UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

Stylized Image Triangulation

Masterarbeit

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

vorgelegt von

Christoph Moritz Löhne

Erstgutachter:	J.-Prof. Dr. Kai Lawonn (Institut für Computervisualistik, AG Medizinische Visualisierung)
Zweitgutachter:	Nils Lichtenberg (Institut für Computervisualistik, AG Medizinische Visualisierung)

Koblenz, im Juli 2019

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Vereinbarung der Arbeitsgruppe für Studien- und Abschlussarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. ja ☐ nein ☐

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu. ja ☐ nein ☐

Koblenz, den 23. Juli 2019

Kurzfassung

Die stilisierte Triangulierung ist ein beliebtes Stilmittel bei der Abstraktion von Bildern. Ergebnisse sind auf Covern von Magazinen zu finden oder als Kunstwerk zu kaufen. Eingesetzt wird diese Stilisierung auch bei mobilen Anwendungen oder gar bei Programmen, die sich ausschließlich mit der automatisierten Triangulation befassen. Diese Arbeit basiert auf einer Veröffentlichung, die die adaptive dynamische Triangulierung als Optimierungsproblem versteht und damit, hinsichtlich der visuellen und technischen Qualität, neue Ergebnisse erzielt. Ziel dieser Arbeit ist es, dieses Verfahren möglichst vielen Nutzern zugänglich zu machen. Dazu wird eine mobile Anwendung - Mesh - entworfen und umgesetzt. Ein Host-Client System wird entwickelt, um die ressourcenbedürftige Berechnung nicht auf dem mobilen Endgerät ausführen zu müssen. Im Zuge dessen wird das Verfahren für die CPU portiert und zusätzlich ein Webserver entwickelt, der die Kommunikation zwischen dem Triangulierungsverfahren und der mobilen Anwendung herstellt. Die App «Mesh» bietet die Möglichkeit, ein beliebiges Bild zu dem Server zu senden, das nach der Bearbeitung heruntergeladen werden kann.
Ein Forschungsaspekt der Arbeit thematisiert die Optimierung des Verfahrens. Dafür wird der Gradientenabstieg, der die Energieminimierung durchführt, anhand verschiedener Ansätze untersucht. Die Einschränkung der Schrittmöglichkeiten, diagonale Schrittrichtungen und eine dynamische Neupositionierung werden getestet. Es zeigt sich, dass sich bei diagonaler Schrittrichtung, anstatt horizontaler und vertikaler, keine Verbesserung verzeichnen lässt. Die Einschränkung der Schrittrichtung, dass ein Punkt seine vorherige Position nicht erneut einnehmen kann, verursacht einen Verlust an optischer Qualität. Jedoch wird der globale angestrebte Approximationsfehler in kürzerer Zeit erreicht. Die vektorbasierte Variante der flexiblen Schrittrichtung resultiert mit längerer Berechnungszeit in qualitativ hochwertigeren Ergebnissen, sodass ästhetischere Resultate erzielt werden.
Ein weiterer Bestandteil dieser Arbeit setzt sich mit der Imitation eines Kunststils auseinander. Die Werke von Josh Bryan dienen als Inspiration. Mittels eines *GLSL-Shaders* soll durch die Verwendung von Pseudozufälligkeit ein natürlicheres Aussehen einer schraffierten Triangulierung erreicht werden. Ergebnisse zeigen, dass der Ansatz Möglichkeiten der Verbesserung aufweist, dass jedoch eine präzisere Triangulierung für eine hochwertige Imitation notwendig ist. Als letzter Bestandteil wird ein Renderstil präsentiert, der ausgehend von einem beliebigen Ausgangspunkt, die Dreiecke der Triangulation versetzt, sodass Lücken entstehen. Durch die freie Wahl des Zentrums des Effekts, ist ein Einsatz bei Animationen denkbar.

Abstract

Stylized image triangulation is a popular tool of image processing. Results can be found on magazine covers or bought as a piece of art. Common use cases are filters by mobile apps or programs dedicated to automated triangulation. This thesis is based upon a paper that achieves new results formulating the adaptive dynamic triangulation as optimization problem. With this approach, new results concerning visual and technical quality are accomplished. One aim of this thesis is to make this approach accessible to as many users as possible. To reach users, a mobile app called Mesh is designed and implemented. A client-host-system is presented which relieves the app from computing the result requiring a lot of resources. Therefore, transferring the approach to a CPU based solution is part of the thesis. Also, a webserver is implemented that handles the communication between app and algorithm. "Mesh" enables the user to send a arbitrary image to the server whose result can be downloaded.
Part of the research deals with optimizing the method. As the main step, the gradient descent method, which minimizes an approximation error, is examined with three different approaches re-defining the movement of a point: The limitation of the directions of movement in a meaningful manner, diagonal directions and a dynamically repositioning of points are analyzed. Results show no improvement of visual quality using diagonal instead of horizontal and vertical steps. Disallowing a point to take its last position, the limitation of step opportunities results in a loss of visual quality but reaches an intended global error earlier. The dynamically repositioning rests upon a vector-based solution that weights the directions and applies a factor to each of them. This results in a longer computational time but also in a higher visual quality.
Inspired by the work of Josh Bryan, another part of research aims at imitating an artists style. With the use of pseudo-random events combined with a *geometryshader*, a more natural look shall be achieved. This method illustrates a way of adding minor details to a rendering. To imitate an artists work, a more complex and more precise triangulation is needed. As the last aspect, a renderstyle is presented. The style uses a center for its effect moving the triangles of a triangulation apart. The arbitrary choice of placing the centrum enables the renderstyle to be used in animations.

Inhaltsverzeichnis

1 Einleitung

Die künstlerische Abstraktion von Bildern ist ein bedeutender Bestandteil von häufig genutzten Anwendungen. Apps, wie Instagram mit monatlich 1 Mrd. aktiven Nutzern weltweit (Stand Juli 2019) [32], begleiten mit ihren Funktionen den Alltag vieler. Eine ihrer Kernfunktionen ist die Anwendung von Filtern auf Bilder. Dabei reicht die Auswahl von Manipulationen der Farben, über zusätzliche projizierte Inhalte, bis zur kompletten Abstraktion eines Bildes.
In dieser Arbeit wird solch eine Abstraktion umgesetzt. Die stilisierte Triangulierung ist ein beliebtes Stilmittel. Dabei wird das Eingabebild mosaikähnlich in Dreiecke unterteilt, mit dem Ziel, das Eingabebild möglichst detailgetreu wiederzugeben. Dieser Stil wird ebenso von Künstlern verwendet, deren Werke beispielsweise für Magazincover verwendet werden [13].
Ein neues Verfahren [LG18] erzielt in diesem Gebiet Ergebnisse, die von handgefertigten Werken nicht zu unterscheiden sind und diese sogar in technischer Hinsicht übertreffen. In dieser Arbeit wird dieser Ansatz der stilisierten Triangulation behandelt. Ein beispielhaftes Eingabe- und Ergebnisbild sind in Abbildung 1 dargestellt.

(a) Eingabebild **(b)** Ergebnisbild

Abbildung 1: Beispiel der stilisierten Triangulation von Lawonn und Günther

Die Triangulierung bietet in der Informatik ein weites Spektrum an Einsatzgebieten. Viele Echtzeitanwendungen arbeiten mit Dreiecken bei dem Rendervorgang. Die Erarbeitung und Darstellung einer Oberfläche einer Punktemenge kann durch verschiedene Triangulationsverfahren erreicht werden. Mit der Delaunay-Triangulierung wird ein gleichmäßig geformtes Dreiecksnetz erstellt, das für die Weiterverarbeitung, wie die Modellierung oder den 3D-Druck, benutzt werden kann. Aufbauend auf einer solchen Triangulierung, wird ein Bild mit dieser Anwendung erstellt. Zusätzlich werden verschiedene Stile der Abstraktion untersucht, um ein optisch ansprechendes Ergebnis zu erzielen.
Die Werke des Künstlers Josh Bryan dienen als Inspiration für einen Stil der Triangu-

lierung, dargestellt in Abbildung 2.

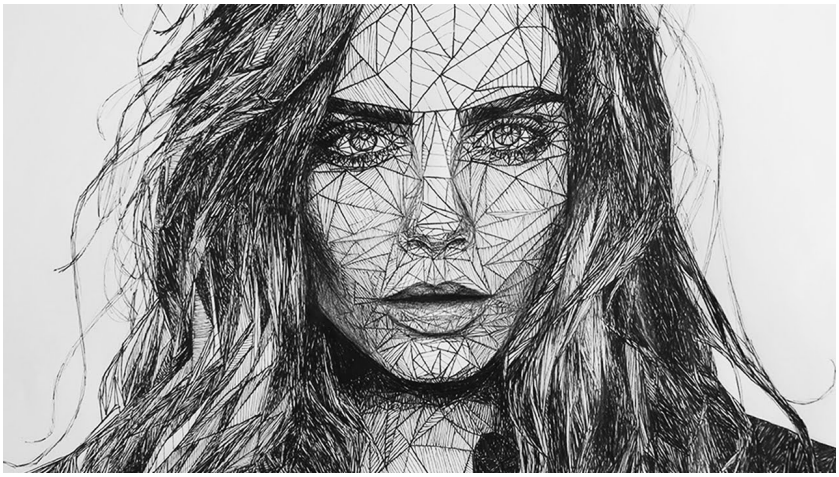


Abbildung 2: Inspiration: ein Werk von Josh Bryan

1.1 Motivation und Ziel

Ziel dieser Arbeit ist es, einen Zugang zu dem neuen Ansatz der stilisierten Triangulation zu schaffen. Aus diesem Grund ist ein Hauptbestandteil dieser Arbeit, die Entwicklung einer mobilen Anwendung - einer App. Die Anwendung - Mesh - hat die Anforderung, dass möglichst vielen Nutzern der Zugang zu dem neuen Verfahren ermöglicht wird. Sie können sich inspirieren lassen oder selbst künstlerisch aktiv sein, was durch die Stilisierungen unterstützt wird.
Für den Gebrauch der App sollen wenige Voraussetzungen gelten. Es wird ein Host-Client System entwickelt, sodass dem Benutzer leistungsstarke und oftmals teure Hardware indirekt bereitgestellt wird. Einzige Voraussetzung ist ein Endgerät mit Internetzugang. Der Nutzer tritt über das mobile Endgerät mit einem Server in Verbindung, der ein Bild als Eingabe entgegen nimmt, die Verarbeitung veranlasst und die Rücksendung des Ergebnisses an den Nutzer umsetzt. Das Host-Client System stellt eine Alternative zur Anwendunng dar, die die Ressourcen des mobilen Endgeräts nutzt. Die in dieser Arbeit umgesetzte Option bietet die Möglichkeit, das Endgerät weiterhin nutzen zu können, während das Bild bearbeitet wird.
Neben der Umsetzung des Verfahrens für die CPU, ist die Untersuchung von Optimierungsansätzen ein Bestandteil dieser Arbeit. Ziel der Optimierung ist die Steigerunng der Performanz, um die Berechnungszeit so gering, wie möglich zu halten.
Aufbauend auf der Veröffentlichung von Lawonn und Günther [LG18], wird ein neuer Ansatz untersucht, der den Stil von Josh Bryan imitieren soll. Der Ansatz verfolgt das Ziel, einen Stil zu erhalten, der einem handgefertigten Werk ähnelt. Die Natürlichkeit der Imperfektion soll durch Pseudozufälligkeit bei der Zeichnung von Kanten erreicht

werden. Es wird untersucht, ob es möglich ist, mittels eines Algorithmus den künstlerischen Stil zu imitieren.

1.2 Gliederung

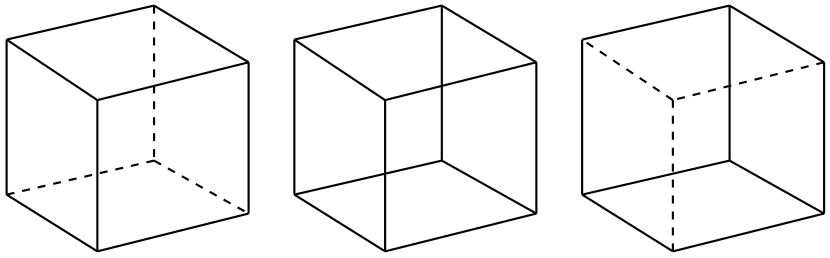
Im folgenden Kapitel werden Grundlagen bezüglich einer Triangulierung, deren Stilisierung, sowie deren Einbettung in ein Host-Client System mit App und Server gegeben. In Kapitel 3 wird auf themenbezogene Arbeiten und Anwendungen eingegangen, in denen alternative Umsetzungen der Thematik vorgestellt werden. Diese werden anhand des vermittelten Grundwissens aus Kapitel 2 analysiert und bewertet.
Das erarbeitete Konzept der einzelnen Komponenten der Anwendung wird in Kapitel 4 erläutert, während sich das darauf folgende Kapitel mit der Implementation des Konzepts befasst. Es folgt im 6. Kapitel die Präsentation und Bewertung der erzielten Ergbenisse. Abgeschlossen wird diese Ausarbeitung mit dem Fazit und dem Ausblick in Kapitel 7.

2 Grundlagen

Das Kapitel der Grundlagen bietet einen Überblick über alle Themen, die für die Arbeit relevant sind. Grundlegendes Wissen, Funktionsweisen, auf denen die umgesetzte Anwendung basiert und einen Ausblick im jeweiligen Themengebiet werden vorgestellt.
Beginnend mit dem Themenbereich der Datenstruktur, wird zur Topologie eines Dreiecksnetzes übergegangen. Anhand dieser Grundlagen soll deutlich werden, welche Verfahren benötigt werden, um eine dynamische Änderung einer Datenstruktur umzusetzen.
Ein Einblick in das nicht-photorealisitische Rendern eines Bildes wird anschließend gewährt. Dazu werden exemplarisch Verfahren und Ergebnisse vorgestellt, um ein Verständnis von Ziel und Möglichkeiten der Abstraktion zu vermitteln.
Die abschließenden Unterkapitel behandeln den Aufbau von Android-Anwendungen und die grundlegende Funktionsweise eines Webservers.
Mit den Informationen aus diesem Kapitel, soll das Verständnis und die Bewertung von themenbezogenen Anwendungen möglich sein. Das darauf aufbauende Konzept dieser Arbeit und die Umsetzung sollen nachvollzogen werden können.

2.1 Datenstrukturen

Eine Datenstruktur dient dazu, die vorhandenen, zu verarbeitenden Daten im Sinn der Anwendung zu speichern und zugänglich zu machen. Die Art der Datenrepräsentation und der Speicherstruktur sind variabel und weisen unterschiedliche Eigenschaften auf, wodurch es abzuwägen gilt, welche Struktur für den jeweiligen Anwendungszweck benötigt wird.
Der Anwendungszweck in dieser Anwendung ist die dynamische Veränderung einer geometrischen Struktur. Beispielweise weist das Szenario der Verwendung von volumetrischen Daten den Anspruch an eine Datenstruktur auf, die speichereffizient arbeitet [Ger07].
In dem Bezug der geometrischen Struktur wird der Begriff *Topologie* eingeführt. Er beschreibt den Aufbau von Geometrie: Punkte und Kanten schließen gemeinsam Oberflächen ein. Sogenannte *Faces*, entstehen.
Ein Polygon besteht aus diesen drei Bestandteilen und nimmt dabei beliebige Formen an [Zdu06]. Das kleinst mögliche Polygon beschreibt ein Dreieck, bestehend aus drei Eckpunkten (Vertices), drei Kanten und einer Fläche. Ebenso kann ein Polygon mit einer Fläche durch mehrere Kanten definiert sein.
Bei der visuellen Darstellung ist es möglich, nur Kanten zu verwenden, um das «Gerüst» zu verbildlichen; ein Drahtgittermodell entsteht. Alternativ werden *Faces* für eine geschlossene Darstellung der Oberfläche genutzt.
Vorteil des Drahtgittermodells ist die Möglichkeit der Transparenz des Modells, wodurch eine Durchsicht ermöglicht wird. Allerdings entstehen dadurch, wie in Abbildung 3 gezeigt, Mehrdeutigkeiten der sichtbaren Elemente [Fra00]. Dieser Vor- und Nachteil kehren sich bei der Darstellung von Polygonen mit geschlossenen Flächen um.



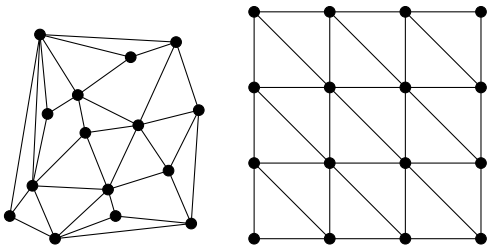
(a) 1. mögliche Interpretation **(b)** Das Drahtgittermodell **(c)** 2. mögliche Interpretation

Abbildung 3: Veranschaulichung der Mehrdeutigkeit von Drahtgittermodellen. Die gestrichelten Linien befinden sich auf der Rückseite des Modells

Im Folgenden wird hinsichtlich der stilisierten Triangulierung das Polygonnetz vorgestellt. Aufbauend auf dieser Struktur, folgt der Übergang zum Dreiecksnetz. Das Themengebiet der Triangulierung umfasst Eigenschaften der gesamten Struktur und einzelner Dreiecke. Es werden Methoden vorgestellt, die die dynamische Verwaltung und Erweiterung einer geometrischen Struktur umsetzen. Zusätzlich werden Auswirkungen auf Performanz und Speicherbedarf bei der Traversierung der Daten erläutert.

2.1.1 Polygonnetz

Das Polygonnetz besteht aus einem Zusammenschluss von multiplen Polygonen und ist eine grundlegende Art, Punkte miteinander zu verbinden, um eine Struktur innerhalb der Daten zu erstellen [Zdu06]. Dabei kann die aufgebaute Repräsentation unstrukturiert erscheinen, wie links in Abbildung 4, oder mit deutlichen Strukturvorgaben, dargestellt in Abbildung 4b.



(a) Polygonnetz ohne Eigenschaften **(b)** Einheitsnetz mit strikten Vorgaben

Abbildung 4: Auswirkung der Eigenschaften eines Polygonnetzes

2.1.2 Triangulierung

Eingeschlossen in der Rubrik der Polygonnetze, ist die Triangulierung, die Repräsentation der Daten durch Dreiecke. Analog zu dem Polygonnetz, wird die Oberfläche durch eine Anordnung von Punkten und Kanten, resultierend in Dreiecken, erreicht.
Die Topologie einer Triangulierung kann durch eine Liste von Vertices ausgedrückt werden:

$$\mathcal{V} = \{v_1, \ldots, v_V\}$$

Von der jeweils drei Vertices die Dreiecke definieren, die, zusammengesetzt, wiederum die Gesamtoberfläche darstellen [BKP+10], vgl. Gleichung 1. Bei dieser Repräsentation werden Punkte mehrfach gespeichert und Kanten wiederholt, wodurch ein zusätzlicher Speicheraufwand entsteht.

$$\mathcal{F} = \{f_1, \ldots, f_F\}, \quad f_i \in \mathcal{V} \times \mathcal{V} \times \mathcal{V} \tag{1}$$

Ebenso kann die Triangulierung durch Kanten dargestellt werden [BKP+10], vgl. Gleichung 2. In dieser Darstellung wird eine Mehrfachspeicherung der Punkte beschränkt.

$$\mathcal{E} = \{e_1, \ldots, e_E\}, \quad e_i \in \mathcal{V} \times \mathcal{V} \tag{2}$$

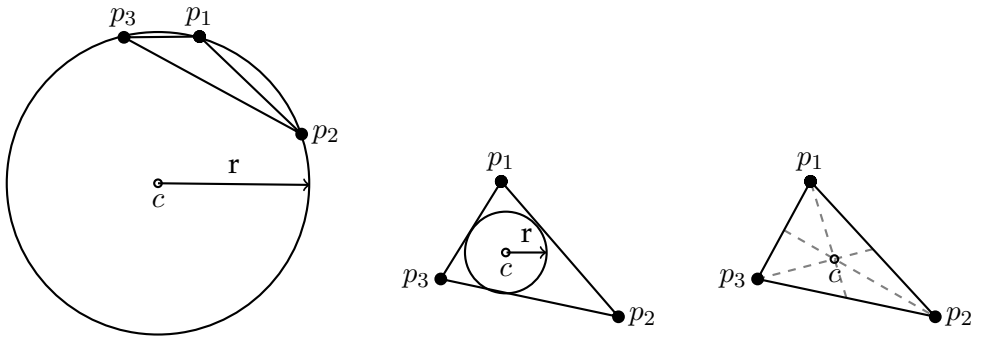
$\mathcal{V}$, $\mathcal{F}$ und $\mathcal{E}$ beinhalten die gesamte Anzahl der jeweiligen Entität. Bei der Definition von f_i und e_i muss berücksichtigt werden, dass $\mathcal{V}$ jeweils distinkte Elemente der Vertexliste sind.

Es werden zwei Domänen vorgestellt, die eine inhaltliche Trennung entstehen lassen. Dabei wird zwischen geometrischen Daten der Punkte und der Topologie - der Anordnung der Geometrie - unterschieden. Die Koordinaten der Punkte sind ohne Redundanzen in einer Liste gespeichert. Die darzustellende Struktur, die Topologie, wird in einer seperaten Liste gespeichert, in der Zeiger auf die Punkte enthalten sind. Dadurch wird der Speicherbedarf gering gehalten, da keine Koordinaten mehrfach gespeichert werden. Als Nachteil dieser Methode ist aufzuführen, dass nur mit Suchaufwand zu ermitteln ist, wie geometrische und topologische Bestandteile miteinander verbunden sind [A.P09]. Diese Eigenschaft besteht bei der redundanten Speicherung der Dreiecke ebenfalls.
Wird eine Kantenliste, etabliert in Definition 2, hinzugefügt, definieren sich Oberflächen durch Zeiger auf Kanten. Dadurch wird die Beziehung der geometrischen Daten hergestellt und der bei Punktelisten genannte Nachteil des nötigen Suchaufwands aufgehoben. Bestehen bleibt der Nachteil, dass nur durch Suchaufwand die Nachbarschaft von Flächen ermittelt werden kann [FDVVD+96].
Komplexere Strukturen ermöglichen Abfragen im Sinne der Nachbarschaft von Flächen und Kanten. Die *Winged Edge* Struktur speichert zusätzlich gerichtete Nachbarkanten, was die Ermittlung von adjazenten Flächen ermöglicht [TM19].
Weiterführende Methoden zeigen auf, wie Datenstrukturen hinsichtlich Performanz optimierbar und mit Nutzungsmöglichkeiten erweiterbar sind. Beispielhaft wird in der

Doktorarbeit [Bog11] erläutert, wie 3D Modelle inkrementell um Kanten, Flächen und Körper erweiterbar sind.
Für diese Arbeit soll an dieser Stelle ein ausreichendes Verständnis der Datenstrukturen geschaffen worden sein, sodass nicht detaillierter darauf eingegangen wird.

Für den folgenden Abschnitt sind markante Punkte eines Dreiecks für das Verständnis des Verfahrens wichtig. Der Umkreis eines Dreiecks und dessen Mittelpunkt sind essenziell für die Delaunay-Triangulierung [CDS16]. Auf diesem Kreis befinden sich alle drei Punkte des Dreiecks, dargestellt in Abbildung 5a, wodurch der Abstand vom Umkreismittelpunkt zu jedem der Punkte gleich des Radius des Kreises ist. Der Mittelpunkt des Umkreises wird aus den drei sich schneidenden Mittelsenkrechten der Seiten gebildet [MF06]. Bei einem stumpfwinkligen Dreieck kann der Mittelpunkt des Umkreises außerhalb des Dreiecks liegen. Diese Eigenschaft findet bei der Delaunay-Triangulierung Bedeutung.

Ein weiterer wichtiger Punkt für diese Anwendung ist der Schwerpunkt eines Dreiecks. Dieser wird durch die drei Seitenhalbierenden des Dreiecks gebildet [MF06]. Er befindet sich innerhalb des Dreiecks. Mit der Formel $\frac{p_0+p_1+p_2}{3} = p_{schwerpunkt}$ lässt er sich mit bekannten Koordinaten berechnen.
Als Ausblick wird an dieser Stelle der Inkreis eines Dreiecks genannt, dargestellt in Abbildung 5b. Die drei Winkelhalbierenden schneiden sich im Zentrum des Inkreises [MF06]. Der Kreis tangiert alle drei Seiten des Dreiecks.



(a) Umkreis mit Kreismittelpunkt c **(b)** Inkreis. Darstellung ohne Winkelhalbierende **(c)** Schwerpunkt c des Dreiecks

Abbildung 5: Bedeutende Punkte eines Dreiecks

Liegen Koordinaten ohne weitere Topologie vor, gibt es Verfahren, um eine Triangulierung zu erstellen. Bekannt dafür ist die Delaunay-Triangulierung [LS80], die sowohl im 2D als auch im 3D ein Dreiecksnetz generieren kann.
Betrachtet wird im Folgenden die grundsätzliche Funktionsweise des Verfahrens für den 2-dimensionalen Fall. Die Erweiterung in die dritte Dimension wird nicht erläutert, da sie in dieser Arbeit nicht umgesetzt wird.

Ausgehend von einer Menge von Punkten, wird bei der Delaunay-Triangulierung ein Startpunkt gewählt. Von diesem Punkt werden inkrementell nächste Punkte gesucht, um Dreicke zu bilden. Es werden Kanten zwischen den Punkten erstellt, sodass möglichst große Innenwinkel in den Dreiecken entstehen. Dafür wird die Bedingung eingehalten, dass im Umkreis jedes Dreiecks kein weiterer Punkt liegt [LS80]. Im Fall, dass mehrere Punkte direkt auf dem Umkreis des Dreiecks liegen, stellt die Mehrdeutigkeit kein Problem dar. Beide Lösungen sind valide.
Analog zu der Delaunay-Triangulierung repräsentiert das Voronoi-Diagramm eine Punktemenge. Das Diagram setzt sich aus mehrkantigen Waben zusammen, die den jeweiligen maximalen überschneidungsfreien Raum zwischen den Punkten darstellt.
Delaunay-Triangulierung und Voronoi-Diagramm gelten dabei als austauschbar [LS80]: Die Mittelsenkrechten der Kanten im Voronoi-Diagram ergeben die Dreiecke in der Delaunay-Triangulierung, ersichtlich in Abbildung 6 [19].

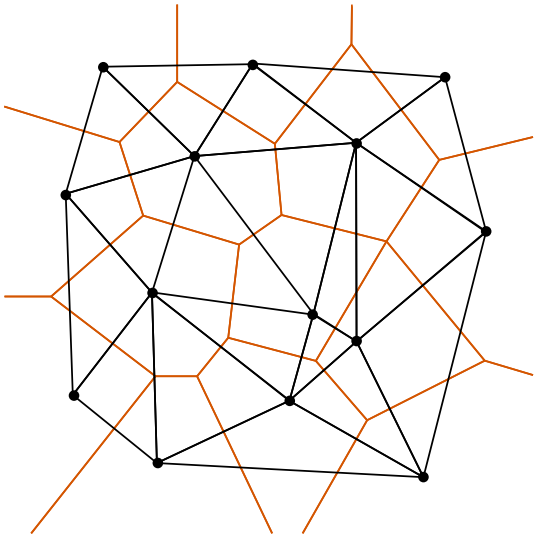


Abbildung 6: Zusammenhang von Voronoi-Diagram und Delaunay-Triangulierung

Eine alternative Methode der Triangulierung [AE87] wählt aus einer gegebenen Menge von Punkten einen Punkt aus, der die Menge so aufteilt, dass nicht mehr als eine bestimmte Anzahl an Punkten in den entstehenden Untermengen vorkommt. Das Verfahren geht von einem initialen Polygon aus, in dem sich die übrigen Punkte befinden. Die nach einer Unterteilung entstehenden Untermenge ergibt ein Simplex in der behandelten Dimension. Ein Simplex ist im Fall von zwei Dimensionen ein Dreieck.
Die Punkte, die die bestimmte Aufteilung erzielen, werden *Splitter* genannt. Diese Methode hat den Nachteil, dass von einer unveränderbaren Anzahl und unbeweglichen Punkten ausgegangen wird.

Der *Verfolgungs-Algorithmus* (englisch: marching method), erstellt, wie der Delaunay-Algorithmus, inkrementell eine Triangulation [Har98]. Ausgehend von einem Punkt nahe der zu triangulierenden Fläche, wird ein Hexagon generiert, dessen Punkte den tatsächlichen Punkten der Fläche zugewiesen werden. Die einzelnen Punkte des Hexagons dienen als Ausgangspunkte der lokalen Operation: In der Nachbarschaft der Punkte wird die Oberfläche je nach Dichte der benachbarten Punkte entweder feiner strukturiert oder als größere Fläche zusammengefasst. Dieser Vorgang wird wiederholt, bis die gesamte Fläche eingeschlossen ist.

Für diese Arbeit ist eine geeignete Datenstruktur unerlässlich. Währende der Laufzeit muss eine dynamische Veränderung der Daten gewährleistet sein. Um auf Punkte, Kanten und Dreiecke, sowie deren jeweilige Nachbarschaft zugreifen zu können, müssen alle Aktualisierungen an der genutzten Datenstruktur zur Laufzeit erfasst werden. Das Hinzufügen und Entfernen von Elementen der Struktur muss ebenfalls unterstützt werden. Eine simple Auflistung von Punkten bzw. Dreiecken, ist somit nicht ausreichend.
Aus dieser Anforderung resultiert der Anspruch an eine adaptive Datenstruktur. Diese Datenstruktur weist die Eigenschaft auf, nicht auf einen Detailgrad festgelegt zu sein, vgl. Abbildung 7 [Mav90].
Feinere Strukturen ermöglichen an bestimmten Regionen eine detailliertere Darstellung des Modells. Homogene Flächen können hingegen durch entsprechend größere geometrische Formen dargestellt werden, was den Speicherbedarf reduziert [BKP$^+$10].

Die Behandlung von impliziten Flächen ist für diese Arbeit nicht relevant, da keine geometrisch-komplexen Strukturen verwendet werden. Die sukzessive Unterteilung einer planaren Fläche in Dreiecke ist für das grundlegende Verständnis dieser Ausarbeitung wichtig. Implizite Flächen werden als Formeln dargestellt, was für die mathematische Verarbeitung von Geometrie geeignet ist. Bei der Visualisierung dieser Flächen ist eine zusätzliche Verarbeitung, wie die Triangulierung, nötig [BBB$^+$97]. Da eine direkte bildliche Darstellung Thema dieser Arbeit ist, wird nicht näher auf implizite Flächen eingegangen.

Die Funktionsweise einer Triangulierung wird im folgenden Unterkapitel erweitert. Es werden Methoden vorgestellt, die eine dynamische Verwaltung der Daten ermöglicht.

2.2 Dynamische Verfahren einer Triangulierung

Bei Anpassungen einer Triangulierung ist eine flexible, dynamische Veränderung der Struktur erforderlich. Solche Anpassungen sind notwendig, wenn Punkte der Triangulierung hinzugefügt oder entfernt werden sollen. Bei Dreiecken, die der Wohlgeformtheit der Triangulierung entgegenwirken, kann die Entfernung des gesamten Dreiecks aus der Struktur erforderlich werden.

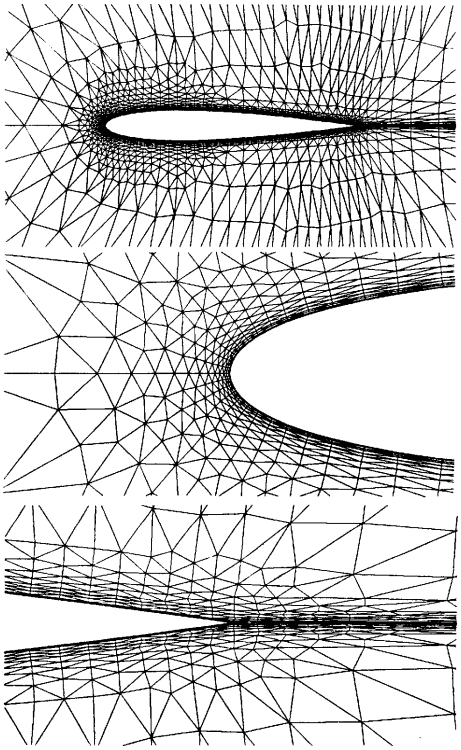


Abbildung 7: Adaptive Datenstruktur

In diesem Unterkapitel werden Methoden erklärt, die in solchen Situationen eingesetzt werden. Eine Voraussetzung für die dynamische Anpassung der Triangulierung, ist die Indizierung der einzelnen Punkte. Durch die Verwendung von Listen mit Indizes, werden alle Elemente der Topologie aktualisiert, sobald sich die Koordinaten eines Punktes ändern.

In dem Fall, dass zusätzliche Punkte einer Delaunay-Triangulierung hinzugefügt werden sollen, ist es wichtig, die Umkreisbedingung zu gewährleisten. Um in diesem Fall die Bedingung einer Delaunay-Triangulation zu erhalten, gibt es mehrere Verfahren. Ein Ansatz arbeitet mit einem inkrementellen Umkehren von Kanten. Dem sogenannten *Edge Flip* Verfahren [Müc98]. Dabei wird ein beliebiger Punkt innerhalb eines bekannten Bereichs in die bestehende Triangulierung eingefügt. Das Dreieck, in dem sich der neue Vertex befindet, wird in drei weitere Dreiecke unterteilt. Um die Umkreisbedingung der Delanaunay-Triangulierung aufrecht zu erhalten, erfolgt der Kanten Flip inkrementell [Müc98]. Bei dem *Edge Flip* wird lokal geprüft, ob ein Dreieck die Umkreisbedignung erfüllt. Ist dies nicht der Fall, liegen mehr als die geforderten drei Punkte im Umkreis des Dreiecks. Da eine Kante, außer am Rand einer Fläche, immer zwei Dreiecke separiert, steht ein alternatives Punktepaar für den Flip zur Verfügung. Die Kante des Dreiecks, die die Umkreisbedingung wiederherstellen kann, wird geflippt. Der Vorgang ist in Abbildung 8 veranschaulicht. Dieser Vorgang wird an an-

liegenden Kanten wiederholt, bis alle Dreiecke in dem Bereich die Umkreisbedingung erfüllen. Da vor Beginn des Hinzufügens des Punktes von einer global erfüllten Umkreisbedingung ausgegangen wird, ist nach lokaler Wiederherstellung der Bedingung auch die globale gewährleistet.

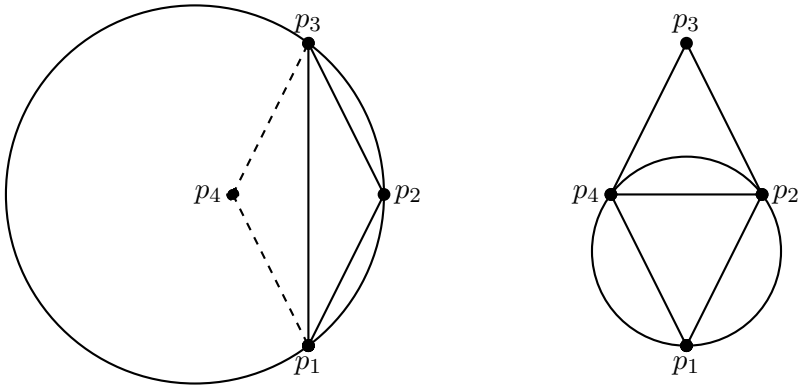


Abbildung 8: Flip einer Kante am Beispiel der Umkreisbedingung

Ein Aspekt der Triangulierung ist die Behandlung von Löchern in der Oberfläche. Bei dem folgenden koordinatenfreien Ansatz werden Löcher gefunden [LH08].
Initial werden Knoten aus einer Menge mit einer kreisförmigen *Sensorreichweite* R überschneidungsfrei gewählt. Entsprechend der Radien der Knoten, wird zwischen aktiven und redundanten Knoten unterschieden. Redundante Knoten liegen innerhalb des Radius eines anderen (aktiven) Knotens. Obwohl das Verfahren koordinatenfrei agiert, muss für die Wahl der aktiven Knoten bekannt sein, ob Knoten den Abstand R zueinander haben. Um Löcher zu detektieren, muss zusätzlich bekannt sein, ob Knoten den Abstand 2R zueinander haben [LH08]. Mit diesen Informationen wird ein Graph für die Struktur erstellt. Bei entsprechendem Abstand werden Knoten, die am Rand eines Lochs liegen, gefunden und als solche deklariert.

Der Bowyer-Watson-Algorithmus löst das Hinzufügen von Punkten mit Hilfe von Löchern in der Struktur. Der Algorithmus erstellt eine Delaunay-Triangulierung aus einer Punktemenge in beliebiger Dimensionalität, die größer als eins ist [Bow81]. Bei diesem Verfahren werden die Punkte inkrementell einer bestehenden Delaunay-Triangulieriung hinzugefügt. Dreiecke, dessen Umkreise den neuen Punkt einschließen, werden mittels der Topologie bestimmt und aus der Triangulierung entfernt. Diese Löcher werden ausgehend von den umschließenden Dreiecken neu trianguliert, sodass der zusätzliche Punkt eingeschlossen ist. Dieser Vorgang wird wiederholt, bis alle Punkte der Punktemenge in die Delaunay-Triangulierung eingepflegt sind.
Da die Behandlung von Löchern in dieser Arbeit keine Relevanz darstellt, soll für den Umgang mit Löchern einer Delaunay-Triangulierung an dieser Stelle nur auf weiterführende Literatur verwiesen werden [She96].

Dreiecke mit einem stumpfen Innenwinkel äußern sich in flachen Dreiecken und ergeben qualitativ minderwertige Ergebnisse beim Rendern, da auf Subpixelebene gearbeitet werden müsste, um den Detailgrad der Dreiecke zu erfassen. Aus diesem Grund werden Dreiecke mit möglichst großen Innenwinkeln als wohlgeformt betrachtet; flache Dreiecke sollen vermieden werden.
Das Teilgebiet der Regularisierung thematisiert Methoden, die die Strukturierung eines Netzes hinsichtlicht optischer und technischer Sicht aufwertet.
Eine Möglichkeit, die Struktur des Netzes an problematischen Stellen zu verbessern, ist der Gebrauch von *Steiner Punkten* [SG04]. Sie werden als zusätzliche Punkte der Struktur beigefügt, um beispielsweise flache Dreiecke so unterteilen zu können, dass diese vermieden werden.
Weitere Optionen, die die einzelnen Dreiecke und damit die gesamte Triangulierung optimieren, sind das Glätten der Struktur (englisch: mesh smoothing) und das Neu-Vernetzen (englisch remesh) [SG04].
Bei der Glättung der Struktur werden einzelne Vertices verschoben, um lokal die Dreiecke zur Wohlgeformtheit zu bewegen. Da dieser Vorgang ein wichtiges Element dieser Arbeit ist, wird in Abschnitt 4.3.4 auf eine besondere Form der Glättung nach Laplace eingegangen. Die einfache Glättung nach Laplace wird durch folgende Formel definiert [HDZ05]:

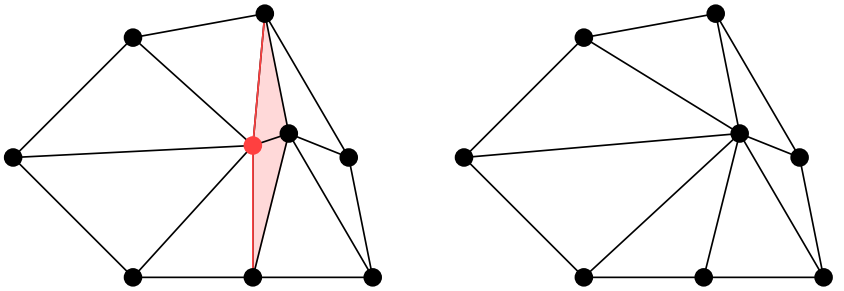
$$x_i = \frac{1}{M} \sum_{j=0}^{M} x_j \tag{3}$$

In Gleichung 3 beschreibt x_i den zu aktualisierenden Punkt. M beschreibt die Anzahl der von x_i ausgehenden Kanten und x_j stellt die adjazenten Punkte von x_i dar. Die Koordinaten von Punkt x_i werden durch die Anwendung dieser Formel auf die gemittelte Position der benachbarten Punkte aktualisiert. Bei dreidimensionalen Strukturen bewirkt die Glättung nach Laplace einen Informationsverlust, da hervorstehende Punkte und Kanten in Richtung des Schwerpunkts der Struktur bewegt werden [OBB01].

Bei der Neu-Vernetzung kann der bereits erklärte Kanten-Flip benutzt werden. Generell wird in diesem Fall eine bestehende Vernetzung, wie bei dem Bowyer-Watson-Algorithmus [Bow81], lokal aufgelöst und eine neue etabliert [SG04].

Dreiecke mit einem großen stumpfen Winkel oder kleine Dreiecke weisen einen vergleichsweise geringen Flächeninhalt auf. Solche Dreiecke können zur Verletzung der Umkreisbedingung einer Delaunay-Triangulierung oder zu Überschneidungen von Flächen führen. Um die Wohlgeformtheit der Triangulierung zu sichern, kann in solchen Fällen eine Zusammenlegung benachbarter Punkte durchgeführt werden [Muk12]. Bei der Vereinigung der Punkte werden Elemente der Struktur entfernt. Die Kante, die die beiden betroffenen Punkte verbindet, wird entfernt. Daher stammt der Begriff *Edge collapse*, der die Methode betitelt. Die anliegenden Dreiecke der Kante werden ebenfalls entfernt, dargestellt in Abbildung 9. Alle Kanten, die am zu löschenden Punkt liegen, werden auf den bleibenden Punkt übertragen. Je nach Topologie müssen weitere, nicht

direkt benachbarte Kanten ebenfalls aktualisiert werden.



(a) Ausgangssituation bei dem *Edge-collapse*-Verfahren. Die rot eingezeichneten Elemente sollen verschwinden

(b) Ergebnis vom *Edge-collapse*-Verfahren

Abbildung 9: Visualisierung der Kantenkollabierung

2.3 Gradientenverfahren

Das bildgebende Verfahren dieser Arbeit implementiert das Anpassen von Dreiecken an die visuellen Strukturen eines Eingabebildes. Die Qualität des Ergebnisbildes resultiert aus der Anordnung der Dreiecke. Im Folgenden wird eine Methode erläutert, die die dynamische Anpassung der Dreiecke an das Eingabebild erklärt.
Wie in mehreren Einsatzgebieten, wird auch in diesem Fall das Gradientenverfahren zur Optimierung genutzt. Im Themenfeld des *Machine Learnings* werden Parameter mit Hilfe des Gradientenabstiegs dem Ziel-Ergebnis angepasst [WFH11].
In diesem Verfahren wird das Gradientenverfahren verwendet, um den Approximationsfehler zu minimieren. Dieser Fehler beschreibt den farblichen Unterschied zwischen dem Originalbild und der stilisierten Abstraktion. Die Berechnung des Fehlers wird im folgenden Kapitel behandelt; an dieser Stelle wird der Optimierungsprozess erklärt, der diesen Fehler minimiert.
Indem die Punkte der Dreiecke im Bild neu platziert werden, wird eine präzisere Erfassung der Bildinhalte angestrebt. Die Bewegung der Punkte wird durch das Gradientenverfahren bestimmt.

Der Gradient ist ein Vektor, dessen partielle Ableitungen zusammengefasst werden:

$$\nabla f(x) = \begin{pmatrix} f_{x_1}(x) \\ \vdots \\ f_{x_n}(x) \end{pmatrix}$$

Diese Tangente weist in Richtung des steilsten Anstiegs im Punkt x. Die Steigung wird durch die Länge des Vektors dargestellt [20].

Für einen beliebigen Punkt x einer Funktion, liefert der Gradientenvektor die Richtung und Steigung zu einem lokalen Minimum und Maximum, je nach Bewegungsrichtung entlang der Tangente.
Die Idee des Gradientenverfahrens besteht darin, sich an dem Vektor orientierend lokalen Extrempunkten schrittweise zu nähern, bis sie erreicht sind. Ist das Ziel eine Minimierung, eine Optimierung bei der ein Minimum gesucht wird, wird vom Gradientenabstieg gesprochen. Es wird die negative Richtung des Gradienten traversiert, dargestellt in Gleichung 4.

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k), \quad k \geq 0 \tag{4}$$

Von einem geratenen Startpunkt x_0 beginnt das Verfahren. Die Punkte x_1, x_2,..., x_n werden darauf aufbauend bestimmt. x_{k+1} ergibt sich aus dem vorrangegangenen Punkt x_k. Mit variabler Schrittweite α und dem Gradienten des aktuellen Punkts ∇ f(x_x) folgt ein Iterationsschritt in Richtung des lokalen Minimums [CH10].
Bei dem Verfahren gilt es mehrere Eigenschaften zu beachten.
Die gewählte Schrittgröße ist entscheidend. Ist sie zu klein gewählt, nähert sich das Verfahren nur langsam dem Extrempunkt. Dadurch kann es vorkommen, dass bei einer limitierten Iterationsanzahlt kein Minimum/ Maximum erreicht wird.
Bei zu groß gewählter Schrittweite kann der Extrempunkt ebenfalls nicht erreicht werden, da eine Oszillation um den gesuchten Extrempunkt stattfinden kann. Ein gängiges Verfahren ist, die Schrittgröße für die Iterationen anzupassen [Xu11].
Ein weiteres Risiko ist in Abbildung 10 dargestellt. Es kann auftreten, dass ein lokaler Extrempunkt zuverlässig gefunden wird, jedoch der globale Extrempunkt als Lösung gewählt worden wäre, wenn Informationen über die Nachbarschaft bekannt gewesen wäre.

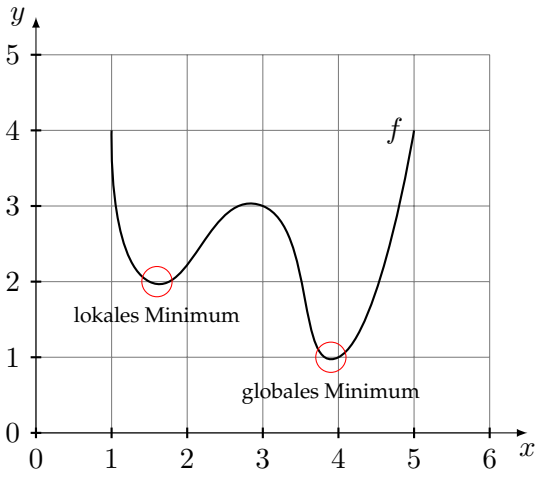
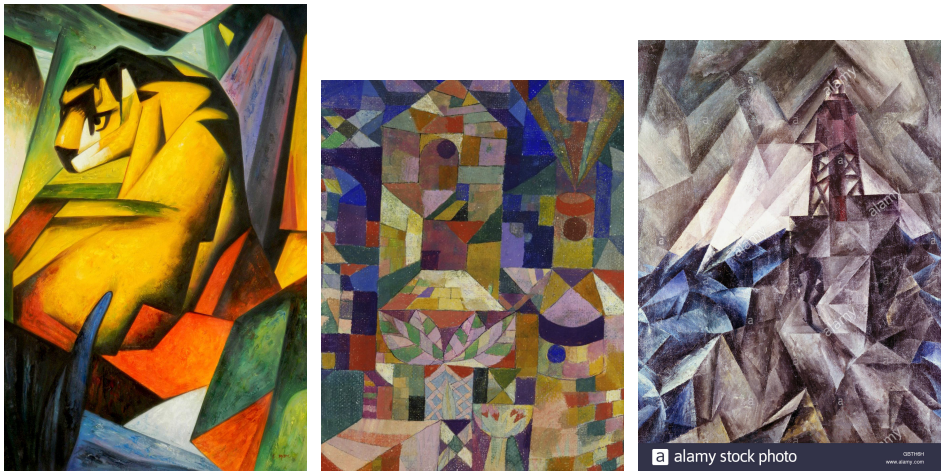


Abbildung 10: Risiko des lokalen Minimums

Durch Verwendung des Gradientverfahrens wird die Position eines Punktes inkrementell angepasst. Beeinflusst wird die Verschiebung des Punktes durch den berechneten Fehler an benachbarten Koordinaten. Diese Optimierung entspricht dem Gradientenverfahren.

2.4 Stilisierung

Das Ziel der Anwendung ist die Abstraktion eines Bildes mit möglichst großer Ähnlichkeit zum Original. Es gilt eine Mischung zu finden, die einen künstlerischen Stil hervorhebt und gleichzeitig das Eingabebild detailgetreu wiedergibt.
Solch eine Art von Kunst ist bereits in der Klassischen Moderne zu finden, die Stilrichtungen, wie den Konstruktivismus, den Kubismus oder den Surrealismus umfasst [Par05].
Um einen künstlerischen Stil mit einem Algorithmus zu imitieren, ist das Verstehen, wie sich ein Bild, ein Stil, zusammensetzt, unumgänglich. Die Abstraktion von gewöhnlichen Bildinhalten mittels geometrischer Formen ist auch Anfang des 20. Jahrhunderts zu finden. Ebenso gibt es heutzutage eine Vielzahl von Möglichkeiten der nicht-photorealistischen Computergrafik. Gerenderte Bilder sehen aus, wie mit Pinsel oder Bleistift gezeichnet, ebenso können Bilder durch geometrische Formen dargestellt werden.
In dieser Sektion werden Grundlagen für die untersuchte Abstraktion einer Triangulierung gegeben. Um den von Josh Bryan verwendeten Stil der «Dreiecks-Portraits» zu verstehen und anschließend umsetzen zu können, werden im Folgenden exemplarische Werke aus der Klassischen Moderne, vgl. Abbildung 11, Ergebnissen aus der nicht-photorealistischen Computergrafik gegenüber gestellt. Künstlerische Relevanz und Möglichkeiten der Abstraktion werden somit vermittelt.



(a) Franz Marc: Tiger **(b)** Paul Klee: Burggarten **(c)** Lyonel Feininger: Der Leuchtturm

Abbildung 11: Abstraktionen in der Klassichen Moderne

In Abbildung 11 sind drei Bilder von unterschiedlichen Künstlern dargestellt, die der Klassischen Moderne zugeordnet werden. Gemein haben alle Kunstwerke, dass ein gewisser Grad von Abstraktion gewöhnlicher Inhalte erreicht wird. «Der Tiger» von Franz Marc [37] und «Burggarten» von Paul Klee [39] weisen keine Einschränkung hinsichtlich der genutzten geometrischen Formen auf. «Der Leuchttum» von Lyonel Feininger [27] nutzt nahezu ausschließlich das Dreieck für die Gestaltung des Inhalts.
Der Kubismus wird in den Anfang des 20. Jahrhunderts eingeordnet und zeichnet sich durch die Abstraktion gewöhnlicher Motive aus. Die Motive werden auf einfache, geometrische Formen reduziert und (teilweise aus verschiedenen Blickwinkeln) wieder zusammengesetzt [Hei06]. Die Abstraktion von Motiven ist in der heutigen Zeit ebenfalls ein häufig verwendetes Stilmittel.

Die stilisierte Triangulierung von Bildern ist Teil der nicht-photorealistischen Rendertechniken. Solche Techniken nutzen ein künstlerisches Verfahren, um das Ergebnisbild in einem Stil darzustellen, sodass eine bestimmte Wirkung bei dem Betrachter erzielt wird. NPR (englisch: nonphotorealistic rendering) kann verwendet werden, um Bildinhalte zu vereinfachen oder zu verdeutlichen. Um bspw. Bildinhalte voneinander abzuheben, kann eine Kantenhervorhebung bei einem Foto genutzt werden. Dafür werden mehrere Bilder mit ausgelöstem Blitz aufgenommen, wodurch unterschiedliche Schatten in der Szene entstehen. Mit Hilfe der zusätzlichen Schatten, wird zwischen Beleuchtungs- und Texturkanten unterschieden, wodurch eine Kontinuität der Bildinhalte ermittelt wird [RTF+04].

Die Veröffentlichung, die dieser Arbeit zugrunde liegt, beinhaltet die Stilisierung von Dreiecken, dargestellt in Abbildung 12.
Folgendermaßen werden die Stilisierungen, ausgehend vom Original aus Abb. 12a, erreicht [LG18]:
Abbildungen 12b & c stellen die grundsätzlichen Farbgebungen (konstant und linearer Gradient) dar, welche im konzeptuellen Teil der Arbeit erläutert werden.
In den Abbildungen 12d & e werden *tonal art maps* verwendet. Diese Texturen haben die Eigenschaft, nahtlos neben- oder übereinander gefügt werden zu können, ohne, dass eine sichtbare, ungewollte Überschneidungen entsteht. Kombiniert mit den ursprünglichen Farben, wird eine Wirkung erzielt, die das Bild wie mit einem Bleistift gezeichnet erscheinen lässt. Der Stil ist Werken des Künstlers Josh Bryan nachempfunden und wird versucht im Verlauf dieser Arbeit, mittels einer alternativen Methode, künstlich zu imitieren.
Abbildung 12f entsteht, indem die Triangulierung durch Kanten dargestellt wird. Bei der Verwendung von OpenGL, kann der Befehl *glPolygonMode* mit den Parametern *GL_POINT*, *GL_LINE* oder *GL_FILL* ausgeführt werden. Wird die Option *GL_LINE* gewählt, werden nur die angegebenen Vertices mit Linien verbunden, wodurch ein Drahtgittermodell entsteht.
Der in Abbildung 12g präsentierte Effekt wird im *Fragmentshader* kreiert. Es wird eine Verzerrung der Dreiecksinhalte erzeugt, die in Richtung der Kanten jedes Dreiecks zu-

(a) Original **(b)** Konstante Farbe **(c)** Lineare Farbe **(d)** Linien **(e)** Kreuzschraffur

(f) Kanten **(g)** Linsen **(h)** Kissentextur **(i)** Dreieckstextur **(j)** Inkreise

Abbildung 12: Stile der Triangulierung

nimmt. Für diesen Stil wird ein Strahl von einer parametrisierten Fläche einer gaussförmigen Linse refraktiert. Die Linse wird dazu im Schwerpunkt des Dreiecks platziert. Der Strahl, von der Linse gebrochen, berechnet die Farbe an der verzerrten Position des darunter positionierten Bildes.
Die Abbildungen 12h & i demonstrieren den Einsatz von beliebigen Texturen. Mit Hilfe einer Kissen- und einer rekursiven Dreieckstextur wird die Helligkeit jedes Dreiecks beeinflusst, wodurch ein dreidimensionaler Effekt erzielt wird.
Als letzter Ansatz, dargestellt in Abbildung 12j, werden Kreise innerhalb der Dreiecke erstellt. Durch Berechnung von Mittelpunkt m und Radius r des Inkreises, und Nutzung einer *Smoothstep*-Funktion, wird die Farbe des Dreiecks angepasst. Radius $r \in [0, 1]$ ist relativ und wird bei der *Smoothstep*-Funktion $3r^2$-$2r^3$ zur Berechnung der Farbintensität genutzt. Bei der verwendeten Triangulierung werden homogene Flächen mit großen Dreiecken, durch die Abwesenheit von Tesselierung, repräsentiert. Feine Strukturen werden mit Hilfe mehrerer kleiner Dreiecke angenähert. Die Stilisierung mittels Inkreisen erfasst durch diese Art der Triangulierung auch die feinen Strukturen, da mit erhöhtem Detailgrad entsprechend viele kleinere Kreise einhergehen.

2.4.1 Farbgebung

Grundlegend für einen Abstraktionsstil ist die Farbgebung der einzelnen Dreiecke. Eine häufig genutzte Methode ist das Einfärben des Dreiecks mit konstanter Farbe. Eine Alternative, die lineare Farbgestaltung, wird in einem späteren Abschnitt (4.3) vorgestellt.

Die durschnittliche, konstante Farbe eines Dreiecks kann durch die Akkumulation der Pixelwerte und die anschließende Mittlung durch die Pixelanzahl ermittelt werden [LG18]:

$$c = \frac{1}{q} \sum_{i=1}^{q} I(x_i, y_i) \tag{5}$$

Bei dem Minimierungsprozess erreicht die resultierende Farbe c eines Dreiecks, als Optimum, die konstante Farbe [LG18]. Darauf wird in folgenden Kapiteln eingegangen. Mit q werden die Pixel eines Bildes I beschrieben.

Der Einblick in diesem Abschnitt, in die Stilisierungen von Triangulierungen, vermittelt ein grundlegendes Verständnis von Möglichkeiten und vom Aufwand einzelner Verfahren.

2.5 App

Ein wissenschaftlicher Schwerpunkt dieser Arbeit ist die Portierung des Ausgangsverfahrens zu einer App, sodass der implementierte Algorithmus einem durchschnittlichen Benutzer bereitgestellt werden kann. Da die Umsetzungsmöglichkeit gezeigt werden soll und keine marktreife App das Ziel ist, fokussiert sich diese Umsetzung auf nur eine Platform: Android. Es folgen daher Grundlagen einer Android App. Um flexibel zu bleiben und mit der Verbreitung im Apple Markt als Ausblick, sind Großteile der Anwendung nicht exklusiv für Android. Diese Erweiterbarkeit wird mittels des *React-Native-Frameworks* umgesetzt, was in der nächsten Untersektion thematisiert wird.

2.5.1 Android

Alle folgenden Informationen stammen von der offiziellen Android Entwickler Webseite [41].
Eine Android App basiert auf vier grundlegenden Komponenten.

Activities sind der Inhalt einer Ansicht der Anwendung. Der Benutzer kann mittels grafischer Schnittstelle, dem *User Interface - UI*, interagieren und so Aktionen der App ausführen oder zu einer anderen Ansicht gelangen. Ein Startbildschirm einer Anwendung stellt eine *Acticity* dar, von der weitere Aktionen ausführbar sind. Beispielhaft das Erreichen des Impressums. Sofern keine Einschränkungen durch eine jeweilige Anwendung gegeben ist, sind alle *Activities* anderer Apps frei zugänglich und erreichbar.

Services sind im Hintergrund ausgeführte Aufgaben. Sie können entweder vom Benutzer initiiert werden, um beispielsweise Musik nach Schließen der App abzuspielen, oder intern aus einer Anwendung gestartet werden. Eine regelemäßige Sicherung von Daten zu erstellen, ist eine weitere mögliche Motivation solcher Aufgaben.

Broadcast receivers realisieren die Kommunikation zwischen Betriebsystem und Anwendungen. Systemweite Nachrichten und Zustände können so anwendungsübergreifend übertragen werden, sogar, wenn eine Anwendung geschlossen ist. Diese Art der Kommunikation ist möglich, indem Anwendungen auf Nachrichten bestimmter Nachrichtenkanäle reagieren. Die Nachrichtenkanäle sind systemweit etabliert.

Content Provider verwalten den Zugriff und den Austausch von Daten. Dabei wird der Zugriff auf Daten der Anwendung selbst oder auf Daten anderer Anwendungen ermöglicht. Zu der Verwaltung der Zugriffe gehört der Umgang mit Zugriffserlaubnissen und die Bereitstellung von Methoden des Datenaustausches. Durch einen *Content Provider* wird der Umgang mit Anwendungsdaten abstrahiert, sodass die Verwaltung vereinfacht und der Austausch von Daten möglich wird.

Für die Sicherheit im Android Betriebssystem und für die Anwendungen sind die Erlaubnisse, die *Permissions*, verantwortlich. Jede App hat dafür die *AndroidManifest.xml* Datei, in der alle Ressourcen und entsprechende Zugriffsrechte gelistet sind. Dadurch werden die Anforderungen und der Nutzungsumfang der App transparent. Während der Nutzung der Anwendung wird der Nutzer nach Zustimmung für Zugriff auf die Ressourcen gefragt (Android Versionen >= 6).

Die Sicherheitsaspekte, der Umgang mit Daten und die beschränkte Verfügbarkeit von Ressourcen des Geräts, stellen die Besonderheiten der Implementation für ein Betriebssystem auf mobilen Geräten dar. Durch Limitierungen des Betriebssystems werden laufende Anwendungen beendet, um Ressourcen zu sparen. Zugriffe auf Hardware des Endgeräts müssen explizit vom Nutzer erlaubt werden.
Bei dem Betriebssystem iOS existieren mehr Einschränkungen, weshalb primär eine Umsetzung für Android angestrebt ist. Um jedoch auch bei weiterführender Arbeit flexibler zu sein, wird das Framework React Native in dieser Umsetzung verwendet.

2.5.2 React Native

React Native ist ein JavaScript Framework, das es ermöglicht, plattformübergreifend, für Android und iOS, Apps zu entwickeln. Es basiert auf der JavaScript Bibliothek React von Facebook. React zielt auf die Entwicklung für Inhalte im Internet ab, React Native auf mobile Endgeräte [Eis15].
Obwohl das Ziel dieser Arbeit eine Anwendung für Android ist, soll die Möglichkeit der plattformübergreifenden Entwicklung erhalten bleiben. Dies ermöglicht React Native.
Anwendungen mit dem Framework werden mit einer Synthese von JavaScript und XML geschrieben - JSX. Die Besonderheit ist, dass XML in JavaScript eingebunden wird, und nicht umgekehrt [48].
Der Vorteil gegenüber anderen plattformübergreifenden Entwicklungsansätzen ist, dass React Native, entsprechend der genutzten Plattform, die nativen APIs benutzt. Eine

API ist eine Programmierschnittstelle. Diese Schnittstelle ermöglicht den Zugriff auf ausgewählte Hard- oder Softwareressourcen. In diesem Fall bedeutet das, dass für iOS der Renderer in Objective-C und für Android der in Java genutzt wird, ohne, dass der Code den Betriebssystemen angepasst werden muss [Eis15]. Ein weiterer Vorteil dieses Aufbaus ist der mögliche Zugriff auf die Hardware der Endgeräte, wie die Kamera oder die Sensoren.
Zusammenfassend: React Native ermöglicht die plattformunabhängige Entwicklung und Nutzung von plattformabhängigen, nativen APIs. Die grundlegende Funktionsweise und Merkmale des Frameworks werden in den folgenden Abschnitten aufgeführt.

Eine Anwendung, die mit React Native erstellt wird, basiert auf Komponenten, die mit einer Java-*Activity* verglichen werden können. Von jeder Komponente wird gefordert, dass sie eine Methode *render* hat - mit JSX als Rückgabewert [48].
Zwei weitere grundlegende Bestandteile des Frameworks, sind *Props* und *States*.
Props sind die Eigenschaften einer Komponente. Sie sind für die «Lebensdauer» einer Komponente fix. Diese Dauer endet, sobald die Komponente mit aktualisierten Werten gerendert wird.
Wenn eine Komponente an mehreren Stellen instanziiert wird, können die Eigenschaften unterschiedlich beschrieben und entsprechend dargestellt werden.
Daten, die in einem **State** einer Komponente gespeichert werden, sind nicht an die Lebensdauer der Komponente gebunden. Sie dienen bei der Verarbeitung von Einagben des Nutzers oder verarbeiten bspw. Aktionen, die zeitbasiert agieren [48]. Mit diesen Werten wird der Zustand einer Komponente beschrieben.
React Native ist so konzipiert, dass der *View*, die grafische Oberfläche, mit aktualisierten Werten neu gerendert wird, wenn *Props* oder der *State* aktualisiert werden. Das ist der *Update Cycle* [Eis15].

Eine markante Eigenschaft des Frameworks ist die Zusammensetzung von Funktionen. Je nach Funktionalität können verschiedene Module eingebunden werden, die auch andere Entwickler bereitstellen. Die Auswahl umfasst dabei unter anderem Module, die die Kommunikation mit Datenbanken ermöglichen, die Animationen von grafischen Inhalten realisieren oder die Verwaltung von Eingabeformularen übernehmen.
Hier noch Quellen?

Bei dem Rendervorgang wird der Vorteil gegenüber anderen plattformübergreifenden *Frameworks* deutlich. Das *Document Object Model* , DOM, ist eine API für HTML und XML, die den logischen Aufbau von Dokumenten und deren Zugriff verwaltet [14]. Das Rendern von interaktiven Inhalten hat aufgrund der Zugriffe auf den DOM "[...] signifikanten Einfluss auf die Performanz" [Eis15]. Mit React Native wird eine zusätzliche Schicht der Architektur hinzugefügt, das virtuelle DOM. Diese Abstraktion beinhaltet zwei Vorteile. Durch die virtuelle Kopie kann React Native Unterschiede der beiden DOMs erfassen und effizienter neu rendern [Eis15].

Der zweite Vorteil, der sich daraus ergibt, ist die Fähigkeit plattformabhängige APIs zu nutzen und entsprechend effizient zu rendern. Diese Eigenschaft ermöglicht React Native auf beliebig vielen Betriebssystemen einsetzbar zu sein. 75% des Codes, geschrieben mit React Native, konnte plattformübergreifend in einer Studienabschlussarbeit genutzt werden [HV16]. Mit dem Ergebnis, dass keine Performanzeinbrüche festgestellt werden konnten. Bezüglich der Benutzererfahrung, ob sich eine React Native Anwendung deutlich von einer nativen Anwendung unterscheidet, wurde wegen nicht aussagekräftiger Umfrage lediglich die Tendenz aufgezeigt, dass kein deutlicher Unterschied zwischen den Versionen bemerkt wurde.

2.6 Webserver

In der Darlegung des konzeptuellen Teils dieser Ausarbeitung wird detailliert erläutert, wie der Server in die Struktur zwischen implementiertem Verfahren und resultierender App integriert wird. Dafür werden in diesem Abschnitt die Grundlagen eines Webservers dargelegt.

2.6.1 Überblick

Als Server wird ein Rechner bezeichnet, der Software betreibt, die von anderen Rechnern, den Clients, genutzt werden kann [YM96]. Dabei ist das Serverprogramm so konzipiert, dass mehrere Clients zeitgleich das Programm nutzen können.
In dieser Arbeit wurde ein Webserver eingebunden, weshalb gezielt Grundlagen aus dem Oberthema “Server” kanalisiert erläutert werden.

Webserver ermöglichen eine weltweite Verbindung zwischen Webbrowsern der Nutzer und dem *World Wide Web*. Mittels des HTTP/HTTPS Protokolls werden dadurch Inhalte aller Art, wie Dokumente, Bilder oder Audiodateien, ausgetauscht [YM96].
Das Modell eines Webservers besteht aus einem Computer mit Internetanschluss, auf dem mehrere Arten von Software läuft. Die Software umfasst die Verwaltung der Kommunikation über das Netzwerk und das Serverprogramm selbst. Zusätzlich gehören Daten dazu, die auf Anfrage, *Request*, zu einer Antwort, *Response*, verarbeitet werden [YM96]. Das Serverprogramm verarbeitet die Anfragen und reagiert entsprechend mit Antworten.

2.6.2 HTTP

Der Ablauf der Kommunikation ist über das HTTP oder das HTTPS definiert. Diese Art der Kommunikation ist für den Webserver charakateristisch. *Hypertext Transfer Protocol*, HTTP, wurde von Tim Berners-Lee Anfang der 1990er Jahre erfunden [50] und ist das Protokoll, das den Dialog zwischen Server und Client definiert.

Um eine erfolgreiche Kommunikation zu erreichen, ist die Einhaltung der Syntax von *Request* und *Response* obligatorisch. Beide Nachrichtenarten bestehen aus zwei Teilen, den Metadaten enthaltenden *Header* und dem eigentlichen Inhalt der Nachricht, dem *Body* [GT02].

Request

```
1 <method> <request-URL> <version>
2 <headers>
3
4 <entity-body>
```

Mit dem Inhalt der ersten Zeile wird der Server über die Art der Anfrage informiert [GT02]. **Method** enthält einen Befehl, wie "Post" oder "GET". *GET* fordert den Server auf, eine Datei zu senden. Mit der *POST* Methode werden Daten an der Server geschickt, die dort verarbeitet werden sollen.
Die URL adressiert eine konkrete Ressource des Servers. Darin kann beispielsweise enthalten sein, welche Datei angefordert wird. Alle möglichen angeforderten Daten des Servers werden als Ressourcen zusammengefasst [40].
Im letzten Element der ersten Zeile wird dem Server mitgeteilt, mit welcher HTTP-Version der *Client* arbeitet [GT02].

Response

```
1 <version> <status> <reason-phrase>
2 <headers>
3
4 <entity-body>
```

Die Antwort, *Response*, ist ähnlich zur Anfrage aufgebaut. In der ersten Zeile steht alternativ der Status der Anfrage und die für Menschen lesbare Variante im Element *reason-phrase*.
Metadaten im *Header* geben Auskunft über die Art des Inhalts im *Body*. Darin enthalten sind bspw. der MIME-Typ einer Anfrage oder die Länge einer Antwort [GT02].
Im *Body* stehen die zu versendenden, beliebigen Daten. Das können bei einer *POST* Anfrage Formulardaten sein, mit denen der Nutzer sich einloggen möchte. Nach einem *Get-Request* kann das angefragte Video darin enthalten sein.

Ein wichtiger Aspekt der Antworten von Servern sind die Fehlermeldungen [RR08]. Benutzer und Entwickler werden durch den *status* und den *reason-phrase* auf die Fehlerquelle hingewiesen.
Einige Fehlerstati sollen kurz aufgeführt werden, um ein Grundverständnis zu vermitteln [BLFF96]:

200 "Ok". Die Kommunikation hat erfolgreich stattgefunden. Alle 2xx-Codes stehen für einen problemfreien Dialog.

301 Die angeforderte Ressource ist unter einer anderen Adresse verfügbar. Die 3xx-Codes repräsentieren die Verschiebung von Ressourcen.

404 "Not found". Die angefragten Daten wurden an der gegebenen Adresse nicht gefunden. Generell stehen die 4xx-Codes für Fehler auf der Seite des Clients.

500 Steht für einen internen Fehler des Servers und befindet sich in der 5xx-Code Klasse der serverseitigen Fehler.

Asynchronität ist bei dem Umgang mit mehreren Clients oder bei rechenintensiven Anfragen ein entscheidender Aspekt der Performanz für den Server.
Die Steigerung der Performanz gegenüber synchronen *Requests* und *Responses* basiert auf dem Verarbeitungsverhalten des Servers. Bei einer asynchronen Anfrage/ Antwort wartet das Serverprogramm nicht auf die vollständige Übertragung der Anfrage/Antwort, sondern steht für weitere Aufgaben zur Verfügung und arbeitet dadurch unabhängig von *Request* und *Response* [KS08].

2.6.3 Weitere Protokolle

HTTPS erweitert HTTP um einen entscheidenden Aspekt in der Domaine des Webs: Sicherheit. *Hypertext Transfer Protocol Secure*, HTTPS, kommuniziert mit einem zusätzlichen Protokoll, dem TLS Protokoll. *Transport Layer Security* dient zur Verschlüsselung der übertragenen Daten [SL11]. Deshalb sollten sensible Daten, wie Zahlungsinformationen, nur via HTTPS übertragen werden.

Die konkrete Übertragung der Daten folgt den Regeln des Zusammenschlusses der *Transmission Control Protocol* und *Internet Protocol* Protokolle, kurz TCP/IP. Die Protokolle sichern eine vollständige und chronologisch korrekte Übertragung von Paketen im Intra- sowie im Internet [YM96].
Auf weitere Arten von Servern, wie etwa der Mailserver oder der Proxyserver, und auf deren spezifische Protokolle, wird an dieser Stelle verwiesen. Für diese Arbeit sind sie nicht relevant.

2.7 Zusammenfassung

In diesem Kapitel werden grundsätzliche Informationen und Vorgehensweisen vorgestellt, die als Vorwissen für darauf aufbauende Verfahren verwendet werden können. Das Ziel ist es, nicht nur die in dieser Arbeit verwendeten Methoden zu verstehen und bewerten zu können, sondern auch weiterführende Verfahren kennenzulernen.
Im Themenbereich der Triangulierung wird ein profundes Grundwissen der Funktionen, Eigenschaften und Qualitätsmerkmale vermittelt. Anhand dieses Wissens sollen die im folgenden Kapitel vorgestellten alternativen Anwendungen verstanden werden können. Eine Hervorhebung der Unterschiede in Methodik und Qualität erfolgt.

In Hinblick auf die Erarbeitung einer mobilen Anwendung und deren Kommunikationspartner, werden Grundlagen einer Android Anwendung und eines Servers gegeben. Somit kann die Erhebung der Anforderungen an diese Komponenten nachvollzogen werden. Die Weiterentwicklung eines Renderstils ist ein weiterer Bestandteil dieser Arbeit. Dafür werden Beispiele und deren Funktionsweise erläutert, um einen Überblick über mögliche Stile herzustellen.
Im folgenden Kapitel wird der Stand der Technik thematisiert. Es werden Arbeiten und Ergebnisse vorgestellt, die mit dem gegebenen Grundwissen bewertet werden können.

3 Stand der Technik und vergleichbare Arbeiten

Dieses Kapitel bietet einen Überblick über alternative Verfahren der stilisierten Triangulierung. Durch die Abgrenzung zu ähnlichen Anwendungsgebieten werden Ziel und Methodik dieser Arbeit deutlicher herausgestellt. Insbesondere werden mobile Anwendungen vorgestellt, die ästhetische Ergebnisse durch Triangulierungen erreichen. Ihre Ergebnisse und Funktionsweisen werden präsentiert und qualitatitv eingeschätzt. Abgeschlossen wird dieses Kapitel mit Methoden der Veröffentlichung von Lawonn und Günther [LG18]. Die dort genannten Aspekte werden mit den alternativen Verfahren verglichen. So werden wichtige Aspekte einer mobilen Anwendung aufgeführt, wodurch ein zusätzlicher Einblick in den Umfang des Themengebiets geleistet wird.

Um die einzelnen Verfahren miteindander vergleichen zu können, werden zwei Ausgangsbilder gewählt [LG18]. Alle Ergebnisbilder (bis auf die Ballons von Grundland et al. [GGD05]) wurden mit den genannten Anwendungen erstellt, sodass ein direkter Vergleich der Resultate möglich ist.

3.1 Eingrenzung des Themengebiets

Die Triangulierung eines Bildes stellt das Ziel dar, weshalb Arbeiten mit gleichem Ziel hervorgehoben werden, während Arbeiten, die auf andere geometrische Strukturen abzielen, sich nicht im Fokus dieser Recherche befinden.
Diese Strukturen können aus Verfahren entstehen, die, basierend auf Farbähnlichkeiten der Pixelnachbarschaft, Regionen erzeugen. Die Algorithmen der Bildsegmentierung oder das L_0-Verfahren stetzen solche Methoden ein [LG18].

Mehrere Anwendungen für den Internetbrowser nutzen eine Triangulierung, um kunstvolle Bilder zu generieren. Der Nutzer beeinflusst dabei die Gleichmäßigkeit der Triangulierung und die Farbgebung der Dreiecke.
Bei der Anwendung Delaunay Triangle Generator [16] wird eine Delaunay-Triangulierung erstellt. Zusätzlich erhält der Nutzer die Kontrolle über eine Beleuchtung der Triangulierung. Durch Farbwahl, Anzahl der Lichter und Größe des Einflusses, kann die Triangulierung farblich gestaltet werden. Die Entfernung von einer Lichtquelle zum Schwerpunkt eines Dreiecks beeinflusst die farbliche Gestaltung und somit den viusellen Effekt.

Trianglify [15] ist für die Gestaltung eines Hintergrundbildes konzipiert. Mit zwei Schiebereglern kann der Nutzer Einfluss nehmen. Mit einem Regler wird die Gleichmäßigkeit der Punkteverteilung kontrolliert, wodurch die einzelnen Punkte verschoben werden und somit die Struktur der Triangulierung beeinflusst wird. Ausgehend vom Einheitsnetz wird mittels Kantenflips dadurch die Delaunay-Triangulierung erreicht. Der zweite Schieberegler verändert die Anzahl und gleichzeitig die Größe der Dreiecke. Als Grundlage für die farbliche Gestaltung werden Bilder mit Farbverläufen

verwendet. Die Triangulierung resultiert in konstanter Einfärbung der Dreiecke.

Low Poly Background Generator [11] bietet ähnlichen Umfang, wie die zuvor genannten Anwendungen. Jedoch resultiert das ursprüngliche Dreiecksnetz in Überschneidungen der Dreiecke, sobald die Struktur der Triangulierung zu sehr verändert wird. Dynamische Anpassungen, wie der Kantenflip, werden nicht eingesetzt.

Obwohl die Kreierung ästhetischer Inhalte ein wichtiger Aspekt dieser Arbeit ist, wird auf oben genannte Verfahren nicht näher eingegangen. Im Fokus der Recherche liegen Anwendungen, die eine automatisierte stilisierte Triangulierung anstreben und umsetzen. Aufgrund dessen wird im Folgenden vorgestellt, wie Verfahren solch eine Triangulation erreichen und wie der Benutzer diese individualisieren kann.

Sun et al. veröffentlichten 2007 einen energiebasierten Minimierungsansatz für Polygonnetze [SLWS07]. Die sogenannte *Optimized-Gradient-Mesh*-Methode erstellt aus einfachen 2D-Bildern Vektorgrafiken mit Gradienten in der Farbgebung. Das semiautomatisierte Verfahren benötigt eine initiale Maskierung der Bildelemente, um qualitativ hochwertige Ergebnisse zu erzielen [36]. Mit initialer Nutzereingabe werden Ergebnisse erreicht, die vom Original nicht zu unterscheiden sind. Das Verfahren optimiert die Position und Ausrichtung rechteckiger Vektorpatches. Die Optimierung aktualisiert die Ausrichtungen der Vektoren und die Positionen der Eckpunkte der *Patches* hinsichtlich eines Approximationsfehlers, der durch den Vergleich vom aktuellen Bild eines Iterationschrittes mit dem Originalbild entsteht. Zusätzlich wird ein Regularisierungsterm verwendet, um ein gleichmäßiges Polygonnetz zu erhalten.
Durch die Kombination von Gradienten und Vektoren an den Punkten der Struktur, werden hochwertige Ergebnisse bei einfachen Motiven erreicht. Feine Strukturen hingegen, können mit diesem Verfahren nicht mit zufriedenstellender Qualität erzeugt werden [36].
Dieses Verfahren differenziert sich deutlich von der Methode, die in dieser Arbeit umgesetzt ist. Während Sun et al. die exakte Wiedergabe eines Eingabebildes anstreben, zielt die stilisierte Triangulierung dieser Arbeit auf eine detailgetreue Abstraktion mit ästhetischer Wirkung ab. Die *Optimized-Gradient-Mesh*-Methode hat eine Darstellung des Motivs als Vektorgrafik zum Ziel, sodass eine vereinfachte künstlerische Weiterverarbeitung möglich ist.
Statt initialer Nutzereingaben, die für dieses Verfahren nötig sind, werden in dieser Arbeit Automatismen genannt, die eine Adaption an beliebig detailreiche Strukturen ermöglichen; unabhängig des Eingabebilds.

3.2 Stilisierte Triangulierungen

In der Veröffentlichung aus 2008 von M. Grundland et al. [GGD05] werden verschiedene nicht-photorealistische Renderverfahren präsentiert. Basierend auf Bildcharakteristiken werden Punkte im Bild gewählt, die anschließend als Grundlage eines Voronoi-

Diagrams oder einer Delaunay-Triangulierung dienen [LG18].
Es werden adaptive Verfahren vorgestellt, die darauf abzielen, den Bildinhalt möglichst genau im Ergebnisbild wiederzugeben. Das Verfahren mit niedrigerem Approximationsfehler geht dazu von einem initialen nicht-adaptiven *farthest-point*-Algorithmus aus. Das visuelle Ergebnis davon ist das Voronoi-Diagram. Um Details des Bildes zu erfassen, werden anschließend weitere Punkte hinzugefügt. Dabei sollen Regionen mit niedriger Punktedichte um Punkte erweitert werden, um eine globale Abdeckung zu gewährleisten und alle Features des Bildes zu erfassen. In inhomogenen Regionen sollen ebenfalls neue Punkte gesetzt werden, damit feinere Strukturen entdeckt und herausgestellt werden können.
Das Verfahren wählt iterativ eine zufällige Untermenge von Voronoi-Vertices C, für die Eigenschaften berechnet und als repräsentativ angenommen werden. Die Eigenschaften der gewählten Vertices $i \in C$ umfassen die quadrierte euklidische Distanz r_i^2 zum nächsten Punkt und die Leuchtdichte l_n der dichtesten Nachbarn $n \in N_\mathrm{i}$. Entfernungen können zusätzlich gewichten werden: $w_i r_i^2$. Mit einem Faktor $w \geq 0$.
Die Leuchtdichte gibt die empfundene Helligkeit einer Lichtquelle an [Wit14].
Für jeden betrachteten Voronoi-Vertex wird mit Gleichung 6 die mittlere absolute Abweichung d_i der Leuchtdichte der Nachbarschaft berechnet.

$$d_i = \frac{1}{\|N_i\|} \sum_{n \in N_i} |l_n - \mu_{N_i}| \qquad mit \quad \mu_{N_i} = \frac{1}{\|N_i\|} \sum_{n \in N_i} l_n \tag{6}$$

Um die Kriterien r_i^2 und d_i miteinander vergleichen zu können, werden mit Hilfe der z-Transformation $\hat{r}_i^2$ und $\hat{d}_i$ berechnet. Durch die z-Transformation werden die Werte unter Berücksichtigung von Erwartungswert und Varianz skaliert [GW09]. Dadurch wird ausgedrückt, inwieweit die Werte vom Erwartungswert abweichen [GGD05].

$$\hat{r}_i^2 = \frac{w_i r_i^2 - \mu_r}{\sigma_r} \quad und \quad \hat{d}_i = \frac{d_i - \mu_d}{\sigma_d} \tag{7}$$

$$\sigma_r = \frac{1}{\|C\|} \sum_{j \in C} |w_j r_j^2 - \mu_r| \quad mit \quad \mu_r = \frac{1}{\|C\|} \sum_{j \in C} w_j r_j^2 \tag{8}$$

$$\sigma_d = \frac{1}{\|C\|} \sum_{j \in C} |d_j - \mu_d| \quad mit \quad \mu_u = \frac{1}{\|C\|} \sum_{j \in C} d_j \tag{9}$$

$$e_i = min(\hat{r}_i^2, \hat{d}_j) + \lambda \, max(\hat{r}_i^2, \hat{d}_i) \tag{10}$$

Erst durch die Standardisierung ist Gleichung 10 auf die einzelnen Punkte anwendbar. Das Ziel dieser Gleichung ist, einen Kompromiss zwischen globaler Abdeckung und lokaler Detailerfassung zu ermitteln. Durch λ wird erreicht, dass ein niedrigerer z-Wert mehr gewichtet wird als der höhere z-Wert.
Als geeignet gelten $c \in C$, wenn beide Kriterien hohe z-Werte vorweisen oder einer von beiden besonders hoch ist. Dadurch werden Vertices in Gegenden ausgewählt, in denen die Punkteverteilung niedrig ist und (noch) keine erfassbaren Details vorkommen

[GGD05].

(a) 2450 Samples

(b) 7350 Samples

(c) Ergebnis mit 7350 Samples

Abbildung 13: Auswirkungen der Samplingdichte

Aus den in Abbildung 13 präsentierten Ergebnisbildern [GGD05] sind folgende Schlüsse zu ziehen:
Ansätze einer adaptiven Struktur sind in der oberen linken Abbildung erkennbar. Der Zusammenschluss von Regionen mit nahezu gleicher Farbe ist auf die globale Samplingdichte beschränkt. Somit ensteht selbst in Regionen mit geringer Dynamik eine engmaschige Einteilung der Waben. Gleichzeitig werden feinere Bildelemente nicht erfasst. Einem Ballon im Vordergrund fehlt der Korb, ein Ballon im Hintergrund fehlt komplett. Wie in Abbildung 13 dargestellt, ist eine erhöhte Dichte an Voronoi-Waben in der Nähe von Kanten sichtbar. Es werden zwei Nachteile deutlich: Im rechten oberen Bild ist zu sehen, dass selbst bei hoher Dichte an Strukturelementen, Kanten aus dem Originalbild etwas undeutlich wiedergegeben werden oder sogar Bildelemente verschwinden.

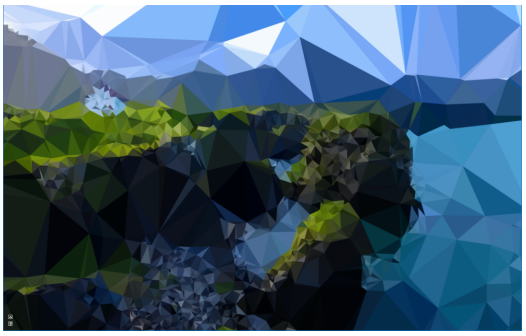
Zusätzlich wird in Abbildung 13b deutlich, dass bei gesteigerter Samplingdichte auch Regionen unterteilt werden, die von einer weiteren Unterteilung nicht profitieren. Die visuelle Qualität wird nicht gesteigert, jedoch steigen Speicherbedarf und Rechenzeit. Die Kanten des adaptiven Polygonnetzes entsprechen nicht den Kanten des Bildinhalts, wodurch eine nicht detailgetreue Abstraktion stattfindet [LG18].

Yun bietet mit Dmesh eine bekannte Anwendung an, die eine automatisierte und manuelle Erstellung einer Triangulierung unterstützt [51]. Bereits die automatisierte Triangulierung erzielt Ergebnisse, die auf einer adaptiven Netzstruktur aufbauen, Details des Bildes erfassen und einen insgesamt harmonischen Eindruck vermitteln. Die Qualität des Ergebnisses des automatisierten Ansatzes ist abhängig vom Eingabebild, sodass, je nach Bild, unterschiedliche Qualitätsstufen erreicht werden.
Die Punkte, die der Triangulierung zugrunde liegen, werden reproduzierbar auf Bildfeatures platziert, dargestellt in Abbildung 14c. Bei der Rekonstruktion des Himmels wird erkennbar, dass Hilfspunkte außerhalb des Bildes hinzugenommen werden. Der Benutzer kann Punkte einer bestehenden Triangulierung hinzufügen oder entfernen, wodurch eine beliebig genaue Wiedergabe erreicht werden kann. Die automatisierte Verarbeitung von Videos ist möglich, ergibt jedoch unstetige Ergebnisse [LG18].

2012 veröffentlichte Akimitsu Hamamuro eine auf JavaScript basierende Lösung [6]. Bei diesem Ansatz wird das Eingabebild vor der stilisierten Triangulierung verändert: Als erster Schritt wird aus dem beliebig wählbaren Eingabebild ein Graustufenbild erstellt. Anschließend werden Filter benutzt, um das Bild weich zu zeichnen und die Kanten hervorzuheben.
Mit dem bearbeiteten Bild als Grundlage, werden Punkte entlang an Kanten erstellt. Diese Punkte werden einer initialen Delaunay-Triangulierung hinzugefügt. Dadurch wird erreicht, dass Dreiecke in der Nähe von Kanten erstellt werden; eine Erfassung der Bilddetails erfolgt. Eine geringere Punktedichte liegt in homogenen Bereichen vor [LG18].
Die Farbgebung der einzelnen Dreiecke erfolgt simpel. Für jedes Dreieck wird der Schwerpunkt berechnet, dessen Koordinaten zum Auslesen der konstanten Farbe des Dreiecks verwendet werden [6].
Das beispielhafte Ergebnisbild, vgl. Abbildung 15, verdeutlicht, dass harte Kanten im Eingabebild zuverlässig verwendet werden, um Strukturen zu erfassen und somit den Bildinhalt wiederzugeben. Auffällig ist auch, dass bei weichen Übergängen eine Unruhe im Ergebnisbild entsteht, sichtbar rechts im Bild beim Horizont. In homogenen Regionen ist eine deutlich niedrigere Punktedichte zu erkennen als bei Grundland et al. [GGD05]. Das verdeutlicht eine fortgeschrittene Umsetzung einer adaptiven Triangulierung. Jedoch entsteht auch in diesem Fall eine ungleichmäßige Aufteilung der Dreiecke, sodass der Hintergrund durch Unstetigkeiten in den Vordergrund tritt.

Polygonize umfasst 15 verschiedene Stilisierungen eines Bildes [22]. Bei allen Varianten werden detailreiche Bildelemente automatisch präzise erfasst, Kanten verschwim-

(a) Eingabebild

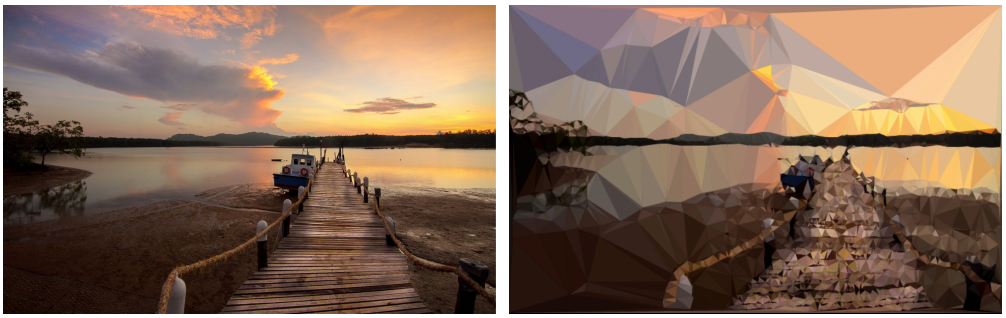


(b) Resultierende Triangulierung

(c) Vertices ohne Triangulierung

Abbildung 14: Eingabebild und Ergebnisse der automatisierten Methode von Dmesh

men nicht und homogene Regionen weisen eine geringere Dichte an Dreiecken im Verlgeich zum restlichen Bild auf. Die Dreiecke werden mit konstanter Farbgebung versehen. Viele Renderstile werden durch Farb- oder Texturanpassungen individualisiert.
Die Wiedergabe der feinen Strukturen im Bild wird über eine hohe Anzahl von verwendeten Dreiecken erreicht. Ausgehend von den Kanten der Motive im Bild, platziert der Algorithmus Punkte entlang der Kanten und verbindet diese, sodass eine Triangulierung entsteht. Auffällig ist der Unterschied der Triangulierungen in Abbildung 16c und 16d. Die farblose Triangulierung fällt durch viele nicht-wohlgeformte Dreiecke auf. Diese Art der Triangulierung wird für Ergebnisse mit Farbe nicht verwendet, wie in Abbildung 16d deutlich wird. Das zweigeteilte Bild gleicht bis auf die Einfärbung dem Ergebnis in Abb. 16b. In beiden Fällen entstehen deutlich wohlgeformtere Dreiecke bei



(a) Eingabebild **(b)** Ergebnis

Abbildung 15: Ein- und Ausgabebild des Triangulation Image Generator Verfahrens

der Triangulierung.

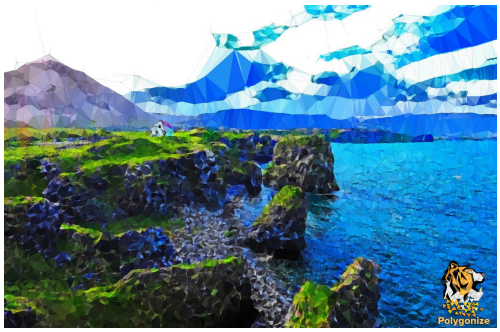
Mit dem L_0 Gradientenverfahren ist eine Abstraktion eines Bildes möglich, sodass Bildinhalte erhalten bleiben und eindeutige Kanten entstehen [XLXJ11]. Die Ergebnisse der Arbeiten, die dieses Verfahren nutzen und erweitern, unterteilen das Bild in Regionen, nicht in Dreiecke. Aus diesem Grund wird auf den Bereich der Bildsegmentierung lediglich verwiesen.

3.3 Apps

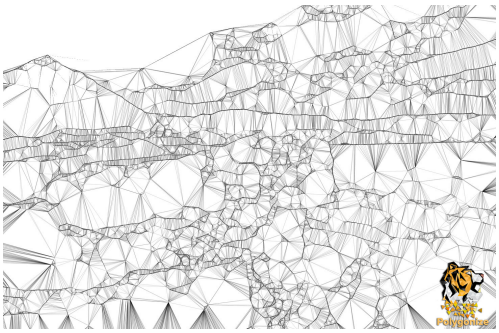
In diesem Abschnitt werden veröffentlichte Anwendungen untersucht und anhand der visuellen Ergebnisse und der Gestaltungsfreiheit beurteilt. Es werden nur Android Apps in Betracht gezogen. Das Smartphone Huwawai G8 oder das Tablet Samsung SM-T580 wurden bei der Nutzung der Anwendungen und der Erstellung der Ergebnisse verwendet.

Polygon Effect
Polygon Effect [29] stellt eine Anwendung bereit, die ein Bild trianguliert und mehrere Farbstile zur Auswahl stellt. Das Eingabe- und das Ergebnisbild stellen separate Ebenen dar. Die Farbstile geben an, wie diese Ebenen miteinander vereint werden, wodurch Kontrast und Dynamik des Ergebnisses beeinflusst werden.
Die Anwendung akzeptiert ein Bild aus der Galerie des mobilen Geräts als Eingabe oder nimmt ein neues Foto aus der App auf.
Nachdem das gewählte Bild verarbeitet ist, hat der Nutzer mit zwei Schiebereglern Einfluss auf das Ergebnis. Es kann die Dichte der Dreiecke der Delaunay-Triangulierung global verändert werden. Der zweite Regler beeinflusst die Opazität der Ebenen: steht der Regler auf 0 ist nur die Triangulierung zu sehen, bei 100 lediglich das Eingabebild.
Durch eine weitere Funktion hebt sich die App hervor. Mit einem Maskierungswerkzeug kann die Opazität der «obenliegenden» Ebene beliebig regional verändert wer-

(a) Eingabebild



(b) Ergebnis der Triangulierung mit konstanter Farbe



(c) Ergebnis mit Kantendarstellung

(d) Ergebnis mit gemischter Kanten- und Farbdarstellung

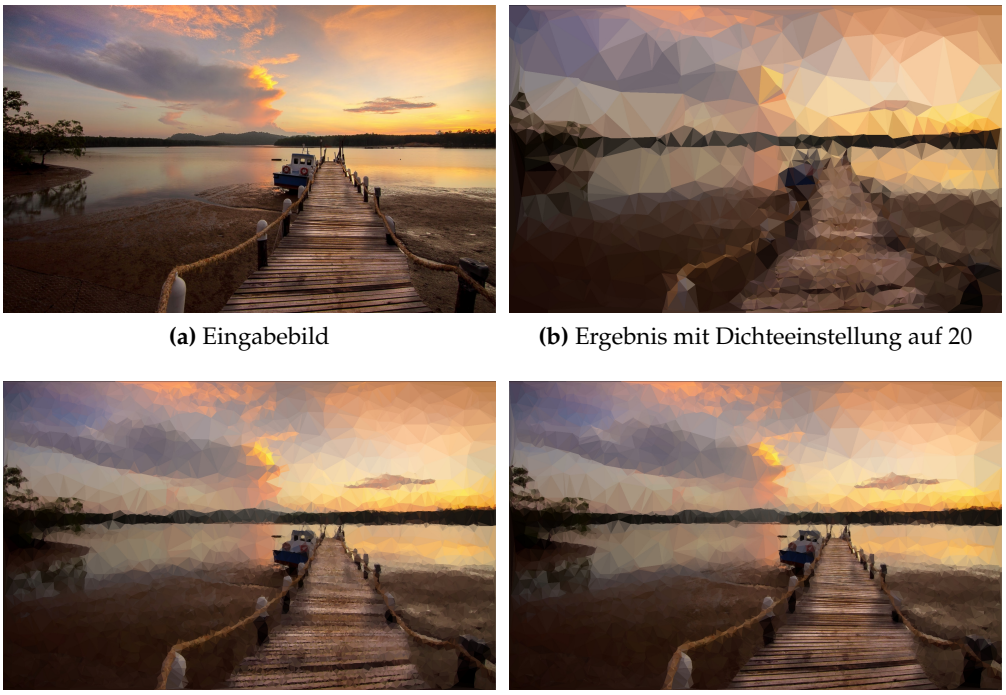
Abbildung 16: Ein- und Ausgabebilder von Polygonize

den [29], dargestellt in Abbildung 17d.

Anhand der Ergebnisse, vgl. Abbildung 17, wird deutlich, dass eine Abstraktion der Bildinhalte grundsätzlich, mit genauer Erfassung und Wiedergabe der Kanten, erreicht wird. Farbliche Übergänge werden eingehalten und erscheinen sanft. Für diese visuelle Qualität werden, im Vergleich zu anderen Ansätzen, viele Dreiecke benötigt. Dazu ist in Abbildung 18 ein vergrößerter Ausschnitt des Stegs dargestellt. Hierbei wird die Dichte von verwendeten Dreiecken in Regionen mit hoher Detaildichte deutlich. Anhand der Vergrößerung ist zu beobachten, dass insgesamt eine detailreiche Wiedergabe mit ästhetischem Eindruck erreicht wird. Die Punkte der Dreiecke unterlaufen jedoch keiner Anpassung der Bildinhalte, was an zahlreichen kleinen Dreiecken in Nähe harter Übergänge deutlich wird.
Scheinbar wird ein initiales, einheitliches Dreiecksnetz erstellt, das sich entlang der Kanten der Bildinhalte adaptiv feiner strukturiert.

Toolwiz Photos

Toolwiz Photos [47] bietet eine Vielfalt an Bildbearbeitungswerkzeugen. Als Kunstfil-



(a) Eingabebild

(b) Ergebnis mit Dichteeinstellung auf 20

(c) Ergebnis mit Dichteeinstellung auf 80

(d) Beispielhafte Benutzung der Opazitätseinstellung am Steg

Abbildung 17: Ergebnisse der Polygon Effect App

ter wird die Triangulierung angeboten, von der fünf Stufen vom Nutzer gewählt werden können, die jedoch frei konfigurierbar sind: die Dichte des Dreiecknetzes und eine Intensitätseinstellung stehen zur Auswahl. Bei hoher «Intensität» wird versucht, bestimmte homogene Bereiche in weniger Dreiecken zusammenzufassen. Die automatisierte Auswahl der homogenen Bereiche ist nicht kohärent. Dadurch entstehen visuelle Unregelmäßigkeiten, siehe Abbildung 19c. Bei sehr niedrigem Intensitätswert wird anstatt der Steilküste der Himmel mit weniger Dreiecken angenähert, wodurch die wahrgenommene Qualität nicht gesteigert wird.
Die Triangulierung wirkt zufällig, Details des Bildes werden nur durch eine hohe Dichte von Dreiecken angenähert, vergleichbar mit der Lösung der Polygon Effect App. In der Nähe von harten Übergängen der Bildinhalte und von Details, ist eine erhöhte Anzahl an kleineren Dreiecken zu beobachten, die der Bildstruktur nicht angepasst werden, wodurch Unstetigkeiten im Ergebnis entstehen.

PolyGen

PolyGen [9] ist auf die stilisierte Triangulation spezialisiert. Der Benutzer hat anfangs die Wahl zwischen der Erstellung eines Gradienten und der Bearbeitung eines Bildes, das neu aufgenommen oder aus der Galerie gewählt werden kann.
Auf das Ergebnis kann der Nutzer auf mehrere Arten Einfluss nehmen:

(a) Ergebnis mit Dichteeinstellung auf 80 **(b)** Vergrößerung des Ergebnis

Abbildung 18: Vergrößertes Ergebnis von Polygon Effect. Dichteeinstellung auf 80

Eine Zufallsfunktion bearbeitet das Bild mit unterschiedlichen Einstellungen. Dabei variiert die Größe der Dreiecke. Farbgebung und Adaption der Bildstrukturen bleiben grundsätzlich unverändert.
Als weitere Einstellungsmöglichkeit, kann auf den Farbeffekt Einfluss genommen werden. Verdunkeln und diversifizieren stehen kostenlos zur Verfügung, weitere können käuflich erworben werden. Die Option der Diversifizierung färbt Dreiecke in Farben ein, die im Bild nicht vorkommen, die es jedoch visuell harmonisch bunter gestalten.
Der Benutzer hat die Möglichkeit, direkt die Triangulation zu steuern. Zwischen sechs Arten der Punkteverteilung und der Option der manuellen Bearbeitung einer Triangulierung kann entschieden werden.
In Abbildung 20 sind die sechs Möglichkeiten der Triangulierung repräsentiert:

Spaced: eine zufällige, meist äquidistante Verteilung der Punkte ohne Clusterbildung; Abb. 20a

Flurry: Strukturelle Komplexitäten sollen durch viele Vertices erfasst werden; Abb. 20b

Edges: Punkte orientieren sich an Kanten im Bild; Abb. 20c

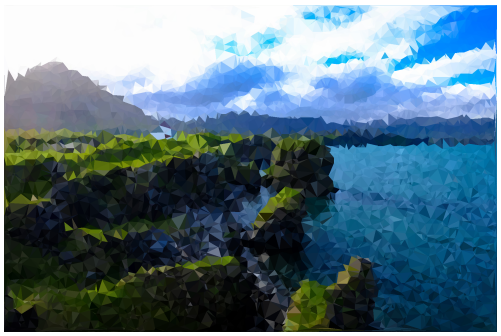
Clustered: zufällige Clusterbildung mit variierendem Abstand untereinander; Abb. 20d

Grid: ein quadratisches Einheitsgitter; Abb. 20e

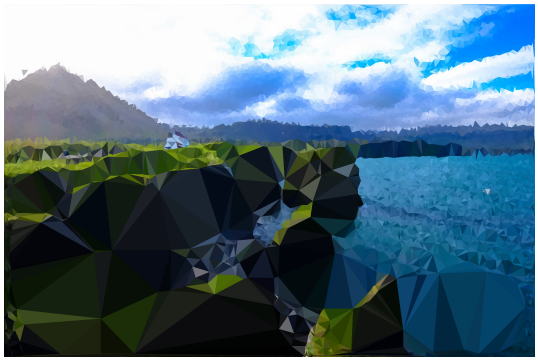
Diamond: ein verschobenes Einheitsgitter; Abb. 20f

Die zusätzliche Option der manuellen Bearbeitung der Vertices ist sehr effektiv für Details und kleinere Regionen, die bereits trianguliert wurden. Der Nutzer hat ebenfalls die Option, das Bild komplett manuell zu triangulieren.
Bei der manuellen Bearbeitung der Vertices kann die Position der einzelnen Punkte gesteuert werden, woraufhin der entsprechende Bereich neu trianguliert wird. Als weitere Optionen können Punkte hinzugefügt oder gelöscht werden.

(a) Eingabebild



(b) Ergebnis ohne strukturelle Adaption



(c) Ergebnis mit Adaption

Abbildung 19: Ergebnisse der Toolwiz Photo App

Zusätzlich kann die Zellgröße auf klein, mittel oder groß eingestellt werden; das Resultat ist beispielhaft in Abbildung 21 dargestellt. Ein Export des Ergebnisses wird ebenfalls angeboten.

Der einzige Ansatz von PolyGen, der die adaptive Triangulierung anstrebt, ist die *Edges*-Option, die die Orientierung an Kanten der Bildelemente umsetzt. Am Beispiel wird deutlich, dass Details in einigen Bildregionen so sehr abstrahiert werden, dass feiner strukturierte Inhalte nicht erfasst werden. Im Beispielbild 20c werden homogene Regionen in zu großen Dreiecken zusammengefasst, sodass Unstetigkeiten in der Segmentierung entstehen. Optisch ähnliche Bereiche werden unterschiedlich verarbeitet. Ein visuell zufriedenstellendes Ergenis wird daher nicht erreicht.

Trimaginator

Die kommerzielle App Trimaginator [12] bietet eine Vielzahl an Einstellungen. Unter anderem stehen Renderstile, Farbeinstellungen von Dreiecken und Kanten, verschiedene Geometrien, die die Triangulierung beeinflussen, beliebige Opazität, eine zufällige und eine manuelle Triangulierung zur Auswahl.
Die Anwendung stellt als besondere Option die Triangulierung durch Formerkennung

(a) Spaced **(b)** Flurry **(c)** Edges

(d) Clustered **(e)** Grid **(f)** Diamond

Abbildung 20: Ergebnisse der Triangulierungsarten in PolyGen

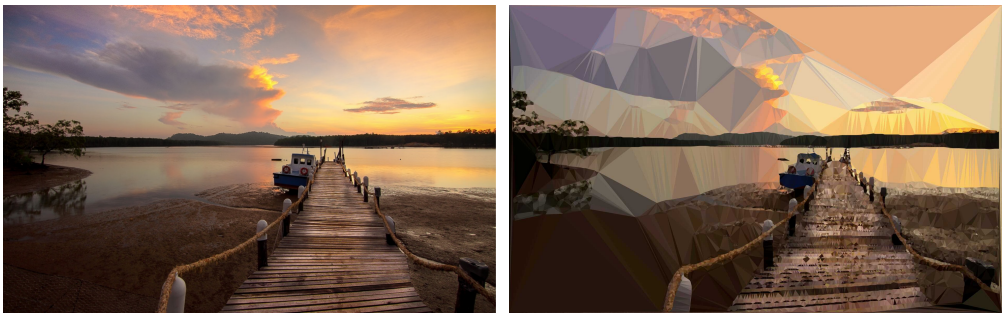
(a) Zellgröße mittel **(b)** Zellgröße klein

Abbildung 21: Ergebnisse der Zellgrößenanpassung in PolyGen

bereit. Das Ergebnis ist in Abbildung 22 zu sehen und zeigt deutlich, dass Bildelemente und deren Kanten präzise erfasst werden. Bei weichen Farbübergängen entsteht eine grobmaschigere Triangulierung, die in dem sonst sehr homogenen Bildeindruck auffällt. Details werden durch eine hohe Anzahl an Dreiecken erfasst. Insgesamt erreicht Trimaginator ein visuell ansprechendes Ergebnis.

PolyColor Game [5] präsentiert dem Nutzer fertige Delaunay-Triangulierungen von Motiven, deren Dreiecke vom Benutzer im Sinne von "Malen nach Zahlen" eingefärbt werden sollen.

Mit **Delaunay Raster** wird ein Framework bereit gestellt, das eine manuelle Delaunay-Triangulierung von Bildern ermöglicht. Dass qualitative Hochwertige Ergebnisse erzielt werden, zeigt die Verwendung bei Covern von Magazinen [13].



(a) Eingabebild **(b)** Ergebnis der Formerkennung

Abbildung 22: Ergebnisse von Trimaginator

3.4 Methoden der Ausgangsveröffentlichung

In dieser Sektion werden Aspekte erläutert, die aus der Veröffentlichung [LG18] stammen, auf der diese Arbeit aufbaut. Diese Thematiken werden in dieser Anwendung nicht umgesetzt, da sie weiterführende Arbeit darstellen und nicht für eine prototypische Anwendung benötigt werden. Stattdessen werden die Aspekte in Bezug zu den alternativen Ansätzen gebracht, was die Einbindung in dieses Kapitel begründet.

3.4.1 Initiale Triangulierung

Eine Triangulierung kann aus verschiedenen initialen Strukturen beginnen. Daraus resultieren unterschiedliche Eigenschaften des Ergebnisses.
Das Einheitsgitter strukturiert homogene und unstetige Regionen gleichermaßen, sodass in homogenen Regionen weniger Vertices nötig wären, um die Bildinhalte zu erfassen. In Regionen mit höherer Detaildichte fehlen hingegen Punkte, mit dem Ergebnis, dass nicht alle Details im Ergebnisbild wiedergegeben werden können. Ein Vorteil des Einheitsgitters ist der beschränkte Einfluss eines Punktes auf seine direkt benachbarten Punkte [LG18]. Bei geringem lokalen Einfluss wird eine effiziente Optimierung unterstützt. Der lokale Einfluss der Vertices ist ein entscheidender Aspekt des Minimierungsprozesses; erläutert im konzeptuellen Teil der Arbeit.
Trimaginator [12] bietet alternativ die Wahl zwischen zwei und sechs Ausgangsdreiecken, die das Bild erfassen und mit denen der Triangulierungsprozess startet.

Vom Benutzer als interaktive Eingabe generiert oder als automatisierter Vorverarbeitungsschritt, ist das Festlegen von Bereichen mit initialen Punkten möglich. Durch diese Option werden mit erhöhter Wahrscheinlichkeit wichtige Details, wie die Augen bei einem Porträt, deutlicher im Ergebnis herausgestellt.
Lawonn und Günther nutzen die inverse kumulative Verteilungsfunktion (englische Abkürzung *inverse CDF*), um Punkte zu generieren, die einer Delaunay-Triangulierung hinzugefügt werden [LG18]. Durch diese Methode werden ausgehend von einer ein-

heitlichen Verteilung Zufallszahlen generiert, die einer kumulativen Verteilungsfunktion folgen [Cha12]. Bei einer kumulativen Verteilungsfunktion werden die Wahrscheinlichkeiten summiert, die ein Unterschreiten (und Zutreffen) der Zufallsvariable betreffen [GS97].

3.4.2 Benutzerinteraktion

Alle evaluierten Anwendungen weisen Formen der Benutzerinteraktion auf. Die Punktedichte konnte indirekt bei jeder Anwendung beeinflusst werden, doch nur PolyGen und Trimaginator erlaubten ein direktes Manipulieren der Punkte.
Eine direkte Manipulation wird ebenfalls in dem Verfahren von Lawonn und Günther unterstützt. Es werden mehrere Möglichkeiten vorgestellt, die der Nutzer/Künstler hat, um den Gestaltungsprozess zu beeinflussen [LG18].

Details. Neue Punkte werden durch die Kanten- oder Dreiecksteilung erstellt. Dadurch verfeinert sich die Triangulierung. Mit dem Kollabieren einer Kante können unpassend liegende Dreiecke entfernt werden. Der Kantenflip kann genutzt werden, um wohlgeformtere Dreiecke zu erreichen. Um die Ausrichtung einer Textur anzupassen, kann eine Ausgangskante des Dreiecks nach belieben gewählt werden.

Regionen. Das Pinselwerkzeug soll die Bearbeitung erleichtern, indem mehrere Punkte nicht mehr einzeln bearbeitet werden müssen. Der variable Radius des Werkzeugs erleichtert das Einfügen oder Entfernen von Punkten in beliebig großen Regionen.

Flächen. Der Benutzer hat die Möglichkeit beliebig große Flächen zu maskieren, in denen anschließend alle darin liegenden Punkte entfernt und mit einer vom Nutzer gewählten Anzahl neu trianguliert werden. Dadurch können ganze Bereiche höher aufgelöst oder zu einem grobmaschigeren Netz zusammengefasst werden.

3.5 Zusammenfassung

Dieses Kapitel fasst den Stand der Technik zusammen. Dafür werden alternative Ansätze erläutert, die eine stilisierte Triangulierung implementieren. Als direkter Vergleich zu Mesh werden mobile Anwendungen analysiert, um Anforderungen und Unterschiede herauszuarbeiten. Diese Unterschiede verdeutlichen, dass das umgesetzte Verfahren bisher in keiner mobilen Anwendung Verwendung findet. Alternative Apps bieten jedoch eine größere Auswahl an Einstellungen, durch die der Nutzer das Ergebnis individualisieren kann.
Im folgenden Kapitel wird die konzeptuelle Ausarbeitung der Anwendung dargelegt. Einzelheiten und Methoden der Triangulierung werden vorgestellt. Die Umsetzung einer mobilen Anwendung, die Entwürfe zweier Stilisierungen und der Optimierungs-

ansätze werden thematisiert. Das Kapitel der vergleichbaren Arbeiten hat dazu einen Einblick gegeben, sodass Ideen und Verfahren im konzeptuellen Teil nachvollzogen werden können.

4 Konzept

Das Ziel dieser Arbeit ist die dokumentierte prototypische Entwicklung einer mobilen Anwendung. Mittels der App, genannt «Mesh », soll dem durchschnittlichen Benutzer die Möglichkeit eröffnet werden, ein neues Bildabstraktionsverfahren nutzen zu können. Dieses Verfahren wird von Lawonn und Günther vorgestellt [LG18] und arbeitet mit einem GPU-basierten Rasterisierungsansatz. Die stilisierte Triangulierung ist rechenintensiv. Um möglichst vielen Nutzern das Verfahren zugänglich zu machen, ohne dass leistungsstarke mobile Geräte benötigt sind, soll die Berechnung des Verfahrens nicht auf dem Endgerät erfolgen. Aufgrund dessen wird ein Host-Client System entwickelt. Zusätzlich zu einer mobilen Anwendung ergänzt ein Server den Aufbau, auf dem die rechenintensive Triangulierung durchgeführt wird. Diese Konstruktion stellt dem Client, dem Nutzer, die stilisierte Triangulierung bereit. Da die grafische Schnittstelle eines Rechners, der X-Server, und eine Grafikkarte auf Servern nicht obligatorisch sind, wird im Folgenden ein CPU-basierter Ansatz vorgestellt, um unabhängig von diesen Ressourcen zu sein.
Dieses Kapitel umfasst zusätzlich die Entwicklung von Optimierungsansätzen des Triangulationverfahrens. Im Lauf der Arbeit erfolgt die Auswertung und Einordnung der erzielten Ergbenisse der Ansätze.
Ein weiteres Ziel stellt die Imitation eines Kunststils dar. Dazu wird eine Renderingmethode ausgearbeitet, die die Natürlichkeit eines handgefertigten Werks des Künstlers erfassen soll. Zusätzlich erfolgt die Vorstellung eines weiteren Renderstils einer Triangulation.

4.1 Aufbau

Aus folgenden Komponenten setzt sich das ausgearbeitete System zusammen:

Die App. Mit der mobilen Anwendung wird die Kommunikation zwischen Nutzer und dem Server realisiert. Die Wahl des Eingabebilds und der Empfang des Ergebnisses findet dort statt.

Der Algorithmus. Das Ergebnisbild, die stilisierte Triangulierung, wird durch den Algorithmus erstellt.

Der Server. Der Kommunikationspartner der App. Der Server nimmt das Eingangsbild entgegen und ist für die Rücksendung des Ergebnisses zuständig.

Die mobile Anwendung dient dem Nutzer als Interaktionsplattform. Von dort wird ein vom Benutzer ausgewähltes Bild an den Server gesendet, der daraufhin mit der Verarbeitung beginnt. Sobald das Ergebnis zur Verfügung steht, kann der Nutzer die App zur indirekten Kommunikation mit dem Server verwenden, um das Bild auf das mobile Endgerät herunterzuladen und es zu speichern.
In den folgenden Sektionen werden die Anforderungen und die Funktionsweise der einzelnen Komponenten dargelegt.

4.2 App

Um ein Programm möglichst vielen Menschen zur Verfügung zu stellen, ist ein einfacher und durchgehend erreichbarer Zugang erforderlich. Diese Anforderungen erfüllt eine mobile Anwendung, die unabhängig von der Plattform ist und somit von möglichst vielen Betriebssystemen unterstützt wird. Durch die Verbreitung in Apple's *App Store* und bei *Google Play* sind viele Nutzer zu erwarten, die aufmerksam auf die Anwendung werden, da dadurch die beiden meist verbreitetsten Betriebsysteme und entsprechend viele Endgeräte unterstützt werden. Das Ziel dieser Arbeit ist keine marktreife Anwendung, sondern ein Prototyp. Es wird untersucht, ob die Portierung des Verfahrens auf die CPU eine vorteilhafte Erweiterung darstellt. Aufgrund der prototypischen Entwicklung, erfolgt in dieser Arbeit keine Studie über Anwenderfreundlickeit oder Gebrauchstauglichkeit.
Die Entwicklung mit React-Native realisiert die plattformunabhängige Implementation. Primär wird die Anwendung für Androidgeräte umgesetzt, da technisch mehr Freiheiten möglich sind. Auf diese wird im Teil der Implementation eingegangen.

Mit der mobilen Anwendung soll ein verlässliches System entwickelt werden. Abstürze, verursacht durch Spitzen plötzlicher Rechenlast, sollen mit Hilfe eines Host-Client Systems umgangen werden. Dadurch wird der ressourcenbedürftige Algorithmus auf ein System verlagert, bei dem von einer konstanten Leistung ausgegangen werden kann. Dem Benutzer wird ermöglicht, ein beliebiges Foto bearbeiten zu lassen, ohne dass das Endgerät durch benötigte Rechenlast oder Berechnungszeiten eingenommen wird. Bei der Benutzung der im vorherigen Kapitel genannten Apps kam es vor, dass eine Anwendung abstürzt oder mit geringer Bildwiederholungsrate arbeitet, sodass eine unterbrechungsfreie Nutzung der App und des Endgeräts nicht möglich ist. Mit der Entwicklung eines Host-Client Systems entsteht für die Nutzung der Anwendung die Anforderung an eine Verbindung mit dem Internet. Aufgrund von stetig wachsender Infrastruktur und sinkenden Kosten der Mobilfunkbetreiber bzw. der WLAN-Zugriffsmöglichkeiten, wird diese Anforderung als zumutbare Hürde für den Benutzer betrachtet.

Die Anwendung ist eine grafische Oberfläche, die dem Nutzer zur Interaktion dient. Es soll dem Benutzer die Wahl zwischen einem bestehenden Foto vom Gerät oder einem neuen, aus der App heraus aufgenommenen, eröffnet werden. Mit anschließender Zustimmung des Anwenders, kann das Bild zum Server gesendet werden. Das Bild wird vor dem Versenden mit der *base-64* Kodierung in eine Zeichenkette verarbeitet und dem Body der Anfrage angehängt. Das gewählte Bild wird an eine Adresse des Servers gesendet. Diese Adresse ist in der Anwendung festegschrieben.
Mit der Darstellung der Bildinformationen als *base-64* werden die Pixelwerte aufgelistet und als Zeichenkette versendet [Bay10]. Die Kodierung resultiert in einem 7-Bit-ASCII-Code, der als internationales Schema für Datenaustausch etabliert ist [Gol02]. Die kodierte Zeichenkette wird in ein JSON-Dokument eingebettet, das von der Anwendung erstellt und an den Server per Anfrage übertragen wird. JSON (JavaScript Object Nota-

tion) ist ein Datenformat, das für den Informationsaustausch konzipiert ist [34].
Sollte die Übertragung des Bildes an den Server fehlschlagen, wird dem Nutzer eine Fehlermeldung angezeigt. Dabei muss der Nutzer über die Fehlerart informiert werden. Bei einer fehlgeschlagenen Übertragung kann der Nutzer den Vorgang innerhalb weniger Minuten mit Erfolgsaussicht wiederholen. Wenn der Server nicht erreichbar ist, soll dem Nutzer eine entsprechende Information dargeboten werden, sodass sich der Benutzer auf den Umstand einstellen kann.
Der Nutzer soll die Möglichkeit haben, die Stilisierung der Anwendung zu beeinflussen. Dazu sollen vom Nutzer gewählte Parameter zusammen mit dem gewählten Bild an den Server übertragen werden.

Sobald das Ergebnis vom Server bereitgestellt wird, soll es die mobile Anwendung dem Nutzer ermöglichen, das Ergebnis herunterzuladen und auf dem Client, dem mobilen Gerät, zu speichern. Für die Umsetzung des Vorgangs ergeben sich zwei Möglichkeiten. Die erste Variante bietet dem Benutzer die Option, den Zeitpunkt des Herunterladens selbst zu wählen und über eine *Activity* der App aktiv das Ergebnis anzufragen. Falls das Ergebnis zu dem Zeitpunkt noch nicht vorliegt, erscheint dem Nutzer eine entsprechende Meldung und der Versuch kann zu einem späteren Zeitpunkt wiederholt werden. Die zweite Option automatisiert den Mechanismus, indem sich der Zustand der App ändert, sobald ein Bild erfolgreich an den Server übertragen wurde. Eine Aufgabe, die im Hintergrund der App arbeitet, schickt alle 15 Minuten eine Anfrage per *Get*-Methode an den Server.
Eine beispielhafte *Get*-Anfrage umfasst 450 Bytes. Zusammen mit der Antwort und TCP/IP *Headern* ergibt sich in dem Beispiel eine Menge von circa 1380 Bytes [FDFL$^+$14]. In einer Stunde, in der das Ergebnisbild noch nicht zur Verfügung steht, summiert sich eine Datenmenge von 5,5kB. Zusätzlich addiert wird die erfolgreiche Übertragung des Ergebnisbildes. Die Größe des Bildes beeinflusst die Übertragungsdatenmenge. Bei einer exemplarischen Bildgröße von 700kB ergibt sich eine Gesamtsumme von circa 706kB, die entsteht, wenn der Nutzer eine Stunde auf das Bild wartet und es anschließend heruntelädt. Die überschüssige Menge an Daten, die bei der Übertragung ohne Ergebnisbild entsteht, ist deutlich geringer als das Bild selbst. Anfragen und Antworten, die das Ergebnisbild nicht enthalten, können als Misserfolg gewertet werden. Um 1MB an Daten zu erhalten, die als Misserfolg gewertet werden, muss die Anwendung etwa 182 Stunden ($\frac{1000kB}{5.5kB*h} = 181.82h$) im Hintergrund arbeiten.

Die mobile Anwendung als Einstiegspunkt soll für den Anwender zwei Aspekte umsetzen. Dem Nutzer soll eine Interaktion mit der Triangulierung ermöglicht werden. Rendereinstellungen und die Wahl des Bildes stehen dabei im Vordergrund. Aufgrund des Host-Client Systems ist eine Manipulation des Bildes mit direktem Feedback nicht möglich. Der Benutzer muss für jede Anpassung des Ergebnisses ein neues Bild an den Server senden.
Der zweite wichtige Aspekt ist eine markante Repräsentation der stilisierten Triangulierung, sodass ein Wiedererkennungswert bei Nutzern etabliert werden kann.

4.3 Formalisierung der Triangulierung

Dieses Kapitel umfasst die Einführung in den umgesetzten Algorithmus. Der Ansatz der Energieminimierung bei einer Triangulation wird in der Veröffentlichung von Lawonn und Günther vorgestellt [LG18]. Die Problemstellung, die Zielsetzung und das Konzept werden daraus übernommen und im Folgenden erläutert.

4.3.1 Problemformulierung

Die Neuformulierung der stilisierten Triangulation setzt die Darstellung der Bildinhalte mit Dreiecken als Optimierungsproblem um. Bei dem Energieminimierungsansatz werden die Punkte der Dreiecke so verschoben, dass die Dreiecke den Bildinhalt wiedergeben. Dazu folgt die Definition eines Fehlermaßes, das den Unterschied zwischen dem Eingabebild und den Ergebnissen der Iterationsschritte erfasst. Um einen Vergleichs- und Zielwert bei der Berechnung zu erhalten, wird im Folgenden die Farbgebung dargelegt. Zunächst erfolgt die Definition der Strukturierung eines Dreiecksnetzes für das Bild.
Gleichung 11 definiert ausgehend von einem Bild I und dessen Domäne U die Abbildung in die Farbdomäne C [LG18].

$$I : U \subset \mathbb{R}^2 \rightarrow C \tag{11}$$

Das Eingabebild wird in Dreiecke $\mathcal{T}$ unterteilt, mit dem Ziel, das Eingabebild so in der Triangulierung darzustellen, dass die Abweichung, gemessen an einem Fehlermaß, möglichst gering ist. Die Unterteilung der Domäne U in Dreiecke $\mathcal{T}$ unterliegt Eigenschaften, die in Gleichung 12 zusammengefasst werden.

$$\bigcup_{T \in \mathcal{T}} T = U \tag{12}$$

Dabei gilt, dass $T_i \cap T_j$ mit T_i, $T_j \in \mathcal{T}$ und $T_i \neq T_j$ keine Überschneidung von Flächen der Dreiecke entsteht. Stattdessen gilt, dass sich adjazente Dreiecke einen Punkt (0-Simplex) oder eine Kante (1-Simplex) teilen. Sind die Dreiecke nicht adjazent, sind T_i und T_j disjunkt [LG18].

4.3.2 Farbgebung

Die Berechnung konstanter Farbe eines Dreiecks wird in Abschnitt 2.4.1 erwähnt. Mit Gleichung 5 wird die Herleitung von c in der Domäne eines Dreiecks T definiert, das die Farbe f des Dreiecks darstellt [LG18]:

$$f(x, y) = c \tag{13}$$

Ziel der Farbgebung ist den durch das Dreieck erfassten Bereich optimal mit einer Farbe zu repräsentieren. Der Farbwert mit gringster Abweichung stellt dabei der Mittelwert

des abgedeckten Bereichs dar.

Lawonn und Günther erläutern weiterführend die Berechnung der linearen Farbe eines Dreiecks [LG18]:

$$f(x, y) = ax + by + c \tag{14}$$

Durch die Minimierung des quadrierten Approximationsfehlers $E(T)$ können die Koefffizienten a, b und c berechnet werden[LG18]:

$$E(T) = \frac{1}{2} \int_T (I(x) - f(x))^2 \, dx \rightarrow min \tag{15}$$

Das Polynom f wird als lineares Gleichungssystem dargestellt und mittels einer Cholesky-Dekomposition gelöst. In dieser Arbeit wird die konstante Farbgebung umgesetzt, weshalb die Berechnung der linearen Farbe eines Dreiecks als Ausblick verbleibt und darauf lediglich verwiesen wird [LG18].
Die komplexere Färbung der Dreiecke resultiert in gesteigerter Qualität des Ergebnisses. Im Fall der linearen Farbe (des Gradienten) ist die Farbe in jedem Dreieck flexibler, was sich in einem niedrigerem Approximationsfehler äußert [LG18].

Das Verfahren hat den Anspruch, eine ästhetische stilisierte Triangulierung zu generieren. Die visuelle Wahrnehmung ist folglich ausschlaggebend. Das Lab-Farbmodell zielt auf die menschliche Wahrnehmung von Farben ab; Distanzen im Farbraum korrelieren mit den wahrgenommenen Farbunterschieden [TM98]. L, a und b stellen bei dem Lab-Farbmodell drei Achsen eines dreidimensionalen Koordinatensystems dar. Mit L wird die Helligkeit der Farbe beschrieben, a steht für die rot-grün und b für die gelb-blau Achse [NFHS12]. Die Imitation des künstlerischen Stils verwendet das Lab-Farbmodell. Die wahrgenommene Helligkeit einer Farbe ist dabei entscheidend.

4.3.3 Minimierungsansatz

Das Ziel des Algorithmus ist die Annäherung einer Triangulierung an ein beliebiges Bild. Die einzelnen Dreiecke repräsentieren dafür den Bereich des Bildes, den sie abdecken. Wie bereits erwähnt, stellt bei konstanter Farbgebung der Mittelwert des abgedeckten Bereichs die Farbe des Dreiecks dar. Um die Annäherung so präzise, wie möglich zu gestalten und dadurch einen geringen Approximationsfehler zu gewährleisten, strebt die Triangulierung gegen ein Minimum:

$$E = \sum_{T \in \mathcal{T}} E(T) + \lambda p(t) \ \rightarrow min \tag{16}$$

E stellt den Term der Energie des gesamten Bildes dar, den es zu minimieren gilt. Er setzt sich aus allen Dreiecken $T \in \mathcal{T}$ der Triangulierung zusammen, für die jeweils die lokale Energie und ein Regularisierungsterm berechnet werden [LG18].
Die Minimierung der Energie eines Dreiecks beeinflusst die Positionierung der einzelnen Punkte des Dreiecks. Das resultiert daraus, dass die Energie für das Potenzial der

Anpassung steht. Hohe Energie bedeutet folglich, dass die optimale Position des Dreiecks nicht erreicht ist. Ein niedriger Wert der Energie äußert sich in einem Dreieick, das den erfassten Bereich des Bildes farblich mit geringer Abweichung darstellt.
Der Regularisierungsterm $p(T)$ mit variablem Einfluss durch λ wirkt dem Energieterm entgegen. Er beeinflusst die Anordnung der Punkte in der Triangulierung mit dem Ziel, dass wohlgeformte Dreiecke erhalten bleiben und keine Überschneidungen benachbarter Dreiecke entstehen.
Für Gleichung 16 gilt, dass die Koordinaten sowie die Anzahl der Dreiecke bisher Unbekannte darstellen [LG18].

Gegeben seien alle Vertices der Triangulierung $\mathbf{v} \in \mathcal{V}$. Für die Triangulierung wird zunächst angenommen, dass sie aus einer festen Anzahl an Dreiecken besteht, wodurch die Unbekannten aus Gleichung 16 auf die Positionen der Punkte beschränkt wird.
Da die Triangulierung aus Dreiecken besteht, die separat voneinander betrachtet werden können, wird deutlich, dass in Folge dessen auch die Energieterme der Dreiecke separat erfasst werden können. Dies äußert sich bei den einzelnen Vertices $\mathbf{v} \in \mathcal{V}$. Sie haben lokalen Einfluss auf den Approximationsfehler/ den Energieterm, aller angrenzender Dreiecke. Angrenzende Dreiecke werden folgendermaßen definiert: $\mathcal{A}(\mathbf{v}) = \{T \in \mathcal{T} : \mathbf{v} \subset T\}$.
Die Erkenntnis des lokal beschränkten Einflusses fordert eine neue Formulierung der ursprünglichen Gleichung 16 [LG18]:

$$E = \sum_{\mathbf{v} \in \mathcal{V}} E(\mathbf{v}) \ \rightarrow min \tag{17}$$

Die Energie E repräsentiert, wie in Gleichung 16, den Approximationsfehler des gesamten Bildes. Statt den Fehler der Dreiecke zu summieren, wird mittels der neuen Formulierung eine Iteration über die Vertices etabliert; mit lokalen Energietermen [LG18]:

$$E(\mathbf{v}) = \sum_{T \in \mathcal{A}(\mathbf{v})} \frac{E(T)}{3} + \lambda p(\mathbf{v}) \tag{18}$$

Der Regularisierungsterm wird ebenso auf einzelne Vertices mit dem selben Einfluss bezogen: $p(T) = \frac{1}{3} \sum_{\mathbf{v} \in T} p(\mathbf{v})$.
Der Faktor von $\frac{1}{3}$ ergibt sich aus der Iteration der adjazenten Dreiecke eines Punkts, da von jedem Punkt eines Dreiecks ausgegangen wird.
Die globale Minimierung ist durch die Minimierung der Approximationsfehler der einzelnen Vertices $\mathbf{v} \in \mathcal{V}$ einer gegebenen Triangulierung weiterhin möglich, da $E(\mathbf{v})$ positiv ist [LG18].

4.3.4 Regularisierung

Überschneidungen von Dreiecken wirken negativ auf die ästhetische Qualität des Ergebnisses. Unstetigkeiten in der Struktur und Färbung in dem entsprechenden Bereich

sind in solch einem Fall zu beobachten. Ebenso kann eine Verletzung der Delaunay-Triangulierung resultieren. Um eine initiale Wohlgeformtheit der Triangulierung aufrechtzuerhalten, darf kein Punkt seine konvexe Hülle verlassen.

$$\mathcal{N}_1(\mathbf{v}) = \{\mathbf{w} \in (\mathcal{V} \setminus \mathbf{v}) : \mathbf{w} \in \mathcal{A}(\mathbf{v})\} \tag{19}$$

Diese Gleichung beschreibt die Positionierung eines Punkts in seiner konvexen Hülle. Die Fläche wird durch die adjazenten Dreiecke und den entsprechenden Kanten erzeugt, visualisiert in Abbildung 23. $\mathcal{N}_1(\mathbf{v})$ steht dabei für alle direkt adjazenten Vertices von Punkt **v**.

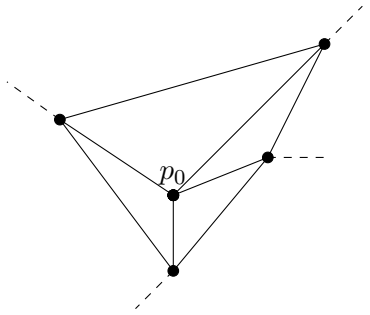


Abbildung 23: Visualisierung der konvexen Hülle eines Vertex

Um diese Eigenschaft der Triangulierung zu erhalten, wird ein Regularisierungsterm eingeführt [LG18]:

$$p(\mathbf{v}) = \frac{1}{2|\mathcal{N}_1(\mathbf{v})|} \sum_{\mathbf{w} \in \mathcal{N}_1(\mathbf{v})} (\mathbf{w} - \mathbf{v})^2 \tag{20}$$

Mittels Funktion 20 wird ein Vertex in Richtung des Zentrums seiner konvexen Hülle verschoben [LGT+19]. Die Funktion stellt eine Glättung nach Laplace dar. Dafür wird $\mathbf{w}^2 := \langle \mathbf{w}, \mathbf{w} \rangle$ als Skalarprodukt definiert [LG18], was den Einfluss der benachbarten Punkten auf die Verschiebung bestimmt. Mit $\lambda = 0.001$ beschränken Lawonn und Günther den Einfluss der Verschiebung, um Vertices die Möglichkeit zu lassen, sich an Bildinhalte anzupassen.
Der Faktor 2 des Terms $-2\mathbf{wv}$ innerhalb der Summe ergibt sich aus der 2. binomischen Formel und wird durch eine Mittlung von $\frac{1}{2|\mathcal{N}_1(\mathbf{v})|}$ mit der doppelten Anzahl an adjazenten Vertices ausgeglichen.

Für diesen Ansatz wird die simplere Glättung, vgl. Formel 3, verwendet, um Rechenaufwand zu verringern. Ausgehend vom betrachteten Vertex werden alle direkt benachbarten Punkte zur Berechnung des Zentrums der konvexen Hülle einbezogen. Ein Vektor wird ausgehend vom berechneten Zentrum mit Richtung zum betrachteten Vertex erstellt.

Je wohlgeformter die Triangulierung ist, desto ungenauer können Details im Bild durch die Struktur der Dreiecke erfasst werden. Aus diesem Grund wird der Einfluss des Vektors in Richtung Zentrum mit einem Faktor von 0.015 beschränkt. Eine Verschiebung in Richtung des Zentrums der konvexen Hülle eines Vertex wird somit erreicht. Dieser Vorgang wirkt gegen Überschneidungen von Dreiecken und schränkt die Anpassungsfähigkeit der Punkte in nicht zu großem Ausmaß ein.

4.3.5 Gradientenverfahren

Das Gradientenverfahren wird bei der Verschiebung der Vertices eingesetzt [LG18]:

$$\mathbf{v} \leftarrow \mathbf{v} - h\frac{dE(\mathbf{v})}{d\mathbf{v}} \tag{21}$$

Die aktualisierte Position eines Punktes ergibt sich aus der resultierenden Energie bei Verschiebung des Punktes. Die Energieergebnisse stellen den Gradienten dar, sodass die Verschiebung in Richtung der geringsten Energie vollzogen wird. Mit h wird der Energieterm gewichtet. Die gesamte Schrittweite bei dem Gradientenabstieg ist essenziell für die Auswirkung eines Iterationsschritts der Vertices. Bei zu großer Schrittweite können bei kleinen Dreiecken in einer Region Überschneidungen der Dreiecke entstehen. Aus diesem Grund beschränken Lawonn und Günther die Verschiebung eines Punktes auf maximal 0,2 Pixel. Zusätzlich werden die Vertices in ihrem Algorithmus in einem *out-of-place* Verfahren aktualisiert [LG18]. Für dieses Verfahren wird zusätzlicher Speicher benötigt, in dem vorerst jede einzelne Aktualisierung zwischengespeichert wird. Sind alle Vertices bearbeitet worden, stellt der zusätzliche Speicher den aktuellen Stand dar und der Ausgangsstand ist veraltet. Dadurch werden Konflikte beim Lesen und Schreiben bei der Bearbeitung der einzelnen Punkte vermieden.
Die maximale Schrittgröße kann während der Laufzeit variieren, damit die in Abschnitt 2.3 genannte Oszillation um das Minimum nicht eintritt [LG18].
Der Abschnitt der Optimierungsansätze beinhaltet unterschiedliche Varianten der Schrittwahl des Gradientenverfahrens. Ohne Optimierungsansatz beträgt die Schrittweite in dem CPU-basierten Ansatz die Hälfte eines Pixels. Die CPU-basierte Version der Energieminimierung ist durch eine deutlich gesteigerte Berechnungszeit limitiert. Die Auswirkungen werden im Teil der Evaluation ausgeführt.

4.4 Triangulierung

4.4.1 Topologie

Für eine stilisierte Triangulierung eines Bildes gelten mehrere Ansprüche an die Topologie. Wie Lawonn und Günther gezeigt haben, äußert sich die Nutzung eines Einheitsgitters als initiale Struktur in einem höheren Approximationsfehler des Ergebnisses. Aufgrund dessen ist eine adaptive Struktur obligatorisch. Die Topologie muss es ermöglichen, Punkte in die bestehende Triangulierung nach Bedarf hinzuzufügen. Aus

diesem Anspruch resultiert die separate Speicherung von Punkten und Kanten. Zusätzlich wird die Relation zwischen Punkten und Kanten auf verschiedene Arten gespeichert. Um anliegende Dreiecke einer Kante erfassen zu können, ist in einer Relation enthalten, welche Punkte mit der gegebenen Kante die adjazenten Dreiecke formen. Explizit bedeutet das, dass ein Eintrag dieser Relation mittels eines Index einer Kante zugeordnet wird. Dieser Eintrag umfasst vier Indizes: Anfangs- und Endpunkt der Kante an den ersten beiden Stellen des Eintrags; an Stelle drei und vier ist jeweils der Punkt gespeichert, der ein Dreieck mit Index eins und zwei formt.
Als weitere Relation ist die Punkt-zu-Punkt-Adjazenz zu nennen. Dazu wird einem Punkt jede anliegende Kante zugeordnet. Durch die zuvor beschriebene Zuordnung ist es möglich, alle direkt benachbarten Punkte zu ermitteln.
Die Farbe eines Dreiecks wird gesondert behandelt. Dazu werden Farben und Dreiecke zusätzlich gespeichert. Eine zusätzliche Relation ist die Zuordnung zwischen Kanten und Dreiecken. Dort wird gespeichert, welche Dreiecke sich eine Kante teilen. Mittels dieser Relationen ist es beispielsweise möglich, mit drei Vertices als Eingabe das eingeschlossene Dreieck und dessen Farbe und Energie zu ermitteln.

4.4.2 Methoden und Ablauf

Pro Iterationsschritt wird ein Dreieck behandelt. Das ermöglicht die Visualisierung des kompletten Prozesses und erfordert eine Liste, die die zu bearbeitenden Dreiecke speichert. Als Iterationsschritt wird zunächst der zuletzt berechnete Fehler des aktuellen Dreiecks ausgewertet. Bei Überschreitung eines Schwellwerts, erfolgt eine Unterteilung des Dreiecks, die Wiederherstellung der Umkreisbedingung und der Gradientenschritt sowie die Regularisierung für alle drei Vertices. Bei Unterschreitung des Schwellwerts werden lediglich das Gradientenverfahren und die Anwendung des Regularisierungsterms ausgeführt. Je nach Fall werden die entsprechenden Dreiecke der Bearbeitungsliste hinzugefügt.

Wenn die Energie eines Dreiecks einen Schwellwert überschreitet, wird es in drei kleinere Dreiecke unterteilt. Dadurch wird die Adaption an Bildelemente eingeleitet. Notwendig für die Unterteilung ist ein zusätzlicher Punkt, der im Schwerpunkt des ursprünglichen Dreiecks platziert wird. Daraufhin folgt eine Aktualisierung der Topologie:

Vertices: Der Vertexliste wird ein Punkt hinzugefügt.

Kanten: Drei neue Kanten entstehen. Jeweils von den Vertices des Ausgangsdreiecks zum neuen Punkt. Die Kanten des veralteten Dreiecks werden aktualisiert: die entstehenden Dreiecke bilden die neuen adjazenten Dreiecke.

Dreiecke: Die drei erstellten Dreiecke werden der Liste der Dreiecke hinzugefügt. Die Zuordnung von Kanten und den anliegenden Dreiecken wird aktualisiert.

Sobald ein Punkt der Struktur hinzugefügt wird, kann die Umkreisbedingung der Delaunay-Triangulierung verletzt und eine Wiederherstellung der Bedingung nötig werden.
Um zu testen, ob die Bedingung besteht und sich keine weiteren Punkte im Umkreis befinden, berechnet ein Ansatz zunächst den Umkreis des Dreiecks und dessen Radius. Mittels der Relation von Kanten und den benachbarten Punkten, werden die adjazenten Dreiecke der Kanten geprüft. Punkte, die innerhalb des Umkreises liegen, werden zwischengespeichert. Anhand dieser Liste wird nach Verbesserung (einer Vergrößerung) der Innenwinkel des Ausgangsdreiecks gesucht. Wird ein Ergebnis gefunden, findet der Flip der Kante statt. Zuordnungen von Kanten, Punkten und den entsprechenden Relationen zueinander werden aktualisiert. Dieser Vorgang wird für die Dreiecke wiederholt, deren Kante einen Flip vollzogen hat, bis die Umkreisbedingung lokal wiederhergestellt ist. Dieser Ansatz beinhaltet so viele Fälle und deren Unterscheidungen, dass ein Unterschied bezüglich der Perfomanz deutlich wird. Stattdessen wird in dieser Ausarbeitung ein Kantenflip umgesetzt, den Lawonn und Günther beschreiben [LG18]: Sobald die gegenüberliegenden Winkel einer Kante summiert 180° überschreiten, wird der Flip der Kante durchgeführt und die Topologie angepasst. Dafür werden Methoden verwendet, die abhängig der Nachbarschaft agieren. Adjazente Kanten bearbeiteter Punkte und die beiden Dreiecke, die an jeder Kante anliegen, müssen bei jeder Methode identifiziert und entsprechend aktualisiert werden. Je nach verwendeter Topologie variiert der Umfang dieser Operationen.

Zusätzliche Relationen der Topologie beanspruchen weiteren Speicherplatz. Bei der oben genannten Zuordnung von Kanten und ihren vier anliegenden Punkten wird bspw. der Speicherplatz von allen Kanten der Triangulation $*$ 4 $*$ Größe eines Integers benötigt. Bei einer Speicherbelegung von 32 Bit pro Integer [28] und angenommenen 5000 Kanten, kann ein Speicherplatzbedarf von 80 KByte angenommen werden. Da ein Client-Host System verwendet wird, ist die Speicheroptimierung kein primäres Ziel der Anwendung.

Der globale Approximationsfehler wird ebenfalls durch die zweite Methode, das Verschieben der Vertices, reduziert. Bei dieser Methode werden pro Vertex eines Dreiecks vier neue Fehlerwerte berechnet. Diese Werte ergeben sich aus der Verschiebung des Vertex in alle vier Hauptrichtungen. Die Auswirkung der Verschiebung manifestiert sich in den Approximationsfehlern der adjazenten Dreiecke. Die benachbarten Dreiecke erfassen durch die Verschiebung unterschiedliche Bereiche des Bildes. Ergibt eine dieser Verschiebungen einen niedrigeren Fehler als die Ausgangsenergie, wird die Position des entsprechenden Vertex aktualisiert [LG18].
Mit $\{\mathbf{a}_T, \mathbf{b}_T, \mathbf{c}_T\} = T \cap \mathcal{V}$ als die drei Ecken eines adjazenten Dreiecks $T \in \mathcal{A}(\mathbf{v})$ ergibt sich der Energieterm folgendermaßen:

$$\frac{dE(T)}{d\mathbf{v}} = \begin{cases} \mathbf{v} = \mathbf{a}_T & : \frac{E(\mathbf{a}_T+\Delta,\mathbf{b}_T,\mathbf{c}_T)-E(\mathbf{a}_T-\Delta,\mathbf{b}_T,\mathbf{c}_T)}{2\Delta} \\ \mathbf{v} = \mathbf{b}_T & : \frac{E(\mathbf{a}_T,\mathbf{b}_T+\Delta,\mathbf{c}_T)-E(\mathbf{a}_T,\mathbf{b}_T-\Delta,\mathbf{c}_T)}{2\Delta} \\ \mathbf{v} = \mathbf{c}_T & : \frac{E(\mathbf{a}_T,\mathbf{b}_T,\mathbf{c}_T+\Delta)-E(\mathbf{a}_T,\mathbf{b}_T,\mathbf{c}_T-\Delta)}{2\Delta} \end{cases} \tag{22}$$

In dieser Gleichung wird die Energie $E(T)$ in der From $E(\mathbf{a}_T, \mathbf{b}_T, \mathbf{c}_T)$ ausgedrückt. Mit Δ wird die Pixelgröße beschrieben.
In dieser Umsetzung der Veröffentlichung wird der Umgang mit der Energie der Dreiecke und Vertices unterteilt. Ein weiterer Wert für Fehler wird auf Kosten von Speicher und Berechnungszeit hinzugefügt. Es werden Fehlerdomänen getrennt, indem der Approximationsfehler die fortlaufende Unterteilung der Dreiecke beeinflusst. Bei dem Optimierungsprozess der Vertexpositionen wird der zweite Energiewert eingesetzt. Das Potenzial eines Vertex errechnet sich weiterhin anhand der adjazenten Dreiecke, die Energie eines Dreiecks erschließt sich aus dem erfassten Bereich. Durch diese Aufteilung kann die Unterteilung eines Dreiecks und das Verschieben der Vertices getrennt behandelt werden.
Der Fehler eines Dreicks ergibt sich weiterhin aus dem Approximationsfehler. Dafür wird der vom Dreieck erfasste Bereich des Bildes genutzt, um einen farblichen Mittelwert zu bilden. Anhand des Mittelwerts wird bei einer zweiten Erfassug des Bereichs der Fehler als Quadratisches Mittel [CD14] (englisch: *root mean square error - RMSE*) berechnet:

$$rmse = \sqrt{\frac{1}{|N|}\sum_{i \in N}(x_i - y_i)^2} \tag{23}$$

In diesem Fall nimmt y_i der Gleichung 23 den Mittelwert des Dreiecks an. x_i steht für die Pixelwerte der erfassten Pixel N. Es wird pro Pixel die Abweichung berechnet, um den kompletten Approximationsfehler zu erhalten. Dabei werden die quadrierten Differenzen der Farbkanäle summiert und durch drei geteilt. Überschreitet der Fehler einen Schwellwert, wird eine Unterteilung des Dreiecks vorgenommen. Der Schwellwert kann frei gewählt werden und beeinflusst die Adaptionsfähigkeit der Triangulierung. Bei niedrigem Schwellwert ($\sim$ 0,08) ist eine deutlich feinere Struktur zu erwarten als bei einem Schwellwert von $\sim$ 0,13. Als künstlerische Freiheit, kann der Benutzer je nach Wunsch des Ergenisses den Schwellwert wählen.
Die Energie eines Vertex ergibt sich aus den Approximationsfehlern der adjazenten Dreiecke. Durch das oben beschriebene Verschieben der Vertices wird der globale Approximationsfehler minimiert.
Bei dem Optimierungsprozess tritt der Fall ein, dass Vertices so verschoben werden, dass die Umkreisbedingung der Delaunay-Triangulierung nicht eigehalten wird. Dieses Verhalten ist jedoch notwendig, weil somit Details und feine Strukturen im Bild durch die Dreiecke erfasst werden können [LG18]. Wie beschrieben, wirkt sich der Regularisierungsterm positiv auf die Wohlgeformtheit und negativ auf die Adaptionsfähigkeit der Dreiecke aus. Es kann beobachtet werden, dass ein zu starker Einfluss des Terms keine adäquate Adaption zulässt. Aus diesem Grund wird der Einfluss minimiert, mit der Folge, dass die Umkreisbedingung nicht eingehalten wird und flache

Dreiecke mit kleiner Fläche entstehen. Lawonn und Günther etablieren ein Kollabieren der kürzesten Kante, *edge collapse*, eines Dreiecks, sobald Punkte zu nah beieinander liegen [LG18].

Zu Beginn des Verfahrens besteht die Triangulierung aus zwei Dreiecken, sodass keine Struktur vorgegeben ist und die Adaption nicht eingeschränkt wird. Die Triangulierung endet, sobald ein globaler Approximationsfehler unterschritten wird. Daraufhin wird das Bild gespeichert.

Lawonn und Günther stellen eine Lösung vor, die Farb- und Fehlerberechnung eines Dreiecks mittels der OpenGL-Rasterisierung realisiert [LG18]. Mit der Beschränkung auf die CPU wird ein Verfahren nötig, das nur Pixel innerhalb eines Dreiecks berücksichtigt und Pixel, die außerhalb liegen, erkennt und verwirft.
Ein Ansatz dafür testet die Punkte, indem das Skalarprodukt gebildet wird. Dabei wird jeweils jede Kante eines Dreiecks als Vektor interpretiert. Der zweite Vektor wird mit einem der Eckpunkte des Dreiecks und dem zu testenden Punkt erstellt. Sobald ein Test mit negativem Skalarprodukt auftritt, wird der Punkt verworfen. Ein eingrenzendes Rechteck, eine *Boundingbox*, wird zur initialen Begrenzung der zu testenden Punkte verwendet. Dieser Ansatz weist je nach Orientierung und Größe des Dreiecks viele Punkte auf, die verworfen werden, wodurch Rechenaufwand entsteht, der nicht zum Ergebnis beiträgt.
Eine Optimierung hinsichtlich dieses Rechenaufwandes bietet der folgende Ansatz:
Jedes Dreieck wird jeweils als zwei Vektoren interpretiert, die das Dreieck aufspannen. Diese Vektoren werden mit der Schrittweite eines Pixels traversiert, wodurch pro Vektor pro Schritt eine neue Koordinate entsteht. Die beiden neuen Koordinaten bilden pro Schritt einen neuen Vektor, der durch die Fläche des aufgespannten Dreiecks verläuft. Mit der gleichen Schrittweite wird auch dieser Vektor traversiert. Die dadurch entstehenden Koordinaten ermöglichen eine Abtastung des gesamten Dreiecks, ohne Tests, ob sich die Koordinaten innerhalb des Dreiecks befinden.
Diese Methode der Abtastung der Dreiecke wird verwendet, um erst die mittlere Farbe eines Dreiecks und daraufhin die Energie des gleichen Dreiecks zu ermitteln.

4.5 Optimierungsansätze

Ein wichtiger Aspekt dieser Arbeit ist die Entwicklung von Verbesserungsmöglichkeiten des Triangulierungverfahrens und deren Evaluation. Die folgenden Ansätze verfolgen das Ziel der Perfomanzsteigerung. Für diese Umsetzung, in einem Host-Client System, ist die Echtzeitfähigkeit kein primäres Ziel. Jedoch ist eine Performanzsteigerung für den Server bedeutend. Bei kürzeren Bearbeitungszeiten der Bilder können mehrere Bilder in der gleichen Zeitspanne bearbeitet werden, sodass eine höhere Zahl an Nutzern das System verwenden kann und das Ergebnis für den Einzelnen schneller erreicht wird.

Die Optimierungsansätze werden in die Methode der Vertexaktualisierung eingebunden. Das Ziel dabei ist es, die Auswirkungen der Ansätze hinsichtlich Geschwindigkeit und visuellem Ergebnis zu untersuchen. Somit soll eine Verbesserung des Gradientenverfahrens erzielt werden. Die Vertices werden pro Iterationsschritt entsprechend ihrer Approximationsfehler bewegt, sodass Bildinhalte in der Triangulieung präziser erfasst werden können.
Der erste Ansatz etabliert dazu eine zusätzliche Speicherstruktur, die mittels des Index pro Vertex addressiert wird. In diesem Speicher wird der zuletzt gewählte Schritt eines Punktes gesichert. So soll vermieden werden, dass ein Punkt, der beispielsweise im vorherigen Iterationsschritt nach links versetzt wurde, im aktuellen Schritt wieder nach rechts gesetzt wird. Durch die vorherige Prüfung, ob ein Schritt zulässig ist, soll Berechnungszeit gespart werden. Zur Diskussion steht, ob in der direkten Umgebung eines Punktes Änderungen geschehen, die eine Verschiebung des Punktes in die vorhergegangene Richtung befürworten. Solche Änderungen können der Flip einer adjazenten Kante oder die Aufteilung eines benachbarten Dreiecks sein.

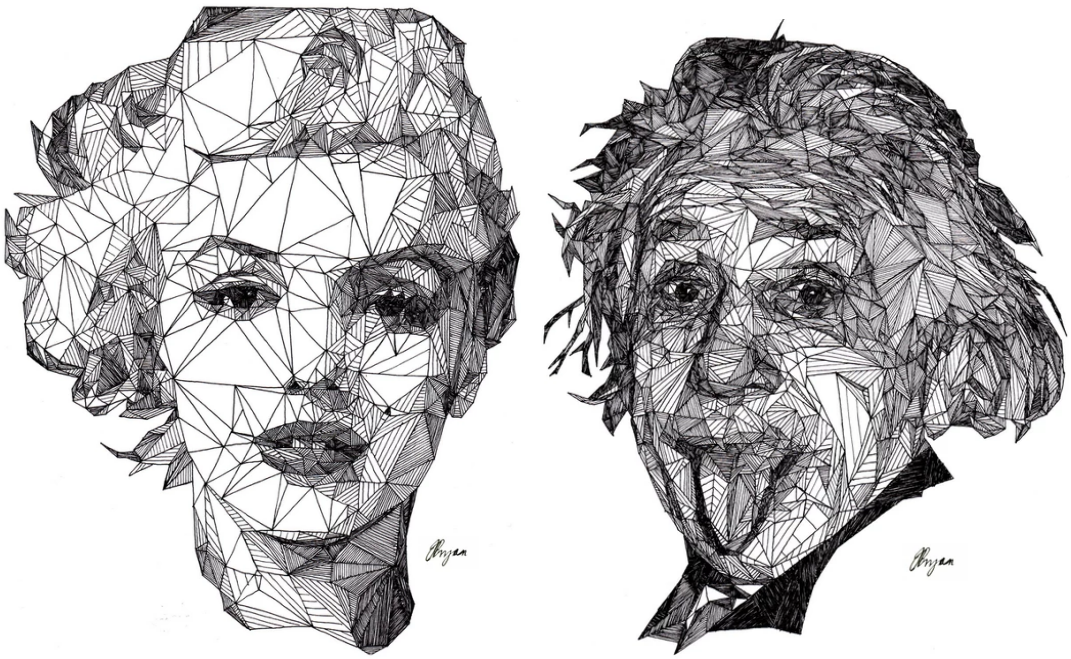
Der nächste Ansatz untersucht die Schrittrichtung eines Punktes. Es wird getestet, ob diagonale Schritte eine schnellere Berechnung des Ergebnisses unterstützen, als die Schritte in die vier Hauptrichtungen des Punktes. Anstelle der Optionen (links, rechts, oben und unten) werden diagonale Schritte getestet (links oben, rechts oben, links unten und rechts unten). Untersucht wird, ob durch dieses Verfahren bei der Berechnung Iterationsschritte gespart werden. Ebenso kann der Fall eintreten, dass eine ungenauere Erfassung der Details im Bild eintritt und somit ein optisch minderwertigeres Ergebnis entsteht.

Der letzte Ansatz, der untersucht wird, stellt einen Bewegungsvektor für das Gradientenverfahren vor. Für diesen Optimierungsversuch werden pro Vertex die Energieterme pro Schrittrichtung berechnet. Anhand der Intensität der Werte werden die Bewegungsrichtungen gewichtet. Statt der Auswahl der Richtung mit dem geringsten Approximationsfehler, wird ein Richtungsvektor erstellt, der sich flexibler auf die Verschiebung des Vertex auswirkt. Somit sind beliebige Verschiebungen innerhalb eines Radius möglich. Der Radius wird beschränkt, um keine zu großen Schritte zuzulassen, die Überschneidungen der Dreiecke verursachen können. Gleichzeitig wird durch die Gewichtung der Richtungen die Richtung mit dem größten Approximationsfehler ausgeschlossen. Die verbleibenden drei Richtungen werden gewichtet summiert und als Verschiebungsvektor auf den aktuellen Punkt angewendet. Ob bei dieser Methode qualitätssteigernde Verschiebungen erreicht werden, ist zu untersuchen.

4.6 Stilisierung

Lawonn und Günther stellen in der Veröffentlichung die Annäherung eines nicht-photorealistischen Renderstils an den Stil eines Künstlers vor [LG18]. Josh Bryan erzielt mit analoger Triangulierung visuell interssante Abstraktionen, dargestellt in Abbildung

24[1]. Motiv dieser Bilder sind berühmte Personen, die als Porträt dargestellt werden.



(a) Marylin Monroe **(b)** Albert Einstein

Abbildung 24: Werke des Künstlers Josh Bryan

Der per Hand enstandene Stil weist mehrere Merkmale auf, die von der computergenerierten Abstraktion angenähert werden sollen. Die Triangulierung orientiert sich an den Kanten der Bildelemente, ohne, dass sich Dreiecke überschneiden. Ingesamt erscheinen die Triangulierungen als wohlgeformt. Deutlich ist hingegen auch, dass die Erfassung der markanten Details des Porträts ein übergeordnetes Ziel einnehmen.
Details werden präzise durch feinere Strukturierung erfasst. Ausgehend von einem Gittermodell mit weißen Dreiecken, entstehen für den Betrachter Grautöne durch Linien innerhalb der Dreiecke. Die Linien verlaufen äquidistant und prallel zu einer beliebigen Kante des Dreiecks. Durch die Dichte der Linien erscheinen Dreiecke variabel in der Helligkeit. Ein lebhafter Stil der Triangulierung wird durch Varianz in der Liniengestaltung erreicht, ob absichtlich oder nicht. Manche Linien verlaufen nicht komplett parallel oder gradlinig [LG18].
Helle Regionen werden durch Belassen der weißen Farbe in den Dreiecke dargestellt, was durch die Schraffur der anderen dunkleren Dreiecke eine Tiefe im Bild entstehen lässt. Homogene Regionen, die keine Details aufweisen, werden durch größere Dreiecke abgebildet. In Abbildung 24a wird deutlich, dass durch diese adaptive Triangulierung eine Hevorhebung der Besonderheiten im Bild erreicht wird. Das linke Auge wird durch die umliegenden größeren weißen Dreiecke betont.

Lawonn und Günther verwenden *Tonal Art Maps* um den Stil von Josh Bryan zu imitieren. Dabei wird der ursprünglichen Farbe eines Dreiecks entsprechend der Helligkeit eine Textur hinzugefügt [LG18], dargestellt in Abbildung 12.

Ein untergeordnetes Ziel dieser Anwendung ist die Entwicklung eines weiteren Ansatzes dieses nicht-photorealistischen Renderverfahrens. Aufbauend auf das Gittermodell mit weißen Dreiecken als Eingabe, soll zur weiteren Verarbeitung eine Pseudozufälligkeit bei der Zeichnung der Linien erreicht werden. Sobald eine pseudo-zufällige Zahl einen Schwellwert unterschreitet, werden einzelne Linie innerhalb eines Dreiecks an dem Startpunkt leicht versetzt, sodass die Linien im Dreieck nicht mehr parallel verlaufen. Der weitere Aspekt, der auf die Imperfektion der Natürlichkeit abzielt, ist die leichte Überschneidung der begrenzenden Linien des Dreiecks. Dieser Aspekt wird ebenfalls bei Eintreten des Zufallereignisses implementiert.
Die Helligkeit der Dreiecke wird aus dem **L**-Anteil des Lab-Farbmodells errechnet. Entsprechend der Helligkeit wird die Distanz der Linien innerhalb des Dreiecks bestimmt. Um die Helligkeit der Farbe eines Dreiecks bestimmen zu können, erfolgt eine Konvertierung der RGB-Werte anfangs in den XYZ und anschließend in den Labfarbraum.
Die Kante, an der sich die Schraffur orientiert, wird zufällig gewählt. Bei Dreiecken mit hoher Helligkeit, soll die Schraffur bewusst ausgelassen werden. Dafür wird ein Schwellwert bei der Berechnung der Schraffur eingeführt.

Eine zusätzliche Stilisierung einer Triangulation wird im Folgenden vorgestellt. Für diese Stilisierung wird ein beliebiger globaler Bezugspunkt in dem Bild gewählt. Zu diesem Punkt nimmt jeder Punkt eines Dreiecks Bezug; je nach Distanz zwischen Vertex und Bezugspunkt, wird der Punkt verschoben. Durch die Verschiebung entstehen variable Lücken in der Triangulierung, wodurch Dreiecke isoliert im Raum stehen.
Um die Punkte verschieben zu können, ohne, dass mehrfach verwendete Punkte an die gleiche Stelle verschoben werden, wird für jedes Dreieck der Schwerpunkt berechnet. Die Distanz zwischen dem beliebigen Bezugspunkt und dem Schwerpunkt wird zur Bestimmung der Richtung benötigt, in die die Punkte des Dreiecks verschoben werden. Der Abstand zum Ausgangspunkt bestimmt, wie weit die Punkte in die Richtung verschoben werden. Befindet sich das Dreieck nah am Ausgangspunkt, wird eine kleine Verschiebung beobachtet, während Punkte am Bildrand weit verschoben werden. Dadurch entsteht ein optischer Effekt, der mit Scherben verglichen werden kann, die wieder zusammengelegt wurden.

4.7 Server

Der Server ist in dieser Anwendung der Kommunikationspartner der App. Da er mit einer mobilen Anwendung in Dialog tritt, entsteht die Anforderung der Erreichbarkeit. Ein Webserver ist durch *HTTP-Requests* und *-responses* mittels des Internets erreichbar, sodass der Server und somit die stilisierte Triangulierung des Bildes für alle Benutzer zur Verfügung stehen, die Zugang zu einem Endgerät haben, das über einen Internet-

zugang verfügt.
Parameter, die vom Nutzer zur Beeinflussung der Triangulierung verwendet werden können, können mittels der verwendeten Protokolle übertragen werden. Sie werden in der URL der Anfrage kodiert und vom Server verarbeitet.

Der Server soll zwei Ressourcen unterstützen: das Hoch- und das Herunterladen eines Bildes.
Die Ressource, die das Hochladen des Bildes realisiert, erwartet eine *Post*-Methode vom Client. In der empfangenen Anfrage, *Request*, soll das Bild enthalten sein. Um ein Bild in einer Anfrage versenden zu können, muss es kodiert sein. In diesem Fall erreicht den Server ein JSON-Dokument, das von der Anwendung erstellt und versendet wird.
Die Ressource, die das Bild empfangen soll, nimmt den Inhalt einer Variable des *Bodys* entgegen. Mit dieser Variable wird das kodierte Bild im JSON-Dokument addressiert. Sobald die Variable nicht vorhanden oder ausgelesen werden kann, schlägt der Vorgang fehl, das Bild kann nicht empfangen werden und der Server antwortet mit einer Fehlermeldung in der *Response*. In dem Fall wird der Fehlercode "400 Bad Request" gewählt. Er steht für eine nicht konforme Anfrage. Dieser Fall tritt ein, sobald die gesendeten Daten nicht konform übertragen werden. Andere Fehlermeldungen sind in diesem Fall möglich, da lediglich der Miss-/ Erfolg der Anfrage unterschieden wird. Der Nutzer hat keine Möglichkeit das Verhalten vom Client oder die gewählte Ressource anzupassen.
Nachdem die Daten des Bildes erfolgreich übertragen wurden, müssen die Daten dekodiert werden, um als Eingabe der Triangulierung genutzt werden zu können. Auf die En- und Dekodierung wird in dieser Arbeit nicht näher eingegangen, da sie nicht Haupbestandteil dieser Arbeit sind. Bei erfolgreicher Übertragung wird dem Client per Statuscode 200 und dem Nutzer mit einer kurzen Nachricht bestätigt, dass die Verarbeitung des Bildes beginnt. Für den Nutzer ist die Rückmeldung über die grafische Oberfläche wichtig. Bei fehlender Antwort kann der Anwender den weiteren Verlauf nur vermuten. Für die mobile Anwendung ist der Statuscode bedeutend, da somit ein problemfreier Programmablauf versichert wird und keine Fehler behandelt werden müssen. Falls dem Nutzer eine Fehlermeldung angezeigt wird, kann er entsprechend darauf reagieren und beispielsweise das Bild erneut senden, falls der Server nicht erreichbar war.

Das asynchrone Bearbeiten von Anfrage und Antwort ist bei der Umsetzung des Serveraufbaus ein wichtiger Aspekt. Nachdem mittels der Anfrage eine Verbindung zu dem Server aufgebaut ist, bleiben dem Client standardmäßig wenige Sekunden (circa fünf), um eine vollständige Anfrage zu übertragen. Die Übertragung einer Datei kann je nach Datenrate diese Zeitspanne überschreiten. In dem Fall wird bei einem synchronen Aufbau dem Client eine Fehlernachricht übermittelt und das Bild kann nicht vollständig versendet werden. Bei einem asynchronen Aufbau des Serverprogramms, wartet der Server nicht auf eine vollständige Übertragung der Daten, sondern sendet eine Antwort, sobald die Datenübertragung erfolgreich beginnt. Durch dieses Verhalten wird

eine kurzfristige Rückmeldung für den Benutzer und eine parallele Arbeitsweise des Servers gewährleistet.
Eine Inkrementierung der begrenzenden Zeitspanne birgt den Nachteil, dass keine zeitnahe Rückmeldung der Übertragung möglich ist. Dadurch wird der Nutzer verleitet, das Bild mehrmals zu versenden, was unvorhergesehenes Verhalten auf Server- sowie auf Clientseite verursacht. Ein weiterer Verbindungsversuch kann vom Server abgelehnt werden, da eine Verbindung bereits besteht. Für den Nutzer entsteht dadurch eine weitere Fehlermeldung.
Eine andere Alternative stellt die persistente Verbindung zwischen Client und Server dar. Für diese Art der Verbindung «einigen» sich beide Kommunikationspartner auf die Verbindungsart. Mehrere Anfragen und Antworten können dadurch über die selbe Verbindung versendet werden; die Zeit und der Aufwand des Verbindungsaufbaus werden gespart, da sie nur einmal benötigt werden [Gre11]. Nach einer gewissen Zeit ohne Interaktion wird diese Verbindungsart jedoch ebenfalls beendet. Für diese Umsetzung der Anwendung ist die persistente Verbindung keine Alternative, da die stilisierte Triangulierung je nach Größe des Bildes unterschiedlich viel Zeit benötigt. Eine Überschreitung des Zeitlimits der Verbindung hat zur Folge, dass das Ergebnisbild nicht übertragen werden kann und dem Benutzer eine Fehlermeldung übermittelt wird.
Durch diese Anforderungen und Beschränkung entsteht der Bederf einer zweite Ressource des Servers. Sie verwaltet das Herunterladen des Ergebnisbildes.

Für das Herunterladen des Bildes wird die *Get*-Methode in der Anfrage verwendet. Zusätzlich zur Angabe, dass ein Bild heruntergeladen werden soll, muss die Datei exakt adressiert werden. Mittels eines regulären Ausdrucks wird die Eingabe vom Client auf Konformität geprüft. Dieses Verfahren dient einem Sicherheitsaspekt:
Cross-Site Scripting ist eine Methode bei der Skripte in HTML-Elementen clientseitig eingebunden werden und auf dem Server ausgeführt werden sollen [Sin08]. Als Ziel haben diese Angriffe, den Zugriff zu sensiblen Daten zu erhalten. Eine mögliche Sicherheitsmaßnahme ist die Validierung und sichere Verarbeitung von Anwendereingaben. In diesem Fall der Umsetzung ist ein Test mittels eines regulären Ausdrucks hinreichend. Reguläre Ausdrücke definieren und validieren Zeichenketten [Fri09]. Dabei kann spezifiziert werden, welche Zeichen an welcher Stelle der Eingabe erwartet werden. Beispielweise kann das explizite Vorkommen ganzer Wörter oder die Beschränkung auf drei Zahlen als Eingabe gefordert werden.

Bei dem automatisierten Herunterladen des Ergebnisses ist es ein wichtiger Aspekt, ob das Ergebnisbild zum Versenden vorliegt. Die mobile Anwendung baut in dem Fall alle 15 Minuten Kontakt zum Server auf, sobald ein Bild vom Nutzer erwartet wird. Liegt das Ergebnis noch nicht vor, erhält der Client eine Fehlermeldung mit dem Status 404. Sobald das Ergebnisbild zur Verfügung steht, wird es dem *Body* der Antwort angehängt und versendet. Der Benutzer erhält das Bild erfolgreich und bearbeitet zurück. Falls das manuelle Herunterladen benutzt wird, erhält der Nutzer eine simple

Meldung, dass das Ergebnis noch nicht bereit liegt

Für die prototypische Entwicklung dieser Anwendung wird der Sicherheitsaspekt des unauthorisierten Zugriffs nicht detaillierter behandelt. Angreifer könnten bei einem *Brute-Force*-Angriff den benötigten Bildnamen erraten und ein Ergebnisbild herunterladen. Maßnahmen, die als weiterführende Aufgaben die Schwachstelle beheben, sind Authentifizierungsmechanismen gegenüber dem Server.

4.8 Zusammenfassung

Dieses Kapitel stellt das Konzept vor, das der Implementation zugrunde liegt. Alle grundlegenden Methoden und Aspekte, die die Anwendung charakterisieren, werden erläutert. Dabei wird die Problemstellung und Lösung der Triangulierung hergeleitet, das Konzept und die Anwendung der Stilisierung präsentiert und die Funktionen und Anforderungen von App und Server dargelegt. Die Darlegung der Optimierungsansätze, deren eventuelle Vorteile und Risiken sind Teil des Kapitels.
Die Erarbeitung des Konzepts vermittelt die Ansätze und Möglichkeiten aller Aspekte der Anwendung. Die technische Umsetzung, die im folgenden Kapitel erläutert wird, kann dadurch nachvollzogen und bewertet werden. Ebenso ist es möglich, die Ergebnisse hinsichtlich Qualität und Perspektive einzuordnen.

5 Implementation

In diesem Abschnitt wird die Umsetzung des Konzepts erläutert. Mit dem Ziel, die praktische Umsetzung darzulegen, werden die generelle Vorgehensweise und markante Passagen aus dem Code erläutert. Es werden nicht alle Teilaspekte der Anwendung in diesem Kapitel aufgeführt, da ihre Umsetzung durch die konzeptuelle Darlegung möglich ist.
In Abbildung 25 wird der abstrahierte Ablauf vom Eingabebild bis zum Ergebnis dargestellt. Die Zusammenarbeit der in vorherigen Kapiteln vorgestellten Komponenten wird somit visualisiert.

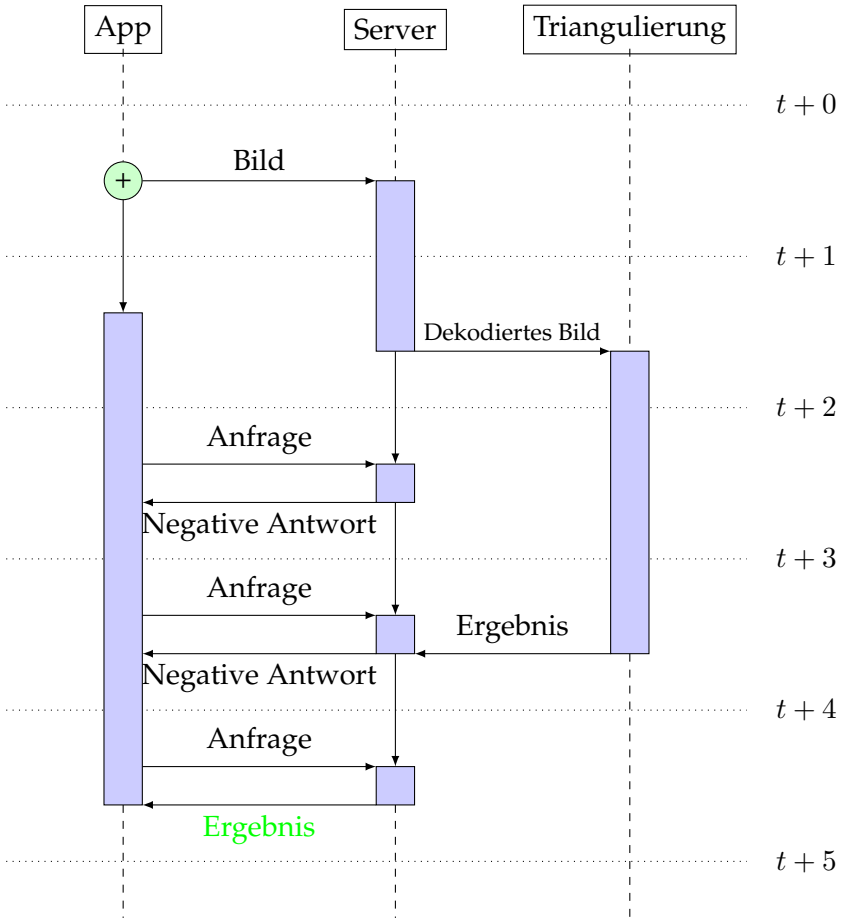


Abbildung 25: Visualisierung des Ablaufs der Anwendung

5.1 Werkzeuge des Entwicklers

Im Folgenden werden die Programme vorgestellt, die in dieser Umsetzung benutzt wurden.
OpenGL mit **GLSL** stellt eine Programmierschnittstelle (API) bereit, die eine Kommunikation mit der Grafikkarte realisiert. Dadurch wird die Entwicklung von Programmen ermöglicht, die die Grafikkarte zur Berechnung und zum Rendern der grafischen Inhalte einsetzt [44].
Das Framework **CVK2** [8] der Universität Koblenz-Landau stellt eine grundsätzliche Funktionalität bereit, die zur Entwicklung von unterschiedlichen Grafikanwendungen benutzt werden kann. Das Framework basiert auf OpenGL und GLSL und richtet eine Programmstruktur ein. Die grundlegende Struktur der Anwendung wurde von diesem Framework übernommen. Ebenso stammen **Cmake** und die Einstellungen, um die Anwendung plattformübergreifend kompilieren zu können, aus dem CVK2.
Grundlegende Bestandteile des Rendervorgangs, wie die Erstellung von *Shaderprogrammen*, die der Kamera in der Szene und die Erstellung der Textur, stammen aus einer anderen Quelle [25].
Um ein Bild mit beliebigem Format importieren [4] und, nach Abschluss der Berechnung, speichern zu können [3], sind zwei Bibliotheken eingebunden.

Facebook stellt mit **React Native** ein freizugängliches Framework bereit, das für die Entwicklung von mobilen Apps konzipiert ist [48]. Für die Entwicklung von Android-Apps wird die Entwicklungsumgebung **Android Studio** [31] mit dem **Android SDK** verwendet. Ebenso wird das **Java Development Kit** [38] benötigt, um Android basierte Anwendungen zu implementieren.

Ein **Webserver** mit kompaktem Funktionsumfang wird für die Kommunikation zwischen App und Triangulationsalgorithmus verwendet [2]. Um das Ergebnisbild über den Webserver versenden zu können, wird eine Kodierung in **Base64** eingesetzt, wofür eine bestehende (De-)Kodierung verwendet wird [10].

Mit **inotify** wird ein Subsytem eines Linux-Kernels eingesetzt, das Änderungen im Dateisystem bemerkt [35].

5.2 Triangulierung

In dieser Sektion wird auf Einzel- und Besonderheiten eingegangen, die wichtige Aspekte der Implementation darstellen. Dabei wird nicht der gesamte Ablauf der Triangulierung erläutert, da die grundlegenden Funktionen im Konzept dargelegt werden.
Die Ausgangssituation der Topologie umfasst zwei Dreiecke, die das gesamte Eingabebild bedecken. Es werden vier Eckpunkte definiert, die von Verschiebungen ausgeschlossen sind. Die Kanten, die die Bildränder darstellen werden bei Operationen, die Kanten betreffen, ausgeschlossen, wodurch nur eine Kante der Topologie zur Bearbeitung anfangs entsteht. Entsprechend werden die Relationen der Topologie mit Werten

vor Beginn der Berechnung befüllt. Als initiale Situation können mehrere Triangulierungen in Betracht gezogen werden. Mit der gewählten Variante ist eine Adaption an die Bildinhalte ermöglicht, da, wie im konzeptuellen Teil beschrieben, keine Struktur vorgegeben wird.
Bis ein beliebig wählbarer globaler Approximationsfehler unterschritten ist, wird eine Adaption durch Verschieben der Punkte untersagt. Stattdessen werden Dreiecke fortlaufend unterteilt, wenn die jeweilige Energie einen weiteren Schwellwert überschreitet. Sobald das Verschieben der Vertices erlaubt wird, wird dieser Schwellwert angepasst, wodurch eine Erfassung von Details des Bildes erfolgt. Die Werte der Parameter werden anhand des Ergebnisses angepasst, sodass mehrere Versuche nötig sind, bis ein adäquates Resultat erzielt werden kann.
Die inkrementelle Adpation der Triangulierung erfordert eine Verwaltung der topoligischen Elemente. Diese Operationen basieren auf der Identifikation und des Austausches von Punkten, was je nach Topologie variiert.
Aktualisierungen dieser Art werden beim Unterteilen eines Dreiecks benötigt, da unter Umständen die Umkreisbedingung nicht eingehalten wird. Bei dem Flip einer Kante müssen die Elemente der Topologie ebenfalls aktualisiert werden.

Eine Schwierigkeit bei der Implementation auf der CPU, ist die Konvertierung zwischen einer Gleitkommazahl und einer Ganzzahl. Bei der Berechnung der mittleren Farbe eines Dreiecks, wird das Dreieck im Abstand von Pixelgröße abgetastet. Die Bildinformationen, die Farbwerte des Bildes, sind als flache Liste gespeichert, welche durch Ganzzahlen addressiert werden können. Folglich muss eine Kompatibilität zweier Domänen hergestellt werden. Texturkoordinaten liegen im geschlossenen Intervall von [0; 1]. Die Koordinaten des Bildes werden durch die Höhe und Breite in Pixelanzahl angegeben und nehmen bspw. Werte von 1000 an.
Die Koordinaten eines Dreiecks befinden sich in Texturkoordinaten; bei einem Bild mit einer angenommenen Weite von 940 und einer Höhe von 680 Pixeln, misst ein horizontaler Schritt eine Weite von $\frac{1}{940} = 0.00106$. Um die Farbwerte der Textur mit einer Koordinate des Dreiecks ermitteln zu können, muss eine Multiplikation mit den Maßen der Textur erfolgen und auf eine Ganzzahl beschränkt werden. Durch diese Konvertierung tritt der Fall ein, dass unterschiedliche Texturkoordinaten auf die gleiche Ganzzahl abgebildet werden.
In der Farb- und Fehlerberechnung wird aufgrund dessen ein zwei-dimensionales *Array* eingeführt, das eine Maske darstellt, die verwaltet, ob die Werte des Bildes bereits ausgelesen wurden und somit zur Berechnung beitragen. Dadurch wird ein mehrmaliges Auslesen verhindert und die Farbtreue gesichert.
Nach Summierung aller Farbwerte eines Dreiecks, wird der Farbwert durch die Anzahl der verwendeten Pixel pro Dreieck geteilt.
Die Fehlerberechnung verläuft analog zur Farbberechnung. Mit der berechneten mittleren Farbe des Dreiecks kann pro Pixel die Abweichung ermittelt werden. In diesem Fall verhindert die Maske die mehrfache Inklusion der Werte.

Komplikationen treten bei dem Gradientenschritt der Optimierung auf. Vertices sollen in die vier Hauptrichtungen verschoben werden, um die zugehörige Energie zu minimieren. Bei vorgeschlagener maximaler Schrittweite von 0.2 Pixeln tritt der Abbildungsfehler der Domänen auf, wodurch Vertices nicht verschoben werden, da keine Verschiebung von Koordinaten in der Bilddomäne erreicht wird. Aus diesem Grund wird die Einschränkung auf eine maximale Schrittweite von 0.2 Pixeln aufgehoben. Die benutzte Schrittweite ergibt sich aus dem Wert der gemittelten Texturmaße. Dieser Wert wird auf die Hälfte beschrankt, wodurch eine Schrittweite gegenüber der ursprünglichen Methode deutlich erhöht wird.

```
1 float mean_tex_width = (tex_width + tex_height) /2.0f;
2 float pixel_step = 1.0f / mean_tex_width;
3 float stepsize = pixel_step * 0.5f;
```

Die Erhöhung hat zur Folge, dass kleine Schritte in der Texturdomäne nicht auf gleiche Werte in der Bilddomäne abgebildet werden. Gleichzeitig ist die Schrittweite nicht zu groß, dass Vertices ihre konvexe Hülle verlassen.

Alle vorkommenden Dreiecke und deren Bearbeitungsreihenfolge werden in einer Liste mit Indizes der einzelnen Vertices festgelegt. Sobald ein Dreieck in drei kleinere unterteilt wird, werden die neuen Dreiecke der Liste angehängt, während ein Index, der das Fortschreiten in der Liste verwaltet um drei inkrementiert wird. Dadurch wird das ursprüngliche Dreieck nicht wiederholt bearbeitet. Sobald nur die Optimierungsmethode bei einem Dreieck angewendet wird, werden die drei Vertices der Liste wieder angehangen.
Eine komplexere Verwaltung ist nach einem Kantenflip nötig. Es müssen die beiden veralteten Dreiecke in der Bearbeitungsliste gefunden und durch andere Indizes der Liste ersetzt werden. Die beiden Dreiecke, die sich aus dem Kantenflip ergeben haben, werden der Liste angefügt.

Um ein Ergebnisbild zu erhalten, muss ein Abbruchkriterium eingeführt werden. Dazu wird der globale Approximationsfehler benutzt. Bei der Fehlerberechnung werden jedoch nicht alle Dreiecke berücksichtigt. Dreiecke, die auf Grund ihrer Größe keiner echten Farbberechnung unterliegen, sind bei der Berechnung nicht inkludiert. Nach Unterschreitung eines Schwellwerts wird der Algorithmus beendet und das aktuelle Bild gespeichert.
Die Wahl des Schwellwerts beeinflusst die Auflösung der Dreiecksstruktur. Der Schwellwert, der die Unterteilung der Dreiecke beeinflusst, ist größer als derjenige, der das Abbruchkriterium des Verfahrens darstellt. Das hat zur Folge, dass Vertices weiterhin verschoben werden können, auch wenn die Unterteilung der Dreiecke nicht erlaubt wird.
Das Ergebnisbild wird mit den Befehlen *glReadBuffer(GL_FRONT)* und *glReadPixels* ausgelesen und mit der oben genannten Bibliothek [3] gespeichert.

5.3 Rendering der Stile

Für die Farbgebung, sowie für die Stilisierung der Dreiecke, werden verschiedene Methoden in einem *Geometryshader* implementiert. Beginnend mit der simpleren, konstanten Farbgebung der Dreiecke, wird im Anschluss auf die technische Umsetzung des neuen Ansatzes der Schraffur eingegangen. Abschließend wird die Stilisierung des «Scherben»-Stils erläutert.
Für die Visualisierungen wird ein *Geometryshader* verwendet, der die Manipulation der Geometrie ermöglicht. Die Koordinaten deier Punkte gelangen von dem *Vertexshader* zum *Geometryshader*, sodass die Bearbeitung eines Vertex ohne Umstände möglich ist. Um Dreiecke in dem *Shader* identifizieren zu können, wird in dem *Vertexshader* eine *Primitive ID* pro Vertex erstellt und an den *Geometryshader* übergeben. Im *Geometryshader* werden Dreiecke als Eingabe spezifiziert, wodurch drei *Primitive IDs* zur Verfügung stehen. Anhand der IDs und der Relationen der Topologie, die als *Shader Storage Buffer* verfügbar sind, kann das eingeschlossene Dreieck und die zugehörige Farbe ermittelt und an den *Fragmentshader* weitergegeben werden. Im *Fragmentshader* werden dadurch alle Pixel, die innerhalb des jeweiligen Dreiecks liegen, mit der zugeordneten Farbe beschrieben.

Um den Stil des Künstlers Josh Bryan zu imitieren, wird im *Geometryshader* das Layout angepasst. Statt Dreiecken werden Linien als Ausgabe erwartet. Da bei dieser Stilisierung nicht bekannt ist, wie viele Linien pro Dreieck benötigt werden, um eine Schattierung zu erreichen, wird die Variable $max\_vertices$ im Layout auf eine Zahl gesetzt, die nicht erreicht wird. Für 50 Linien pro Dreieck genügt es, die Einstellung $max\_vertices = 106$ zu verwenden.
Im nächsten Schritt dieser Stilisierung wird das jeweilige Ausgangsdreieck durch drei Linien dargestellt ausgegeben. Daraufhin findet die eigentliche Stilgebung statt:
Die ursprüngliche Farbe des Dreiecks wird mittels der Topologie ermittelt. Diese Farbe wird in den Lab-Farbraum konvertiert. Da lediglich die Helligkeit benutzt wird, kann Rechenaufwand vermieden und die Konvertierung vereinfacht werden. Als Zwischenschritt ist eine Umrechnung in den XZY-Farbraum nötig, dafür werden die Rot-, Grün- und Blaukanäle jeweils zunächst linearisiert. Anschließend werden die drei linearisierten Farbwerte verwendet, um den Y-Wert des Übergang-Farbraums zu erhalten. Die X- und Z-Werte werden für die Berechnung der Helligkeit nicht benötigt, weshalb statt einer Matrixmuliplikation eine komponentenweise Vektormultiplikation durchgeführt und Rechenaufwand gespart werden kann.
Anhand des Y-Werts wird abschließend die Helligkeit, der L-Wert aus dem Lab-Farbmodell, berechnet. Dieser wird durch eine Teilung des Maximalwerts auf das geschlossene Intervall $[0; 1]$ normiert. Der Vorteil der Normierung wird im folgenden Abschnitt deutlich.

Zusammen mit der Größe eines Dreiecks wird der Helligkeitswert verwendet, um die Anzahl an Linien innerhalb eines Dreiecks zu berechnen. Dabei soll eine höhere Dichte an Linien benutzt werden, wenn die Fläche dunkler erscheinen soll. Die Anzahl

an Linien ergibt sich aus einer initialen Anzahl an Linien, die je nach Helligkeitswert und Größe des Dreiecks skaliert wird. Mit bekannter Anzahl an Linien werden zwei Seiten eines Dreiecks gleichmäßig traversiert, um an den Seiten Start- und Endpunkte der zusätzlichen Linien zu bestimmen.
Ein Aspekt, der nicht durch *Tonal Art Maps* erfasst werden kann, ist die Natürlichkeit, die handgefertigte Werke aufweisen. In diesem Ansatz wird untersucht, ob mit dem *Geometryshader* ein gewisser Grad dieser Natürlichkeit nachgeahmt werden kann.
Bei Werken von Josh Bryan sind feine asymetrische Linienzüge erkennbar, wodurch Linien innerhalb eines Dreiecks nicht vollkommen parallel zueinander verlaufen. Selten sind die Seiten eines Dreiecks zu weit gezeichnet, sodass eine fehlerhafte Triangulierung entsteht. Eine Funktion, die Pseudozufallszahlen generiert [33] wird in diesem Ansatz benutzt, um den Eindruck von Natürlichkeit zu erreichen. Bei Eintreten eines Zufallereignisses wird der Startpunkt einer Linie innerhalb eines Dreiecks um ein Pixel verschoben oder die begrenzenden Linien eines Dreiecks werden etwas weiter gezeichnet als nötig, um das Dreieck zu bregrenzen.

Um eine angenäherte Freistellung der Motive zu erhalten, werden Dreiecke, die von den Bildecken ausgehen, nicht gezeichnet. So wird das im Mittelpunkt liegende Motiv zusätzlich inszeniert.

Die Realiserung des «Scherben»-Effekts findet ebenfalls im *Geometryshader* statt, da die Manipulation von Geometrie benötigt wird. Der Bezugspunkt, der als Ausgangspunkt des Effekts betrachtet werden kann, stellt eine variable Eingabe dar. Dieser Punkt sollte in der Nähe des Hauptmotivs sein, damit in dessen Umgebung die Detaildichte am höchsten ist. Dass der Bezugspunkt beliebig gewählt werden kann, ermöglicht eine variable Hervorhebung von Bildregionen.

Für alle Stilisierungen ist ledigleich ein Renderdurchlauf nötig. Für die Gestaltung mit konstanter Farbe und schwarz gezeichneten Kanten der Dreiecke, ist ein weiteres *Shaderprogramm* nötig, das das Gittermodell zeichnet.

5.4 App

Die Implementation eines Host-Client-Systems bedingt die Gestaltung der App Mesh. Editieren des Ergebnisses oder eingreifen in den Bearbeitungsprozess sind für den Benutzer nicht möglich. Der Nutzer sendet ein Bild an den Server und erhält das bearbeitete Ergebnis zurück. Um diesen Ablauf zu realisieren, müssen mehrere Module in die mobile Anwendung integriert werden:
Die Einbindung von **Ionicon** [45] bietet die Auswahl von mehreren Hundert Icons. Für Android und für iOS stehen unterschiedliche Varianten der Icons zur Verfügung. Dass in der Anwendung plattformübergreifend das jeweils richtige Icon für die Darstellung gewählt wird, entscheiden Fallunterscheidungen im Code der Anwendung.
Mit dem **Imagepicker**-Modul [21] wird ein Dialog zwischen Nutzer und grafischer

Oberfläche realisiert, der dem Nutzer die Wahl zwischen Bildern aus dem Speicher und der Kameraanwendung des Geräts ermöglicht. In beiden Fällen kann ein Bild ausgewählt werden, das bearbeitet werden soll. Nach der Auswahl durch das Modul, wird das Bild in der Anwendung präsentiert. Um auf Bilder, die auf dem mobilen Gerät gespeichert sind, zugreifen zu dürfen, muss der Nutzer den Zugriff auf den Speicher gestatten.
Push Notifications [24] sind kurze Mitteilungen, die über Geschehnisse oder Zustandsänderungen der Anwendung informieren. Die Mitteilungen werden dem Nutzer als Popup-Nachricht zugestellt und in der Statusleiste des Geräts angezeigt. Der Nutzer kann entscheiden, ob eine Anwendung solche Mitteilungen zustellen darf. Der Benutzer erhält solch eine Nachricht bei erfolgreichem Erhalt des Ergebnisbildes.
Die **Fetch** API ermöglicht die Kommunikation mittels des Internets. Der Zugriff auf Ressourcen von Servern wird via eines HTTP *Requests*, der frei konfigurierbar ist, umgesetzt [46]. In dieser Anwendung wird die API verwendet, um ein ausgewähltes Bild in JSON einzubinden, es an den Server zu senden und dem Nutzer das Ergebnis zur Verfügung zu stellen. Für die Übertragung an den Server, wird das Bild in base64 kodiert im *Body* der Anfrage übertragen. Die Versendung eines kodierten Bildes hat den Vorteil, dass der Empfänger, der Server, von einer bestimmten Repräsentation der Daten ausgehen kann. Die Weiterverarbeitung wird dadurch gewährleistet. Der Nachteil, der sich daraus ergibt, ist eine erhöhte Datenmenge, die durch die Kodierung entsteht. Im folgenden Abschnitt wird dargelegt, wie die *Fetch* API verwendet wird, um das bearbeitete Bild vom Server auf das mobile Gerät zu übertragen.
Mit **Headless JS** kann eine **Task** erstellt und für die Anwendung registriert werden, sodass diese Funktion bei jedem Zustand der Anwendung ausgeführt werden kann. Eine *Task* ist eine JavaScript Methode, die bei Auftreten eines vom Entwickler bestimmten Events ausgeführt wird [17]. Unter der Bedingung, dass keine grafischen Komponenten durch die *Task* benutzt und verändert werden, kann sie auch ausgeführt werden, wenn die App im Hintergrund arbeitet und nicht aktiv vom Benutzer verwendet wird. In dieser Anwendung wird eine *Task* dazu verwendet, um das Ergebnisbild vom Server zurückzuerhalten. Unter der Voraussetzung, dass ein Ergebnisbild erwartet wird, erfolgt in regelmäßigen Abständen ein *GET-Request* per *Fetch* API an den Server geschickt, der das Bild anfordert. Bei negativer Antwort findet eine Wiederholung der Anfrage zu einem späteren Zeitpunkt statt. Sobald das Ergebnisbild zur Verfügung steht, erfolgt somit das automatische Herunterladen.
Bei der Implementation von *Tasks*, die im Hintergrund agieren - *Backgroundtasks* - findet eine unterschiedliche Behandlung je nach Betriebssystem statt. Android unterstützt *headless JS*, für iOS müssen weitere APIs installiert werden, um Arbeiten bei inaktiven Apps zu verrichten. iOS beschränkt zusätzlich die Zeit, die eine Aufgabe hat, um bearbeitet zu werden. Wird die Zeitspanne überschritten, bevor die App die Bearbeitung abgeschlossen hat, beendet das Betriebssystem die Anwendung, sodass sie neu gestartet werden muss. Eine weitere Einschränkung von iOS ist die Frequenz, in der eine Aufgabe im Hintergrund gestartet werden kann. 15 Minuten Pause werden von iOS als kleinstes Intervall festgelegt, in dem Aufgaben im Hintergrund bearbeitet werden

können [7].
Da nur für Android eine direkte Lösung ohne zusätzliche APIs bereitgestellt ist, entsteht an dieser Stelle eine plattformabhängige Programmierung. Die Erweiterung auch bei iOS im Hintergrund Aufgaben auszuführen, überschreitet die prototypische Ausarbeitung dieser Anwendung.
Alternativ zu einer *Task*, die im Hintergrund arbeitet und Anfragen an den Server stellt, ist eine regelmäßige Erinnerung möglich, die dem Nutzer angezeigt wird. Dieser hat die Möglichkeit, manuell den Server über die App mittels der *Fetch* API zu kontaktieren. In der protoypischen Umsetzung wird das manuelle Herunterladen des Ergebnisses unterstützt. Die Übertragung der Rendereinstellungen ist ebenso exemplarisch umgestzt worden. Dieser Ansatz bestätigt, dass eine individuelle Gestaltung des Ergebnisses mittels eines Host-Client Systems möglich ist.
Die Interaktionsmöglichkeiten von Mesh sind in Abbildung 26 als Aktivitätsdiagramm dargestellt.

Ein untergeordnetes Ziel der Anwendung ist es, die grafische Oberfläche so zu gestalten, dass Wiedererkennungswert für den Benutzer generiert wird. Ein simpler, sauberer Stil findet dafür Verwendung, wie er von bekannten Firmen eingesetzt wird. Zusätzlich wird eine Farbe gewählt, die häufig im Layout der Anwendung und im Icon im Menü des Geräts verwendet werden soll. Das Icon wird im gleichen Stil, wie die Anwendung gestaltet, um eine visuelle Kohärenz zu erreichen. Der prototypische Entwurf weist Tendenzen dieser Planung auf, vgl. Abbildung 27.

5.5 Verbindung der Komponenten

Die Trennung von Kommunikation und Verarbeitung ist ein entscheidender Aspekt der Softwarearchitektur. Der Server ist gelöst von der stilisierten Triangulation. Er verwaltet das Empfangen und Senden der Bilddateien. Die Verarbeitung des Eingangsbildes wird durch ein anderes Programm umgesetzt. Dieses Programm ist rechenintensiv, was an der benötigten Rechenzeit deutlich wird. Die Zeiten sind der Evaluation zu entnehmen. Wenn ein Webserver ausgelastet ist, kann es vorkommen, dass Anfragen nicht oder nur mit Verzögerung bearbeitet weden können. Aufgrund dessen wird die Bearbeitung nicht vom Webserver ausgeführt.
Die Kommunikation zwischen Webserver und Triangulationsalgorithmus ermöglicht *inotify*. Dieses Subsystem von Linux registriert Dateiänderungen innerhalb eines Ordners.
Ein simples Skript initiiert die Überwachung eines Ordners. Sobald der Webserver ein Bild empfängt, startet das Skript den Triangulationsalgorithmus. Nach Abschluss der Berechnung, findet die Speicherung des Bildes in einem gesonderten Ordner statt, in dem der Webserver regelmäßig, auf Anfrage der App, nach dem Ergebnis “sucht”. Liegt das Ergebnisbild vor, wird es über den Webserver an die App gesendet.

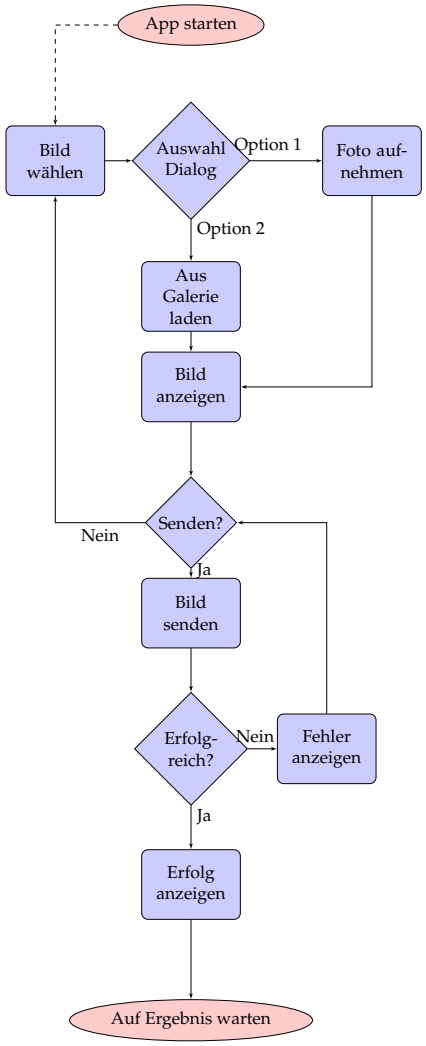


Abbildung 26: Aktivitätsdiagramm der App Mesh

5.6 Zusammenfassung

Dieses Kapitel hebt Details der technischen Umsetzung des Verfahrens hervor. Es umfasst Herausforderungen und Lösungsansätze, sowie detaillierte Beschreibungen wichtiger Aspekte aus technischer Sicht. Anhand der Informationen können folgende Ergebnisse eingeordnet und eine Perspektive der Anwendung erstellt werden.
Nachdem verwendete Bibliotheken und Programmierschnittstellen vorgestellt werden, wird auf wichtige Methoden der Triangulierung explizit eingegangen. Domänenspezifische Problemlösungen stehen dabei im Vordergrund der Erläuterung. Anschließend wird prägnant auf Details der Renderstile eingegangen. Durch eine ausführliche Darlegung im konzeptuellen Teil, ist in diesem Kapitel keine tiefergehende Erklärung nötig. Der letzte Unterpunkt thematisiert die Funktionsweise und die Zusammensetzung der

(a) Icon der Anwendung **(b)** Grafisches Beispiel der Anwendung

Abbildung 27: Layout der App Mesh

Programmierschnittstellen in der App Mesh.
Im folgenden Kapitel werden die Ergebnisse dieser Umsetzung präsentiert und ausgewertet, sodass ein Fazit der Arbeit erstellt werden kann.

6 Evaluierung

In diesem Kapitel werden die Ergebnisse präsentiert und ausgewertet, die aus dem umgesetzten Verfahren resultieren. Die Unterschiede, die bei der Portierung des Verfahrens von der GPU auf die CPU entstehen, werden hervorgehoben. Anschließend folgen die Evaluation der Stilimitation und die Präsentation des «Scherbenstils». Als letzter Unterpunkt gilt es, die Ergebnisse der Optimierungsansätze auszuwerten. Durch die Beobachtungen und Schlussfolgerungen können ein Fazit und ein Ausblick mit Perspektive des Verfahrens herausgearbeitet werden.

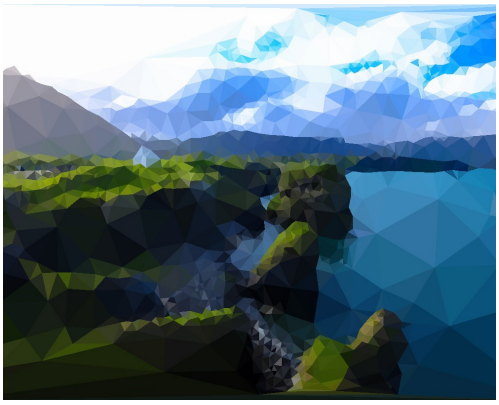
6.1 Host-Client System

Die Verbindung von mobiler Anwendung und dem Webserver ermöglicht einen simplen Zugang zu dem Triangulationsverfahren. Diese Umsetzung verlagert erfolgreich die rechenlastige Triangulierung auf den Server, wodurch die Ressourcen des mobilen Geräts nicht verbraucht werden und eine kontinuierliche Verwendung des Endgeräts gesichert ist.
Als Nachteil dieses Systems gilt die fehlende Rückmeldung über die Auswirkungen der Stilisierungseinstellungen. Diese Rückmeldung erfolgt erst nach Erhalt des Ergebnisses. Daraus resultiert eine mehrfache Berechnung, bis das angestrebte Ergebnis erreicht ist. Die Funktionalität von Mesh umfasst die wesentlichen Methoden. Die Entscheidung zwischen einem Bild aus dem Speicher des Geräts und der Aufnahme eines neuen Fotos ermöglicht dem Nutzer die freie Wahl des Ausgangbildes. Das Herunterladen des Ergebnisses erfolgt ebenso nach Belieben des Nutzers, wodurch dem Benutzer einerseits die freie Wahl ermöglicht wird, andererseits jedoch keine automatisierte und somit eine den Nutzer entlastende Lösung präsentiert wird.
Die Zusammenarbeit von Webserver und *inotify* ergibt einen problemfreien Ablauf bei der Bearbeitung des Ergebnisses. Zwischen Eingang des Bildes und der Verabeitung entsteht keine zeitliche Verzögerung.

6.2 Ergebnisse der Triangulierung

In diesem Abschnitt werden die Ergebnisse der CPU-basierten Variante, die im Rahmen dieser Arbeit umgesetzt wurde, präsentiert und evaluiert. Sie werden mit dem ursprünglichem Verfahren verglichen [LG18]. Die Ergebnisse sind mit konstanter Farbgebung entstanden. Das Referenzbild wurde aus der Veröffentlichung [LG18] entnommen; es misst 450 Pixel in der Breite und 299 Pixel in der Höhe und wird ebenfalls für die Evaluierung der Optimierungsansätze verwendet.
Anhand der Ergebnisbilder sind lediglich tendenzielle Unterschiede der Verfahren zu erfassen, da nicht alle Einstellungen der GPU-basierten Methode bekannt sind.

(a) Eingabebild



(b) Ergebnis dieses Verfahrens mit vektorbasiertem Optimierungsansatz

(c) Ergebnis von Lawonn und Günther

Abbildung 28: Vergleich der Ergebnisse der unterschiedlichen Ansätze

6.2.1 Beobachtungen

Für jedes Bild ist eine individuelle Einstellung der Parameter nötig, um qualitativ ähnliche Ergebnisse zu erhalten. Diese Eigenschaft resultiert aus unterschiedlichen Maßen der Bilder, da Dreiecke unterschiedlich viele Details erfassen. Dass kleinere Dreiecke eine Limitierung der Methode sind, wird im Verlauf des Kapitels deutlich. Der Inhalt der Bilder ist ebenso beeinflussend. Bilder, die große homogene Regionen aufweisen, jedoch wenige, aber wichtige Details beinhalten, unterscheiden sich von Bildern, die eine eher durchgängige Dichte an Details aufweisen.
Bei größeren Bildern sind erhöhte Berechnungszeiten zu beobachten, da mehr Pixel in den adjazenten Dreiecken eines Punktes zu finden sind. Somit entsteht ein direkter Zusammenhang zwischen Bildgröße und Berechnungszeit.

Der Unterschied der Güte der Ergebnisse zwischen dem Ansatz von Lawonn und Günther, und dem Ansatz dieser Arbeit, wird in Abbildung 28 deutlich. Beeinflussende Parameter stellen Schwellwerte der Fehlertoleranz dar. Dazu zählt der Schwellwert, der die Unterteilung einzelner Dreiecke beeinflusst und derjenige, der das Abbruchkriterium darstellt.
Beide Verfahren verzeichnen eine adaptive Strukturierung der Triangulation. Homogene Regionen werden durch eine konstante Farbgebung mit weichen Übergängen und im Vergleich größeren Dreiecken erfasst. Details im Bild werden hingegen durch mehrere kleinere Dreiecke wiedergegeben, die sich ebenfalls durch den Optimierungsprozess an Übergängen von Bildinhalten anpassen. Dadurch wird ingesamt eine adäquate Wiedergabe der Bildinhalte erreicht.
Deutlich ist jedoch auch die unterschiedliche Anzahl der verwendeten Dreiecke. In der GPU-basierten Anwendung von Lawonn und Günther werden erkennbar weniger Dreiecke verwendet, sodass ein ästhetischeres und visuell ruhigeres Ergebnis erzielt wird. Die Umsetzung auf der CPU hingegen benötigt so viel Zeit pro Iterationsschritt, dass nicht ohne zeitkritischen Mehraufwand eine vergleichbare Lösung erreichbar wäre. Besonders an harten Übergängen der Bildinhalte sind mehrere kleinere Dreicke zu verzeichnen. Dadurch entstehen unstetige Übergänge und eine Minderung der Ergebnisqualität. Insgesamt entsteht ein visuell kohärentes Bild. Farben bleiben, wie bei der GPU-Lösung, bis auf einen geringen Intensitätsverlust, erhalten.

6.2.2 Auswertung des CPU-basierten Ansatzes

Methoden, die die Dreiecksstruktur optimieren, bewirken einen deutlichen Qualitätsunterschied. Ein Kollabieren der Kante von kleinen Dreiecken steigert wahrscheinlich merkbar die Qualität dieses Verfahrens. Übergänge, an denen viele Dreiecke zu finden sind, würden durch weniger Dreiecke vereinfacht, sodass eine gleichmäßigere Strukturierung erzielt werden kann, vergleichbar mit der GPU-Variante.
Die Limitierung dieses Ansatzes ist die Berechnungszeit des Optimierungsschritts. Anhand Abbildung 29 wird ein exemplarisches Ergebnis präsenstiert, das ohne Verschiebung der Punkte entstanden ist. Innerhalb von 143 Sekunden ist die Erstellung möglich, doch nur durch die Minimierung der Energie der Punkte und der Dreiecke ist eine exakte Darstellung von harten Kanten im Bild möglich. Die Qualität dieser Ergebnisse manifestiert sich in der benötigten Zeit pro Iterationsschritt.
Eine weitere Limitierung der CPU-basierten Variante äußert sich bei der vektorbasierten Traversierung der Dreiecke. Durch die Konvertierung zwischen der Bild- und der Texturdomäne entstehen Ungenauigkeiten: Die Typänderung von Gleitkomma- zur Ganzzahl bewirkt einen Präzisionsverlust der Koordianten. Die Farbe und die Energie relativ kleiner Dreiecke können nicht genau berechnet werden, da durch Rechenungenauigkeiten keine Pixel als innerhalb eines Dreiecks erkannt werden, vgl. Abbildung 30.

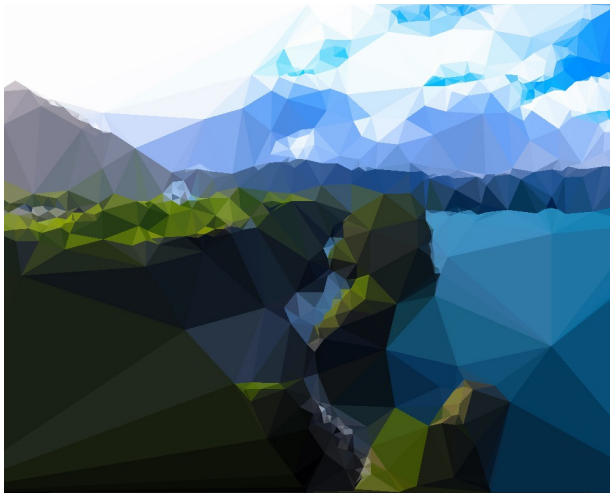


Abbildung 29: Ergebnis ohne Optimierungsverfahren

Eine mehrstündige Berechnungszeit ist für ein Verfahren, das Ergebnisse für eine mobile Anwendung erstellt, nicht akzeptabel. Aufgrund dessen wird ein Kompromiss hinsichtlich der Qualität eingegangen und die Anzahl der Schritte pro Punkt pro Iteration verringert. Dieses Verhalten äußert sich in unruhigen Übergängen an Kanten, verursacht durch kleinere Dreiecke in der Region. Würde das Minimum, die Kante, schneller gefunden werden, müssten keine zusätzlichen Dreiecke dort gebildet werden. Je öfter die Position eines Punktes mit dem Gradientenverfahren aktualisiert wird, desto weniger Dreiecke werden benötigt, um den Bildinhalt zu erfassen.
Eine weitere Limitierung der CPU-Variante ist die Schrittweite der Optimierungsmethode. Bei zu gering gewählter Verschiebung der Koordinaten in der Texturdomäne, resultiert es in der Bilddomäne in keiner Verschiebung, sodass kein Optimierungsschritt möglich ist. Dieser Umstand verlangt eine erhöhte Schrittweite, die wiederum bei Verschiebungen der Punkte bewirkt, dass Überschneidungen der Dreiecke häufiger auftreten können. Um Überschneidungen zu vermeiden, wird der Einfluss der Regularisierung erhöht.
Die Umsetzung auf der Grafikkarte mit der Einbeziehung der Rasterisierung resultiert in ästhetischeren Bildern, die ruiger wirken, keine Unstetigkeiten an Kanten aufweisen und ingesamt weniger Dreiecke für die Stilisierung benötigen. Diese Qualität wird durch die geringe Berechnungszeit der Iterationsschritte ermöglicht, wodurch eine höhere Anzahl an Schritten in kürzerer Zeit durchgeführt werden kann.
Trotz der qualitativ hochwertigeren Lösung, ist die Einbettung der GPU-basierten Variante für mobile Geräte fragwürdig, da selbst bei mehreren Minuten Berechnungszeit das Gerät nicht für den Nutzer anderweitig zur Verfügung steht. Da die Berechnung dieses Verfahrens auf einem Host ausgeführt wird, ist hier ein Vorteil der CPU-basierten Variante zu sehen. Zusätzlich werden hinreichend ästhetische Ergebnisse erzielt, die konkurrenzfähig mit anderen mobilen Anwendungen sind.
Zwei weitere Beispiele befinden sich am Ende des Kapitels, vgl. Abbildung 37 [43] [LG18].

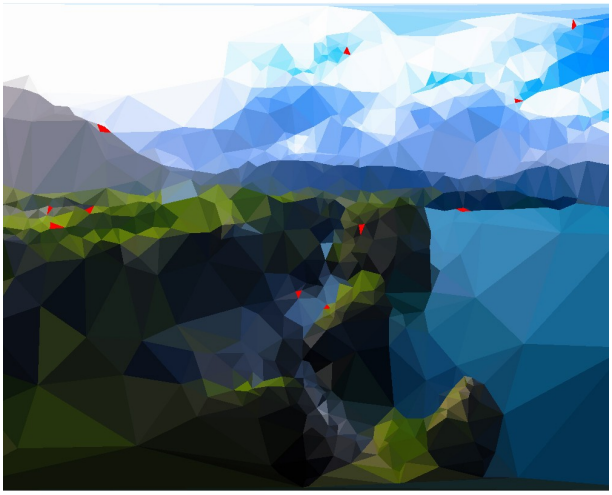


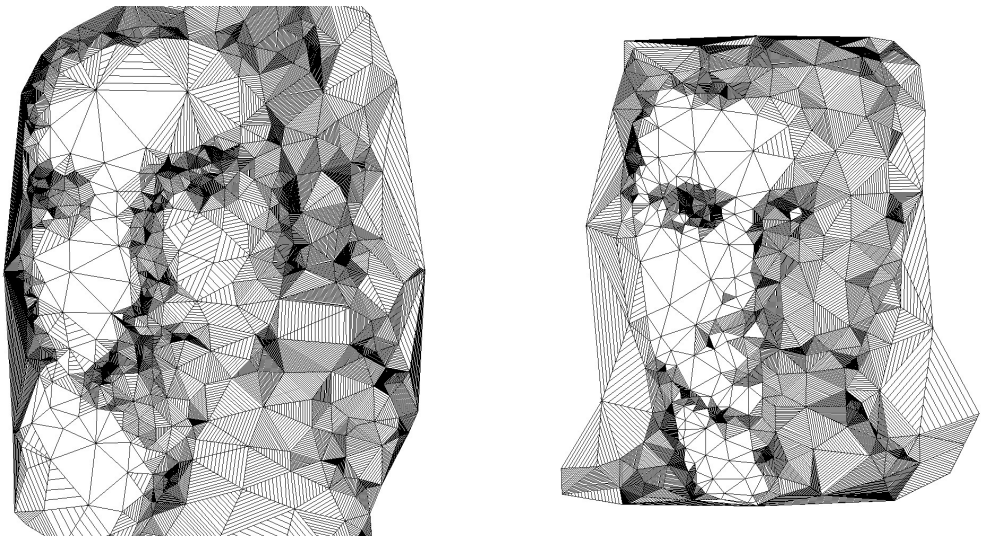
Abbildung 30: Visualisierung von Dreiecken mit problematischer Größe

6.3 Ergebnisse der schraffierten Triangulierung

Ein Forschungsaspekt dieser Arbeit umfasst die Ausarbeitung und Bewertung einer alternativen Methode einer Stilisierung. Ziel ist es dabei, einen handgefertigten Stil zu imitieren. Dafür soll Natürlichkeit nachempfunden werden, die bei handangefertigten Werken üblich ist. Diese Natürlichkeit wird in dieser Arbeit versucht mit dem Einsatz eines *Geometryshaders* zu erreichen. Ergebnisse sind in Abbildung 36 dargestellt. Der Stil ist von Werken von Josh Bryan adaptiert [1].
Die Effekte und deren Auswirkungen auf die Stilisierung sind beliebig veränderbar. Die Häufigkeit und die Intensität der «natürlichen Effekte» resultiert folglich in visuellen Unterschieden. Eine vergrößerte Aufnahme eines Ergebnis ist in Abbildung 32 dargestellt. Die Schrägstellung der Linie ist deutlich erkennbar. Die verlängerten Kanten eines Dreiecks sind weniger deutlich. Der Effekt kann jedoch nach Belieben intensiviert werden.
Die groben Strukturen der Gesichter werden durch die Triangulierung gefunden. Details der Motive, die die Gesichter charakterisieren werden hingegen nicht so präzise erfasst, wie bei den Werken von Bryan.

Ein Unterschied zu den Werken von Bryan ist die Freistellung der Motive. Die Nahaufnahmen der Personen werden zusätzlich hervorgehoben, da keine Strukturen im Hintergrund gezeichnet sind. Ein Ansatz der Freistellung ist in Abbildung 31 dargestellt. Dieser Porträtmodus strebt das Herausstellen der Gesichter an. Bei dieser Methode werden Dreiecke, die von den vier Bildecken ausgehen, nicht gezeichnet, um eine Freistellung des Hauptmotivs zu erlangen. Voraussetzung ist die Platzierung des Mo-

tivs im Mittelpunkt des Bildes. Durch die Freistellung von Bryan werden die Konturen der Nahaufnahmen präziser erfasst und das Motiv steht konkurrenzlos im Mittelpunkt.



(a) Ergebnis des Porträtmodus **(b)** Ergebnis des Porträtmodus

Abbildung 31: Ergebnisse des Portätmodus mit Stilisierung nach Josh Bryan

6.3.1 Auswertung

Anhand der Beispiele ist zu erkennen, dass eine Imitierung des Stils von Josh Bryan prinzipiell möglich ist. Jedoch mit Einschränkungen. Gleichzeitig ist ein Unterschied zur Methode der *Tonal Art Maps* erkennbar. Die Verwendung der pseudozufälligen Ereignisse lässt ein gewisses Maß an Natürlichkeit zu, vgl. Abbildung 32. Die Versetzung einzelner Linien im Dreieck sind erkennbar und produzieren ein natürlicheres Aussehen. Die verlängerten Kanten der Dreiecke treten hingegen nicht hervor. Aliasing bewirkt, dass viele Linien Stufen aufweisen, wodurch Punkte, an denen sich Kanten treffen, undeutlich gezeichnet werden und der Effekt nicht zu erkennen ist.
Weitere Einschränkungen betreffen die Qualität der Imitation. Details und charakteristische Merkmale der Motive werden von Bryan deutlich präziser herausgearbeitet. Dieses Qualitätsmerkmal wird durch die Triangulation nicht erreicht, wie im vorherigen Kapitel beschrieben wird.

6.4 Ergebnisse des «Scherbeneffekts»

Der in Abbildung 33 präsentierte Renderstil wurde bei der Recherche dieser Arbeit nicht in dieser Umsetzung gefunden. Durch den Einsatz des *Geometryshaders* wird eine nicht zusammenhängende Triangulation erstellt. Außerdem entsteht ein Rahmen um

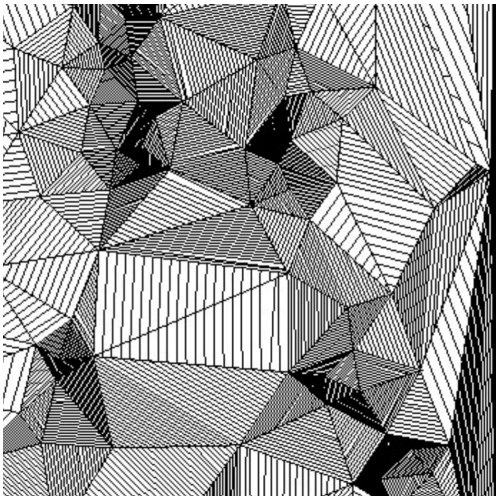
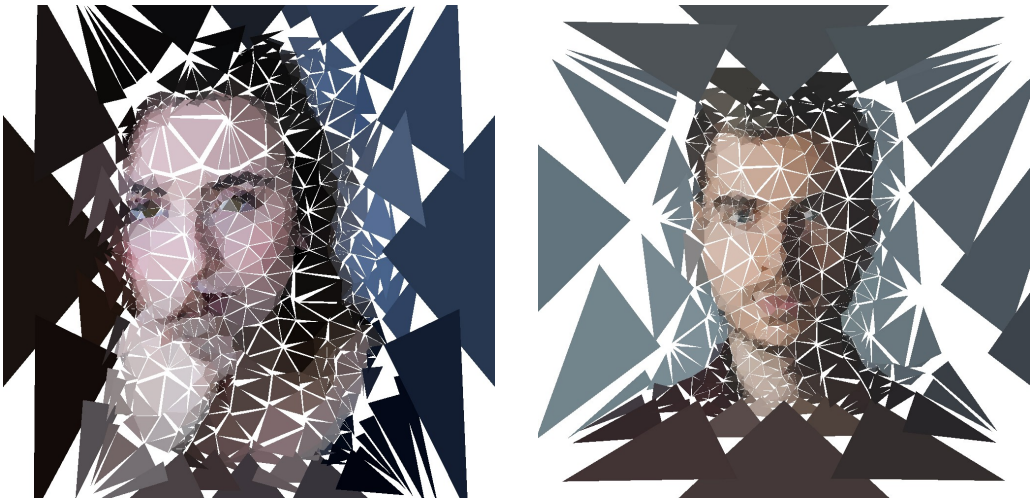


Abbildung 32: Vergrößerter Ausschnitt der Schraffur

das Motiv, wodurch eine Hervorhebung des Bildinhaltes erzielt wird.
Die Besonderheit des Renderstils ist die Variabilität des Ausgangspunktes. Diese Ei-



(a) Ergebnis 1 **(b)** Ergebnis 2

Abbildung 33: Ergebnisse des Scherbeneffekts

genschaft versetzt den Nutzer in die Lage, eine beliebige Region des Bildes hervorzuheben.

6.5 Auswertung der Optimierungsansätze

Um die unterschiedlichen Ansätze miteinander vergleichen zu können, gelten für die Ansätze die gleichen Bedingungen. Anhand eines berechneten Referenzbildes ohne Verwendung eines Optimierungsansatzes, entsteht ein optischer Vergleichswert, vgl. Abbildung 35a. Gleichzeitig werden die verwendeten Parameter übernommen. Die Parameter beschreiben, ab welchem Fehlerwert das Gradientenverfahren eingesetzt, ab welchem Wert der Algorithmus beendet wird und mehrere Parameter, die das Verhalten der Verschiebung der Punkte beeinflussen. Der Wert des Abbruchkriteriums ergibt sich aus den summierten Approximationsfehlern der Dreiecke, geteilt durch die Anzahl der Dreiecke im Bild. Mit diesen Einstellungen wurden die Ansätze jeweils fünf Mal durchgeführt und die Ergebnisse gemittelt, sodass eine repräsentative Darstellung der Ergebnisse folgt, vgl. Tabelle 1. Der Approximationsfehler des gesamten Bildes wird ebenfalls angegeben. Als Vergleichswert ist die Lösung von Lawonn und Günther [LG18] in der Tabelle aufgenommen. Um die Ergebnisse grundsätzlich miteinander vergleichen zu können, werden die Bilder mittels des Bildbearbeitungsprogramms Gimp [26] auf eine einheitliche Größe skaliert. Anschließend wird der Fehler mit dem Programm Imagemagick [23] für die Bilder berechnet. Als Fehlermetrik wird die Wurzel der mittleren Fehlerquadratsumme (*rmse*) verwendet.

Eine marginale Anpassung der Parameter ist bei dem diagonalen Schrittverfahren und dem Einschränken der Schritte vorgenommen worden. Dadurch, dass der Algorithmus bei jeder bestimmten Anzahl an Dreiecken im Bild, den globalen Approximationsfehler prüft, wird das Verfahren bei den genannten Methoden früher beendet als bei dem Verfahren ohne Optimierungsansatz und bei dem dritten zu testenden Ansatz. Der Schwellwert, der das Abbruchkriterium darstellt, wurde um eine Nachkommastelle verringert, sodass vergleichbare Ergebnisse entstehen. Dass visuell vergleichbare Ergebnisse entstanden sind, verdeutlicht Abbildung 35.
Der Verlauf der Verfahren ist grafisch in Abbildung 34 dargestellt. Auf der x-Achse

Ansatz	Iterationen	Fehler	Dauer in s
Lösung L&G	n.b.	0.0624	n.b.
Ohne Opt.	3714	0.0846	2435.32
Bewegung merken	3172	0.09	1619.69
Diagonale Schritte	3204	0.0849	2144.14
Bewegungsvektor	3904	0.0943	3057.98

Tabelle 1: Messdaten der Optimierungsansätze

befindet sich die Anzahl der Iterationen. Auf der y-Achse die Zeit in Sekunden, die pro Iterationsschritt benötigt wird. Zu Anfang aller Methoden ist eine niedrigere Zeit pro Iterationsschritt zu beobachten. Das wird durch die Einschränkung erwirkt, die ein Bewegen der Punkte untersagt. Sobald die Energieminimierung ausgeführt wird, steigt die Berechnungszeit pro Schritt deutlich. Je nach Einfluss eines Punkts auf sei-

ne Umgebung, entsprechend der Größe der adjazenten Dreiecke, wird mehr Zeit für die Berechnung der aktualisierten Position benötigt. Dadurch, dass im fortlaufenden Algorithmus eine Unterteilung der Dreiecke in kleinere stattfindet, erfolgt ein sinkender Einfluss der Punkte auf ihre Umgebung. Entsprechend müssen weniger Pixel pro Iterationsschritt in die Berechnung einbezogen werden und die Zeit pro Schritt sinkt tendenziell.

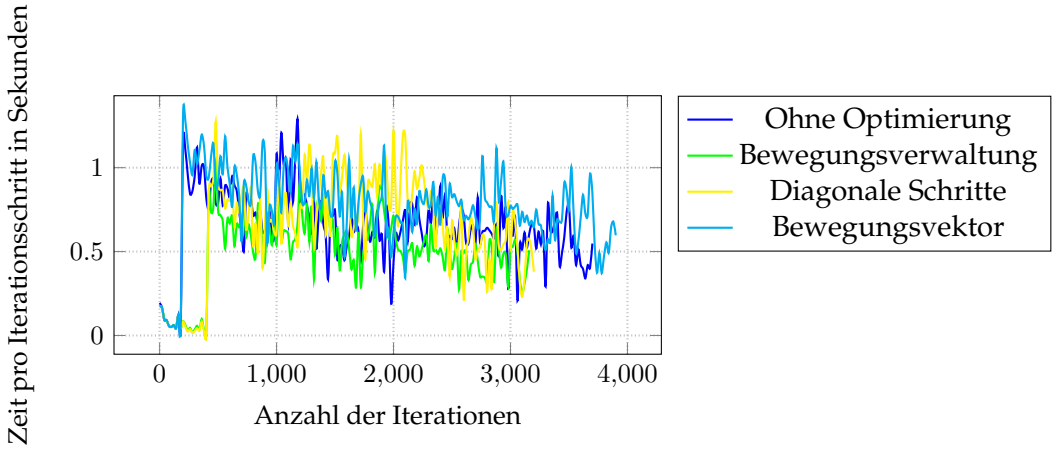
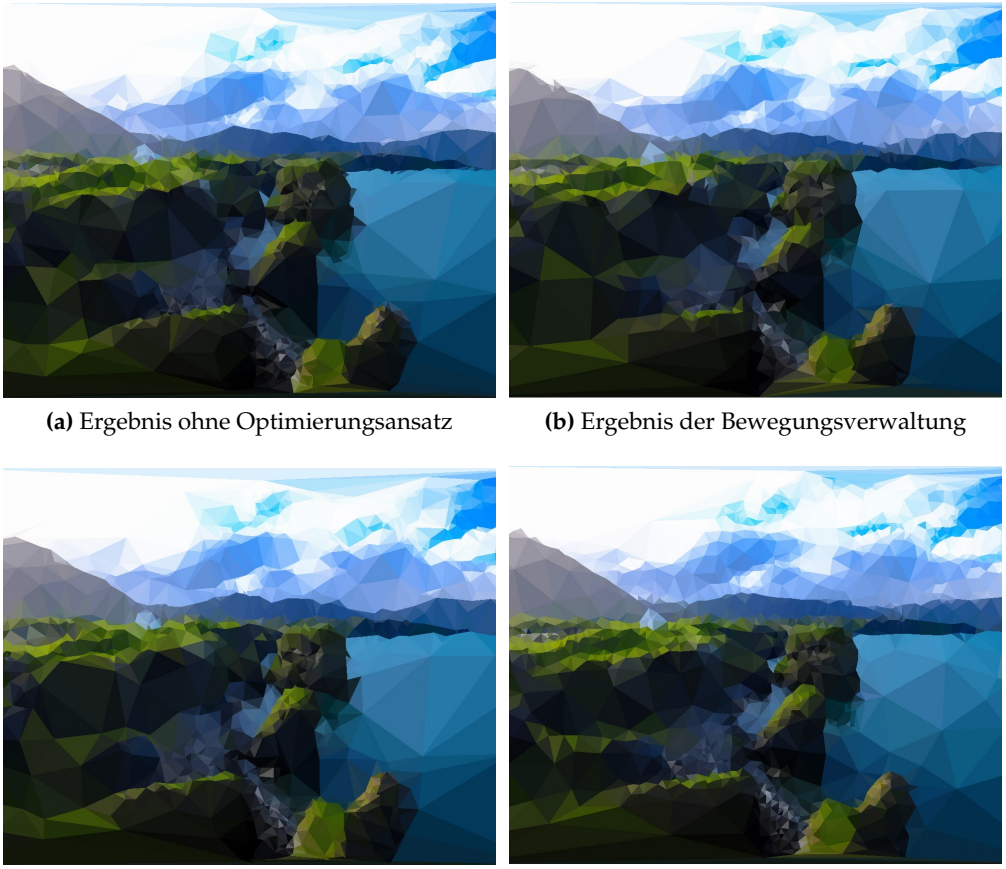


Abbildung 34: Verlauf der Optimierungsansätze

Verwaltung der Bewegung

Dieser Optimierungsansatz implementiert die Verwaltung der letzten Bewegung eines Punktes. Bei dieser Methode wird die letzte Bewegung jedes Punktes gespeichert. Im nächsten Iterationsschritt wird die Richtung des Gradienten beschränkt, indem die gegenüberliegende Richtung des zuletzt getätigten Schritts keine Option der Verschiebung darstellt. Mit diesem Verfahren soll Rechenzeit gespart werden. Der Verlauf des Graphen verifiziert die Intention, vgl. Tabelle 1. Die benötigte Zeit pro Iterationsschritt und die gesamte Berechnungszeit weisen die niedrigsten Werte im Vergleich zu den anderen Ansätzen auf. Im Vergleich zum Ausgangsverfahren, ergeben die Ergebnisse geringfügige visuelle Unterschiede. In einigen Regionen entstehen optisch unruhigere Bereiche, in denen Kanten nicht so präzise wiedergegeben werden, wie bei dem Ausgangsverfahren, sodass mehr Dreiecke in diesem Bereich verwendet werden. In anderen Regionen wird eine ähnliche Erfassung der Bildinhalte erzeugt, sodass ingesamt ein etwas höherer Approximationsfehler im Vergleich zur nicht optimierten Variante entsteht. Ein Grund für die regionalen Unterschiede kann in der Dynamik des Verfahrens gefunden werden. Sobald Änderungen der Struktur in der Nachbarschaft eines Punktes durchgeführt werden, kann die Unterbindung einer Bewegungsrichtung kontraproduktiv wirken.
Aus Tabelle 1 geht hervor, dass bei nahezu gleichem Fehlerwert deutlich weniger Iterationen und weniger Zeit für die Berechnung benötigt wurde.



(a) Ergebnis ohne Optimierungsansatz

(b) Ergebnis der Bewegungsverwaltung

(c) Ergebnis der diagonalen Schritte

(d) Ergebnis des Bewegungsvektors

Abbildung 35: Ergebnisse des Optimierungsansätze

Aus dem visuellen Ergebnis und den Daten folgt, dass dieser Optimierungsansatz für den Fall eine Lösung darstellt, in dem eine schnellere Berechung wichtiger als eine höhere Qualität ist. Der Ansatz bietet außerdem die Möglichkeit, die nicht benötigte Zeit in das Verfahren zu investieren, und einen niedrigeren Approximationsfehler anzustreben.

Diagonale Schrittrichtungen

Bei dem Ansatz der diagonalen Schritte werden statt der vier Hauptrichtungen die Positionen oben links, oben rechts, unten links und unten rechts des Punktes nach geringerer Energie untersucht.
Diese Variante verhält sich ähnlich wie der Ausgangsansatz ohne Optimierung. Eine kürzere gesamte Berechnungsdauer wird verzeichnet, mit weniger Iterationen, vgl. Tabelle 1. Ein qualitativer Unterschied der Ergebnisbilder ist zu erkennen. Einige Bereiche werden detaillierter von diesem Ansatz wiedergegeben, während in anderen Regionen

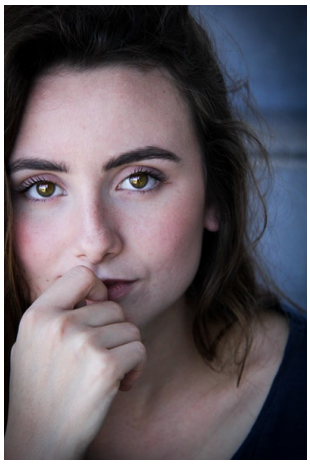
eine erhöhte Unstetigkeit gegenüber der Vergleichsvariante entsteht, vgl. Abbildung 35. Insgesamt wird keine nennbare Verbesserung des Approximationsfehlers erreicht. Die möglichen Schrittrichtungen können diese Eigenschaft verursachen, sodass in einigen Regionen schneller ein lokales Minimum erreicht wird.
Ein Verfahren, das die Energie aller benachbarten Pixel auswertet, würde eine deutliche Steigerung der Rechenzeit verursachen.

Bewegungsvektor
Der Ansatz des Bewegungsvektors soll eine flexiblere Möglichkeit des Optmierungsschritts realisieren. Die Positionen, die sich aus dem Gradientverfahren erschließen, werden als Vektor dargestellt. Anhand der Energiewerte, die durch eine Verschiebung resultieren würden, werden sie skaliert und gemittelt. Dadurch sollen mehrere Positionen bei dem Minimierungsschritt ermöglicht werden.
Aus den Werten aus Tabelle 1 geht hervor, dass trotz längerer Berechnungszeit ein höherer Fehlerwert resultiert. Entgegen der Erwartung ist das Ergebnisbild von gesteigerter ästhetischer Qualität. Details des Eingabebildes werden präziser wiedergegeben. An harten Kanten werden weniger Dreiecke eingesetzt, sodass eine exakte Abgrenzung der Bildinhalte erzielt wird. An weichen Übergängen werden hingegen mehr Dreiecke verwendet, wodurch eine ruhige Wirkung des Bildes erzielt wird. Insgesamt wird eine Steigerung der visuellen Qualität verzeichnet.
Grund für diese Steigerung kann in der flexibleren Verschiebung der Punkte gesehen werden. Dies ermöglicht vermutlich eine gesteigerte Adaptionsmöglichkeit, wodurch Übergänge der Bildinhalte insgesamt schneller erfasst werden. Ein weiterer Grund der gesteigerten Ästhetik kann durch die Schrittweite begründet werden. Durch Mittlung der Schrittrichtungen entstehen vergleichsweise kleinere Schritte, die eine exaktere Positionierung der Punkte erlaubt. Der Graph in Abbidlung 35 zeigt eine deutlich höhere Rechenzeit pro Iterationsschritt. Das resultiert aus Rechenoperationen, die im Vergleich zu anderen Ansätzen, zusätzlich stattfinden. Die Steigerung an Qualität wird durch eine 25%ige Steigerung der benötigten Zeit, im Vergleich zum Ausgangsverfahren, erwirkt.
Sobald ein qualitativ hochwertigeres Ergebnis angestrebt wird und keine zeitliche Beschränkung vorliegt, kann dieser Ansatz verwendet werden.

6.6 Zusammenfassung

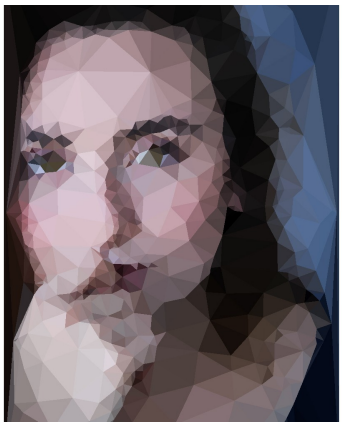
Die Ergebnisse der umgesetzten Verfahren verdeutlichen Herausforderungen und Möglichkeiten der einzelnen Ansätze. Der Vergleich zwischen der GPU- und der CPU-basierten Variante der stilisierten Triangulierung hebt die qualitativen Differenzen deutlich hervor. Mittels des Rasterisierers der Grafikkarte werden hochwertigere Ergebnisse erzeugt. Soll das Verfahren auf einem Server ohne Grafikkarte eingesetzt werden, zeigt die CPU-Variante, dass Ergebnisse erreichbar sind, die mit alternativen Ansätzen mobiler Anwendungen konkurrieren können. Weiterhin umfasst dieses Kapitel die Präsentation der Ergebnisse der Optimierungsansätze und die Ergebnisse der Stilisierungen.

Es werden Eigenschaften hervorgehoben und Vergleiche erstellt, sodass Einsatzgebiete ermittelt werden. Erörtert werden auch die Limitierungen der Verfahren. Im folgenden Kapitel werden, basierend auf den Ergebnissen, mögliche weiterführende Arbeiten aufgezeigt



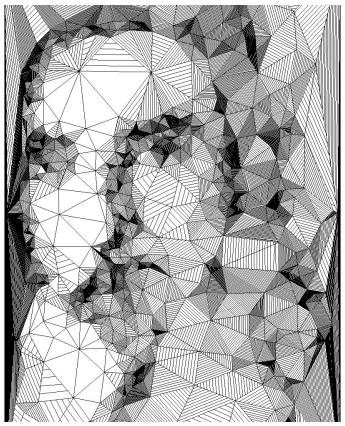
(a) Eingabebild 1

(b) Eingabebild 2

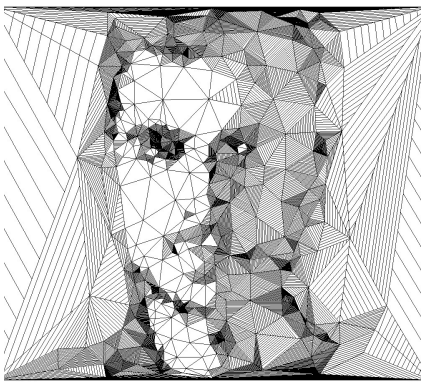


(c) Trianguliertes Ergebnis 1

(d) Trianguliertes Ergebnis 2



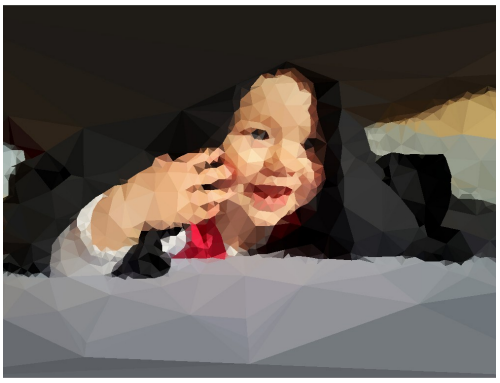
(e) Stilisierte Triangulierung nach Josh Bryan mit verringerter Dynamik



(f) Stilisierte Triangulierung nach Josh Bryan mit verringerter Dynamik

Abbildung 36: Stilisierte Triangulierung basierend auf Linien

(a) Beispielergebnis 1



(b) Beispielergebnis 2

Abbildung 37: Beispielergebnisse des CPU-basierten Verfahren

7 Fazit

Dieses abschließende Kapitel umfasst eine Zusammenfassung der Kapitel und schließt mit einem Ausblick, der Möglichkeiten weiterführender Arbeit darstellt.

7.1 Zusammenfassung

Diese Arbeit strebt mehrere Ziele an. Aufbauend auf einer Veröffentlichung [LG18], soll das neue verwendete Verfahren möglichst vielen Nutzern zugänglich gemacht werden. Als Grundlage der Umsetzung dienen essentielle Vorgehen der Triangulierung. Methoden der dynamischen Erweiterung eines Dreiecknetzes und dessen Topologie stehen neben dem Optimierungsverfahren im Mittelpunkt. Zusätzlich umfasst diese Thematik die Regularisierung der Triangulierung und die Sichtung von alternativen Ansätzen und Verfahren, die mobile Anwendungen und thematikbezogene Arbeiten einschließen. Mit dieser Grundlage wird ein Konzept entworfen. Das Verfahren soll als App zur Verfügung gestellt werden. «Mesh» wird vorgestellt. Die App soll anfangs für Android-Geräte entwickelt werden, jedoch mit der Aussicht ebenfalls mit Apple-Geräten kompatibel zu sein. Dieser Ansatz wird von dem React Native Framework unterstützt.
Die mobile Anwendung soll als Host-Client System umgesetzt werden, damit die ressourcenbedürftige Berechnung auf einen Server ausgelagert werden kann und Nutzer nicht auf leistungsfähige Hardware angewiesen sind. Dafür wird ein Webserver erstellt, der die Kommunikation zwischen der App und der stilisierten Triangulation ermöglicht.
Grafikkarten bei Servern sind nicht weit verbreitet, weshalb die Anwendung für die CPU umgesetzt wird.
Der Rasterisierer und die parallele Verarbeitungsmöglichkeit der Grafikkarte haben deutlichen Einfluss auf den Unterschied der Ergebnisse der beiden Verfahrensvarianten. Um ähnliche Ergebnisse mit der CPU zu erreichen, ist ein Vielfaches der Berechnungszeit nötig. Zusätzlich erweist es sich als problematisch, dass Pixel innerhalb eines Dreiecks nicht durch atomare Funktionen der Grafikkarte erfasst werden können. Bei der CPU-basierten Variante werden Pixel, die sich innerhalb eines Dreiecks befinden, mit einer vektorbasierten Abtastung erfasst, was in rechnerischen Ungenauigkeiten resultiert. Die Ungenauigkeiten sind auf die Konvertierung zwischen Gleitkomma- und Ganzzahlen zurückzuführen und verursachen ungenauere Ergebnisse. Eine mögliche Steigerung der Qualität besteht bei der Verwendung von strukturoptimierenden Verfahren, die eine präzisere Erfassung des Bildinhalts durch Entfernen redundanter Dreiecke erzielt.
Berechnungszeiten von mehreren Minuten sind bei einer mobilen Anwendung nicht akzeptabel, weshalb sich die Auslagerung auf einen Server als sinnvoll erweist. Nach erfolgreichem Versenden des Eingabebilds, erfolgt nach gewisser Zeit automatisch eine Erinnerung. Die durchschnittliche Berechnungszeit ist erreicht, weshalb der Nutzer mittels der App das Ergebnis vom Server herunterladen und speichern kann.

Als weiteres Ziel dieser Arbeit wird die Imitation eines künstlerischen Stils aufgenommen. Die handgefertigten Werke von Josh Bryan dienen dafür als Vorlage. Der möglichen Lösung mittels *Tonal-Art-Maps* fehlt es an Natürlichkeit. Mittels eines *Geometryshaders* soll eine zufallsbasierte Natürlichkeit imitiert werden. Durch ein pseudozufälliges Auftreten von schrägen und zu weit gezeichneten Linien, wird die Möglichkeit verdeutlicht, nicht-photorealistische Effekte flexibel anpassen zu können. Gleichzeitig wird deutlich, dass eine Triangulierung von Bryan nur schwer nachzuahmen ist. Die sehr präzise Erfassung von Details ist mit der genutzten Triangulierung nicht erreichbar, sodass es bei den Ergebnissen dieser Arbeit an Wirkung und Qualität fehlt. Das Potenzial des Verfahrens verdeutlicht jedoch die Möglichkeit der Imitation.

Der Ansatz, der der Veröffentlichung zugrunde liegt, löst die adaptive Triangulierung, indem ein Energieminimierungsverfahren verwendet wird. Einzelne Dreiecke werden mit einer konstanten Farbe dargestellt. Diese Farbe ergibt sich aus der mittleren Farbe des Bereichs, der durch ein Dreieck erfasst wird. Die Abweichung der gemittelten Farbe von den original Farbwerten der Pixel im Ausgangsbild, ergibt einen Approximationsfehler, den es zu minimieren gilt. Durch Verschieben der Ecken der Dreiecke werden unterschiedliche Bereiche erfasst, wodurch sich die Farbe des Dreiecks anpasst. Die Verschiebung wird durch das Optimierungsverfahren erreicht, bei dem neue Positionen der Punkte auf geringere Fehler untersucht werden. Bestandteil dieser Arbeit ist die Vorstellung von Optimierungsansätzen der Energieminimierung.
Dazu werden drei Ansätze getestet und evaluiert. Sie untersuchen die Auswirkung der Schrittwahl und Richtung des Punktes auf die ästhetische Qualität und die benötigten Ressourcen, Zeit und Rechenaufwand.
Der erste Ansatz versucht die Schrittrichtungen sinnvoll einzugrenzen, um Rechenaufwand zu sparen. Die Ergebnisse zeigen, dass bei geringerem Zeitaufwand eine Lösung von etwas geringerer Qualität erreicht werden kann.
Der zweite Ansatz ändert die möglichen Schrittrichtungen von horizontalen und vertikalen Schritten zu diagonalen. In diesem Fall wird bei gleichem zeitlichen Aufwand ein visuell ähliches Ergebnis erzielt.
Der letzte Ansatz lässt eine flexiblere Verschiebung der Punkte zu. Die als Vektoren interpretierte Verschiebung wird anhand der Fehlermaße der Nachbarpositionen gewichtet. Dadurch wird eine Verschiebung in die Richtung des höchsten Fehlers ausgeschlossen und eine dynamische Aktualiserung der Position ermöglicht. Die Ergebnisse weisen eine erhöhte Darstellung von Bilddetails, und eine päzisere Lokalisierung von Kanten auf. Die ästhetische Steigerung findet bei deutlichem Mehraufwand von Zeit statt.
Zusammenfassend gilt für die Optimierungsansätze, dass, je nach Kriterium, ein Ansatz profitabel verwendet werden kann. Für zeitkritische Anforderungen kann die Beschränkung der Schrittrichtung vorteilhafte Wirkung zeigen. Sobald die höchst mögliche Qualität angestrebt ist, kann der Bewegungsvektor verwendet werden.

7.2 Ausblick

Dieses Unterkapitel bietet einen Ausblick der Anwendung. Dabei werden alle größeren Unterthemen aufgegriffen und Möglichkeiten aufgezeigt, die Verfahren in weiterführenden Arbeiten zu nutzen und zu verbessern.
Optimierungen der Dreiecksstruktur sind ein wichtiger Aspekt, um das Verfahren in einer marktreifen App zu verwenden. Die Vereinfachungen der Netzstruktur, wie die Methode nach Garland und Heckbert [GH97], oder die vereinfachte Variante der Kantenkollabierung bieten Aussichten auf qualitativ hochwertigere Ergebnisse. Die Ergebnisse von höchster Qualität werden mit der GPU-Variante erreicht, sodass die Implementation dieses Verfahrens für ein mobiles Endgerät als weiterführende Arbeit gilt. Anschließend ist die Erstellung einer Benutzerzufriedenheitsstudie sinnvoll. Zu klären gilt, ab welcher Dauer ein Nutzer es als störend empfindet, dass das Gerät mit der Berechnung für andere Aufgaben nicht verfügbar ist. Ebenso gilt es zu testen, wie hoch die Akzeptanz des Host-Client Systems ausfällt. Sobald das Verfahren auf einem mobilen Gerät ausgeführt wird, kann die Anwendung als zeitkritisch gesehen werden. In dem Fall gilt es zu testen, ob einer der Optimierungsansätze Vorteile bietet.
Die Unabhängigkeit der Serveranbindung ist ein Ziel weiterführender Arbeiten. Ebenso denkbar ist eine hybride Variante, bei der der Nutzer zwischen der lokalen und ausgelagerten Berechnung wählen kann.
Der umfangreiche Bereich der Benutzerinteraktion ist ein entscheidender Punkt der Qualitätssteigerung. Durch Möglichkeiten der Manipulation der Triangulierung, können individuelle Ergebnisse erwirkt werden. Dieser Aspekt gilt für eine lokal arbeitende Version der Anwendung. In dem Fall ist die Umsetzung von verschiedenen Renderstilen ebenso als Ziel zu sehen. Ein zusätzliches Ziel ist das automatisierte Herunterladen der Ergebnisse. Ob diese Option die Benutzerfreundlichkeit steigert, wäre ebenfalls zu testen.
In der Rubrik der Stilisierungen sind zusätzliche weiterführende Arbeiten möglich. Die variable Einstellung des Ausgangspunktes bei dem «Scherbenstil » eröffnet Möglichkeiten in der Animation oder bei der Stilisierung von Videos. In diesem Fall kann der Ausgangspunkt des Effekts dem Hauptmotiv im Video folgen, sodass eine zusätzliche Hervorhebung stattfindet.
Bei der Stilisierung nach Josh Bryan fehlte die Herausstellung von Merkmalen und Details. Mit Nutzereingaben ist eine Steigerung der Qualität wahrscheinlich. Mit zusätzlicher Detektion der Augen oder weiteren Features im Gesicht, ist eine automatisierte Stilisierung mit erhöhter Qualität ebenso möglich.

8 Hinweise

Mathematische Notation:
Mengen (Großbuchstaben): $\mathcal{V}$
Vektoren (Kleinbuchstaben, fett): $\mathbf{v}$
Skalare (Kleinbuchstaben): s

Instagram ist eines der Facebook Produkte [42] und gehört zur Firma Facebook Inc.

Die Plattform «App Store» und das Betriebssystem iOS entstammen der Firma Apple mit dem Hauptsitz in den USA [18].

Die Plattform «Google Play» und das Betriebssystem Android sind Eigentum der Firma Google LLC mit dem Hauptsitz in den USA [30].

Aufgrund der Thematik und des technischen Bezugs, wird ein fachbezogenes Vokabular vorausgesetzt, sodass nicht alle Abkürzungen erklärt und englische Begriffe übersetzt werden.

Tabellenverzeichnis

Abbildungsverzeichnis

Literatur

[AE87] AVIS, David ; ELGINDY, Hossam: Triangulating point sets in space. In: *Discrete & Computational Geometry* 2 (1987), Nr. 2, S. 99–111

[A.P09] A.P.GODSE: *Computer Graphics*. Technical Publications, 2009

[Bay10] BAYER, Jürgen: *"Das C# 2010 Codebook"*. Pearson Deutschland GmbH, 2010

[BBB+97] BLOOMENTHAL, Jules ; BAJAJ, Chandrajit ; BLINN, Jim ; WYVILL, Brian ; CANI, Marie-Paule ; ROCKWOOD, Alyn ; WYVILL, Geoff: *Introduction to Implicit Surfaces*. Morgan Kaufmann, 1997

[BKP+10] BOTSCH, Mario ; KOBBELT, Leif ; PAULY, Mark ; ALLIEZ, Pierre ; LÉVY, Bruno: *Polygon mesh processing*. AK Peters/CRC Press, 2010

[BLFF96] BERNERS-LEE, Tim ; FIELDING, Roy ; FRYSTYK, Henrik: Hypertext transfer protocol–HTTP/1.0. 1996. – Forschungsbericht

[Bog11] BOGUSLAWSKI, Pawel: *Modelling and analysing 3d building interiors with the dual half-edge data structure*, University of Glamorgan Pontypridd, Wales, UK, Diss., 2011

[Bow81] BOWYER, Adrian: Computing dirichlet tessellations. In: *The computer journal* 24 (1981), Nr. 2, S. 162–166

[CD14] CHAI, T. ; DRAXLER, R. R.: Root mean square error (RMSE) or mean absolute error (MAE)? In: *Geoscientific Model Development Discussions* 7 (2014), Februar, S. 1525–1534. `http://dx.doi.org/10.5194/gmdd-7-1525-2014`. – DOI 10.5194/gmdd–7–1525–2014

[CDS16] CHENG, Siu-Wing ; DEY, Tamal K. ; SHEWCHUK, Jonathan: *Delaunay Mesh Generation*. CRC Press, 2016

[CH10] CHESI, Graziano ; HASHIMOTO, Koichi: *Visual Servoing via Advanced Numerical Methods*. Springer, 2010

[Cha12] CHANG, Mark: *Adaptive Design Theory and Implementation Using SAS and R*. CRC Press, 2012

[Eis15] EISENMAN, Bonnie: *Learning react native: building native mobile apps with javascript*. Ö'Reilly Media, Inc.", 2015

[FDFL+14] FORTINO, Giancarlo ; DI FATTA, Giuseppe ; LI, Wenfeng ; OCHOA, Sergio F. ; CUZZOCREA, Alfredo ; PATHAN, Mukaddim: *Internet and Distributed Computing Systems: 7th International Conference,...* Springer, 2014

[FDVVD+96] FOLEY, James D. ; DAN VAN, Foley ; VAN DAM, Andries ; FEINER, Steven K. ; HUGHES, John F. ; HUGHES, J. ; ANGEL, Edward: *Computer Graphics: Principles and Practice*. Addison-Wesley Professional, 1996

[Fra00] FRANK, Anton C.: *Organisationsprinzipien zur Integration von geometrischer Modellierung, numerischer Simulation und Visualisierung*. Herbert Utz Verlag, 2000

[Fri09] FRIEDL, Jeffrey E. F.: *Reguläre Ausdrücke*. O'Reilly Germany, 2009

[Ger07] GERTH, Holger: *Verwaltung sehr großer volumetrischer Datensätze unter Verwendung derhierarchischen Lauflängenkodierung*, Technische Universität Chemnitz, Diplomarbeit, 2007

[GGD05] GRUNDLAND, Mark ; GIBBS, Chris ; DODGSON, Neil A.: Stylized rendering for multiresolution image representation. In: *Human Vision and Electronic Imaging X* Bd. 5666 International Society for Optics and Photonics, 2005, S. 280–293

[GH97] GARLAND, Michael ; HECKBERT, Paul S.: Surface simplification using quadric error metrics. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* ACM Press/Addison-Wesley Publishing Co., 1997, S. 209–216

[Gol02] GOLENIEWSKI, Lillian: *Telecommunications Essentials: The Complete Global Source for Communications Fundamentals, Data Networking and the Internet, and Next-generation Networks*. Addison-Wesley Professional, 2002

[Gre11] GREEN, Rue: *Cisco Unified Customer Voice Portal: Building Unified Contact Centers*. Cisco Press, 2011

[GS97] GRINSTEAD, Charles M. ; SNELL, James L.: *Introduction to Probabilty*. American Mathematical Soc., 1997

[GT02] GOURLEY, David ; TOTTY, Brian: *HTTP: The Definitive Guide*. Ö'Reilly Media, Inc.", 2002

[GW09] GRAVETTER, Frederick J. ; WALLNAU, Larry B.: *Statistics for the Behavioral Sciences*. Cengage Learning, 2009

[Har98] HARTMANN, Erich: A marching method for the triangulation of surfaces. In: *The Visual Computer* 14 (1998), Nr. 3, S. 95–108

[HDZ05] HANSEN, Glen A. ; DOULGASS, R. W. ; ZARDECKI, Andrew: *Mesh Enhancement: Selected Elliptic Methods, Foundations and Applications*. Imperial College Press, 2005

[Hei06] HEINTZE, Florian v.: *Kunst und Architektur: 1000 Fragen und Antworten*. wissenmedia Verlag, 2006

[HV16] HANSSON, Niclas ; VIDHALL, Tomas: *Effects on performance and usability for cross-platform application development using React Native*. 2016

[KS08] KONGENT SOLUTIONS, Inc.: *Java Server Programming Java Ee5*. Dreamtech Press, 2008

[LG18] LAWONN, Kai ; GÜNTHER, Tobias: Stylized Image Triangulation. In: *Comput. Graph. Forum* 38 (2018), S. 221–234

[LGT+19] LAGA, Hamid ; GUO, Yulan ; TABIA, Hedi ; FISHER, Robert B. ; BENNAMOUN, Mohammed: *3D Shape Analysis: Fundamentals, Theory, and Applications*. John Wiley & Sons, 2019

[LH08] LI, Xiaoyun ; HUNTER, David K.: Distributed coordinate-free hole recovery. In: *ICC Workshops-2008 IEEE International Conference on Communications Workshops* IEEE, 2008, S. 189–194

[LS80] LEE, Der-Tsai ; SCHACHTER, Bruce J.: Two algorithms for constructing a Delaunay triangulation. In: *International Journal of Computer & Information Sciences* 9 (1980), Nr. 3, S. 219–242

[Mav90] MAVRIPLIS, Dimitri J.: Adaptive mesh generation for viscous flows using delaunay triangulation, 1990

[MF06] MÜLLER-FONFARA, Robert: *Mathematik verständlich*. Weltbild, 2006

[Müc98] MÜCKE, Ernst P.: A robust implementation for three-dimensional Delaunay triangulations. In: *International Journal of Computational Geometry & Applications* 8 (1998), Nr. 02, S. 255–276

[Muk12] MUKUNDAN, Ramakrishnan: *Advanced Methods in Computer Graphics: With examples in OpenGL*. Springer Science & Business Media, 2012

[NFHS12] NISCHWITZ, Alfred ; FISCHER, Max ; HABERÄCKER, Peter ; SOCHER, Gudrun: *Computergrafik und Bildverarbeitung: Band II: Bildverarbeitung*. Springer-Verlag, 2012

[OBB01] OHTAKE, Yutaka ; BELYAEV, Alexander ; BOGAEVSKI, Ilia: Mesh regularization and adaptive smoothing. In: *Computer-Aided Design* 33 (2001), Nr. 11, S. 789–800

[Par05] PARTSCH, Susanna: *Die 101 wichtigsten Fragen: moderne Kunst*. C.H.Beck, 2005

[RR08] RICHARDSON, Leonard ; RUBY, Sam: *RESTful Web Services*. Ö'Reilly Media, Inc.", 2008

[RTF+04] RASKAR, Ramesh ; TAN, Kar-Han ; FERIS, Rogerio ; YU, Jingyi ; TURK, Matthew: Non-photorealistic camera: depth edge detection and stylized rendering using multi-flash imaging. In: *ACM transactions on graphics (TOG)* Bd. 23 ACM, 2004, S. 679–688

[SG04] SURAZHSKY, Vitaly ; GOTSMAN, Craig: High quality compatible triangulations. In: *Engineering with Computers* 20 (2004), Nr. 2, S. 147–156

[She96] SHEWCHUK, Jonathan R.: Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In: *Workshop on Applied Computational Geometry* Springer, 1996, S. 203–222

[Sin08] SINGH, Abhishek: *Vulnerability Analysis and Defense for the Internet*. Springer Science & Business Media, 2008

[SL11] SULLIVAN, Bryan ; LIU, Vincent: *Web Application Security, A Beginner's Guide*. Mcgraw Hill Professional, 2011

[SLWS07] SUN, Jian ; LIANG, Lin ; WEN, Fang ; SHUM, Heung-Yeung: Image vectorization using optimized gradient meshes. In: *ACM Transactions on Graphics (TOG)* Bd. 26 ACM, 2007, S. 11

[TM98] TOMASI, Carlo ; MANDUCHI, Roberto: Bilateral filtering for gray and color images. In: *Iccv* Bd. 98, 1998, S. 2

[TM19] TOBLER, Robert ; MAIERHOFER, Stefan: A Mesh Data Structure for Rendering and Subdivision. (2019), 06

[WFH11] WITTEN, Ian H. ; FRANK, Eibe ; HALL, Mark A.: *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier, 2011

[Wit14] WITTING, Walter: *Licht.Sehen.Gestalten.* Birkhäuser, 2014

[XLXJ11] XU, Li ; LU, Cewu ; XU, Yi ; JIA, Jiaya: Image smoothing via L 0 gradient minimization. In: *ACM Transactions on Graphics (TOG)* Bd. 30 ACM, 2011, S. 174

[Xu11] XU, Wei: Towards Optimal One Pass Large Scale Learning with Averaged Stochastic Gradient Descent. (2011), 07

[YM96] YEAGER, Nancy J. ; MCGRATH, Robert E.: *WebServer Technology*. Morgan Kaufman Publishers, Inc., 1996

[Zdu06] ZDUNCZYK, Bozena: *Analyse von Repräsentationen für 3D-Modelle aus Sicht der Softwaretechnik*, Universität Koblenz-Landau, Diss., 2006

Internet-Quellen

[1] Josh Bryan. https://www.joshbryanart.com/, letztmals besucht am 23.06.2019.

[2] Ole Christian Eidheim. https://github.com/eidheim/Simple-Web-Server, letztmals besucht am 2.07.2019.

[3] Sean Barrett et al. https://github.com/nothings/stb/blob/master/stb_image_write.h, letztmals besucht am 2.07.2019.

[4] Sean Barrett et al. https://github.com/nothings/stb/blob/master/stb_image.h, letztmals besucht am 2.07.2019.

[5] Exosmart. https://play.google.com/store/apps/details?id=com.exosmart.polycolorand, letztmals besucht am 15.06.2019.

[6] Akimitsu Hamamuro. http://jsdo.it/akm2/xoYx, letztmals besucht am 15.06.2019.

[7] James Isaac. https://github.com/jamesisaac/react-native-background-task, letztmals besucht am 4.07.2019.

[8] Gerrit Lochmann Kevin Keul, Detlev Droege. https://gitlab.uni-koblenz.de/CVK/CVK_2, letztmals besucht am 2.07.2019.

[9] Bartlomiej Niemtur. https://play.google.com/store/apps/details?id=com.bitbotany.polygen, letztmals besucht am 16.06.2019.

[10] Rene Nyffenegger. https://github.com/ReneNyffenegger/cpp-base64, letztmals besucht am 2.07.2019.

[11] Charles Ojukwu. https://codepen.io/cojdev/full/vgmxmj, letztmals besucht am 12.07.2019.

[12] Paul Ollivier. http://trimaginator.com/, letztmals besucht am 16.06.2019.

[13] Jonathan Puckey. https://puckey.studio/projects/delaunay-raster, letztmals besucht am 15.06.2019.

[14] Jonathan Robie. https://www.w3.org/TR/WD-DOM/introduction.html, letztmals besucht am 9.06.2019.

[15] Quinn Rohlf. https://trianglify.io/, letztmals besucht am 12.07.2019.

[16] Maks Surguy. https://msurguy.github.io/triangles/, letztmals besucht am 12.07.2019.

[17] Unbekannt. https://facebook.github.io/react-native/docs/headless-js-android, letztmals besucht am 3.07.2019.

[18] Unbekannt. https://www.apple.com/contact/, letztmals besucht am 16.07.2019.

[19] Unbekannt. https://upload.wikimedia.org/wikipedia/de/1/17/Voronoi-Delaunay.svg, letztmals besucht am 16.07.2019.

[20] Unbekannt. http://statistik.wu-wien.ac.at/~leydold/MOK/HTML/node123.html, letztmals besucht am 06.06.2019.

[21] Unbekannt. https://github.com/react-native-community/react-native-image-picker, letztmals besucht am 3.07.2019.

[22] Unbekannt. http://polygonize.net/, letztmals besucht am 11.07.2019.

[23] Unbekannt. https://imagemagick.org/index.php, letztmals besucht am 16.07.2019.

[24] Unbekannt. https://www.npmjs.com/package/react-native-push-notification, letztmals besucht am 3.07.2019.

[25] Unbekannt. https://learnopengl.com/, letztmals besucht am 2.07.2019.

[26] Unbekannt. https://www.gimp.org/, letztmals besucht am 16.07.2019.

[27] Unbekannt. https://www.alamy.de/stockfoto-bildende-kunst-feininger-lyonel-1871-1956-malerei-der-leuchtturm-museum-folkwang-essen-des-kunstlers-urheberrecht-auch-geraumt-werden-muss-110563785.html, letztmals besucht am 10.06.2019.

[28] Unbekannt. https://de.cppreference.com/w/cpp/language/types, letztmals besucht am 21.06.2019.

[29] Unbekannt. https://play.google.com/store/apps/details?id=com.palabs.polygon, letztmals besucht am 15.06.2019.

[30] Unbekannt. https://www.google.com/contact/, letztmals besucht am 16.07.2019.

[31] Unbekannt. https://developer.android.com/studio/index.html, letztmals besucht am 15.07.2019.

[32] Unbekannt. https://instagram-press.com/our-story/, letztmals besucht am 9.07.2019.

[33] Unbekannt. https://stackoverflow.com/questions/4200224/random-noise-functions-for-glsl?rq=1, letztmals besucht am 2.07.2019.

[34] Unbekannt. https://www.json.org/, letztmals besucht am 10.07.2019.

[35] Unbekannt. https://www.archlinux.org/packages/community/x86_64/inotify-tools/, letztmals besucht am 2.07.2019.

[36] Unbekannt. https://www.youtube.com/watch?v=txjFhnsdhA4, letztmals besucht am 13.07.2019.

[37] Unbekannt. https://www.lorenz24.com/kd24/Artikelbilder/6042018/p95048.jpg, letztmals besucht am 10.06.2019.

[38] Unbekannt. https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html, letztmals besucht am 15.07.2019.

[39] Unbekannt. https://www.pgm.de/media/image/thumbnail/PK-63X_998x831.jpg, letztmals besucht am 10.06.2019.

[40] Unbekannt. https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Identifying_resources_on_the_Web, letztmals besucht am 24.06.2019.

[41] Unbekannt. https://developer.android.com/guide/components/fundamentals, letztmals besucht am 07.06.2019.

[42] Unbekannt. https://www.facebook.com/help/1561485474074139?ref=igtos, letztmals besucht am 16.07.2019.

[43] Unbekannt. https://en.wikipedia.org/w/index.php?curid=20658476, letztmals besucht am 16.07.2019.

[44] Unbekannt. https://www.opengl.org/, letztmals besucht am 2.07.2019.

[45] Unbekannt. https://ionicons.com/, letztmals besucht am 3.07.2019.

[46] Unbekannt. https://facebook.github.io/react-native/docs/network, letztmals besucht am 3.07.2019.

[47] Unbekannt. https://play.google.com/store/apps/details?id=com.exosmart.polycolorand, letztmals besucht am 15.06.2019.

[48] Unbekannt. https://facebook.github.io/react-native/docs/tutorial, letztmals besucht am 07.06.2019.

[49] Unbekannt. https://help.instagram.com/581066165581870, letztmals besucht am 16.07.2019.

[50] Duke Vukadinovic. https://www.globalsign.com/de-de/blog/unterschied-zwischen-http-und-https/, letztmals besucht am 9.06.2019.

[51] Dofl Y. H. Yun. dmesh.thedofl.com, letztmals besucht am 8.07.2019.