzwischen findet man dieses Verfahren in vielerlei Software zur leichtgewichtigen Visualisierung feiner Oberflächenstrukturen. Die meisten *3D-Sculpting*-Programme verwenden diese Technik unter dem Namen *Matcap*[32]. Obwohl dieses Verfahren die Anforderungen der Echtzeitfähigkeit und Anpassungsfähigkeit zu hohem Maß erfüllt, kann es die anderen beiden Anforderungen im Umkehrschluss nicht erfüllen. Obwohl es sich bei *Lit Sphere* um eine beleuchtete Kugel handelt, fehlt dem Verfahren ein elementarer Bestandteil der globalen Beleuchtung: der Schatten. Es handelt sich um ein Texturierungsverfahren und nimmt somit keine Rücksicht auf die gesamte Szene. Es findet keine Interaktion zwischen Licht und den Oberflächen statt. Es findet ebenso keine Interaktion mit einer Lichtquelle statt, denn die Beleuchtung ist quasi auf die Kugeloberfläche gemalt. Damit ist die Beleuchtung unabhängig vom Betrachter und somit höchst unplausibel. Bezüglich der visuellen Qualität kritisieren *Fiser et al.*[11], dass aufgrund der Texturierung pro Pixel der flächige Eindruck der Vorlage verloren geht. Dies sei besonders auffällig bei hochfrequenten Modellen in Kombination mit groben Strukturen in der Vorlage.

2.2 Textursynthese

Man kann schlecht von *EBR* reden, ohne einen Bezug zur Textursynthese zu nehmen. Diese beiden Bereiche sind stark verknüpft und es gibt viele Techniken, die in beiden Bereichen Verwendung finden. Fortschrittliche Verfahren von *Jamriska et al.*[19], *Kaspar et al.*[20] oder *Kwatra et al.*[23] versuchen die Struktur einer Vorlage auf ein Bild zu projizieren, welches um ein Vielfaches größer ist. Dabei werden Muster und Bereiche aus der Vorlage wiederholt und abgewandelt, sodass das resultierende Bild aussieht, als wäre die Vorlage nur ein Bildausschnitt davon. Andere Techniken wie die von *Darabi et al.*[7], *Barnes et al.*[4] oder *Wexler et al.*[33] sind daran interessiert, fehlende Bereiche mit Hilfe des restlichen Bildes zu füllen. Dabei wird ein defektes Bild in definierte und nicht-definierte Bereiche segmentiert. Die nicht-definierten Bereiche stellen das Zielbild dar, während die definierten Bereiche die Vorlage bilden.

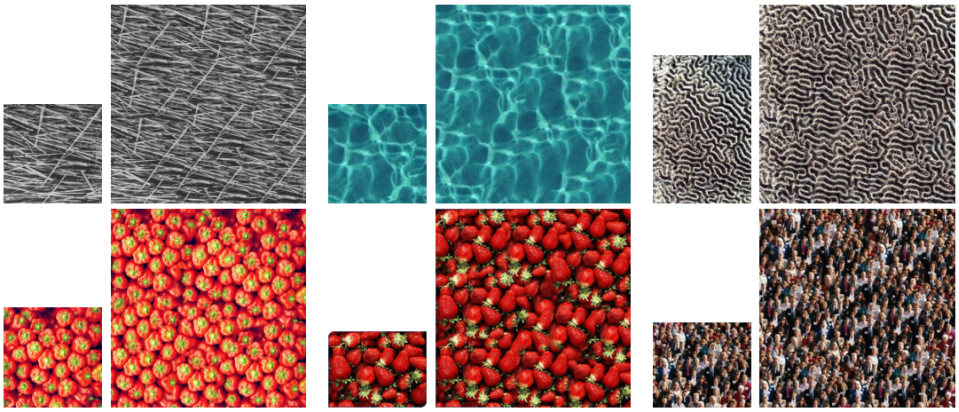


Abbildung 2: Die Ergebnisse von *Kwatra et al.*[23]. Hier wird eine Textur auf ein größeres Bild übertragen. Dabei wird die Struktur der Textur erweitert und fortgeführt. *Quelle: Kwatra et al.[23]*

Jedes dieser Verfahren hat folgendes gemeinsam: Sie alle generieren Bereiche im Zielbild auf Basis gegebener Information und Struktur der Vorlage. Dabei wird allerdings keinerlei Rücksicht auf räumliche Eigenschaften oder deren Beleuchtung genommen. *Barnes et al.*[4] sind in der Lage, perspektivische Nebenbedingung einzuhalten und *Diamanti et al.*[9] kann Bereiche bezüglich ihrer Beleuchtung zuordnen, jedoch werden dafür Benutzereingaben benötigt. Diese sind in einem Echtzeitszenario nicht praktikabel.

2.3 *Neural Algorithm of Artistic Style*

Vor nicht allzu langer Zeit verfolgten *Gatys et al.*[15] den Ansatz gewöhnliche Bilder durch den Einsatz eines *Deep Neural Network*[14] mit abstrakten Kunstwerken zu kombinieren. Dieses *Deep Neural Network* ist auf Objekterkennung trainiert. Es stellt eine Abbildung von Merkmalen in der Vorlage auf jene in einem Zielbild dar.
Auf einer hauseigenen Web-Plattform[35] lassen sich einige Beispiele für die visuelle Qualität dieses Verfahrens finden. Landschaftsmotive und Portraits können dort mit einer beliebigen Vorlage stilisiert werden. Trotz der beachtlichen Ergebnisse sollte jedoch erwähnt werden, dass sich die Trainingsdaten des *Deep Neural Networks* auf Fotos beschränken. Das bedeutet wiederum für 3D-Renderings, dass sie ein Maß an Fotorealismus erfüllen müssen, um stilisiert zu werden. Tatsächlich kommt es bei abstrakten Szenen tendenziell eher zu Artefakten (siehe Abbildung 4), da diese dem Verfahren nicht natürlich erscheinen[11]. Damit ist gemeint, dass 3D-Renderings Strukturen und Farbkombinationen aufweisen können, die in der Natur nicht vorkommen. Daraus folgt, dass das *Deep Neural Network* keine befriedigenden Ergebnisse liefert, weil es nicht darauf trainiert

Abbildung 3: Die Ergebnisse von *Gatys et al.*[15]. Mit Hilfe der einzelnen Stilvorlagen, wird ein Stil auf das Referenzbild (oben links) übertragen. *Quelle: Gatys et al.*[15]

wurde 3D-Renderings zu verarbeiten.

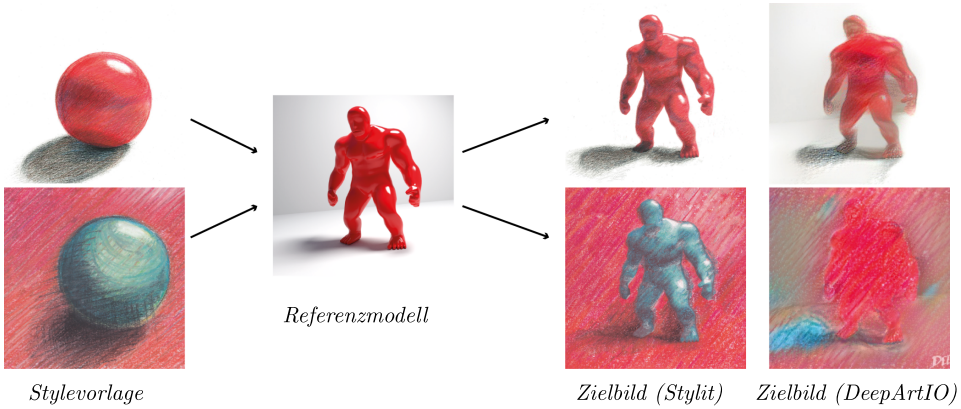


Abbildung 4: *Fiser et al.*[11] vergleichen ihr Verfahren mit dem von *Gatys et al.*[15]. *Eigene Darstellung auf Grundlage von Fiser et al.[11].*

2.4 *Stylit*

Die Hauptinspiration für diese Arbeit ist ein Verfahren von *Fiser et al.*[11] namens *Stylit*. Dabei handelt es sich um eine Anwendung, die mit Hilfe von 3D-Renderings und handgemalten Vorlagen eine Vielzahl an stilisierten Bildern erstellen kann. Um gemalte Bilder korrekt auf die Oberfläche einer beliebigen Geometrie zu übertragen, ist es notwendig, die Beleuchtungssituation des Renderings als Referenz einzubeziehen. Daher verwenden sie neben dem gerenderten Bild noch Farb-*Buffer*, welche bestimmte Beleuchtungseffekte beinhalten. Dies dient als Inspiration für die dritte Anforderung dieser Arbeit (siehe Abschnitt 1.1).
Trotz befriedigender Ergebnisse und der daraus entstandenen Inspirationen ist das Vorgehen von *Stylit* ungeeignet für diese Arbeit. Aufgrund einer Technik namens *Uniform Source Patch Usage* wächst die Anzahl der Iterationen bei *Stylit* um ein Vielfaches. Hierdurch kommt es zu enorm hohen Laufzeiten und ist damit ungeeignet für diese Arbeit. Darüber hinaus ist diese Technik mit einer hohen Zahl von *Memory Barriers* verbunden. Diese erschweren eine parallele Verarbeitung von Daten und wirken sich somit ebenfalls auf die Laufzeit aus. Die gleiche Problematik findet sich auch bei *Jamriska et al.*[19] und *Kaspar et al.*[20].

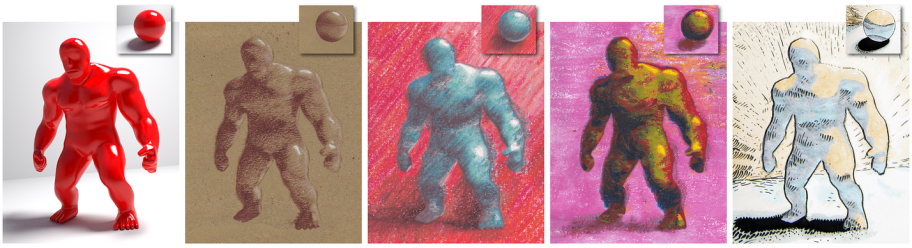


Abbildung 5: Die Ergebnisse von *Fiser et al.*[11]. Ihr Verfahren (*Stylit*) stilisiert ein Modell mit der jeweiligen Stilvorlage, welche oben rechts an jedem Bild ist. *Quelle: Fiser et al.*[11]

In diesem Abschnitt wurde eine Reihe von verwandten Themen und Algorithmen aufgezählt. Diese sind nicht unwesentlich für diese Arbeit, allerdings lassen sich die hier verwendeten Techniken klar von den anderen trennen. Ziel dieser Aufzählungen ist zunächst eine Übersicht zum Stand der Technik und darüber hinaus ein Vergleich bezüglich der definierten Anforderungen. Als nächstes werden die Themen erläutert, die die größte Bedeutung für die Forschungsfrage haben.

3 Grundlagen

In diesem Kapitel werden die Grundlagen erklärt. Zunächst wird das allgemeine Vorgehen erläutert. Außerdem werden einige Algorithmen und Datenstrukturen aufgezählt, die sich großer Beliebtheit bei der Implementierung von *EBR*-Algorithmen erfreuen. Der Umfang und die Vielfalt in diesem Gebiet sind bemerkenswert, allerdings können hier nur die wichtigsten Details genannt werden, welche für diese Arbeit relevant sind.

3.1 Workflow

Das Gebiet des *EBR* findet viele Anwendungsmöglichkeiten. In dieser Arbeit wird allerdings ein bestimmter *Workflow* bevorzugt. Künstler können mit klassischen Werkzeugen der Kunst ausdrucksstarke Bilder erzeugen, welche als Standbild taugen, jedoch ungeeignet für Animationen sind. In anderen Fällen möchte man die Ästhetik eines Kunststils an unterschiedlichen Objekten austesten oder deren Blickwinkel verändern. Dieser Arbeitsablauf kann stark vereinfacht werden, indem man ein bereits gemaltes Bild dazu verwendet, Folgebilder zu generieren. Ein von Hand angefertigtes Bild wird im Folgenden als Stilvorlage und ein generiertes Folgebild als Zielbild bezeichnet.

Das allgemeine Vorgehen beim *EBR* sieht folgendermaßen aus: In der ersten Phase wird die Stilvorlage erstellt. Dessen Motiv ist nicht willkürlich, sondern folgt einer bereits existierenden Vorlage. Jene kann ein fotografiertes oder ein gerendertes Bild sein. Es sollte erkennbar sein, dass die Stilvorlage eine stilisierte Version davon ist. Dieses Bild wird Modellvorlage genannt. Im nächsten Schritt wird ein Bild festgelegt, von dem man eine stilisierte Version erzeugen möchte. Dieses wird Referenzmodell genannt.
Dieses Vorgehen und diese Namen beschreiben ein Modell, welches in späteren Abschnitten genauer beleuchtet wird. Dem Verständnis und der Lesbarkeit zuliebe, werden für die vier genannten Bilder Abkürzungen eingeführt. Modellvorlage wird als A, Stilvorlage als A', Referenzmodell als B und das Zielbild als B' bezeichnet.

3.2 Image Analogies

Viele Arbeiten, die sich mit dem Thema *EBR* beschäftigen[5][10][11], liegen einem Verfahren namens *Image Analogies*[18] zugrunde. Auf Basis eines Eingabebildes wird ein Ausgabebild erzeugt, welches dessen grobe Struktur beibehält. Ergänzend zu der einfachen Synthese über eine einzelne Eingabe, verwenden *Image Analogies* drei Eingabebilder (siehe Abbildung 6). Der Name dieser Technik beschreibt, dass sich die Farbwerte zwischen B' und B analog zur Abbildung von A nach A' verhalten. Mit anderen Worten sucht man ein B' welches sich zu B verhält, wie A zu A'. Während diese Formulierung plausibel erscheint, ist dessen Lösung nicht trivial zu beantworten.
Eine Grundvoraussetzung für die Problemformulierung von *Image Analogies* ist eine Ähnlichkeitsdefinition. Diese wird als Maß verwendet, mit dem der Zusammenhang zwischen sowohl jeweiligem Modell und seiner stilisierten Form als auch Input und Output gemessen wird. Diese Aufgabe erweist sich als nicht unkompliziert, denn es muss lokal nach Korrespondenzen gesucht werden, während die globale Struktur erhalten bleibt.

3.2.1 Patches

Zunächst wird ein lokaler Punkt $\boldsymbol{p} \in S_I$ auf einem Bild $I \in \{A, A', B, B'\}$ betrachtet. Dabei gilt, dass $S_I =: \{\begin{pmatrix} p_1 \\ p_2 \end{pmatrix} \in \mathbb{N}^2 \mid p_1 < b_I \wedge p_2 < h_I\}$, wobei b_I und h_I die Breite und Höhe von I sind. Dieser Bildpunkt $\boldsymbol{p}$ ist in der konkreten Anwendung auch als Pixelposition zu bezeichnen. Die Funktion $I(\boldsymbol{p})$ entspricht der Abbildung eines Bildpunktes auf einen Farbwert, sodass $I : S_I \mapsto F_I$. Der Farbwert $I(\boldsymbol{p})$ kommt aus $F_I \subset \mathbb{R}^n$, wobei F_I der Farbraum von I ist. $I^N(\boldsymbol{p})$ bezeichnet einen erweiterten Farbwert, beziehungsweise einen Merkmalsvektor, in Bild I an der Stelle $\boldsymbol{p}$.

Abbildung 6: Der Aufbau von *Image Analogies*. Bei gegebenen Input A und A' und einem neuen Output B wird ein Zielbild B' generiert. *Quelle: Benard et al. 2013[5].*

Die Menge der benachbarten Bildpunkte $N_{\boldsymbol{p}}$ wird Patch genannt. Dabei handelt es sich um ein quadratisches Fenster, welches um einen Pixel aufgespannt wird. Ein Patch hat folgende Form:

$$N_{\boldsymbol{p}} = \{\boldsymbol{q} \in \mathbb{N}^2 \mid \boldsymbol{q} = \boldsymbol{p} + \boldsymbol{r}, \boldsymbol{r} \in [-\lfloor \frac{w}{2} \rfloor, \lfloor \frac{w}{2} \rfloor]^2\} \tag{1}$$

wobei w die Höhe und Breite eines Patches ist. Anders gesagt handelt es sich bei einem Patch um einen $w \times w$ Bereich um einen Punkt $\boldsymbol{p}$.
Der Merkmalsvektor selbst ist keine rein lokale Eigenschaft. Gegenüber der traditioneller Notationen der Bildverarbeitung besteht $I^N(\boldsymbol{p})$ aus der Konkatenation der benachbarten Farbwerte.

$$I^N(\boldsymbol{p}) = \begin{pmatrix} I(\boldsymbol{q}_0) \\ \vdots \\ I(\boldsymbol{q}_j) \\ \vdots \\ I(\boldsymbol{q}_n) \end{pmatrix} \tag{2}$$

wobei $\boldsymbol{q}_j \in N_{\boldsymbol{p}}$ und $n = |N_{\boldsymbol{p}}|$. So wird ein Vektor gebildet, der jeden Farbwert innerhalb eines quadratischen Fensters aneinanderkettet.
Bezüglich des Merkmalsraum heißt dies, dass $I^N : S_I \mapsto \mathbb{R}^{|N_{\boldsymbol{p}}| \cdot dim(F_I)}$, wobei $|\cdot|$ der Kardinalität und *dim* der Dimension einer Menge entspricht. Im Szenario der *Image Analogies* haben A und A', beziehungsweise B und B' die gleiche Größe. Daher gilt für eine Vorlage $I \in \{A, B\}$ und seine stilisierte Version $I' \in \{A', B'\}$, dass $\forall \boldsymbol{p} \in \mathbb{N}^2, \boldsymbol{p} \in S_I \Leftrightarrow \boldsymbol{p} \in S_{I'}$. Daher werden $\boldsymbol{p} \in S_I$ und $\boldsymbol{p} \in S_{I'}$ in dieser Arbeit synonym verwendet.

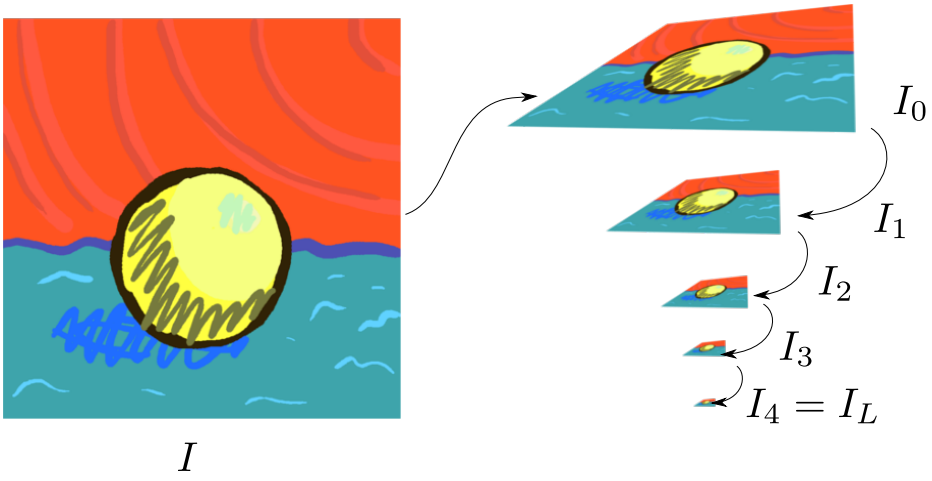


Abbildung 7: Die Generierung einer Bildpyramide. Mit steigendem Index 0 bis L sinkt dir Auflösung der Ebenen.

3.2.2 Bildpyramide

Image Analogies ist ein relativ rechenaufwändiger Algorithmus, der mehrere Iterationen durchläuft, bevor er zu einem angemessenen Ergebnis führt. Zur Beschleunigung wird eine Datenstruktur namens Bildpyramide eingeführt (siehe Abbildung 7). Die Ebene I_0 beschreibt die oberste und I_L die unterste Stufe einer Bildpyramide. Jedes I_l mit $0 \leq l \leq L$ ist ebenso wie I ein Bild und $I_l(\boldsymbol{p})$ beschreibt eine Abbildung $I_l : S_{I_l} \mapsto F_I$. Man beachte, dass I_l und I einige Eigenschaften teilen. Sowohl I als auch I_0 haben die gleiche Auflösung. Darüber hinaus ist jeder Farbwert von I_l, also in dem gleichen Farbraum wie von I: $F_I = F_{I_l}, 0 \leq l \leq L$. Für jedes I_l mit $l \in \{1, ..., L\}$ gilt, dass dessen Auslösung halb so groß ist wie die von I_{l-1}.
Nach der Initialisierung einer Bildpyramide wird jedes I_l mit Farbwerten befüllt. Angefangen wird mit I_0. Danach können die Farbwerte von I_l iterativ nach unten propagiert werden. Diesen Vogang nennt man *Downsampling*. Dabei wird der Farbwert aus dem Mittelwert der 3×3 Nachbarschaft des Bildpunktes $\boldsymbol{p}_{l-1} = \boldsymbol{p}_l \cdot 2$ berechnet.

$$I_l(\boldsymbol{p}) \leftarrow \frac{1}{9} \sum_{i=-1}^{1} \sum_{j=-1}^{1} I_{l-1}(\boldsymbol{p}_{l-1} + \begin{pmatrix} i \\ j \end{pmatrix}) \tag{3}$$

Analog zu Gleichung 2 gilt für die Verkettung von Farbwerten zu einem Merkmalsvektor auf einer Pyramidenebene I_l folgendes:

$$I_l^N(\boldsymbol{p}) = \begin{pmatrix} I_l(\boldsymbol{q}_0) \\ \vdots \\ I_l(\boldsymbol{q}_j) \\ \vdots \\ I_l(\boldsymbol{q}_n) \end{pmatrix} \tag{4}$$

3.2.3 Algorithmus

Mit der gegebenen Notation lässt sich der Algorithmus nun beschreiben. Der erste Schritt umfasst eine Initialisierungsphase. Für die Eingabebilder A, A' und B wird eine Bildpyramide berechnet, zusammen mit deren Merkmalsvektoren und gegebenenfalls nötigen Datenstrukturen zur Nachbarschaftssuche. *Hertzmann et al.*[18] verwenden eine *Approximate-Nearest-Neighbor*-Suche (*ANN*)[3] für ihren Ansatz. Die Synthese erfolgt von der geringsten Auflösung der Bildpyramide und geht von grob nach fein. Auf jedem Level wird eine Version von B'_l erzeugt, welche ihre Farbwerte durch *Upsampling* an B'_{l-1} weitergibt.
Der *Image Analogies*-Algorithmus kann durch folgenden Pseudocode veranschaulicht werden.

Algorithm 1 Das Grundgerüst des *Image Analogies*-Algorithmus

```
1:  function ImageAnalogies(A, B, A', L)
2:      Generate Image Pyramid for A, A', B, B' and s
3:      Fill Pyramid of A, A' and B with respective values
4:      Initialize Pyramid of B' and s randomly
5:      for each l ∈ {0, ..., L} in reverse order do
6:          for each pixel q ∈ S_{B'_l}, in scanline order do
7:              p ← BestMatch(A_l, B_l, A'_l, B'_l, s, l, q)
8:              B'_l(q) ← A'_l(p)
9:              s_l(q) ← p
10:         end for
11:         B'_{l-1} ← UPSAMPLE(B'_l)
12:     end for
13:     B' ← B'_0 return B'
14: end function
```

Das Kernstück dieses Algorithmus ist die BestMatch-Funktion. Sie nimmt alle drei Eingabebilder und das teilweise synthetisierte Bild B'_l, die vorherige Korrespondenzposition s_l und die Zielposition $\boldsymbol{q}$ entgegen, und findet dafür die beste Position. Im Anschluss wird B'_l mit Hilfe dieser Korrespondenzen auf Basis von A'_l neu synthetisiert.
Die BestMatch-Funktion sucht dabei nach zweierlei Faktoren. Zunächst

wird mit Hilfe einer *Nearest-Neighbor*-Suche (im Fall von *Hertzmann et al.*[18] ist dies der *ANN*-Algorithmus) eine Korrespondenz über die Merkmalsvektoren I^N gefunden. Im Anschluss wird die nächstbeste Korrespondenz in Bezug auf s gesucht. Es dient der räumlichen Kohärenz, Patches zu finden, welche in allen Bildern räumlich nah beieinander sind.

Algorithm 2 BESTMATCH

function BESTCOHERENCEMATCH($A, B, A', B', s, \boldsymbol{q}$)
  $\boldsymbol{r}^* \leftarrow \arg\min_{\forall \boldsymbol{r} \in N_q} \textit{distance}(A, B, A', B', s(\boldsymbol{r}) + (\boldsymbol{q} - \boldsymbol{r}), \boldsymbol{q})$
return $s(\boldsymbol{r}^*) + (\boldsymbol{q} - \boldsymbol{r}^*)$
end function
function BESTMATCH($A, B, A', B', s, l, \boldsymbol{q}$)
  $\boldsymbol{p}_{appr} \leftarrow$ BESTAPROXIMATEMATCH$(A, B, A', B', \boldsymbol{q})$
  $\boldsymbol{p}_{cohe} \leftarrow$ BESTCOHERENCEMATCH$(A, B, A', B', s, \boldsymbol{q})$
  $d_{appr} \leftarrow \textit{distance}(A, B, A', B', \boldsymbol{p}_{appr}, \boldsymbol{q})$
  $d_{cohe} \leftarrow \textit{distance}(A, B, A', B', \boldsymbol{p}_{cohe}, \boldsymbol{q})$
  if $d_{cohe} < d_{appr}(1 + 2^{-l}\kappa)$ **then**
    return $\boldsymbol{p}_{cohe}$
  else
    return $\boldsymbol{p}_{appr}$
  end if
end function

Die BESTMATCH-Funktion lässt sich dann durch den Algorithmus 2 ausdrücken. Hier wird die Ähnlichkeit mit Hilfe der *distance*-Funktion ermittelt. Dabei werden die Merkmalsvektoren der In- und Outputbilder an der Stelle $\boldsymbol{p}$ und $\boldsymbol{q}$ verglichen. Sie geht gegen 0 für korrespondierende Patches, die eine hohe Ähnlichkeit aufweisen.

$$\textit{distance}(A, B, A', B', \boldsymbol{p}, \boldsymbol{q}) = \|A^N(\boldsymbol{p}) - B^N(\boldsymbol{q})\|_2 + \|A'^N(\boldsymbol{p}) - B'^N(\boldsymbol{q})\|_2 \quad (5)$$

Die Funktion BESTCOHERENCEMATCH berechnet eine Korrespondenz, auf Basis zuvor berechneter Korrespondenzen innerhalb der Nachbarschaft. Sie dient der Annahme, dass benachbarte Patches an ähnlichen Stellen zu suchen sind. Wie zuvor erwähnt, findet BESTAPROXIMATEMATCH seine Korrespondenzen mit Hilfe von *ANN*. Damit wird eine beinahe-optimale globale Korrespondenz gefunden. Auf genauere Details wird hier allerdings verzichtet, da diese Arbeit ein anderes *Nearest Neighbor*-Verfahren verwendet. Zuletzt wird verglichen, welche Korrespondenz sich besser eignet. Da BESTAPROXIMATEMATCH ein globales Minimum erreichen kann, wird d_{appr} mit einem Wert κ gewichtet, um es künstlich zu vergrößern. Je höher κ ist, desto eher wird räumliche Kohärenz gegenüber einem globalem Minimum bevorzugt. Außerdem wird es zusätzlich mit 2^{-l} gewichtet, um die wachsende Distanz bei höherer Auflösung

auszugleichen.

Der *Image Analogies*-Algorithmus, der hier vorgestellt wurde, ist die Basis dieser Arbeit in Bezug auf *EBR*. Er verdeutlicht das grobe Vorgehen und die zugrundeliegenden Datenstrukturen. Dennoch ist diese Technik in dieser Form nicht geeignet für zufriedenstellende Ergebnisse und wird daher in späteren Kapiteln weiter ergänzt und ersetzt.

3.3 PatchMatch

Da viele *EBR*-Techniken darauf basieren, Pixel- oder Patchkorrespondenzen zu finden, ist eine gute *Nearest-Neighbor*-Suche im Merkmalsraum von elementarer Bedeutung. Dabei stehen zwei Probleme im Vordergrund. Zunächst spielt die Größe des Merkmalsraums eine entscheidende Rolle. Wie zuvor erwähnt, wird in dieser Arbeit die Konkatenation von Farbwerten in einem Patch I^N verwendet. Außerdem ist die Anzahl der Daten von hoher Bedeutung. Hier wird die Menge an Bildpunkten als Daten verwendet. Die Höhe dieser Faktoren kann sich stark auf die Laufzeit auswirken. Besonders bei der Verarbeitung jedes Bildpunktes in einem Bild kann es zu einer enormen Menge an Daten kommen. Da eine Korrespondenzfindung oft mit quadratischem Aufwand verbunden ist, wird diese Tatsache noch problematischer. In den meisten Fällen ist eine niedrige Laufzeit erstrebenswert, was ein beschleunigtes Verfahren besonders attraktiv macht.

3.3.1 *Nearest-Neighbor*-Suche

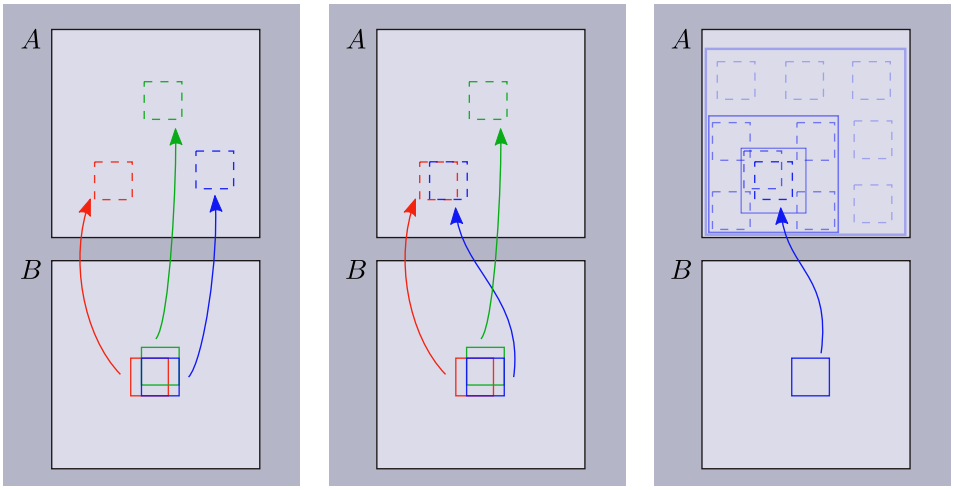
Man betrachte die Funktion BESTMATCH der *Image Analogies* und geht davon aus, dass für jedes $\boldsymbol{q} \in S_B$ ein $\boldsymbol{p} \in S_A$ gefunden werden muss. Der primitive Ansatz wäre, jedes $\boldsymbol{q}$ mit jedem $\boldsymbol{p}$ zu vergleichen und dabei das beste Ergebnis zwischenzuspeichern. Bei Bildern mit einer hohen Auflösung von beispielsweise $1000\text{px} \times 1000\text{px}$ resultiert dieses Vorgehen in mindestens eine Billionen Vergleichsoperationen. Um dieses *Brute-force*-Vorgehen zu vereinfachen, werden Algorithmen wie *ANN*[3] oder *kd-Tree*[6] verwendet, bei denen die Daten zunächst in einer Graphenstruktur vorsortiert werden, um die Suche zu vereinfachen.
Im Jahr 2010 entwickelten *Barnes et al.*[4] für den Bereich der patchbasierten Korrespondenzfindung den sogenannten *PatchMatch*-Algorithmus. Das ursprüngliche Ziel davon ist es, einen fehlenden Bildbereich mit Patches zu füllen, die eine hohe Ähnlichkeit zu den angrenzenden Bereichen haben. Andere aus den Bereichen der Textursynthese[19][20] und des *EBR*[11][5] haben diese Technik für sich entdeckt, um Patches über mehrere Bilder hinweg zu vergleichen.

3.3.2 *Nearest-Neighbor Field*

Wie bereits erwähnt, wird eine Abbildung von Inputbildern A und A' nach B und B' gesucht. Bei einer genaueren Betrachtung von Algorithmus 1 sieht man, dass jede gefundene Korrespondenz in s gespeichert wird. Somit ist s eine temporäre Abbildung von Bildpunkten über die Bilder hinweg. Gewissermaßen ist $s : S_B \mapsto S_A$, also eine Abbildung von den Bildpunkten in B (beziehungsweise B') zu einem Punkt in A (beziehungsweise A'). *Barnes et al.* nennen diese Abbildung *Nearest-Neighbor Field* (kurz *NNF*).

3.3.3 Algorithmus

Bei *PatchMatch* handelt es sich um einen *Nearest-Neighbor* Algorithmus, der iterativ ein *NNF* berechnet, welches bezüglich deren Patches ein optimales Mapping zwischen zwei[1] Bildern darstellt. Jede Iteration folgt dieser Reihenfolge: Initialisierung, *Propagation* und *Random Search*. Seien jene Schritte kodiert als $\mathcal{I}$, $\mathcal{P}$ und $\mathcal{S}$, dann werden sie in folgender Reihenfolge ausgeführt: $\mathcal{I}, \mathcal{P}_0, \mathcal{S}_0, ..., \mathcal{P}_i, \mathcal{S}_i, ..., \mathcal{P}_n, \mathcal{S}_n$. Dabei entspricht i dem Index eines Patches.



(a) Initialisierung *(b) Propagation* *(c) Random Search*

Abbildung 8: Die drei Phasen der Zufallssuche des *PatchMatch*: *(a)* Patches werden initial zugewiesen; *(b)* der blaue Patch sucht oben (grün) und links (rot), ob diese die Abbildung von Blau verbessern; *(c)* der Patch sucht zufällig nach Verbesserung konzentrisch um die Nachbarschaft herum. *Eigene Darstellung auf Grundlage von Barnes et al.[4]*

Der erste Schritt des Algorithmus ist die Initialisierungsphase. Hier

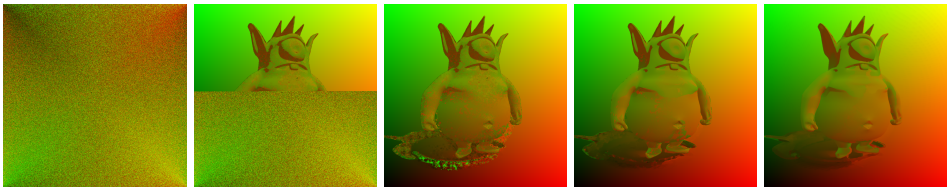
[1] In dieser Arbeit handelt es sich zwar um vier Bilder, jedoch wird hier erst einmal die Notation von *PatchMatch* erklärt, welche auf zwei Bildern arbeitet.

wird das *NNF* an jedem Bildpunkt $\boldsymbol{p}$ (ab jetzt *NNF*($\boldsymbol{p}$) anstatt $s(\boldsymbol{p})$) mit zufälligen Bildpunkten aus S_A gefüllt. Handelt es sich nicht um die erste Iteration und es liegen bereits Initialwerte vor, werden stattdessen diese verwendet. Dieser Schritt wird einmal pro Iteration für das gesamte Bild ausgeführt.
Mit Hilfe der Werte von *NNF*($\boldsymbol{p}$) aus der Initialisierung ist man nun in der Lage, den *Propagation*-Schritt auszuführen. Dabei geht es darum, *NNF*($\boldsymbol{p}$) mit Hilfe der benachbarten Werte zu verbessern. Unter der Annahme, dass benachbarte Patches an ähnlichen Stellen zu finden sind, betrachtet man nun *NNF*($\boldsymbol{p} - \begin{pmatrix}0\\1\end{pmatrix}$) und *NNF*($\boldsymbol{p} - \begin{pmatrix}1\\0\end{pmatrix}$). Wenn deren Werte bessere Ergebnisse bezüglich A und B liefern, wird *NNF*(p) auf den besseren der beiden - inklusive dem Offset - gesetzt. Dies bedeutet konkret folgendes: Sei die Menge aller Offsets $O = \{\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}1\\0\end{pmatrix}\}$ und $\boldsymbol{o}^* = \underset{\forall \boldsymbol{o} \in O}{\arg\min}\, D(A, B, \boldsymbol{p}, NNF(\boldsymbol{p}+\boldsymbol{o}))$, dann gilt: $NNF(\boldsymbol{p}) \leftarrow NNF(\boldsymbol{p}+\boldsymbol{o}^*)-\boldsymbol{o}^*$.
Bei D handelt es sich um eine beliebige Distanzfunktion über A und B. Der Grund für diesen Offset besteht darin, dass der Algorithmus ursprünglich in *Scanline Order* ausgeführt wird. Somit wird während einer Iteration Rücksicht auf die Pixel genommen, die den *Propagation*-Schritt bereits ausgeführt haben. Nach jeder zweiten Iteration wird die Offsetmenge auf $O_r = \{\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}-1\\0\end{pmatrix}, \begin{pmatrix}-1\\0\end{pmatrix}\}$ gesetzt und der Algorithmus wird in umgekehrter *Scanline Order* ausgeführt, um korrekte Lösungen in beide Richtungen zu propagieren.
Der letzte Schritt nennt sich *Random Search*. Dabei wird in einem Fenster um *NNF*($\boldsymbol{p}$) nach einem Korrespondenzkandidaten gesucht. Sei $\boldsymbol{v_0} = NNF(\boldsymbol{p})$ die aktuell beste Korrespondenz laut *Propagation*, dann ist es das Ziel der Suche, dies noch zu verbessern, indem lokal nach einer besseren Korrespondenz gesucht wird.

$$\boldsymbol{u}_i = \boldsymbol{v_0} + w\alpha^i \boldsymbol{r}_i \tag{6}$$

wobei $\boldsymbol{r}_i \in [-1, 1]^2$ ein gleichverteilter Vektor, w eine maximale Suchweite und i eine Laufvariable ist. Es werden Patches für $i = 1, 2, 3, 4, ...$ untersucht, bis der Suchradius $w\alpha^i$ unterhalb einer Pixelbreite ist. Dann errechnet sich ein neues *NNF* wie folgt:

$$NNF(\boldsymbol{p}) \leftarrow \underset{\forall \boldsymbol{u}_i, i \in 1,...,n}{\arg\min}\; D(A, B, \boldsymbol{p}, \boldsymbol{u}_i) \tag{7}$$



0 *Iterationen* $\frac{1}{2}$ *Iterationen* 1 *Iterationen* 2 *Iterationen* 3 *Iterationen*

Abbildung 9: Hier wird das Vorgehen des *PatchMatch* dargestellt. Dabei wird in jeder Iteration ein *NNF* erstellt auf Basis von Initialwerten. Ganz links zu sehen ist ein zufällig initialisiertes *NNF*. Die zweite Grafik von links zeigt, dass der Algorithmus in *Scanline Order* ausgeführt wird. So würde das *NNF* aussehen, wenn man die Iteration nach der Hälfte anhält. Mit jeder weiteren Iteration wird es feiner und konvergiert zu einer *ANN*-Lösung. *Eigene Darstellung auf Grundlage von Barnes et al.[4]*

Diese drei Schritte ergeben eine Iteration des *PatchMatch*. Nach der ersten und jeder folgenden Iteration kann das Ergebnis von $NNF(\boldsymbol{p})$ als Initialwert für die nächste Iteration verwendet werden. Nach einer Anzahl von n Iterationen konvergiert der Algorithmus in Richtung des *Nearest-Neighbors* für jeden Patch. *Barnes et al.*[4] zeigen in ihren Ergebnissen, dass bereits nach einer geringen Anzahl von Iterationen ein zufriedenstellendes Ergebnis vorhanden ist. Sie führen 4-5 Iterationen durch, bevor die Konvergenzgeschwindigkeit marginal wird. *Fiser et al.*[11] führen 6 und *Benard et al.*[5] führen 3 Iterationen durch, bevor sie ihr Resultat als visuell ansprechend empfinden. Darüber hinaus betonen *Barnes et al.*[4], dass der Algorithmus besonders effektiv in den ersten Iterationen arbeitet. Das bedeutet, dass die Veränderung des *NNF* dort am schnellsten voranschreitet (siehe Abbildung 9). Obwohl diese Ergebnisse im Feld der Textursynthese vollkommen ausreichend sind, sei *PatchMatch* bestenfalls als Annäherung zu verstehen. Diese Beschränkung ist aber im Tausch gegen eine niedrige Laufzeit zu verkraften.

3.4 Patchsynthese

In der Textursynthese ist es kein triviales Unterfangen, auf lokaler Ebene zu optimieren und dabei eine globale Struktur beizubehalten. Obwohl patchbasierte Verfahren das Ziel verfolgen, das Umfeld eines Bildpunkts zu beachten, garantiert dies nicht, dass ein synthetisiertes Bild es auch fortführt. Neben der Korrespondenzfindung von Patches offenbart sich eine weitere Problematik, denn es stellt sich die Frage, in welcher Form das Patch übertragen werden soll. Bei einer Korrespondenz zweier Regionen $N_{\boldsymbol{q}} \subset S_{B'}$ und $N_{\boldsymbol{p}} \subset S_A$, kann es passieren, dass sie zwar eine hohe Ähnlichkeit aufweisen, jedoch im Zielbild nicht übereinstimmen (siehe Abbil-

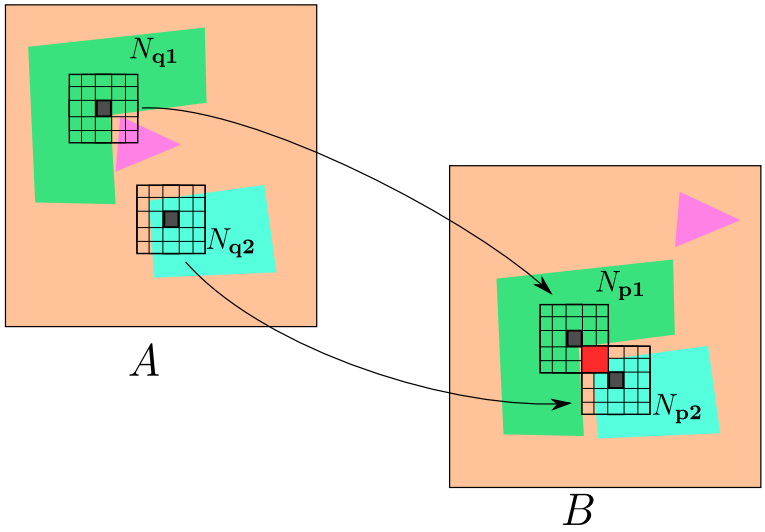


Abbildung 10: Für zwei Patches in B wird je eine Korrespondenz in A gefunden. Deren Ähnlichkeit untereinander ist dabei maximal, jedoch hat die Überschneidung der Korrespondenzen im Zielbild eine Differenz. *Eigene Darstellung auf Grundlage von Kwatra et al.[23]*

dung 10). In der Synthesephase heißt das, man muss darauf achten, sowohl die Umgebung des Ursprungspatches als auch des Zielpixels zu berücksichtigen.

3.4.1 *Texture Optimization for Example-based Synthesis*

Ein Verfahren namens *Texture Optimization for Example-based Synthesis* (kurz *TOES*) von *Kwatra et al.*[23] beschäftigt sich mit der Synthese von Texturen anhand von Vorlagen. Es verwendet ein Ähnlichkeitsmaß zwischen synthetisierter Textur und dessen Vorlage. Dadurch ist es ihnen erlaubt, die Synthese als ein Energieminimierungsproblem zu formulieren, welches über einen *Expectation Maximization* (*EM*)-ähnlichen[8] Algorithmus optimiert wird. Dabei wird im Vergleich zu anderen Verfahren nicht pro Pixel optimiert, sondern nur über eine Teilmenge von uniform-angeordneten Punkten. Die verbleibenden Bildpunkte werden über ein *Blending*-Verfahren der ermittelten Patches erzeugt.

3.4.2 *Sparse Grid*

Zunächst wird davon ausgegangen, dass eine Eingabetextur A gegeben ist, mit dessen Hilfe eine Textur B generiert werden soll. Darüber hinaus gilt die gleiche Notation für Nachbarschaft, Farbwerte und Bildpunkte

wie zuvor. Korrespondierende Bildpunkte werden einfachheitshalber mit $\boldsymbol{p}$ und $NNF(\boldsymbol{p})$ bezeichnet[2]. Man betrachte alle Bildpunkte $\boldsymbol{p} \in S_B$ und $NNF(\boldsymbol{p}) \in S_A$. Es wird außerdem eine Teilmenge an Bildpunkten von B definiert

$$S_B^\dagger = \{\boldsymbol{p}^\dagger \in S_B \mid \exists \begin{pmatrix} a \\ b \end{pmatrix} \in \mathbb{N}^2 : \boldsymbol{p}^\dagger = r \cdot \begin{pmatrix} a \\ b \end{pmatrix} \wedge a < \frac{m}{r} \wedge b < \frac{n}{r}\} \tag{8}$$

wobei m und n die Ausmaße von B sind und w die Fenstergröße einer beliebigen Nachbarschaft $N_{\boldsymbol{p}}$ ist. Außerdem gilt, dass $r = \frac{w+1}{2}$ die aufgerundete halbe Fenstergröße ist. Diese Teilmenge bezeichnet alle Punkte, die maximal eine halbe Fenster- oder besser gesagt Patchgröße voneinander entfernt sind. Dabei gilt für jeden Punkt $\boldsymbol{p}^\dagger \in S_B^\dagger$ und jeden anderen Punkt $\boldsymbol{q}^\dagger \in S_B^\dagger$, dass $\boldsymbol{q}^\dagger \notin N_{\boldsymbol{p}^\dagger} \wedge \boldsymbol{p}^\dagger \notin N_{\boldsymbol{q}^\dagger}$, unter der Voraussetzung, dass $\boldsymbol{p}^\dagger \neq \boldsymbol{q}^\dagger$. Allerdings ist zu beachten, dass für benachbarte Patches gilt: $N_{\boldsymbol{p}^\dagger} \cap N_{\boldsymbol{q}^\dagger} \neq \emptyset$.

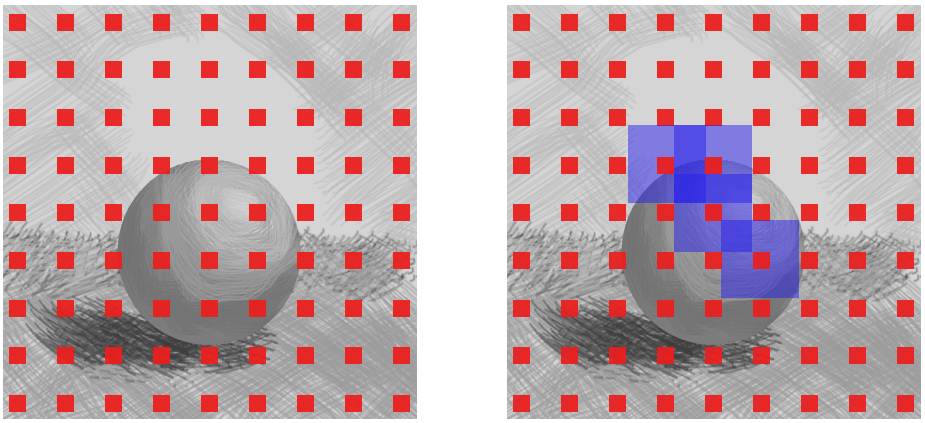


Abbildung 11: Die roten Punkte markieren ein *Sparse Grid* auf einem Bild. Auf der rechten Seite sieht man die quadratischen Fenster von $N_{\boldsymbol{p}}$ in blau. Untereinander überschneiden sich diese Nachbarschaften, jedoch nur so weit, dass keine anderen Punkte des *Sparse Grid* drin sind.*Eigene Darstellung auf Grundlage von Kwatra et al.[23]*

Diese Aufteilung von Bildpunkten nennt sich *Sparse Grid* und bedeutet, dass nur Punkte betrachtet werden, die sowohl gleichmäßig auf dem Bild verteilt sind, als auch nicht-leere Schnittmengen bezüglich ihrer Nachbarpatches haben. Diese Datenstruktur hat den naheliegenden Vorteil, dass eine kleinere Menge an Daten berechnet wird, was sich wiederum

[2]Die Originalnotation von *Kwatra et al.*[23] unterscheidet sich stark von jener, die hier verwendet wird. Dennoch wird in dieser Arbeit ein besonderer Wert auf eine einheitliche Notation gelegt. Dies gilt auch für noch kommende Formulierungen. Es sei außerdem zu beachten, dass *Kwatra et al.* ihre Arbeit formuliert haben, bevor der Begriff *Nearest-Neighbor Field* überhaupt definiert wurde.

in der Laufzeit bemerkbar macht. Dabei geht man davon aus, dass nur die Teilmenge aller Pixel optimiert wird. Obwohl es so aussieht, als ob es sich hauptsächlich auf die Laufzeit auswirkt, ist diese Datenstruktur sehr wichtig für die korrekte Anwendung des *TOES*-Verfahren.

3.4.3 Energieterm und Synthese

Hier wird der Energieterm von *Kwatra et al.*[23] vorgestellt, welcher die Ähnlichkeit einer Vorlage zu einem synthetisiertem Bild zeigt. Diese globale Energie wird über die Ähnlichkeit lokaler Nachbarschaften (Patches) beider Bilder erzeugt. Unter der euklidischen Norm wird die Energie über B gebildet:

$$E(A, B, NNF) = \sum_{\boldsymbol{p} \in S_B^\dagger} \|B^N(\boldsymbol{p}) - A^N(NNF(\boldsymbol{p}))\|_2^2 \tag{9}$$

Diese Formulierung wird benutzt, um durch das iterative Verbessern einer initialen Schätzung eine Textursynthese zu betreiben, bei der die Texturenergie in jeder Iteration abnimmt.
Pro Iteration wird abwechselnd nach einem *NNF* und einem B gesucht, welches diese Energie minimiert.

$$NNF \leftarrow \arg\min_{NNF^*} E(A, B, NNF^*) \tag{10}$$

$$B \leftarrow \arg\min_{B^*} E(A, B^*, NNF) \tag{11}$$

Nach B zu optimieren heißt effektiv, den Farbwert mit dem geringsten quadratischen Abstand zu korrespondierenden Farbwerten in A zu berechnen. Dieser bildet sich aus dem Mittelwert der überlappenden Patches an dieser Position. Damit lässt sich Gleichung 11 für jedes $\boldsymbol{p} \in S_B$ umformen und darstellen als:

$$B(\boldsymbol{p}) \leftarrow \frac{1}{|P_{\boldsymbol{p}}|} \sum_{\boldsymbol{q} \in P_{\boldsymbol{p}}} A(NNF(\boldsymbol{q}) + (\boldsymbol{p} - \boldsymbol{q})) \tag{12}$$

wobei $P_{\boldsymbol{p}}$ die Menge aller Patchpositionen ist, für die gilt, dass $\boldsymbol{p}$ in deren Nachbarschaft ist. Anders formuliert:

$$P_{\boldsymbol{p}} = \{\boldsymbol{q} \in S_B \mid \boldsymbol{q} \in S_{B'}^\dagger \wedge \boldsymbol{p} \in N_{\boldsymbol{q}^\dagger}\} \tag{13}$$

Das heißt, dass die Farben in B über die korrespondierenden Patches in A gemittelt werden. Für Patches, die sich sowohl in A als auch in B überlappen bedeutet es, dass die Struktur aus A übernommen wird. Für Patches, die sich nicht in A überlappen, wird ein Mittelwert berechnet. Für darauf folgende Iterationen bedeutet dies, dass bei der Berechnung eines neuen *NNF* bezüglich der Patches in A Rücksicht auf die Patchfarbe in B genommen wird.

3.4.4 *Expectation Maximization*

Der Ansatz von *TOES* ist algorithmisch ähnlich zu *Expectation Maximization* (*EM*)[8]. *EM* wird für die Optimierung unter der Voraussetzung verwendet, dass neben den gesuchten Variablen auch die statistischen Parameter der Energiefunktion unbekannt sind. Dabei wird so vorgegangen, dass abwechselnd die Variablen und die Parameter geschätzt werden. Diese beiden Schritte lassen sich in einen E- und einen M-Schritt einteilen. In diesem Fall handelt es sich bei den Variablen um B und bei den Parametern um *NNF*. Die Berechnung einer minimalen Energie über B entspricht dabei dem E-Schritt, während das Finden von *NNF* den M-Schritt darstellt.

3.4.5 Algorithmus

Intuitiv versucht der Algorithmus eine gute relative Anordnung von Input-Patches zu finden, um eine Textur zu generieren. Während jeder Iteration wählt der M-Schritt eine Anordnung, welche zur aktuellen Schätzung passt. Das Mitteln von Farbwerten im E-Schritt erlaubt überlappenden Patches Informationen über die lokale Beschaffenheit der Input-Patches untereinander zu kommunizieren. Patches, deren Schnittmengen nicht gut zueinander passen, resultieren in weichgezeichneten Bildausschnitten in B. Diese weichgezeichnete Region repräsentiert den Übergang zweier inkonsistenter Regionen und kann der visuellen Struktur der Ursprungsregionen widersprechen. Dies erlaubt der nächsten Iteration des M-Schritts die aktuellen Korrespondenzen gegen welche auszutauschen, die konsistenter untereinander sind. So werden Patches gewählt, die die weiche Struktur ersetzen und deren Schnittmenge höhere Strukturen aufweisen.

Algorithm 3 Das Grundgerüst des *TOES*-Algorithmus

1: $NNF^0(\boldsymbol{p}) \leftarrow$ random patch in S_A
2: **for each** $i = 0 : N$ **do**
3: **for each** $\boldsymbol{p} \in S_B^\dagger$ **do**
4: $B^{i+1}(\boldsymbol{p}) \leftarrow \frac{1}{|P_{\boldsymbol{p}}|} \sum_{\boldsymbol{q} \in P_{\boldsymbol{p}}} A(NNF^i(\boldsymbol{q}) + (\boldsymbol{p} - \boldsymbol{q}))$
5: **end for**
6: $NNF^{i+1} \leftarrow \underset{NNF^*}{\arg\min}\ E(A, B^i, NNF^*)$
7: **if** $NNF^{i+1}(\boldsymbol{p}) \approx NNF^i(\boldsymbol{p}), \forall \boldsymbol{p} \in S_B^\dagger$ **then**
8: **break**
9: **end if**
10: **end for**

Hier wurde ein Algorithmus vorgestellt, welcher iterativ eine Textur synthetisiert und als Eingabe eine Vorlage verwendet. Ziel davon ist es, die Struktur der Eingabe auf das Ausgabebild zu übertragen. In kommenden

Kapiteln wird gezeigt, wie dieses Verfahren sowohl positive Auswirkungen auf die Laufzeit als auch auf die Struktur der Ergebnisse haben kann.

3.5 Merkmalsvektoren

Ein Bild aus der Computergrafik, welches einem physikalisch-basiertem Beleuchtungsmodell folgt, hat prinzipiell die gleichen Farbräume wie ein Foto. Für das Vorgehen von *Image Analogies* spielt es daher grundsätzlich keine Rolle, um welche Art von Bild es sich handelt. *Hertzmann et al.*[18] haben ihre Ergebnisse zwar unter dem Einsatz von Fotografien gezeigt, dabei wurden jedoch ausschließlich die verfügbaren Farbeigenschaften der Fotos verwendet. Eine konsistente Eigenschaft vieler Fotos ist der *RGB*-Farbraum. Viele Verfahren des *EBR* und der Textursynthese basieren darauf, diese als Merkmalsvektor für ihre Optimierung zu verwenden. Alternativ lassen sich *RGB*-Farben in *Lab**, *HSV* etc. umwandeln. Diese Transformationen haben natürlich Auswirkungen auf das Ergebnis der Distanzmetrik. Man kann den Farbraum also mit Rücksicht auf die Distanz und die Qualität der Resultate verändern. Dennoch gewinnt der Farbraum durch eine Transformation nicht zwangsläufig an Prägnanz, denn der Informationsgehalt bleibt dabei erhalten. *Benard et al.*[5] implementieren eine Weiterentwicklung von *Image Analogies* für Animationen. Sie verwenden neben einfachen *RGB*-Farben zusätzliche Kanäle, wie *Optical Flow*[1] oder vorherige Frames als Orientierungshilfe. Mit Hilfe dieser zusätzlichen Kanäle erreichen sie neben einer höheren Prägnanz auch eine temporale Kohärenz, was bedeutet, dass eine Stilisierung über mehrere Frames eine gewisse Konsistenz aufweist. Für die Stilisierung von Fluiden verwenden *Jamriska et al.*[19] zusätzlich den α-Kanal um Partikeltexturen auf ein sogenanntes *Flow Field* zu übertragen und dabei Rücksicht auf deren Transparenz zu nehmen. Der Vorteil davon ist, dass dadurch die Prägnanz von Randregionen verstärkt wird und somit realistischere Fluide darstellbar sind.
Fiser et al.[11] stellen ihr *EBR*-Verfahren namens *Stylit* vor, welches Renderings aus einem *Pathtracer* verwendet, um ein stilisiertes Bild zu generieren.

3.5.1 *Light Pass Expressions*

Beim Transport von Licht können Interaktionen zwischen Licht und Oberfläche generell als diffus oder spiegelnd bezeichnet werden. Untersucht man alle Interaktionen, die zwischen Lichtquelle und Sensor stattfinden, können diese in die wichtigsten Komponenten globaler Beleuchtungsmodelle eingeteilt werden. Diese Kategorisierungen werden als *Light Path Expressions* (kurz *LPE*s)[17] bezeichnet und zählen seit geraumer Zeit

als Bestandteil[3] der Computergrafik. Einer ihrer Verwendungszwecke ist die Trennung diverser Beleuchtungseffekte in einzelne Zwischenspeicher (engl. *Buffer*). Mit Hilfe dieser *Buffer* lassen sich Bildverarbeitungsfilter auf einzelne Effekte anwenden, bevor sie zum finalen Bild zusammengefügt werden. Im Fall von *Stylit* liegt der Vorteil darin, dass die Prägnanz eines Merkmalsvektors im Vergleich zu einfachen *RGB*-Daten stark erhöht wird. Außerdem kann auf diese Art und Weise Rücksicht auf eine gerenderte Szene und deren Beleuchtung genommen werden. Das kommt der dritten Anforderung dieser Arbeit stark entgegen.

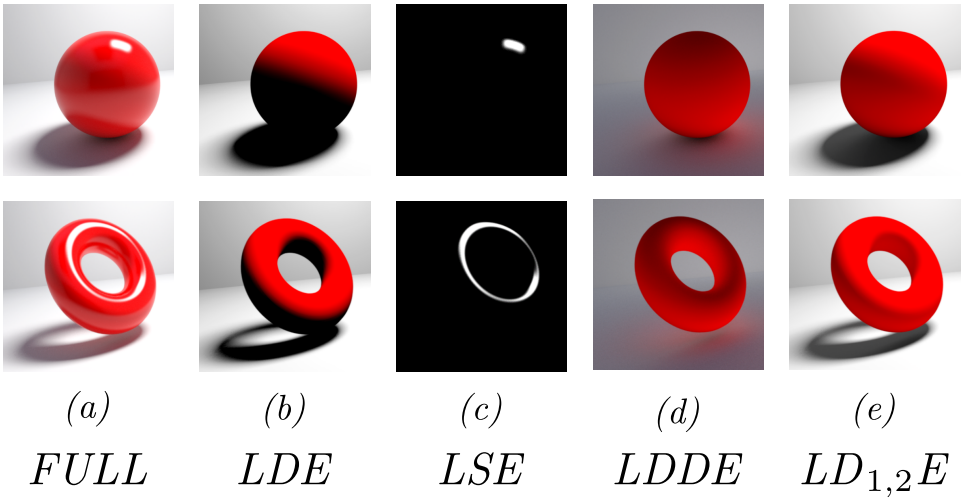


Abbildung 12: Dies sind die von *Fiser et al.*[11] verwendeten *Light Pass Expressions: (a) ist die vollständig gerenderte Szene, (b) direkte diffuse Beleuchtung, (c) direkte spekulare Beleuchtung, (d) diffuse Interreflexionen und (e) die ersten beiden diffusen Indirektionen. Quelle: Fiser et al.[11]*

3.5.2 Notation

Die Verwendung von *LPE*s ist wesentlich für diese Arbeit, daher wird hier kurz auf die erweiterte Notation eingegangen. Wie zuvor erwähnt, handelt es sich bei $I(\boldsymbol{p})$ um den Farbwert an der Position $\boldsymbol{p}$ im Bild I. Man beachte, dass $I(\boldsymbol{p}) \in F_I$, wobei F_I der Farbraum von Bild I ist. Andere verfolgen den Ansatz $F_I = F_{RGB} \subset \mathbb{R}^3$. *Fiser et al.* definieren stattdessen folgendes:

$$F_I = F_1 \times \cdots \times F_b \times \cdots \times F_n \tag{14}$$

wobei $\times$ die nicht-kommutative Verkettung von Mengen über das Kreuzprodukt ist. Für den Index gilt $0 < b \leq n$ mit $n = |LPE|$. Das

[3] Der Begriff *Light Path Expression* selbst ist nur einer von vielen Bezeichnungen für die gleiche Datenstruktur.

bedeutet, dass der Farbraum F_I sich aus n Farbräumen zusammensetzt, welche je einem *Buffer* von I zuzuordnen sind. Bei *Stylit* beschreibt *LPE* die Menge aller Bezeichner von *Buffern* für bestimmte *Lightpass Expressions* (siehe Abbildung 12). Diese setzt sich folgendermaßen zusammen: $LPE = \{FULL, LDE, LSE, LDDE, LD_{1,2}E\}$. Für jeden Farbraum eines *LPE*-Buffers gilt dabei, dass $F_b = F_{RGB} \subset \mathbb{R}^3$, da sich das Pathtracing von *Stylit* auf den *RGB*-Farbraum beschränkt.
Mit *LPE*s verwenden *Fiser et al.* Daten, welche tatsächlich auf der Beleuchtungssituation eines Renderings basieren. Dadurch werden sowohl Artefakte als auch Fehlzuweisungen minimiert. Jeder Pixel, beziehungsweise jedes Patch in A und B haben somit eine sehr hohe Prägnanz bezüglich globaler Beleuchtung. Sie schaffen es damit, komplexe Situationen wie indirekte Beleuchtung im Zielbild zu rekonstruieren. Diese Eigenschaft dient in erster Linie der Anforderung, die Beleuchtung bei der Stilisierung zu beachten.

4 Hauptteil

Im letzten Kapitel wurden viele wichtige Techniken aufgezählt, die in dieser Arbeit Verwendung finden. Der *Workflow* soll dabei erhalten bleiben. Gegeben sei eine Kollektion von Vorlagen, die per Hand angefertigt wurden (siehe Abbildung 13). Diese liegen in digitaler Form vor und können von einem Programm eingelesen werden. Anhand von Benutzereingaben kann man im Vorfeld Eingabeparameter verändern, ein 3D-Modell (das Referenzmodell B) selektieren und per Knopfdruck das Programm starten. Dann wird in einer Art *GUI* (*Graphical User Interface*) das Resultat angezeigt. Die Implementierung folgt dem Grundprinzip von *Hertzmann et al.* und deren *Image Analogies*. Es werden zwei Vorlagen (Modellvorlage A und Stilvorlage A') zusammen mit dem Referenzmodell B verwendet, um ein Zielbild B' zu generieren. Währenddessen werden sowohl Patches als auch eine Bildpyramide als Datenstruktur beachtet und zweckgemäß erweitert oder vereinfacht. Als Korrespondenzfindung wird außerdem eine vereinfachte Version des *PatchMatch* verwendet. Dabei wird eine globale *Nearest Neighbor*-Suche über die vier Bilder ausgeführt. Für die eigentlich Generierung des Zielbildes wird das *Blending*-Verfahren für überlappende Patches angewendet, wie es *Kwatra et al.*[23] vorgestellt haben. Um die Prägnanz von A und B zu erhöhen, werden *LPE*s verwendet, welche die Ergebnisse hinsichtlich der Beleuchtung von A und B verbessern sollen.

4.1 Herangehensweise

Image Analogies zählen als Grundgerüst für viele *EBR*-Verfahren[5][11][16]. In dieser Arbeit ist es nicht anders. Unabhängig davon, ob es sich um Fotos

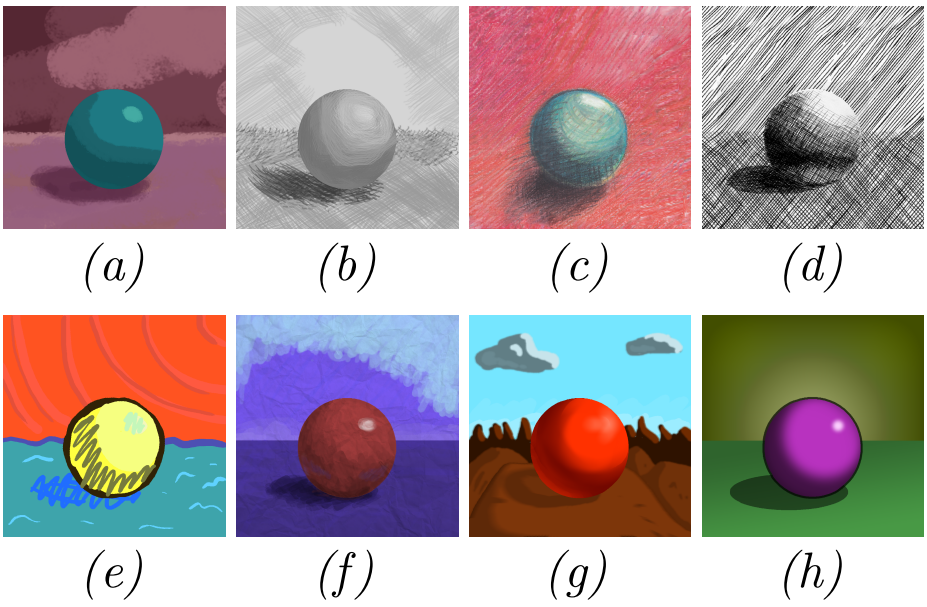


Abbildung 13: Mit Hilfe dieser Stilvorlagen sollen 3D-Modelle stilisiert werden. *Eigene Darstellung auf Grundlage von Fiser et al.[11] für (c)*

oder 3D-Renderings handelt, ist die Analogie $A : A' :: B : B'$ maßgeblich für dieses Vorgehen.

4.1.1 Aufbau

Da sich diese Arbeit mit der Stilisierung von 3D-Renderings beschäftigt, wird ein ähnlicher Aufbau wie jener von *Stylit* verwendet. Dabei werden keine willkürlichen Bilder verwendet, sondern welche, die einer Szene mit konkreter Beleuchtung, Farbkodierung und Modellen entsprechen. Für sowohl A als auch B gilt, dass sie von einer einzigen Punktlichtquelle beleuchtet werden. Der Hintergrund wird weiß und das Objekt im Vordergrund wird rot kodiert. Ähnlich wie bei *Lit Sphere* wird für A eine Kugel als Motiv verwendet. Dies hat zwei Vorteile: Zunächst hat jeder Punkt auf einer Kugel eine andere Orientierung. Das bedeutet, jede Normale einer Kugel zeigt in eine andere Raumrichtung. Für diese Arbeit bedeutet dies, dass es ähnlich wie bei *Lit Sphere*, für jede Normale in B eine Normale in A gibt. Das kommt einer Zuordnung sehr entgegen. Der zweite Punkt ist, dass man besser in der Lage ist, eine Vorlage A' zu erstellen. Ein Benutzer ohne Erfahrungen in der Erstellung von Kunstwerken ist eher in der Lage, eine Kugel zu malen als ein komplexeres Objekt. Für B gilt, dass es ein beliebiges untexturiertes Objekt darstellt. Dieses Objekt wird dann durch das *EBR*-Verfahren stilisiert.

4.1.2 Energieminimierung

Ähnlich wie in Gleichung 5 und 9 wird zunächst eine Energiefunktion definiert, welche für zwei Bildpunkte eine Ähnlichkeit zwischen den Patches widerspiegelt.

$$\begin{aligned} E(A, A', B, B', \boldsymbol{p}, \boldsymbol{q}, \mu) =& (1-\mu)\|A^N(\boldsymbol{p}) - B^N(\boldsymbol{q})\|_2^2 \\ &+ \mu\|A'^N(\boldsymbol{p}) - B'^N(\boldsymbol{q})\|_2^2 \end{aligned} \tag{15}$$

wobei μ ein Gewichtungsfaktor ist, für den gilt $\mu \in [0, 1]$. Je niedriger dieser Energieterm ist, desto ähnlicher sind sich der Patch $A^N(\boldsymbol{p})$ und $B^N(\boldsymbol{q})$, beziehungsweise $A'^N(\boldsymbol{p})$ und $B'^N(\boldsymbol{q})$, je nach Gewichtungsfaktor. Als Distanzmetrik wird die euklidische Norm $\|\cdot\|_2$ über n-dimensionale Vektoren verwendet. Bei den Vektoren $A'^N(\boldsymbol{p})$ und $B'^N(\boldsymbol{q})$ handelt es sich um die Konkatenation aller Farbwerte innerhalb des Patches $N_{\boldsymbol{p}}$, wobei zu beachten ist, dass A' und B' sich im *RGB*-Farbraum befinden. Währenddessen sind A und B in F_{LPE+}, einem erweiterten Farbraum der *LPE*-Buffers. Dieser wird hier zu einem späteren Zeitpunkt erklärt. Damit soll jedoch zunächst vor Augen geführt werden, dass es nicht praktikabel ist, $A^N(\boldsymbol{p})$ mit $A'^N(\boldsymbol{p})$ und analog $B^N(\boldsymbol{q})$ und $B'^N(\boldsymbol{q})$ zu vergleichen, da deren Farbräume nicht zueinander korrespondieren.
Vorherige Arbeiten[23][33][5][10] zeigen, dass gute Resultate erzielt werden können, indem ein Optimierungsschema verwendet wird, welches folgende Energien minimiert:

$$\sum_{\boldsymbol{q} \in S^{\dagger}_{B'}} \min_{\boldsymbol{p} \in S_{A'}} E(A, A', B, B', \boldsymbol{p}, \boldsymbol{q}, \mu) \tag{16}$$

Dabei wird für jeden Patch im *Sparse Grid* $\boldsymbol{q} \in S^{\dagger}_{B'}$ eine minimale Energie gefunden und aufsummiert.

4.1.3 Merkmalsvektoren

Wie zuvor erwähnt, wird eine erweiterte Version der vorgeschlagenen *LPE*s von *Fiser et al.*[11] verwendet. Der zuvor deklarierte Farbraum F_{LPE} ist die Konkatenierung aller *Buffer*-Farbräume F_b) für einen Bezeichner in $LPE = \{FULL, LDE, LSE, LDDE, LD_{1,2}E\}$. Zusätzlich werden nun weitere *Buffer* definiert, welche über schiere *LPE*s hinausgehen. Hier wird der Farbraum F_{LPE+} für die Menge an Bezeichnern $LPE_+ = (LPE \backslash \{LD_{1,2}E\}) \cup \{IDMASK, FRESNEL, LIGHTMASK\}$ benutzt. Diese drei zusätzlichen *Buffer* erhöhen die Prägnanz von A und B. Auf den Bezeichner $LD_{1,2}E$ wird außerdem verzichtet, da hier im Rahmen des Echtzeitrenderings optimiert wird, bei dem maximal eine Indirektion berechnet wird. Dadurch würde F_{LPE+} keine neue Information hinzugefügt. Der Bezeichner *IDMASK* ist eine farbliche Kodierung von Hintergrund, Vordergrund und Schlagschatten. Es ist

zwar wichtig, dass die Anforderung an die korrekte Beleuchtung erfüllt bleibt, jedoch ist es noch wichtiger, dass ein Patch des Vordergrundpatches weder einem Hintergrund- noch einem Schlagschattenpatch zugewiesen wird. Das gleiche gilt gegenseitig. Bei *FRESNEL* handelt es sich um *Schlick's Annäherung*[30] an den Fresnelterm f. Dieser hängt vom Blickwinkel und der Normalen der Oberfläche ab. Da es sich bei A um eine Kugel handelt, ist in A für *FRESNEL* jedes mögliche f zu finden. Dieser Bezeichner verstärkt die Korrespondenz von Patches bezüglich des Blickwinkels und der Normalen. Bei weichen Lichtgradienten erweist sich eine Zuordnung als eher schwierig. Dagegen wirkt der Bezeichner *LIGHTMASK*, welcher Glanzpunkt, Verschattungen und direkt beleuchtete Bildbereiche kodiert. Dabei wird ein Schwellwert verwendet, welcher die Bereiche zuordnet. Der Sinn dahinter ist, es Lichtgradienten zu untermalen und Übergangsregionen gegenüber ähnlichen Bereichen hervorzuheben.
Insgesamt handelt es sich dabei um Daten, die nicht direkt Teil der *LPE*s sind, sondern Charakteristika kritischer Bildbereiche abdecken. Deren Auswirkungen werden in einem späteren Abschnitt genauer beleuchtet.

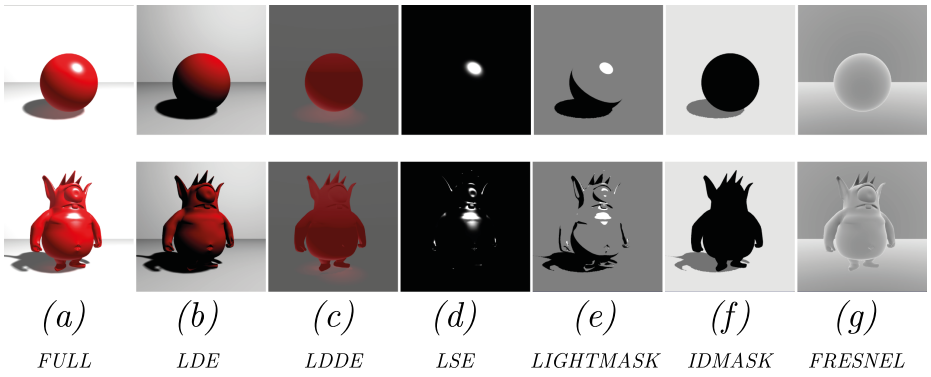


Abbildung 14: Diese Abbildung zeigt die *LPEs*, welche in dieser Arbeit verwendet werden. *(a)* ist die vollständig gerenderte Szene, *(b)* direkte diffuse Beleuchtung, *(c)* diffuse Interreflexionen, *(d)* direkte spekulare Beleuchtung, *(e)* ist die Kodierung von Licht und Schatten, *(f)* ist die Kodierung von Vordergrund, Hintergrund und Schatten und *(g)* der Fresnelterm.

4.1.4 Algorithmus

Ähnlich wie Algorithmus 3 kann nun ein Algorithmus definiert werden, der *EM*-ähnliche Iterationen durchführt. *Fiser et al.*[11] weisen auf diese Herangehensweise bezüglich *EBR* hin.

Algorithm 4 *EM*-ähnliche Iteration zu Minimierung des Energieterms

1: **function** REFINETARGETIMAGE(A, A', B, B')
2: **for each** $\boldsymbol{q} \in S^{\dagger}_{B'}$ **do**
3: $NNF(\boldsymbol{q}) \leftarrow \underset{\boldsymbol{p} \in S_{A'}}{\arg\min}\ E(A, A', B, B', \boldsymbol{p}, \boldsymbol{q}, \mu)$
4: **end for**
5: **for each** $\boldsymbol{q} \in S^{\dagger}_{B'}$ **do**
6: $B^{tmp}(\boldsymbol{q}) \leftarrow Average(A', NNF, \boldsymbol{q})$
7: **end for**
8: **return** B^{tmp}
9: **end function**

Dieser Algorithmus lässt sich in zwei Phasen unterteilen. Zunächst wird ein *NNF* ermittelt, für welches die Energie in Gleichung 15 minimal ist. Dabei wird effektiv auch die Energie von Gleichung 16 minimiert. Im darauffolgenden Schritt wird für jeden Bildpunkt in B^{tmp} ein neuer Farbwert ermittelt. Bei der Funktion *Average* handelt es sich ähnlich wie in Gleichung 12 um das *Blending* überlagernder Pixel mehrerer Patches.

$$Average(I, NNF, \boldsymbol{p}) = \frac{1}{|P_{\boldsymbol{p}}|} \sum_{\boldsymbol{q} \in P_{\boldsymbol{p}}} I(NNF(\boldsymbol{q}) + (\boldsymbol{p} - \boldsymbol{q})) \tag{17}$$

wobei $P_{\boldsymbol{p}}$ die Menge aller Patchpositionen ist, für deren Patches gilt, dass sie $\boldsymbol{p}$ überlagern. Dieser Algorithmus hat B^tmp als Rückgabewert und erhält A, A', B und B' als Eingabewerte. Das bedeutet, dass ein neues B' auf Basis seines aktuellen Wertes berechnet wird. Somit kann man mit diesem Algorithmus iterativ den Inhalt von B' verändern, sodass man einer gewünschten visuellen Qualität entgegenkommt.
Dieser iterative Ansatz wird noch um eine Bildpyramide erweitert. Der Sinn dahinter ist, dass niedrig-frequente Strukturen bereits auf niedriger Auflösung zugewiesen werden können. Ähnlich wie Algorithmus 1 wird zusätzlich über die vorhandenen Bildpyramidenlevel iteriert. Auf jeder dieser Ebenen wird dann zusätzlich Algorithmus 4 ausgeführt.

Algorithm 5 Iteration über die Bildpyramide

1: **function** TRAVERSEPYRAMID(A, A', B, B')
2: Calculate L pyramid layers for A, A' and B
3: Fill pyramid layers respectively
4: Create L empty pyramid layers for B'
5: **for each** l = L-1 ... 0 **do**
6: $B'_l \leftarrow Upscale(B'_{l+1})$
7: **for each** k = 0 ... m **do**
8: $B'_l \leftarrow$ *refineTargetImage*(A_l, A'_l, B_l, B'_l)
9: **end for**
10: **end for**
11: $B' \leftarrow B'_0$
12: **return** B'
13: **end function**

Hier werden zunächst L Pyramidenlevel für A, B und A' berechnet. Bei *refineTargetImage* handelt es sich um die Funktion, welche von Algorithmus 4 beschrieben wird. Dabei ist zu beachten, dass der Inhalt von jeder Ebene I_l - wie in Abschnitt 3.2.2 beschrieben - mit einer weichgezeichneten Version von I selbst befüllt werden kann. Für B'_l gilt hingegen, dass es nicht abwärts befüllt wird, sondern während des Verfahrens selbst. Somit arbeitet man sich von der niedrigsten Auflösung zur originalen Auflösung hoch. In den unteren Ebenen hat man einerseits den Vorteil, dass bei einer geringeren Auflösung weniger Bildpunkte berechnet werden müssen und andererseits findet hier die größte globale Annäherung an B' statt. Das bedeutet, die groben Strukturen werden bereits am Anfang synthetisiert (siehe Abbildung 15).

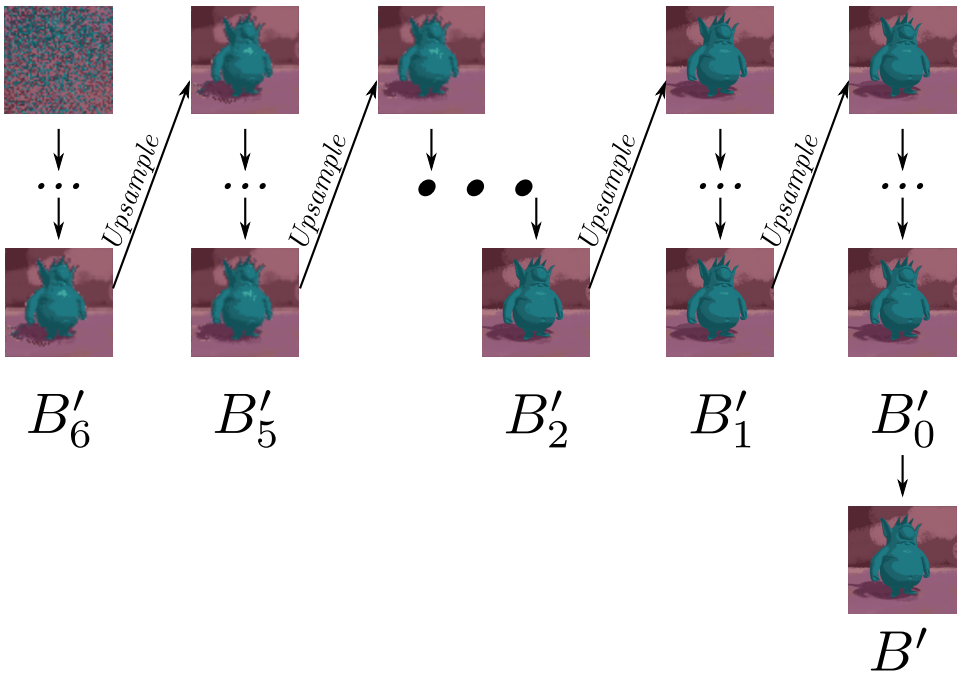


Abbildung 15: Hier ist zu sehen wie B' in jedem Schritt feiner wird. Dabei wird eine Anzahl an Iterationen durchgeführt pro Pyramidenlevel. Danach wird durch *Upsampling* das Ergebnis an die nächsthöhere Ebene weitergegeben. Am Ende, wenn die höchste Stufe erreicht ist und durchiteriert ist, wird der Inhalt von B'_0 an B' weitergegeben.

Prinzipiell handelt es sich bei der hier beschriebenen Herangehensweise um eine Version von *Image Analogies*, welche um die Patchsynthese auf einem *Sparse Grid*, *LPE*s und *PatchMatch* erweitert wurde. Nachdem nun das Grundvorgehen bekannt ist, werden an dieser Stelle Maßnahmen beschrieben, die sich positiv auf die Laufzeit auswirken.

4.2 Laufzeitminimierung

Betrachtet man verwandte Arbeiten, haben viele die Gemeinsamkeit einer hohen Laufzeit. *Fiser et al.*[11] produzieren mit ihrem *Stylit*-Verfahren zweifellos hervorragende Ergebnisse, wenn es darum geht visuelle Eigenschaften einer Vorlage zu übertragen. Jedoch hängen diese Ergebnisse mit intensiven Berechnungen und einer hohen Anzahl an Iterationen zusammen. Das Synthetisieren eines 1Mpx-großen Bildes dauert ungefähr 15 Minuten auf einer 3GHz CPU mit vier Kernen und 3 Minuten unter der GPU-Implementierung auf einer GeForce GTX Titan Black. Auf der *Stylit*-Webseite[12] findet man eine interaktive Demo, die mit einer leichtgewichtigen GPU-Implementierung unter halber Auflösung 3-6 Sekunden

dauert. *Jamriska et al.*[19] brauchen für ihre Synthese eines $1\mathrm{Mpx}$ Bildes, auf einer 3 GHz CPU mit vier Kernen, ungefähr 5 Minuten . Bei der Stilisierung eines Bildes unter dem Verfahren von *Benard et al.*[5] kommt es zu Laufzeiten von bis zu 12 Minuten. Diese Zeiten sind zwar akzeptabel für ein Offline-Setting, aber unbrauchbar für Echtzeitanwendungen. Da diese Arbeit viele Gemeinsamkeiten mit den genannten Verfahren hat, enthält die beschriebene Herangehensweise die gleichen Probleme bezüglich der Laufzeit. In diesem Abschnitt werden daher zunächst die größten Engpässe identifiziert und versucht, deren Auswirkungen auf die Laufzeit zu beheben.

4.2.1 Engpässe

Der in Abschnitt 4.1.4 vorgeschlagene Algorithmus orientiert sich stark an *Stylit*[11], *Image Analogies*[18] und der Textursynthese nach *Kwatra et al.*[23]. Daraus lässt sich schließen, dass es zu ähnlichen Engpässen kommt. Mit Engpässen sind Eigenschaften und Operationen gemeint, die negative Auswirkungen auf die Laufzeit haben. Manche Engpässe kann man umgehen, während andere unumgänglich sind.
Wie bereits erwähnt, werden mit dem Farbraum F_{LPE+} eine Anzahl von Merkmalen betrachtet. Mit einem höheren Merkmalsraum steigt die Anzahl der Abfragen. Der Aufwand, einen solchen Vektor auszulesen, wird hier mit $\mathcal{O}(|F_{LPE+}|) = \mathcal{O}(k)$ dargestellt. Vergleicht man zwei Patches miteinander, wird der Farbwert jedes $\boldsymbol{p}' \in N_{\boldsymbol{p}}$ mit einem $\boldsymbol{q}' \in N_{\boldsymbol{q}}$ verglichen. Wenn w die Breite eines quadratischen Patches ist, beläuft sich deren Komplexität für einen einfachen Vergleich auf $\mathcal{O}(2 \cdot 2 \cdot w^2) = \mathcal{O}(4 \cdot w^2) = \mathcal{O}(w^2)$. Der Faktor 4 wird einerseits benutzt, weil man mit A, A', B und B' Zugriff auf vier Bilder hat und andererseits, weil ein Vergleich auf zwei Bildern stattfindet. Dieser Faktor kann jedoch vernachlässigt werden, da es sich um eine Konstante handelt. Bei jedem Zugriff muss der Merkmalsraum mit einbezogen werden, daher ist es ein eigentlicher Aufwand von $\mathcal{O}(w^2 \cdot k)$ pro Patch. Für einen *Bruteforce*-Ansatz der *Nearest Neighbor*-Suche von Algorithmus 4 kann man von einem Aufwand von $\mathcal{O}(s_A \cdot s_B)$ reden, wobei $s_A = |S_A| = |S_{A'}|$ und $s_B = |S_B| = |S_{B'}|$ die Anzahl der Bildpunkte in den jeweiligen Bildern ist. Durch die Verwendung von *PatchMatch* lässt sich dieser jedoch auf $\mathcal{O}(s_B \cdot h) = \mathcal{O}(s_B)$ reduzieren, denn es wird für jeden Punkt $\boldsymbol{q} \in S_{B'}$ die gleiche konstante Anzahl an Vergleichen h durchgeführt, unabhängig von der Bildgröße[4]. Damit beläuft sich die Komplexität einer Bildpyramidenebene l auf $\mathcal{O}(s_{B_l} \cdot g \cdot w^2 \cdot k)$, wobei g die Anzahl an *PatchMatch*-Iterationen pro Iteration ist.
Man ist durchaus in der Lage, die Komplexität dieses Verfahrens weiter zu untersuchen und in seine Bestandteile zu zerlegen. An dieser Stelle wird jedoch darauf verzichtet. Die genannten Aufwandsklassen sollen veranschaulichen, dass es sich um durchaus kostspielige Operationen han-

delt. Aufgrund der Anforderung, ein echtzeitfähiges Verfahren zu implementieren, werden Maßnahmen aufgezählt um die Zahl drastisch zu reduzieren.

4.2.2 Redundanzminimierung

Unter den zuvor genannten Engpässen leiden viele vergleichbare Verfahren. Allerdings werden sie der Qualität zuliebe in Kauf genommen. Diese Arbeit zielt in erster Linie darauf ab, eine akzeptable Geschwindigkeit zu erreichen.
Angefangen wird hier bei der Minimierung der Redundanz bezüglich des Merkmalsraums. *Fiser et al.*[11] führen bei ihrem beschleunigten Verfahren eine Hauptkomponentenanalyse[29] (engl. *Principal Component Analysis*, kurz *PCA*) zur Laufzeit durch. Damit wird der Farbraum F_{LPE} um einige Dimensionen und gleichzeitig die Anzahl der Zugriffe reduziert. Anstatt Dimensionen während der Laufzeit zu reduzieren, wird hier stattdessen der Farbraum F_{LPE+} in der Implementierung fest reduziert. Eine *PCA* entfernt Dimensionen, welche stark mit anderen korrelieren und projizieren sie auf eine Hauptachse. Aufgrund des Aufbaus können folgende Eigenschaften ausgenutzt werden, Dimensionen vorzeitig zu reduzieren. Erstens entspricht jeder *Buffer* entweder einem *LPE* oder einer farblich kodierten Eigenschaft der Szene. Zuvor wurde bereits erwähnt, dass $F_b = F_{RGB} \subset \mathbb{R}^3$ für *Buffer*, die beim Rendering dem *RGB*-Farbraum unterliegen. Betrachtet man beispielsweise F_{LSE} (siehe Abbildung 14d), dann sind dort ausschließlich weiße Helligkeitsstufen abgebildet. Das bedeutet, dass im Endeffekt drei Farbkanäle verwendet werden, welche den gleichen Wert haben. Für die Laufzeit- beziehungsweise Redundanzminimierung bedeutet dies, dass zwei von drei Kanälen redundant sind und somit wegfallen können, ohne dass *LSE* an Information verliert. Mit einem ähnlichen Prinzip lassen sich alle anderen *Buffer* reduzieren. Dadurch, dass die Szene durch ein rotes Vordergrundobjekt und ein weißes Hintergrundobjekt dargestellt wird, kann ein Drittel des *RGB*-Spektrums vernachlässigt werden. Betrachtet man hiervon den Grün- und Blaukanal, sieht man, dass sie für jeden Bildpunkt korrelieren. Somit lässt sich einer der beiden Kanäle problemlos entfernen.

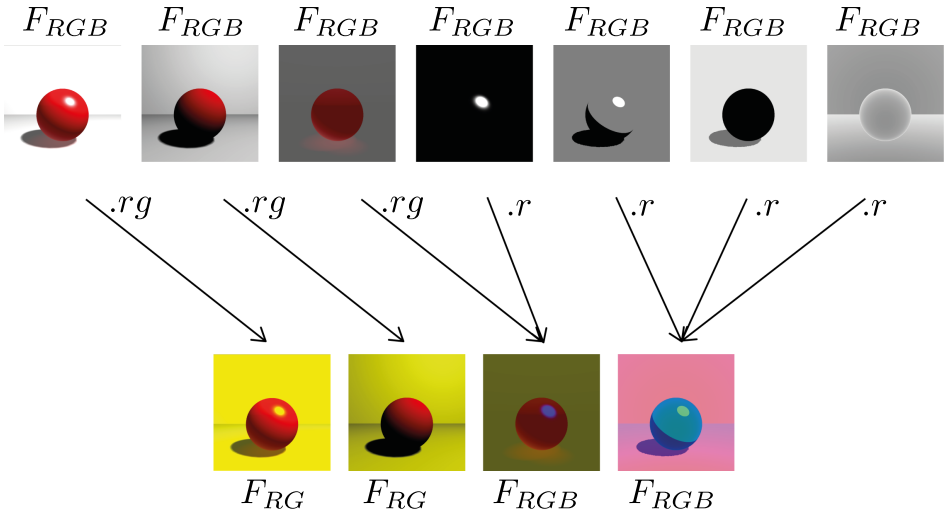


Abbildung 16: Hier ist zu sehen wie die Dimensionen der einzelnen Farbräume minimiert werden. Dabei werden einzelne Farbkanäle selektiert und neu zusammengefügt. Dadurch verringert sich nicht nur die Anzahl der Dimensionen selbst, sondern auch die Anzahl der Texturen, welche die Bilder speichern.

Für die Merkmale *IDMASK*, *FRESNEL* und *LIGHTMASK* gilt, dass sie als Intensitätswerte kodiert werden können, ohne dass Information verloren geht. Daher kann ein einzelner *Buffer* angelegt werden, welcher alle drei als *RGB*-Kanäle innehält (siehe Abbildung 16).

4.2.3 *Scale-Space Representation*

Wie bereits erwähnt, ist der Vergleich von zwei Patches mit $\mathcal{O}(w^2)$ eine Operation mit quadratisch wachsendem Aufwand. Der Sinn dieses Vergleichs besteht darin, im Gegensatz zu einfachen Pixelvergleichen, größere Bereiche eines Bildes miteinander zu vergleichen. Ein Patch hat quasi Informationen über die Umgebung eines Pixels. Es stellt sich die Frage, auf welche Art und Weise man einen ähnlichen Vergleich anstellen kann, welcher weniger Rechenzeit in Anspruch nimmt.
Betrachtet man die Norm der Gleichung 15, kann man folgende Formulierung aufstellen:

$$\|I^N(\boldsymbol{p}) - J^N(\boldsymbol{q})\|_2 = \sum_{i=-n}^{n} \sum_{j=-n}^{n} \|I(\boldsymbol{p} + \begin{pmatrix} i \\ j \end{pmatrix}) - J(\boldsymbol{q} + \begin{pmatrix} i \\ j \end{pmatrix})\|_2 \qquad (18)$$

wobei I und J zwei Bilder sind und $n = \lfloor \frac{w}{2} \rfloor$. Dabei sind I^N und J^N deren Merkmalsvektoren. Letztere sind eine Verkettung der Farbwerte in einem $w \times w$ Fenster um einen Bildpunkt. Dadurch lässt sich die Norm auf

die einzelnen Komponenten verteilen. Die Anzahl dieser Komponenten ist ein ausschlaggebender Faktor für die Laufzeit, wodurch die Reduzierung dieser Vergleiche positive Auswirkungen mit sich bringen kann. Alternativ kann man die Norm umformulieren, sodass sie folgendes berechnet:

$$(\sum_{i=-n}^{n}\sum_{j=-n}^{n}\|I(\boldsymbol{p}+\binom{i}{j})\|_2)-(\sum_{i=-n}^{n}\sum_{j=-n}^{n}\|J(\boldsymbol{q}+\binom{i}{j})\|_2) \tag{19}$$

Dies ist der Vergleich der nicht-normierten Mittelwerte innerhalb eines Fensters in I und J. Ein Vorteil dieser Norm wäre, dass sich die jeweiligen Werte vorberechnen lassen. Experimentell hat sich herausgestellt, dass dieses Vorgehen durchaus befriedigende Ergebnisse liefert. Allerdings wird noch ein Schritt weiter gegangen.
Inspiriert durch Verfahren aus der Bildverarbeitung[26], wird hier die *Scale-Space Representation* (kurz *SSR*) vorgestellt. Sie kann beispielsweise zur Berechnung von Bildmerkmalen verwendet werden[27]. Dabei wird ein Bildpunkt unter verschiedenen Skalierungen betrachtet. Dieser kann als ein Vektor zusammengefasst werden. Dazu nimmt man die Bildpyramidendarstellung und verwendet sie für den Merkmalsvektor:

$$I^D(\boldsymbol{p})=\begin{pmatrix} I_0(\boldsymbol{p}) \\ \vdots \\ I_l(\boldsymbol{p}_l) \\ \vdots \\ I_D(\boldsymbol{p}_D) \end{pmatrix} \tag{20}$$

wobei D die *SSR*-Tiefe und I_l eine Ebene einer Bildpyramide von I ist, deren Inhalt die weichgezeichnete Version von I ist. Außerdem gilt, dass $0 \leq l \leq D$. Das bedeutet, dass $I^D(\boldsymbol{p})$ die Konkatenation von D Pyramidenebenen aneinander ist. Damit ist hier eine Repräsentation gegeben, in der ein Mittelwert für mehrere Fenstergrößen vorhanden ist. Das bedeutet, dass man damit einen ähnlichen Vergleich wie in Formel 19 durchführen kann. Analog wird für eine Pyramidenebene folgende Notation verwendet:

$$I_l^D(\boldsymbol{p})=\begin{pmatrix} I_l(\boldsymbol{p}_l) \\ \vdots \\ I_j(\boldsymbol{p}_j) \\ \vdots \\ I_{D+l}(\boldsymbol{p}_{D+l}) \end{pmatrix} \tag{21}$$

wobei $l \leq j \leq D+l$. Das bedeutet, dass sich I_l^D aus der Verkettung der Werte von I_l bis I_{D+l} zusammensetzt.

Eine weichgezeichnete Version eines Bildes enthält den Mittelwert der näheren Umgebung eines Pixels, je nach Weichzeichnungsfilter. Das bedeutet, dass Informationen in benachbarte Regionen übergehen. Die wiederholte Anwendung eines Weichzeichnungsfilters bedeutet, dass Regionen immer weiter ineinander übergehen und Information über weite Teile des Bildes transportiert werden. Diese Eigenschaft wird hier ausgenutzt, um die Anzahl der Operationen eines Patchvergleichs stark zu reduzieren.
Während beim Merkmalsvektor $I^N(\boldsymbol{p})$ genau w^2 Zugriffe notwendig sind, verwendet man bei $I^D(\boldsymbol{p})$ genau D Zugriffe mit einer Komplexität von $\mathcal{O}(D)$. Bei einem einfachen Mittelwertfilter werden Informationen aus einer Fenstergröße von 3×3 zusammengefasst. Bei einer wiederholten Anwendung des Filters wird das gleiche Fenster wieder gefiltert. Das bedeutet, dass $I^D(\boldsymbol{p})$ mit $D = 2$ bereits Information über Pixel hat, welche in einem 9×9 Fenster in I_0 zu finden sind. Vergleicht man I^D bei $D = 3$ mit I^N und $w = 25$, hat I^N eine Reichweite von 25×25 mit mindestens 625 Zugriffen, während I^D eine Reichweite von 27×27 mit nur 3 Zugriffen hat. Jedoch ist I^N viel prägnanter, wenn man bedenkt, dass bei einer Weichzeichnung weder direktionale Information, noch hochfrequente Strukturen beachtet werden. Das Merkmal von I^D basiert allein auf den tieferen Frequenzen von I. Einschnitte in die visuelle Qualität sowie Beschleunigung der Laufzeit werden in einem späteren Abschnitt erläutert.

4.2.4 Bedingte Energiefunktion

Nachdem ein vereinfachter Merkmalsvektor definiert wurde, wird hier versucht weitere Einsparungen vorzunehmen, die sich auf die Laufzeit auswirken können. Es mag zwar nur ein kleiner Faktor sein, jedoch ist der Vergleich von zwei Patches mit dem Speicherzugriff von vier Bildern verbunden. Die Energiefunktion der Gleichung 15 besitzt den Parameter μ, welcher entweder die gerenderten Bilder A und B, oder die stilisierten Bilder A' und B' gewichtet. Da $\mu \in [0, 1]$, liegt der Wert der Energie zwischen $\|A^D(\boldsymbol{p}) - B^D(\boldsymbol{q})\|_2^2$ und $\|A'^D(\boldsymbol{p}) - B'^D(\boldsymbol{q})\|_2^2$. Für den Fall, dass $\mu = 1$ oder $\mu = 0$, ist es genau einer der beiden Terme. Das heißt:

$$E_2(A, A', B, B', \boldsymbol{p}, \boldsymbol{q}, \mu) = \begin{cases} \|A^D(\boldsymbol{p}) - B^D(\boldsymbol{q})\|_2^2 & \textit{falls } \mu = 0 \\ \|A'^D(\boldsymbol{p}) - B'^D(\boldsymbol{q})\|_2^2 & \textit{falls } \mu = 1 \\ E(A, A', B, B', \boldsymbol{p}, \boldsymbol{q}, \mu) & \textit{sonst} \end{cases} \tag{22}$$

Dies vereinfacht die Energiefunktion im Spezialfall um zwei Speicherzugriffe. Der gesamte Aufwand lässt sich damit also halbieren. Es ist zu beachten, dass diese Gleichung bereits um die Verwendung von I^D ergänzt ist.

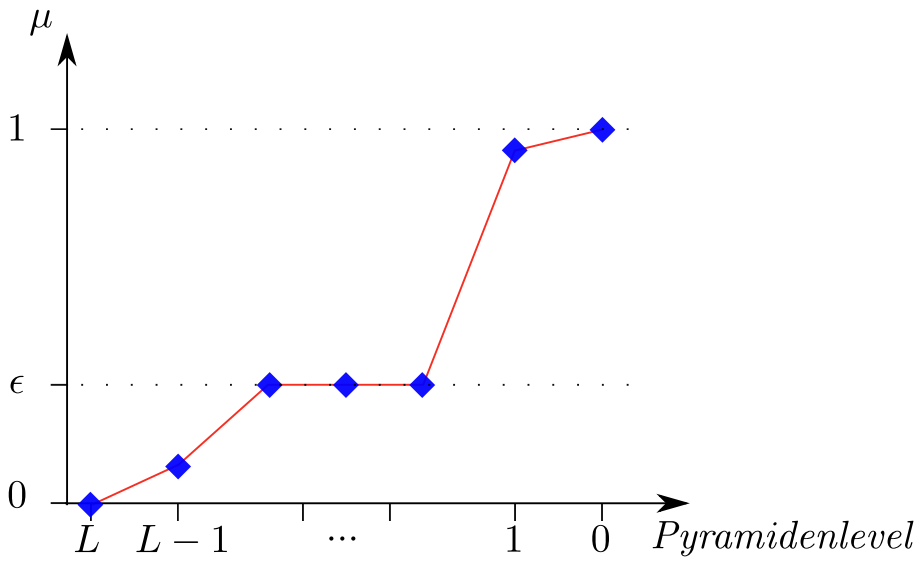


Abbildung 17: Dieser Graph zeigt die Konfiguration von μ abhängig von dem Pyramidenlevel, auf dem das Verfahren arbeitet. Da anfangs eine Zuweisung über die Renderings A und B größeren Einfluss haben, wird μ zunächst mit 0 initialisiert. Im mittleren Bereich wird es auf einen konstanten Wert ϵ gesetzt. Erst in den letzten Stufen wird $\mu = 1$ gesetzt, sodass die Struktur der Stilvorlage stärker gewichtet wird.

Beobachtet man den Verlauf des Verfahrens (siehe Abbildung 15), stellt man fest, dass die größten Veränderungen in den frühen Iterationen stattfinden. Das bedeutet, dass der Algorithmus am schnellsten in den unteren Bildpyramidenleveln konvergiert. Gleichzeitig wird dort die grobe beziehungsweise niedrig-frequente Struktur von B' manifestiert. Es hat sich empirisch herausgestellt, dass der Term $\|A^D(\boldsymbol{p}) - B^D(\boldsymbol{q})\|_2^2$ während dieser Phase den größeren Einfluss hat. Man kann sogar soweit gehen und sagen, dass der andere Term nicht relevant für diese Konvergenz ist. Für das Gewicht μ bedeutet das, dass es durchaus gleich null sein kann, ohne die Struktur von B' zu korrumpieren. Beobachtet man spätere Iterationen, im Besonderen die letzten paar, wird dort die feine, beziehungsweise hochfrequente Struktur von B' determiniert. Dort ist eine höhere Gewichtung von $\|A'^D(\boldsymbol{p}) - B'^D(\boldsymbol{q})\|_2^2$ angebracht. Sind die Regionen bezüglich ihrer Beleuchtung in A und B bereits weit genug konvergiert, kann man sie sogar vernachlässigen und nur noch auf Basis der Stilvorlage A' optimieren. Damit betont man die gemalte Struktur der Vorlage. Für diesen Fall kann man durchaus ein Gewicht von $\mu = 1$ verwenden. Im Idealfall lässt sich hier wieder die Anzahl der Speicherzugriffe halbieren. Durch diese Gegebenheit kann man einen Verlauf von μ modellieren, welcher sich auf

den aktuellen Fortschritt des Algorithmus bezieht (siehe Abbildung 17).
Die hier aufgezählten Maßnahmen dienen in erster Linie der Laufzeitoptimierung. Es liegt auf der Hand, dass hierdurch derbe Einschnitte in die visuelle Qualität stattfinden. Diese werden in späteren Abschnitten näher behandelt. Das nächste Kapitel beschäftigt sich mit der Implementierung des hier konzipierten Verfahrens. Unabhängig von der Laufzeitoptimierung wird dort ein Programm erklärt, welches dieses Verfahren ausführt.

5 Implementierung

Nachdem nun sowohl das Verfahren bekannt ist, inklusive dessen Problematiken, als auch Maßnahmen zur Beschleunigung, werden hier Details zur Implementierung genannt. Die Programmierung des Verfahrens ist in einem eigens entwickelten Framework geschrieben, wodurch Optimierungen auf tieferen Ebenen ermöglicht werden. Zu Vergleichszwecken werden die zuvor genannten Maßnahmen zur Laufzeitminimierung als austauschbares Modul modelliert. Dies hat den Vorteil, dass man die vorgeschlagenen Verbesserungen in späteren Kapiteln messen kann.
Bei der Implementierung handelt es sich um ein *C++-Framework* mit Schnittstellen zur Grafikbibliothek *OpenGL*. Mit dessen Hilfe ist es nicht nur möglich, Renderings in Echtzeit zu generieren, sondern auch *General Purpose*-Programmierung - in Form sogenannter *Compute Shader* - auf der GPU zu betreiben. Viele hochkomplexe Aufgaben lassen sich mit Hilfe von *Compute Shadern* parallelisieren. Beispielsweise sind Operationen pro Pixel oder pro Patch unabhängig voneinander und damit parallel berechenbar. Als Basis für die Implementierung werden die Versionen *C++11* und *OpenGL 4.3* verwendet. Außerdem werden *Glfw 3.0* für das Window-Management, *Glew* als Extension-Handler, *Glm* als Mathebibliothek und *ImGui* als UI-Bibliothek verwendet.

5.1 Komponenten

Das Grundgerüst ist dabei ein Hauptprogramm, welches eine Dauerschleife (den sogenannten *Mainloop*) ausführt. Hier werden in der Initialisierungsphase alle globalen Variablen, die Renderingkomponente, die UI-Komponente und die sogenannte Synthesekomponente erstellt. Alle Texturen, 3D-Modelle und Shader werden in dieser Phase erstellt, geladen oder kompiliert. Auch die Stilvorlagen werden hier geladen, sodass sie in der Anwendung benutzt werden können. Sobald eine dieser Aktionen nicht gelingt, bricht das Programm ab. Andernfalls wird eine Maske geladen, welche erste Laufzeitparameter als Nutzereingabe verlangt. Diese lassen sich über eine UI konfigurieren, bevor man die Anwendung startet. Erst wenn dies gelungen ist, kann der *Mainloop* ausgeführt werden.

5.1.1 Renderingkomponente

Die Renderingkomponente ist dafür zuständig, dass A und B zur Laufzeit erstellt werden können. Sie ist insofern geschlossen, dass sie alle Daten beinhaltet, die für den Renderingprozess benötigt werden. Ihre Hauptfunktion, welche die Szene neu rendert, schreibt Daten in eine Liste von Bildern. Diese Liste beinhaltet alle *Buffer* von A und B und kann für das Hauptverfahren eingesetzt werden.

5.1.2 Synthesekomponente

Die Synthesekomponente ist der Teil des Programmes, der das eigentliche Verfahren dieser Arbeit ausführt. Darin befinden sich folgende Funktionen: die Erstellung und das Füllen von Bildpyramiden, das *Up-* und *Downsampling* innerhalb einer Bildpyramide und einzelne Iterationen zur Stilisierung. Alle diese Operationen werden auf die GPU ausgelagert, was bedeutet, dass CPU-seitig ein sogenannter *Shader Call* aufgerufen wird. Dabei werden die Daten parallel auf der GPU verarbeitet. Von dort aus können sie gegebenenfalls auf die CPU oder zur Weiterverarbeitung auf der GPU gespeichert werden.
Die genannten Maßnahmen zur Laufzeitoptimierung sind hier als austauschbares Modul implementiert. Das bedeutet, über Vererbung und verschiedene *Compute Shader* sind Funktionalitäten so programmiert, dass sie sich leicht auswechseln lassen. Auf diese Art und Weise lassen sich unterschiedliche Ergebnisse mit Hilfe der gleichen Programmstruktur erstellen.

5.1.3 UI-Komponente

Die UI-Komponente ist die Schnittstelle zum Benutzer. Hier können Parameter der Anwendung angepasst und Funktionen aufgerufen werden. Für die Renderingkomponente bedeutet es beispielsweise, dass sich über die UI einstellen lässt, ob sie die Szene auf Knopfdruck oder in jedem Zeitschritt rendert. Nebenbei kann man die Orientierung des Modells in B verändern, um ein animiertes Bild zu erhalten. Für die Synthesekomponente können hier Parameter und Variablen konfiguriert werden. Außerdem gibt es in der UI Knöpfe, mit denen sich einzelne Operationen und Iterationen Schritt für Schritt ausführen lassen. Die UI bietet Viewports, mit denen sich unterschiedliche *Buffer* betrachten lassen.

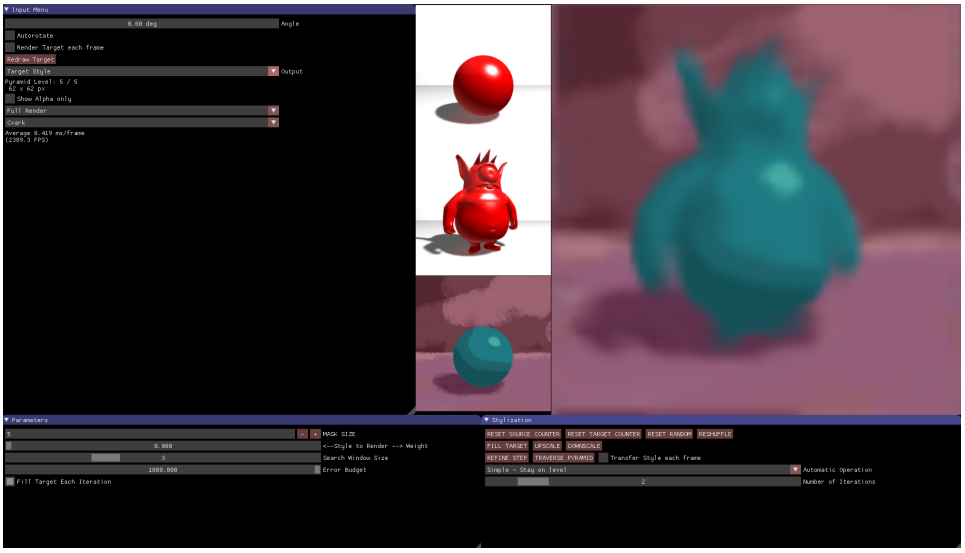


Abbildung 18: Ein Screenshot der Anwendung. Auf der rechten Seite sieht man eine Modellvorlage, ein Referenzmodell, eine Stilvorlage und eine niedrig-aufgelöste Version des Zielbilds. Der Rest der UI besteht aus Knöpfen und Reglern, welche die Anwendung konfigurieren.

5.2 GPU-Implementierung

Da hier größtenteils mit Bildern gearbeitet wird, kann die Synthesekomponente viele Operationen aus Algorithmus 5 parallel berechnen.

5.2.1 Paralleler *PatchMatch*

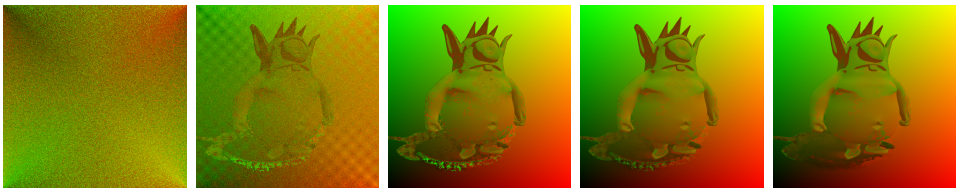


Abbildung 19: Die Implementierung des *PatchMatch* auf der GPU ist nicht weiter auf die Ausführung in *Scanline Order* angewiesen. Stattdessen wird in einer halben Iteration jeder zweite Pixel verarbeitet.

Angefangen wird mit der *Nearest Neighbor*-Suche aus Algorithmus 4. Hier ist man in der Lage, autonom für jeden Pixel eine unabhängige Suche zu starten. Da in der *Propagation*-Phase des *PatchMatch* Wert auf die *Scanline Order* gelegt wird, muss hier ein leicht abgewandelter Ansatz verfolgt werden. Eine *PatchMatch* Iteration wird hierfür in zwei Teile zerlegt.

In jeder dieser Teiliterationen wird nur jeder zweite Pixel bearbeitet (siehe Abbildung 20). Danach sind die übrigen Pixel dran, wodurch letztendlich alle abgearbeitet werden. Dies hat nicht nur den Vorteil, dass *PatchMatch* dadurch parallel läuft, sondern auch, dass die *Propagation*-Phase nicht nur Rücksicht auf Pixel einer bestimmten Richtung nehmen muss. Stattdessen kann jeder Pixel im näheren Umfeld als Referenz genommen werden, der in der vorherigen Teiliteration bearbeitet wurde.

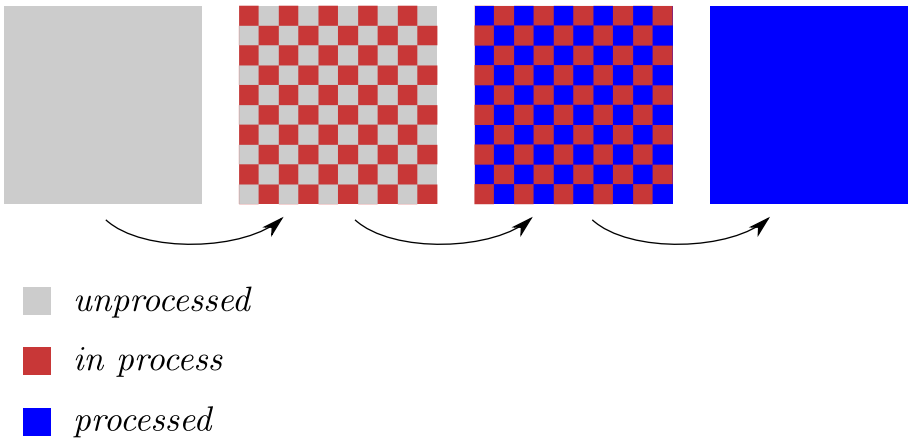


Abbildung 20: Die Aufteilung des *PatchMatch* in zwei Teiliterationen. Dabei wird abwechselnd jeder zweite Pixel optimiert.

5.2.2 Bildpyramidenerstellung

Die Berechnung einer Bildpyramide ist mit dem Weichzeichnen eines Bildes verbunden. Im Normalfall wendet man einen Filter auf ein Bild an und wiederholt den Vorgang solange, bis eine bestimmte Pyramidentiefe erreicht ist. Eine Herangehensweise wäre, einen *Compute Shader* zu implementieren, welcher diesen Filter auf ein Bild anwendet. Glücklicherweise bietet *OpenGL* bereits eine eigene Funktionalität an, die diese Aktion durchführt. Eine sogenannte *Mipmap* ist das Texturequivalent zu einer Bildpyramide. Sie wird verwendet, um beim Rendern einer weit entfernten Oberfläche dafür zu sorgen, dass Details weichgezeichnet werden und Aliasingeffekte vermieden werden. Sobald ein Bild als Textur auf der GPU gespeichert ist, kann über einen einzigen CPU-seitigem Befehl eine *Mipmap* davon erstellt werden. Da alle Berechnungen bezüglich der Eingabebilder ohnehin auf der GPU stattfinden, vereinfacht es die Implementierung und kommt nebenbei noch der Laufzeit entgegen.

5.2.3 Patch-Blending

Sobald ein *NNF* berechnet ist, kann über die *Average*-Funktion in Algorithmus 4 ein neuer Farbwert pro Pixel berechnet werden. Die Synthesekomponente beinhaltet einen Shader, welcher pro Pixel gestartet wird. Für jeden Pixel wird dann überprüft, welche Patches sich an diesem Pixel überschneiden. Da sich alle Patches in einem *Sparse Grid* befinden, ist dies kein schwieriges Unterfangen. In dieser Anordnung können es maximal vier Patches auf einmal sein. Danach werden die vier Farbwerte aus A' geladen, an der Pixelposition gemittelt und eingefügt.

5.2.4 Upsampling

Beim *Upsampling* wird der Inhalt von $B'_{l+1}(\boldsymbol{p})$ nach $B'_l(\boldsymbol{p})$ übertragen für jedes $\boldsymbol{p} \in S_{B'}$. Hier kann für jeden Pixel in B'_l ein paralleler Prozess gestartet werden, bei dem ein Farbwert aus der korrespondierenden relativen Position in B'_{l+1} ausgelesen wird, da sich bei jeder höheren Bildpyramidenstufe die Anzahl der Pixel verdoppelt. Daher muss jeder zweite Pixel in x- und y-Richtung durch bilineare Interpolation erzeugt werden.

Die Implementierung dieses Verfahrens ist eher übersichtlich, dennoch können kleine Änderungen große Auswirkung auf die Laufzeit haben. Allein die Verwendung einer GPU-Variante hat einen enormen Einfluss darauf. Die größten Auswirkungen dürften allerdings die Laufzeitoptimierungen haben. Um welche Folgen es sich genau handelt, wird im nächsten Abschnitt erörtert.

6 Ergebnisse

Im Vordergrund dieser Arbeit steht selbstverständlich in erster Linie die Erarbeitung und Implementierung eines *EBR*-Verfahrens. Zuvor wurden bereits die Grundlagen, deren konzeptuelle Erweiterungen, Verbesserungsmaßnahmen und Details der Implementierung erläutert. Hier werden nun die Ergebnisse präsentiert. Diese nehmen Bezug auf die gestellten Anforderungen. Dabei wird in jedem Punkt zwischen der einfachen und der optimierten Programmierung verglichen. Diese werden in zukünftigen Abschnitten auch so genannt. Dabei wird eine Pyramidentiefe von $L = 6$ mit zwei Iterationsschritten pro Ebene, eine Patchbreite $w = 5$ und eine *SSR*-Tiefe $D = 3$ bei einer Auflösung von $500\text{px} \times 500\text{px}$ und $1000\text{px} \times 1000\text{px}$ verwendet.

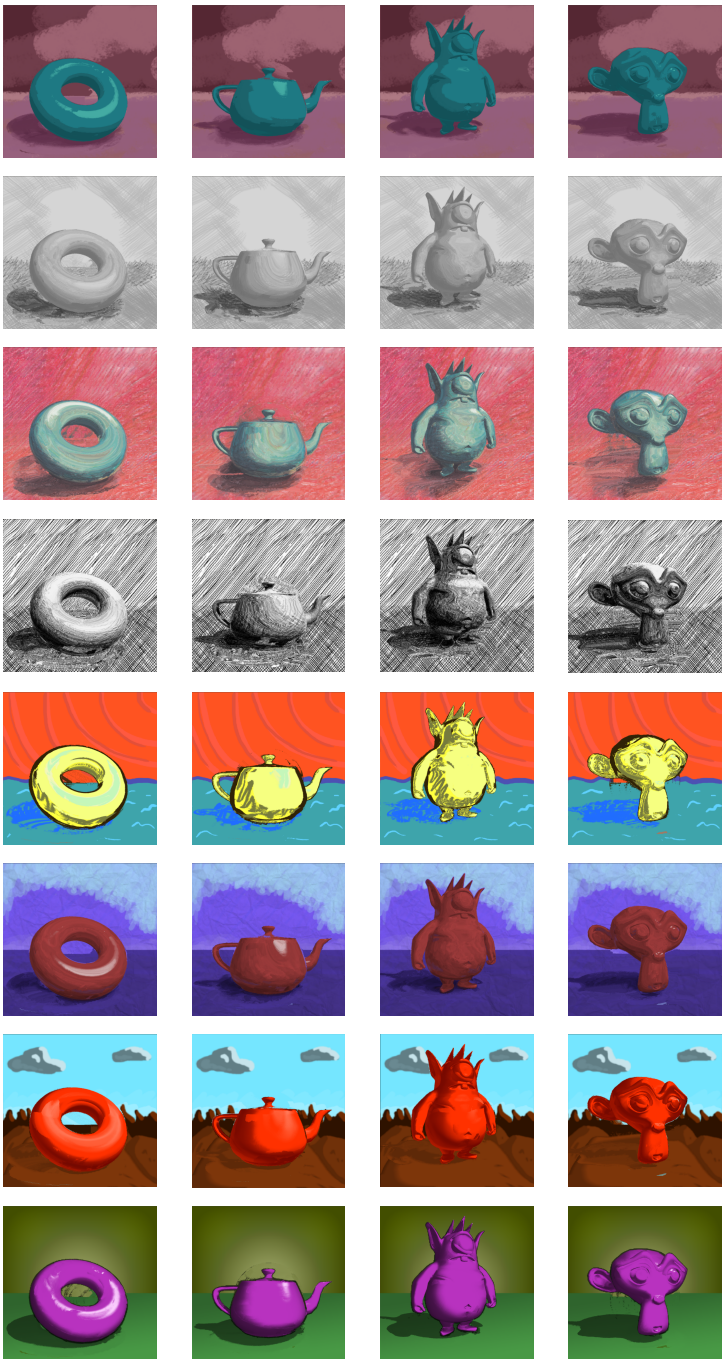


Abbildung 21: Die Ergebnisse des optimierten Verfahrens.

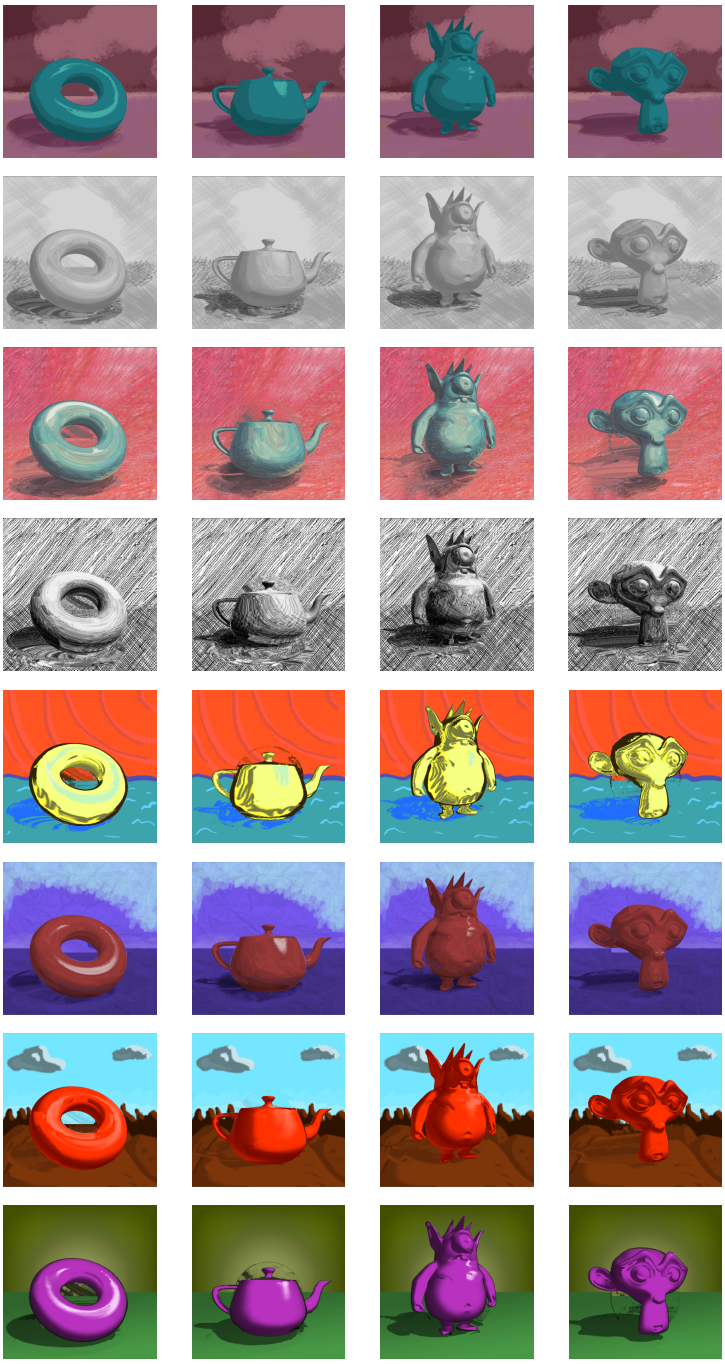


Abbildung 22: Die Ergebnisse des einfachen Verfahrens.

6.1 Echtzeitfähigkeit

Verwandte Verfahren haben in der Regel eine sehr hohe Laufzeit (siehe Abschnitt 2), um im Gegenzug hervorragende Resultate zu präsentieren. Durch die Minimierung der Laufzeit, kommt es oft zu Einschnitten in der Qualität der Ergebnisse. Bei dem Versuch das Niveau einer Echtzeitanwendung zu erreichen, kommt es ebenfalls dazu, dass sich ein deutlicher Qualitätsverlust bemerkbar macht. Um auf eine Bildrate zu kommen, die als angenehm empfunden wird, wurden hier starke Eingriffe vorgenommen. Als Basis für die hiesigen Messungen wurde eine GTX GeForce 1080 GPU zusammen mit einer Intel i7 CPU mit 6 Kernen verwendet.

	$500\text{px} \times 500\text{px}$		$1000\text{px} \times 1000\text{px}$	
Verfahren	EV	OV	EV	OV
mit Rendering	$49,87ms/f$ $(20,05fps)$	$21,69ms/f$ $(46,10fps)$	$146,75ms/f$ $(6,81fps)$	$47,93ms/f$ $(20,86fps)$
ohne Rendering	$45,81ms/f$ $(21,82fps)$	$18,36ms/f$ $(54,46fps)$	$143,93ms/f$ $(6,94fps)$	$43,01ms/f$ $(23,25fps)$

Tabelle 1: Diese Tabelle beinhaltet die Benchmarks für die beiden Auflösungen $500\text{px} \times 500\text{px}$ und $1000\text{px} \times 1000\text{px}$, für sowohl das einfache Verfahren (EV) als auch das optimierte Verfahren (OV).

Bereits durch die Betrachtung der Optimierungsmaßnahmen kann man davon ausgehen, dass das optimierte Verfahren rein laufzeittechnisch überlegen ist (siehe Tabelle 1). Neben der Berechnung selbst wird in der oberen Zeile auch noch die vorangehende Zeit für das Rendering von A und B beachtet. Beim einfachen Verfahren mit $1000\text{px} \times 1000\text{px}$ kann zwar nicht direkt von Echtzeit geredet werden, allerdings ist ein gewisses Maß an Interaktivität vorhanden. Währenddessen kann sich das optimierte Verfahren durchaus als echtzeitfähig bezeichnen. Den größten Einfluss auf die Laufzeit hat bei beiden Verfahren allerdings die Bildauflösung. Mit ihr skaliert gleichzeitig die Anzahl der *Nearest Neighbors* und damit die Anzahl der *Compute Shader-Threads*. Allerdings ist es sehr auffällig in welchem Maß die beiden Verfahren skalieren. Direkt gefolgt davon hat die Größe eines Patches w^2 größere Laufzeitauswirkungen. Davon ist allerdings hauptsächlich die einfache Implementierung betroffen, da die optimierte Implementierung unter Verwendung von I^D eher linear skaliert.

6.2 Visuelle Qualität

Bei dem hier beschriebenen Verfahren handelt es sich um eine Erweiterung der *Image Analogies*[18]. Obwohl der Grundalgorithmus gleich bleibt, wird er um Herangehensweisen und Datenstrukturen ergänzt, die zur Zeit der ursprünglichen Definition noch nicht existierten oder damit in Verbindung

gebracht wurden. *Hertzmann et al.*[18] beschreiben *Image Analogies* als befriedigend in vielen Einzelfällen, sehen es allerdings nicht als Universallösung für Stilisierung. *Jamriska et al.*[19] kritisieren, dass *Image Analogies* dazu tendieren hochfrequente Strukturen zu verlieren. Diese Eigenschaft beobachten auch *Fiser et al.*[11] und vergleichen *Image Analogies* mit einem *Greedy*-Algorithmus, aufgrund seiner schnellen und direkten Vorgehensweise.

6.2.1 Artefakte

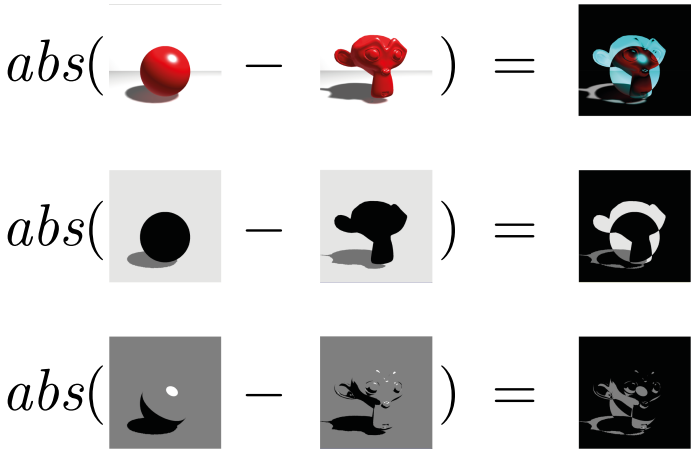


Abbildung 23: Für zwei unterschiedliche Motive, gibt es deckungsungleiche Regionen. Hier sind die Unterschiede der Modellvorlage und dem Referenzmodell zu sehen bezüglich *FULL*, *IDMASK* und *LIGHTMASK*.

Obwohl A und B eine ähnliche Szene repräsentieren, tendiert das Verfahren dazu, in bestimmten Regionen Artefakte zu bilden. Hier wird zunächst auf eine bestimmte Problematik eingegangen, nämlich die deckungsungleichen Regionen, welche besonders kritisch erscheinen. Mit deckungsungleich ist gemeint, dass sich bestimmte Bereiche in A und B stark unterscheiden und nicht deckungsgleich sind. Auf Abbildung 23 sind Differenzen zwischen den Motiven bezüglich ihrer Beleuchtung und Objekteigenschaft zu sehen. In diesen Regionen ist das Verfahren gezwungen, Vorder- und Hintergrundstrukturen zu erzeugen, die lokal nicht vorhanden sind. Die zuvor definierten Bezeichner *IDMASK* und *LIGHTMASK* beschreiben die Kodierung sehr wichtiger Bedingungen an die Synthese. Sie unterstützen beispielsweise, dass Schatten auf Schatten, Vordergrund auf Vordergrund oder Glanzpunkt auf Glanzpunkt gemappt wird. In Abbildung 24 ist zu sehen, dass deckungsungleichen Regionen die auffälligsten Artefakte aufweisen. Man sieht, dass A und B sich in den

Buffern IDMASK beziehungsweise *LIGHTMASK* am meisten unterscheiden. Davon sind sowohl das einfache als auch das optimierte Verfahren betroffen.

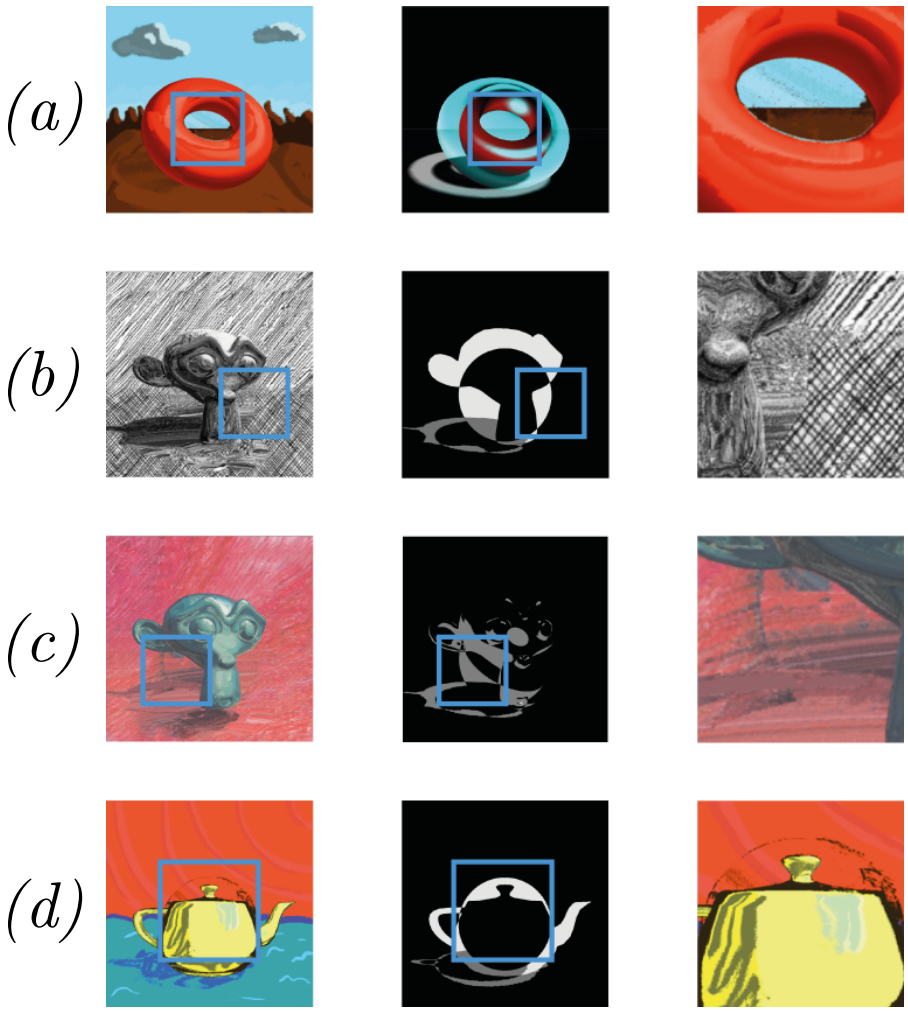


Abbildung 24: Einige Beispiele, bei denen Artefakte entstehen. Diese kann man mit den respektiven deckungsungleichen Regionen in Verbindung bringen. Bei *(a)* sieht man, dass ein Teil des Hintergrunds nicht konsistent fortgesetzt wird, *(b)* zeigt eine Schraffur, welche nicht in die benachbarten Bereiche übergeht, in *(c)* sind Vordergrundbereiche im Hintergrund zu sehen und in *(d)* zeichnet sich der Rand der Stilvorlage ab.

Ein weiteres Problem sind Regionen, an denen sich Vorder- und Hintergrund treffen. Dadurch entstehen oft dünne Strukturen entlang einer

Kante. In Abbildung 25a, 25g und 25n sind solche Strukturen zu sehen. Ein möglicher Grund dafür ist, dass ein Patch an einer Kante zwar dem Vordergrund, jedoch nicht dem Hintergrund ähnelt oder umgekehrt. Das ist auf die *greedy* Natur[11] der *Image Analogies* zurückzuführen.
Da A und B nicht zwangsläufig eine ähnliche Anzahl an Pixeln haben, welche zu einer bestimmten Region gehören, kommt es dazu, dass einige Strukturen gestaucht oder gestreckt werden (siehe Abbildung 25e, 25j, 25l und 25m). *Fiser et al.*[11] und *Jamriska et al.*[19] verwenden *Uniform Source Patch Usage*, damit diese Strukturen nicht gestreckt, sondern wiederholt werden.

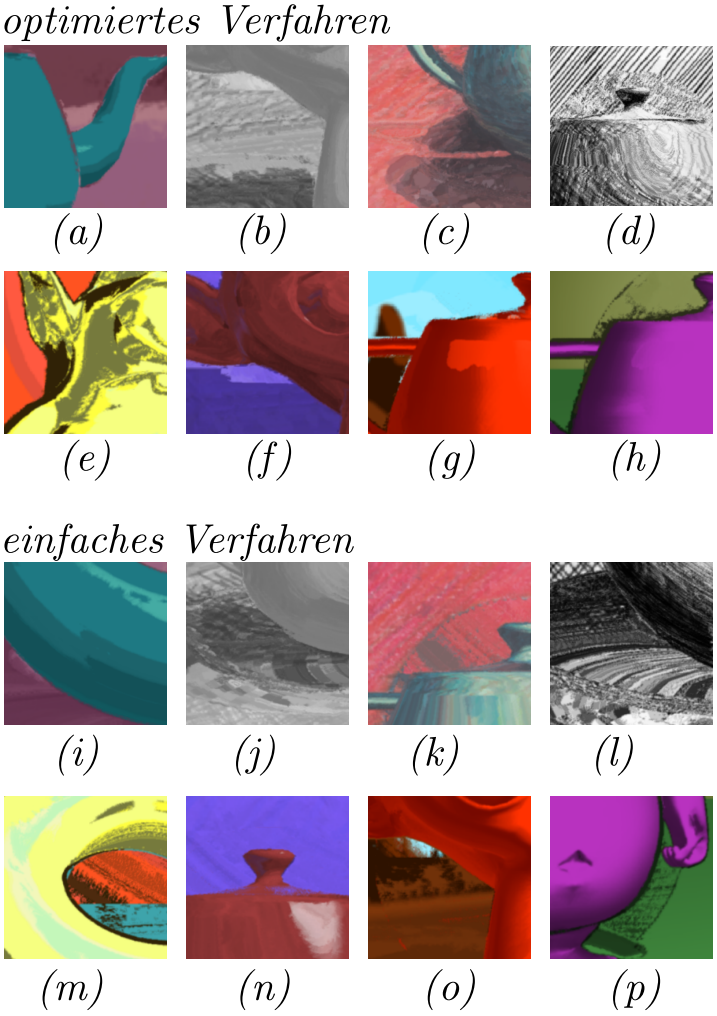


Abbildung 25: Charakteristische Artefakte des Verfahrens.

6.2.2 Vergleich

Das hier entwickelte Verfahren ist stark inspiriert durch *Stylit*[12], wozu hier ein Vergleich vorgenommen wird. Ebenso wird mit *Lit Sphere*[31] verglichen, da es sich um das einzige hier genannte *EBR*-Verfahren handelt, dass ebenfalls in Echtzeit läuft (siehe Abbildung 26).

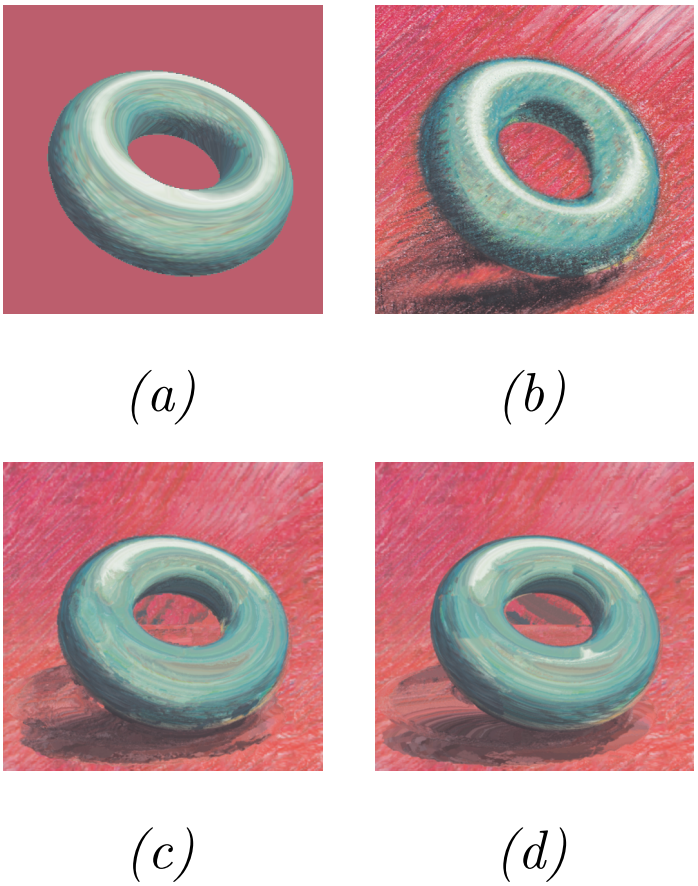


Abbildung 26: Der Vergleich unterschiedlicher Verfahren bei der Verwendung der gleichen Vorlage. *(a)* zeigt *Lit Sphere*[31]; *(b)* ist das *Stylit*-Verfahren [11]; *(c)* ist das optimierte Verfahren dieser Arbeit und *(d)* die nicht-optimierte Version. *Eigene Darstellung auf Grundlage von Fiser et al.[11] und Sloan et al.[31]*

Betrachtet man die Ergebnisse von *Stylit*, sieht man, dass es in der Lage ist, gewisse Strukturen zu reproduzieren. Deckungsungleiche Regionen werden mit kontinuierlicher Struktur fortgesetzt. Pinselstriche sind gut erkennbar und Kanten werden korrekt weitergeführt. Das hier entwickelte Verfahren hingegen hat, neben den bereits genannten Artefakten, die Eigenschaft, die Struktur aus der Stilvorlage zu verformen, sodass ein ge-

rader Pinselstrich aus der Stilvorlage im Zielbild gekrümmt erscheint.
Wenn man sich hingegen *Lit Sphere* ansieht, erkennt man, dass ebenfalls gerade Pinselstriche verformt werden. Dennoch ist es nicht in der Lage, die Hintergrundstruktur oder einen Schlagschatten zu produzieren.
Die Messung visueller Qualität ist normalerweise eine eher subjektive Aufgabe und fällt für gewöhnlich in den empirischen Bereich. Die hier genannten Aussagen basieren auf Beobachtungen des Autors und spiegeln offensichtliche Fakten über das Aussehen der Resultate wider.

6.3 Beleuchtung

Im Vergleich zu den herkömmlichen *Image Analogies* wird in dieser Arbeit ausschließlich Rücksicht auf 3D-Renderings genommen. Beachtet man die zuvor genannten Ergebnisse, erkennt man, dass im Vergleich zu *Lit Sphere* eine eher globale Beleuchtung vorhanden ist. Dies ist in erster Linie der Verwendung von *LPEs* zur Erstellung von Merkmalsvektoren zu verdanken. In Abbildung 27 wird das Ergebnis des einfachen und des optimierten Verfahrens unter dem Einfluss verschiedener Merkmale gezeigt. Dabei werden auffällige Unterschiede beim Weglassen einzelner *Buffer* aufgezählt. Die alleinige Verwendung des *Buffers FULL* $\in$ *LPE*, führt beispielsweise dazu, dass der Glanzpunkt des Vordergrundobjektes mit dem Hintergrund verwechselt werden kann. Erst durch die Benutzung anderer *LPEs* kommt es zu befriedigenden Ergebnissen. Dennoch muss man gestehen, dass unter der Verwendung des optimierten Verfahrens schon eine Vielzahl an korrekten Zuweisungen stattfinden. Wie zuvor beschrieben, kommt es bei einigen Regionen zu kritischen Artefakten. Glanzpunkte und Schattenkanten scheinen besonders davon betroffen zu sein (siehe Abbildung 26c, 26d, 28a, 25l und 25m).

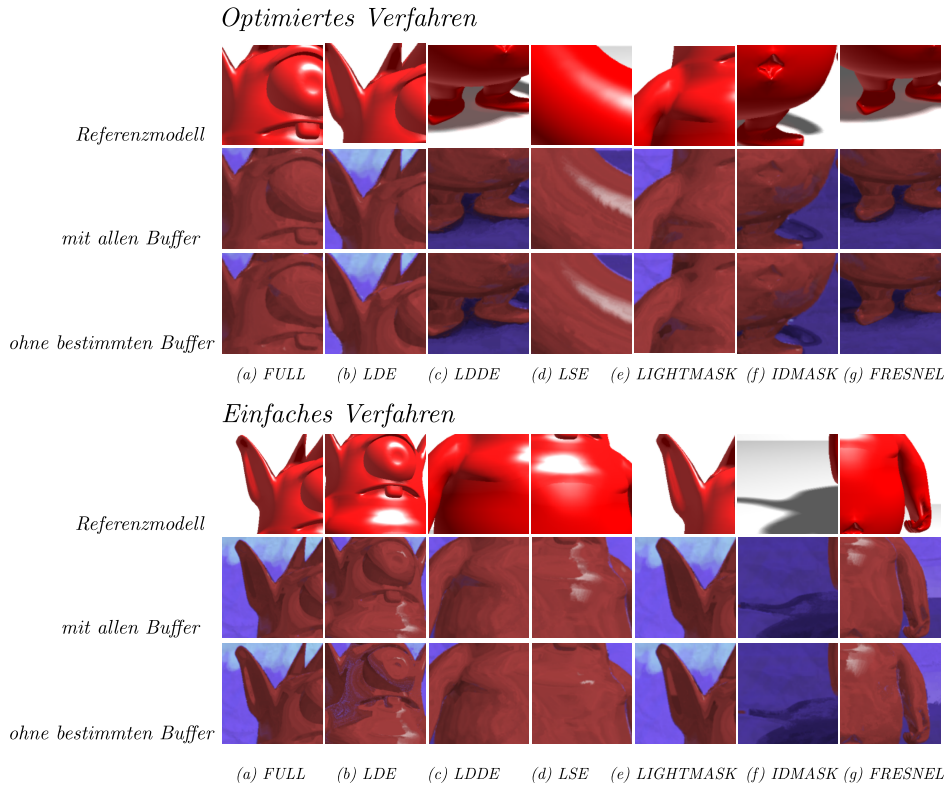


Abbildung 27: In dieser Abbildung wurden bestimmte *Buffer* weggelassen für das optimierte und das einfach Verfahren. Die Ausschnitte, die zu sehen sind, zeigen einige Bereiche in denen die stärkste Veränderung vorkommt.

Es ist sehr auffällig, dass sich zwar durch das Weglassen eines *Buffers* das Resultat verändert, allerdings bewegt sich dies in einem kleineren Rahmen. Das bedeutet, dass durchaus noch weitere Redundanzminimierung möglich wäre. Betrachtet man Abbildung 27b für das einfache Verfahren, sieht man dass der *Buffer LDE*, für das direkte Diffuse Licht, die größte Veränderung hervorruft.

6.4 Anpassungsfähigkeit

Zuvor wurde bereits erwähnt, dass *Image Analogies* dazu tendieren, bestimmte Vorlagen zu bevorzugen. Dies macht sich bei der Betrachtung unterschiedlicher Ergebnisse durchaus bemerkbar. Hochfrequente Pinselstriche und Schraffuren werden anders synthetisiert als grobe Farbkleckse (siehe Abbildung 28). Besonders auffällig sind Strukturen, die eine bestimmte Orientierung aufweisen wie beispielsweise das Hatching. Diese werden oft stark verformt und erscheinen teilweise eher als Rauschen.

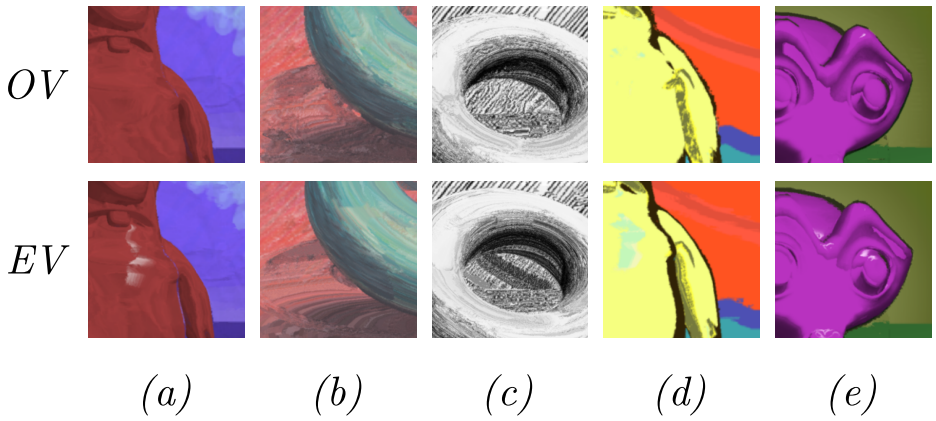


Abbildung 28: Beispiele für Strukturen, die entweder das optimierte Verfahren (OV) oder das einfache Verfahren(EV) besser synthetisiert.

Vergleicht man die zwei Verfahren unter Berücksichtigung der Vorlage, so kann man erkennen welche Strukturen die beiden hervorbringen. Das einfache Verfahren ist gut dazu in der Lage feinere Bereiche zu synthetisieren. Betrachtet man Abbildung 28d und 28e, konnten hier die Konturen der Stilvorlage besser übertragen werden, als bei dem optimierten Verfahren. Eine weitere Sache, die hervorsticht, ist die Abwesenheit eines Glanzpunktes in Abbildung 28a für das optimierte Verfahren. Für diese und auch andere Vorlagen war das einfache Verfahren besser dazu geeignet diese zu rekonstruieren. Sowohl Glanzpunkte als auch Konturen sind eher feine und hochfrequente Strukturen. Betrachtet man Abbildung 28b und 28c, dann sieht man, dass das optimierte Verfahren besser dazu in der Lage ist Oberflächen zu synthetisieren. Das einfache Verfahren scheitert oft an der kontinuierlichen Fortsetzung einer Struktur. Dies macht sich beispielsweise bei Schatten oder deckungsungleichen Regionen bemerkbar.

Die hier genannten Ergebnisse wurden mit bestem Gewissen gesammelt und analysiert. Sowohl für das einfache als auch für das optmierte Verfahren wurden die gestellten Anforderungen überprüft und dokumentiert. Unabhängig von deren Umfang, kann man jedoch den Schluss ziehen, dass es an dieser Stelle noch viel zu erforschen gibt. Der nächste Abschnitt knüpft daran an und bietet dem Leser Einblicke und Ideen, die helfen könnten sich weiter mit dem Thema zu beschäftigen.

7 Ausblick

Das Ziel dieser Arbeit war das hier genannten Stilisierungsverfahren zu implementieren und unter dem Einsatz von Echtzeitrenderings zu unter-

suchen. Die Anforderungen diktieren dabei ein gewisses Maß an Qualität in bestimmten Bereichen. Zu einem gewissen Grad konnten diese erfüllt und aussagekräftige Ergebnisse erzielt werden. Dennoch gibt es noch Verbesserungspotential. Betrachtet man die Ergebnisse von *Fiser et al.*[11], *Diamanti et al.*[9] oder *Jamriska et al.*[19], sieht man, dass besonders im Bereich der visuellen Qualität durchaus bessere Ergebnisse möglich sind. Diese Arbeit liefert zwar eine leichtgewichtige Alternative zu diesen Verfahren, jedoch kann zukünftige Forschung diesen Bereich auf weitere Laufzeitminimierungen untersuchen. In Abschnitt 6.3 wurde bereits erwähnt, dass sich das Weglassen einzelner *Buffer* nicht sehr stark auf die Qualität auswirkt. An diesem Punkt könnte man anknüpfen und untersuchen, wie man weiter *Buffer* reduzieren kann, ohne den Informationsgehalt zu verringern. Des Weiteren wurde in Abschnitt 6.4 dargestellt, dass sich sich beide Verfahren besser für bestimmte Strukturen eignen. Das wirft die Frage auf, wodurch diese Unterschiede entstehen und ob es einen Kompromiss dafür gibt. Zusätzliche Laufzeitoptimierungen erlauben der Anwendung weitere Ressourcen in die Qualität zu investieren. Ein Nutzen davon könnte die Verbesserung der Ergebnisse sein. Wie zuvor beschrieben, hängt die feine Struktur des Resultats stark von der Vorlage ab. Es ist gut möglich, dass dies nicht einmal an dem Verfahren selbst, sondern an der Wahl der Parameter liegt. Zukünftige Arbeiten könnten dokumentieren, wo die Grenzen und Auswirkungen dieser Parameter liegen.
Während diese Arbeit entstanden ist, haben *Fiser et al.*[13] eine neues Verfahren entwickelt, welche sich mit der Stilisierung von Gesichtern beschäftigt. Ein Kern-Feature dieses Verfahrens ist temporale Kohärenz. Das bedeutet, dass es die Stilisierung über eine Animation hinweg fortsetzt, ohne dabei zu rauschen. Diese Arbeit hat vollständig auf temporale Kohärenz verzichtet, jedoch wäre dies ein interessantes Forschungsthema der Echtzeitstilisierung.

8 Fazit

In dieser Arbeit geht es um die Stilisierung von 3D-Renderings unter Verwendung handgemalter Vorlagen. Es wurde ein Überblick über die Ziele und die Aufgabenstellung gegeben. Die Anforderungen umfassen die Echtzeitfähigkeit, die visuelle Qualität der Ergebnisse, die korrekte Darstellung von Licht und Schatten und die Anwendbarkeit unterschiedlicher Vorlagen. Ähnliche und verwandte Arbeiten wurden oberflächlich behandelt. Dabei wurden deren relevantesten Eigenschaften in Bezug zu dieser Arbeit aufgezählt. Auch deren Nachteile bezüglich der gestellten Anforderungen wurden genannt. Diese Arbeit setzt eine eigene Implementierung voraus. Diese wurde im Hauptteil detailliert beschrieben und deren Zusammensetzung und Verbesserungsvorschläge behandelt.

Zuvor wurde jedoch im Grundlagenteil darauf eingegangen, auf welchen Verfahren diese Arbeit basiert. Anschließend wurden die Ergebnisse präsentiert. Dabei wurde jeweils Bezug auf einen naiven Ansatz und einen optimierten Ansatz genommen. Diese wurden auf die anfangs definierten Anforderungen geprüft. Zuletzt wurde ein Ausblick verfasst, bei dem noch einmal auf die Schwächen des Verfahrens eingegangen wird. Dies dient der Formulierung weiterer Forschungsfragen.

Literatur

[1] Kelson R. T. Aires, Andre M. Santana, and Adelardo A. D. Medeiros. Optical flow using color information: Preliminary results. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 1607–1611, New York, NY, USA, 2008. ACM.

[2] Tomas Akenine-Moller, Tomas Moller, and Eric Haines. *Real-Time Rendering*. A. K. Peters, Ltd., Natick, MA, USA, 2nd edition, 2002.

[3] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, November 1998.

[4] Connelly Barnes, Eli Shechtman, Dan B Goldman, and Adam Finkelstein. The generalized PatchMatch correspondence algorithm. In *European Conference on Computer Vision*, September 2010.

[5] Pierre Bénard, Forrester Cole, Michael Kass, Igor Mordatch, James Hegarty, Martin Sebastian Senn, Kurt Fleischer, Davide Pesare, and Katherine Breeden. Stylizing animation by example. *ACM Trans. Graph.*, 32(4):119:1–119:12, July 2013.

[6] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[7] Soheil Darabi, Eli Shechtman, Connelly Barnes, Dan B Goldman, and Pradeep Sen. Image Melding: Combining inconsistent images using patch-based synthesis. *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2012)*, 31(4):82:1–82:10, 2012.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[9] Olga Diamanti, Connelly Barnes, Sylvain Paris, Eli Shechtman, and Olga Sorkine-Hornung. Synthesis of complex image appearance from limited exemplars. *ACM Transactions on Graphics*, 34(2), 2015.

[10] Jakub Fiser, Michal Lukác, Ondrej Jamri ka, Martin Cadik, Yotam Gingold, Paul Asente, and Daniel Sykora. Color Me Noisy: Example-based Rendering of Hand-colored Animations with Temporal Noise Control. *Computer Graphics Forum*, 2014.

[11] Jakub Fišer, Ondřej Jamriška, Michal Lukáč, Eli Shechtman, Paul Asente, Jingwan Lu, and Daniel Sýkora. StyLit: Illumination-guided

example-based stylization of 3d renderings. *ACM Transactions on Graphics*, 35(4), 2016.

[12] Jakub Fišer, Ondřej Jamriška, Michal Lukáč, Eli Shechtman, Paul Asente, Jingwan Lu, and Daniel Sýkora. Stylit demo webpage. `http://dcgi.fel.cvut.cz/home/sykorad/stylit`, 2017.

[13] Jakub Fišer, Ondřej Jamriška, David Simons, Eli Shechtman, Jingwan Lu, Paul Asente, Michal Lukáč, and Daniel Sýkora. Example-based synthesis of stylized facial animations. *ACM Transactions on Graphics*, 36(4), 2017.

[14] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, Apr 1980.

[15] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[16] Ryota Hashimoto, Henry Johan, and Tomoyuki Nishita. Creating various styles of animations using example-based filtering. In *Computer Graphics International*, pages 312–317. IEEE Computer Society, 2003.

[17] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '90, pages 145–154, New York, NY, USA, 1990. ACM.

[18] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 327–340, New York, NY, USA, 2001. ACM.

[19] Ondřej Jamriška, Jakub Fišer, Paul Asente, Jingwan Lu, Eli Shechtman, and Daniel Sýkora. LazyFluids: Appearance transfer for fluid animations. *ACM Transactions on Graphics*, 34(4), 2015.

[20] Alexandre Kaspar, Boris Neubert, Dani Lischinski, Mark Pauly, and Johannes Kopf. Self tuning texture optimization. *Comput. Graph. Forum*, 34(2):349–359, May 2015.

[21] Allison W. Klein, Wilmot Li, Michael M. Kazhdan, Wagner T. Corrêa, Adam Finkelstein, and Thomas A. Funkhouser. Non-photorealistic virtual environments. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 527–534, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[22] J. Kruger and R. Westermann. Acceleration techniques for gpu-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, pages 38–, Washington, DC, USA, 2003. IEEE Computer Society.

[23] Vivek Kwatra, Irfan Essa, Aaron Bobick, and Nipun Kwatra. Texture optimization for example-based synthesis. *ACM Transactions on Graphics, SIGGRAPH 2005*, August 2005.

[24] Adam Lake, Carl Marshall, Mark Harris, and Marc Blackstein. Stylized rendering techniques for scalable real-time 3d animation. In *Proceedings of the 1st International Symposium on Non-photorealistic Animation and Rendering*, NPAR '00, pages 13–20, New York, NY, USA, 2000. ACM.

[25] Nils Lichtenberg, Noeska Smit, Christian Hansen, and Kai Lawonn. Sline: Seamless line illustration for interactive biomedical visualization. In *Proceedings of the Eurographics Workshop on Visual Computing for Biology and Medicine*, VCBM '16, pages 133–142, Goslar Germany, Germany, 2016. Eurographics Association.

[26] Tony Lindeberg. Scale-space theory: A basic tool for analysing structures at different scales. *Journal of Applied Statistics*, pages 224–270, 1994.

[27] Tony Lindeberg. Image matching using generalized scale-space interest points. *J. Math. Imaging Vis.*, 52(1):3–36, May 2015.

[28] Jason Mitchell, Moby Francke, and Dhabih Eng. Illustrative rendering in team fortress 2. In *Proceedings of the 5th International Symposium on Non-photorealistic Animation and Rendering*, NPAR '07, pages 71–76, New York, NY, USA, 2007. ACM.

[29] Karl Pearson. On lines and planes of closest fit to systems of points in space. *Phil. Mag.*, 6(2):559–572, 1901.

[30] Christophe Schlick. An inexpensive brdf model for physically-based rendering. *Computer Graphics Forum*, 13:233–246, 1994.

[31] Peter-Pike J. Sloan, William Martin, Amy Gooch, and Bruce Gooch. The lit sphere: A model for capturing npr shading from art. In *Proceedings of Graphics Interface 2001*, GI '01, pages 143–150, Toronto, Ont., Canada, Canada, 2001. Canadian Information Processing Society.

[32] Blender Documentation Team. Blender manual 2.79. `https://docs.blender.org/manual/en/dev/editors/3dview/properties/shading.html`, 2017.

[33] Yonatan Wexler, Eli Shechtman, and Michal Irani. Space-time completion of video. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(3):463–476, March 2007.

[34] Georges Winkenbach and David H. Salesin. Computer-generated pen-and-ink illustration. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 91–100, New York, NY, USA, 1994. ACM.

[35] Alexander Ecker Łukasz Kidziński. Deepart.io. `https://deepart.io/latest/`, 2017.