



Studienarbeit

Identifizierung von Gitternetzstrukturen in Bildern von Objekten mit regelmäßigen Oberflächenstrukturen

Oliver Stadie

stadie (at) informatik.hu-berlin.de

Gutachter: Prof. Dr. Peter Eisert

Betreuer: Anna Hilsmann

Kurzfassung

Muster in Bildern zu erkennen ist eine typische Aufgabe im Bereich der Computer Vision. Das Finden und Identifizieren deformierter Strukturen von sich wiederholenden Kacheln ist jedoch erst wenig erforscht. Diese Arbeit stellt einen vollautomatischen Ansatz vor, um Netzstrukturen in Bildern, basierend auf den Bilddaten und einigen Feature-Punkten, zu identifizieren.

Schlagwörter: Computer Vision, Netz-Detektion, Muster-Erkennung

Abstract

Finding patterns in pictures is a typical task in the area of computer vision. But when it comes to finding and identifying deformed structures of repeating tiles in pictures, relatively little research has been done in this area. This work presents a fully automated approach to identify the lattice structure of a picture, based on the information in the picture and some feature points.

Keywords: Computer vision, Lattice detection, pattern finding

Inhalt

Kurzfassung	3
Abstract	3
Inhalt	4
1. Einleitung	7
1.1 Einordnung	7
1.2 Motivation	8
1.3 Stand der Technik	10
1.4 Problemstellung & Rahmenbedingungen	13
1.5 Begriffe und Definitionen.....	14
1.5.1 Bild- oder Pixelkoordinaten.....	14
1.5.2 Topologische Koordinaten.....	15
1.5.3 Feature-Punkte	16
1.5.4 Cluster.....	17
1.5.5 Quad & Texel	17
1.5.6 Mapping.....	18
1.6 Überblick der folgenden Kapitel	20
2. Lösungsansatz	20
2.1 Iterative Arbeitsweise	20
2.2 Ständiges globales Mapping	21
2.3 Quads und Texel als elementare Größe	21
2.4 Durchschnitts-Texel	22
2.5 Relativer Fehler	22
2.6 Cluster-Offsets.....	22
3. Theorie.....	23
3.1 Cluster initialisieren.....	23
3.2 Initialisierung.....	24
3.2.1 Bestimmen des ersten Quad-Kandidaten.....	25
3.2.2 Bestimmen des zweiten Quad-Kandidaten.....	26
3.2.3 Plausibilität der Quad-Kandidaten überprüfen	27
3.2.4 Bewertung der Ebenheit	29
3.2.5 Auswahl eines Quad-Paars	30
3.3 Cluster-Iteration.....	31
3.3.1 Cluster-Offset initialisieren	32
3.3.2 Akzeptiere Feature-Punkte in fertigen Quads.....	33
3.3.3 Akzeptiere neu entstandener Quads.....	33

3.3.4	Führe Quad-Iterationen aus	34
3.4	Quad-Iteration.....	34
3.4.1	Bestimme Kontroll-Punkt-Kandidaten	34
3.4.2	Verwerfen ungeeigneter Kontroll-Punkt-Kandidaten.....	35
3.4.3	Validierung der Quad-Kandidaten.....	36
3.4.4	Löschen von Feature-Punkten in neuen Quads	38
3.5	Texel-Vergleichs-Methode.....	38
3.5.1	Beschreibung der Methode	38
3.5.2	Vor- und Nachteile	40
3.5.3	Mögliche Verbesserung	41
4.	Umsetzung	41
4.1	Eingabe- und Ausgabedaten	41
4.2	Programm Architektur.....	42
5.	Ergebnis.....	43
5.1	Eingestellte Parameter und Schwellwerte	44
5.2	Beispielergebnisse	44
5.3	Quantitative Bewertung.....	50
5.3.1	Hinzufügen neuer Kontroll-Punkte	51
5.3.2	Verwendung gegebener Feature-Punkte.....	51
5.3.3	False Positives	52
5.3.4	Verwerfen von Clustern.....	54
5.3.5	Laufzeit	55
5.4	Qualitative Bewertung und Ursachen der Probleme	57
5.4.1	Fehlerhafte Eingabedaten	58
5.4.2	Fehlende Extrapolation oder unvollständige Eingabe-Daten	58
5.4.3	Diskontinuitäten in der Topologie	59
5.4.4	Verwerfen von Clustern.....	60
5.4.5	Wahl des initialen Quad-Paars	60
5.4.6	Nutzung mehrerer Cluster.....	60
5.4.7	Globales Mapping.....	61
5.4.8	Quads und Texel als elementare Größe	62
5.4.9	Texel-Vergleichs-Methode	62
5.4.10	Thin-Plate-Splines als Mapping-Methode.....	62
5.4.11	Ergänzen neuer Feature-Punkte und Quads.....	63
6.	Zusammenfassung und Ausblick	64
6.1	Zusammenfassung	64
6.2	Ausblick.....	65
6.2.1	Laufzeit	65

6.2.2	Texel-Vergleichs-Methode	65
6.2.3	Extrapolation und Diskontinuitäten	66
6.2.4	Fehlerhafte Eingabedaten	66
Anhang: Ergebnisse		67
Literatur		105

1. Einleitung

Dieser Abschnitt gibt eine kurze Einführung in das Thema dieser Arbeit.

1.1 Einordnung

Fast überall in der echten Welt kann man sich regelmäßig wiederholende Strukturen finden. Das können Muster auf Kleidung, regelmäßig angeordnete Fenster an einem Haus oder die Maschen eines Maschendrahtzauns sein. Sie alle haben gemeinsam, dass sie eine Regelmäßigkeit haben, welche durch die gleichmäßige Kachelung eines oder mehrerer Texturelemente entsteht.

Fotografiert oder filmt man diese Strukturen, so bleiben die Texturelemente keine Kopien voneinander. Stattdessen erscheinen die Texturelemente, durch Belichtungseffekte, Perspektive und Verformung, verzerrt und in unterschiedlichen Helligkeiten. Es liegt auf dem Bild also nur noch ein *regelmäßigkeitsnahes* Muster vor (Vgl. Abbildung 1).



Abbildung 1: Links ein regelmäßiges Muster, rechts ein regelmäßigkeitsnahes Muster.

Ein Thema der Computer Vision ist es, solche regelmäßigkeitsnahen Muster auf zweidimensionalen Bildern automatisch zu erkennen, zu analysieren und zu verstehen und ggf. Rückschlüsse über die zugrundeliegende dreidimensionale Struktur zu ziehen.

Diese Arbeit beschäftigt sich ausschließlich mit Strukturen, welche man in einem Gitter anordnen kann. Ein solches Muster kann durch einen *4-uniformen Graph* beschrieben werden. Zwei Beispiele für Strukturen welche durch 4-uniforme Graphen beschreibbar sind ist in Abbildung 2 zu sehen.

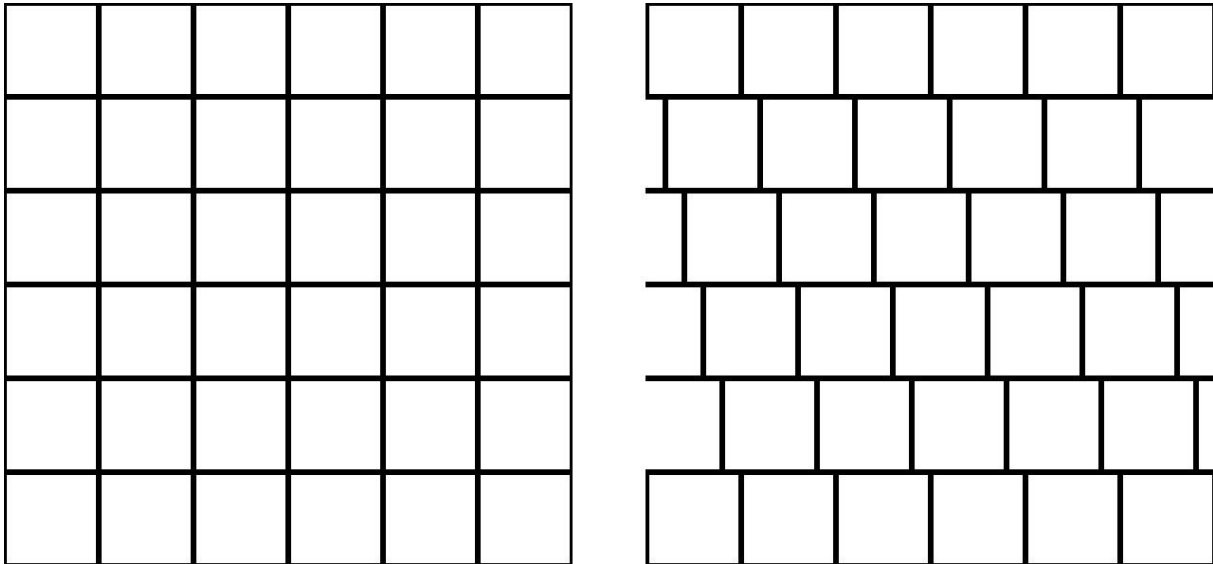


Abbildung 2: Zwei Strukturen welche durch einen 4-uniformer Graph beschreibbar sind. Mit Bildern denen eine solche Struktur zugrunde liegt beschäftigt sich diese Arbeit.

1.2 Motivation

In diesem Abschnitt wird kurz motiviert warum Bilder mit regelmäßigen Strukturen verarbeitet werden? Die potentiellen Anwendungen sind zahlreich, jedoch ist die Bildanalyse stets nur eine Teilaufgabe einer komplexeren Anwendung.

So ist gibt es beispielsweise ein Projekt Virtual Mirror ([1]), welches, wie in Abbildung 3 dargestellt, mittels einer Kamera die Kleidung einer Person aufnimmt und im Spiegelbild andere Kleidung darstellt, als die reale Kleidung.



Abbildung 3: Idee des Virtuellen Spiegels. Eine Kamera nimmt die Person vor dem Spiegel auf und die Software des Spiegels ermöglicht es, die Kleidung, welche man im Spiegelbild trägt, auszutauschen, ohne seine reale Kleidung zu wechseln. Quelle: [1].

Eine Aufgabe der Software des virtuellen Spiegels ist es also, ein gegebenes Bild einer regelmäßigen Struktur in ein anderes Bild der gleichen Struktur und einer anderen Textur umzuwandeln, wie in Abbildung 4 zu sehen.



Abbildung 4: Links: Reale Eingabe (Foto) des virtuellen Spiegels. Rechts: Synthetische Ausgabe des Virtuellen Spiegels, bei dem das originale Muster durch ein neues ersetzt wurde. Quelle: [2]

Ebenso könnte man auf dem Foto von einer Hausfassade die Fenster gegen andere Modelle austauschen indem man nur noch angibt, wie eines der neuen Fenster aussehen soll.

Eine Möglichkeit, diese Aufgabe zu lösen, besteht darin, die Lage und Verzerrung der originalen Texturelemente zu ermitteln, wie in Abbildung 5 zu sehen, um diese Verzerrung später auf eine neue Textur anzuwenden.



Abbildung 5: Eine Möglichkeit die originale Textur (links) in eine andere umzuwandeln, besteht darin zuerst sogenannte Feature-Punkte an markanten ähnlichen Stellen des Bildes zu ermitteln (Mitte). Diese Feature-Punkte können dann in einem weiteren Schritt zur zugrundeliegenden Struktur (rechts) verbunden werden. Quelle: [2], nachbearbeitet.

In diese Arbeit stellt einen Algorithmus vor, welcher aus einem gegebenen Foto einer regelmäßigkeitsnahen Topologie (Abbildung 5, links) und einer gegebenen Menge von Feature-Punkten (Abbildung 5, Mitte) automatisch die zugrundeliegende Topologie und Verzerrung der einzelnen Texturelemente (Abbildung 5, rechts) ermittelt.

1.3 Stand der Technik

Es gibt bereits erste Ansätze zur Analyse von nahe-regelmäßigen Gitter-Strukturen, jedoch noch keinen allumfassenden Ansatz, der alle Bilder problemlos und korrekt analysiert. Einige der Ansätze erfordern zusätzliche menschliche Interaktion während der Analyse, andere arbeiten vollautomatisch.

Das Identifizieren der zugrundeliegenden Topologie und der Verzerrung der einzelnen Texturelemente ist in den hier vorgestellten Ansätzen, oft nur eine Teilaufgabe innerhalb eines größeren Kontexts. Im Folgenden soll kurz gezeigt werden, wie in verschiedenen Ansätzen ausgehend von einem Bild auf die zugrundeliegende Struktur geschlossen wird.

In [3] wird ein halbautomatischer Ansatz vorgestellt. Dabei muss ein Benutzer manuell auf einer grafischen Benutzeroberfläche die Kontroll-Punkte für ein Mapping definieren. Dazu definiert er zuerst drei Kontroll-Punkte, welche die Basis-Vektoren bilden. Daraus extrapoliert das Verfahren weitere Kontroll-Punkte mit gleichem Abstand, ohne die Bilddaten zu beachten. Dann muss der Benutzer die generierten Kontroll-Punkte noch einmal manuell korrigieren. Eine Visualisierung des Verfahrens ist in Abbildung 6 zu sehen.

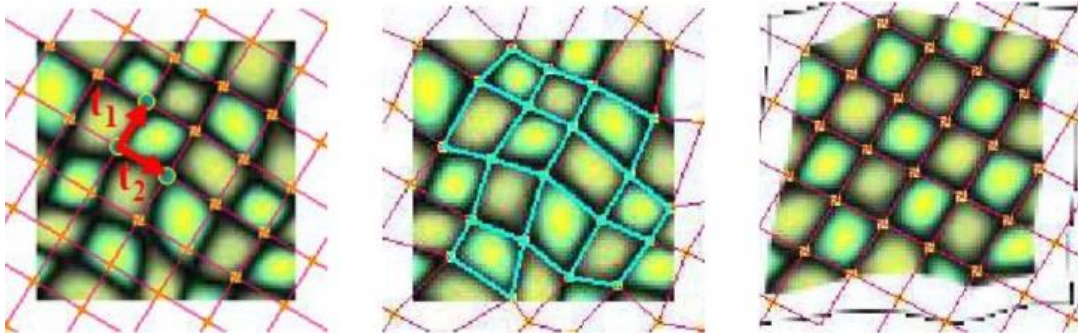


Abbildung 6: Links: Ein Benutzer definiert zwei Basisvektoren $(\mathbf{t}_1, \mathbf{t}_2)$ und der Algorithmus extrapoliert daraus ein erstes Gitter. Mitte: Der Benutzer verschiebt die Kontrollpunkte im Bildraum und der Algorithmus passt das Mapping entsprechend an. Rechts: Das entzerrte Bild im topologischen Koordinatensystem. Quelle: [3].

Hays et al. [4] stellen eine Methode vor, welche einen Mittelweg zwischen globalen und lokalen Ansätzen darstellt. Diese Methode ermittelt iterativ immer mehr Texturelemente und verwendet in jeder Iteration die bereits ermittelten Texturelemente dann in global arbeitenden Algorithmen. Das Verfahren ist in Abbildung 7 dargestellt und genauer beschrieben.

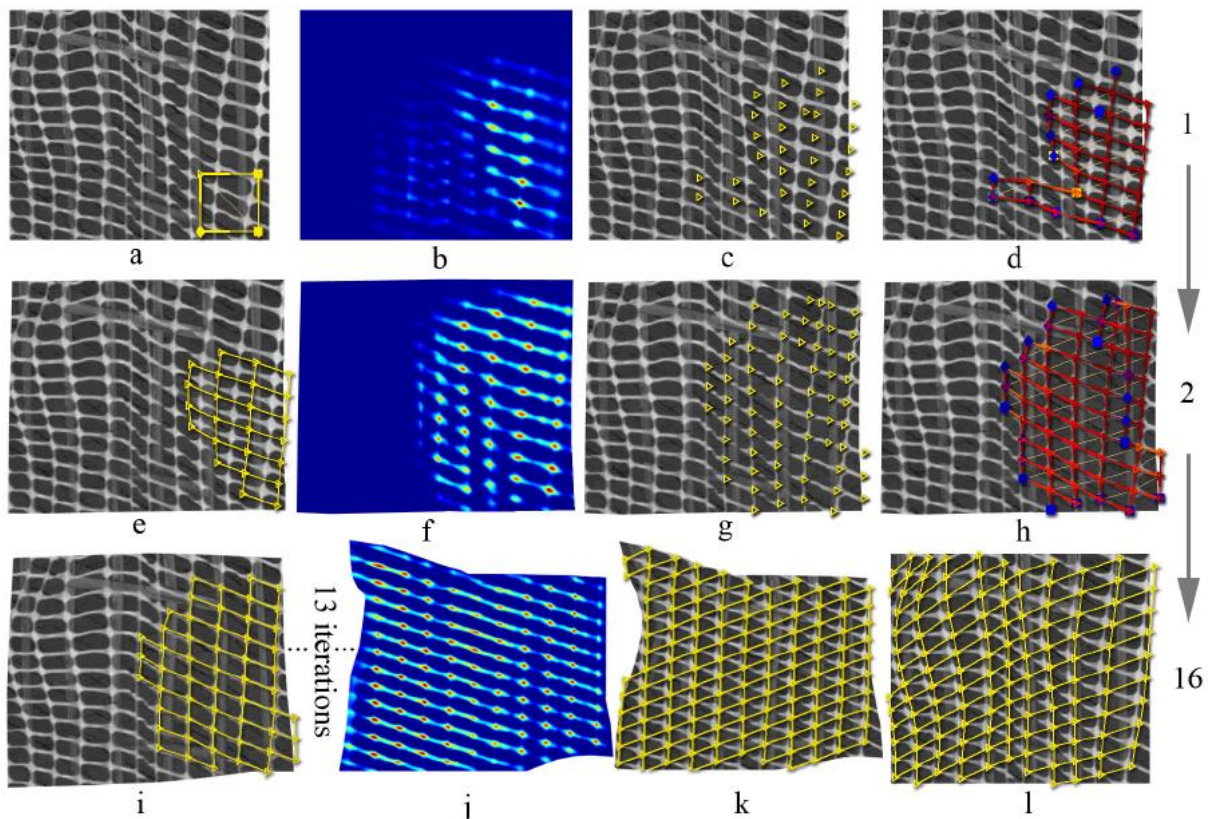


Abbildung 7: Der Ansatz von Hays et al. [4]. Zuerst werden mittels MSER [5] erste potentielle Texturelemente vorgeschlagen, oder falls MSER fehlschlägt wird ein zufälliger Punkt und zufällige Stützvektoren ausgewählt (Bild a). Nach Anwendung normalisierter Kreuzrelation (Bilder b, f und i) werden Maximalwerte mittels der Region Dominance Methode [6] festgestellt (Bilder c und g). Nun wird versucht mittels [7] ein global optimales Gitter (Bilder d und h) auf den Maximalwerten der aktuellen Iteration zu finden. Mittels einiger Heuristiken werden Teile des Gitters wieder verworfen, um nur die potentiellen Texturelemente für die nächste Iteration übrig zu behalten (Bilder e und i). Quelle: [4].

Park et al. [8] formulieren das Problem als „spatial, multitarget tracking problem“. Sie modellieren das gesuchte Gitter als 4-uniformes Markov Random Field¹, und ordnen jedem Knoten seine Position im Bild zu. Des Weiteren werden zwei Funktionen definiert, welche die Ähnlichkeit zum durchschnittlichen Texturelement und die paarweise Kompatibilität zwischen zwei Knoten beschreiben. Mittels Mean Shift Belief Propagation (MSBP) [9] werden die Positionen jedes Knotens iterativ optimiert. Konvergieren die Positionen aller Knoten, so wird das Bild mittels Thin Plate Splines entzerrt. In einer Schleife wird solange das Gitter erweitert und mittels MSBP optimiert bis der Rand des Bildes erreicht ist.

¹ Ein Markov Random Field ist ein Graph bei dem jedem Knoten eine Zufallsvariable zugeordnet ist. Die Zufallsvariable eines Knotens kann dabei, nach speziellen Regeln, von den Zufallsvariablen anderer Knoten abhängen.

In [2] wird ein Ansatz vorgeschlagen, welcher lokal arbeitet und geometrische Beziehungen zwischen Feature-Punkten nutzt, um aus ihnen die gesuchte Struktur zu ermitteln. Dieser Ansatz nutzt einen Cluster ähnlicher Feature-Punkte zur Ermittlung der Struktur.

Nachdem die Feature-Punkte ermittelt und Clustern zugeordnet wurden, wird wie in Abbildung 8 beschrieben vorgegangen um die zugrundeliegende Topologie² zu ermitteln.

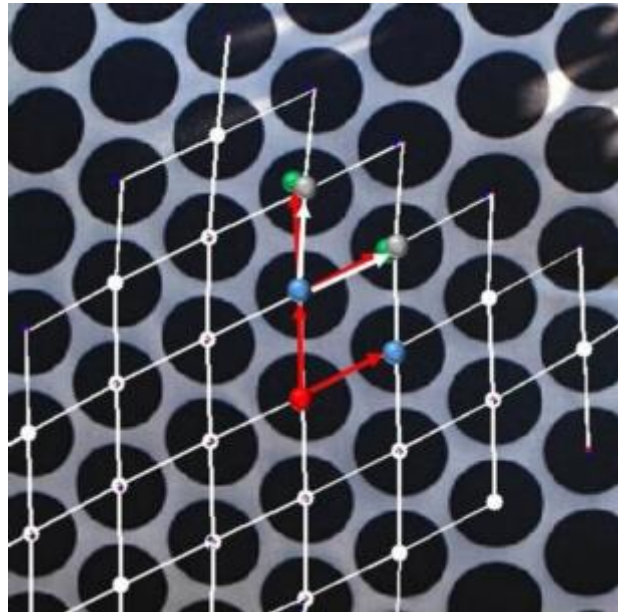


Abbildung 8: Darstellung des Ansatzes aus [2]. Von einem ersten Feature-Punkt (rot) aus werden zwei Basis-Vektoren (rot) ermittelt. Diese Basis-Vektoren zeigen auf zwei weitere Feature-Punkte (blau). Diese „erben“ zuerst die Basis-Vektoren ihres Vorgängers (rote Pfeile) und mithilfe dieser Vektoren werden nun die nächsten Feature-Punkte nahe den grünen Positionen vermutet. Die Feature-Punkte (grau) nahe der vermuteten Position werden als Nachbarn bzw. Nachfolger benutzt. Die Vektoren werden entsprechend der neu gefundenen Punkte aktualisiert (weiße Pfeile). Die gleiche Prozedur wird für alle so gefundenen Feature-Punkte wiederholt und auch in die negative Richtung ausgeführt. Quelle: [2].

Die in dieser Arbeit vorgestellte Methode baut auf [2] auf und erweitert diese.

1.4 Problemstellung & Rahmenbedingungen

Die in dieser Arbeit bearbeitete Problemstellung kann folgendermaßen beschrieben werden.

Gegeben seien ein Bild einer naheregulären Struktur und eine oder mehrere Mengen von ähnlichen Punkten im Bild. Gesucht ist die zugrundeliegende Topologie der Textur.

² In [2] wird das was hier „Topologie“ genannt wird, stattdessen „lattice“ (engl. für Gitter o. Netz).

Dabei sollen die Informationen mehrerer Cluster – ein Cluster ist eine Menge untereinander ähnlicher Feature-Punkte – mit ins Ergebnis einfließen. Denn so können bei den gegebenen Feature-Punkt-Mengen ggf. Fehler durch Redundanz ausgeglichen werden.

Der gesuchte Algorithmus soll später in einem datenbankbasierten Ansatz benutzt werden, weswegen es keine Anforderungen an die Laufzeit gibt. Er muss also nicht in Echtzeit 60 Bilder pro Sekunde analysieren können, sondern darf für einzelne Bilder wesentlich länger brauchen.

Da der Algorithmus in einem datenbankbasierten Ansatz benutzt werden soll, ist auch anzunehmen, dass sehr viele Bilder mit ihm verarbeitet werden sollen. Deshalb soll er vollautomatisch arbeiten und keine neue Anpassung für jedes Bild, oder gar während der Verarbeitung eines Bildes nötig sein.

1.5 Begriffe und Definitionen

1.5.1 Bild- oder Pixelkoordinaten

Als Bildkoordinate oder auch Pixelkoordinate wird im Folgenden die Position im gegebenen Bild bezeichnet. Dabei hat ein Pixel eine Höhe und Breite von 1. Das zugehörige Koordinatensystem wird Bildkoordinatensystem, Pixelkoordinatensystem oder einfach Bild-Raum genannt. Ein Beispiel ist in Abbildung 9 zu sehen.

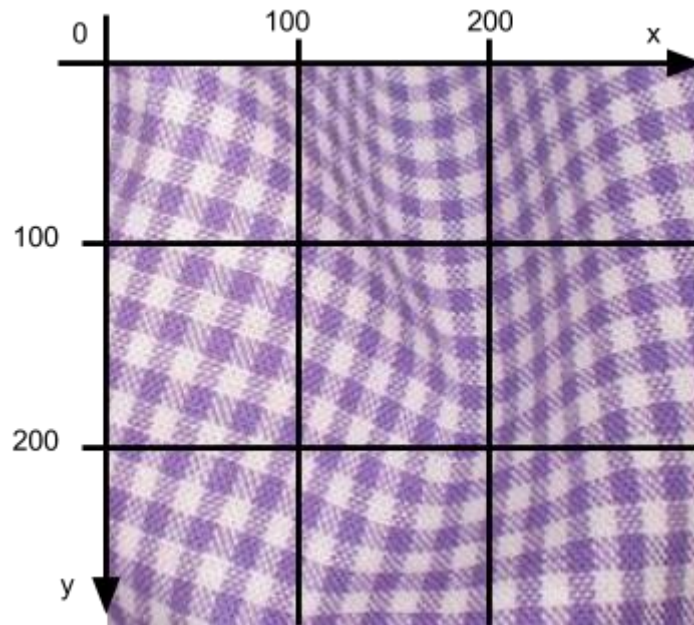


Abbildung 9: Darstellung des Bildkoordinatensystems.

1.5.2 Topologische Koordinaten

Als topologische Koordinate wird im Folgenden die Position in der unverformten Struktur bezeichnet. Eine Wiederholung des zugrundeliegenden Basisrechtecks (später Quad oder Texturelement genannt), hat dabei eine Höhe und Breite von 1. Das zugehörige Koordinatensystem wird topologisches Koordinatensystem genannt. Ein Beispiel ist in Abbildung 10 zu sehen.

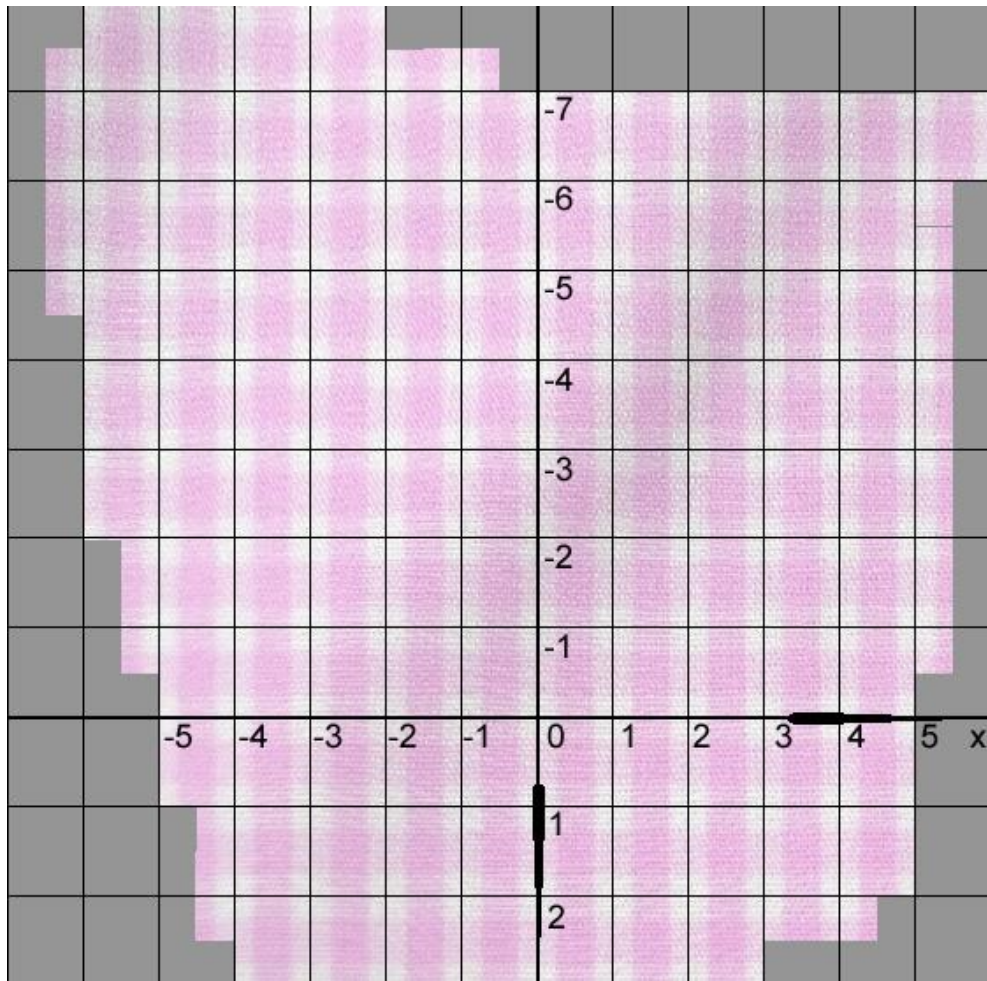


Abbildung 10: Darstellung eines gegebenen Bildes im topologischen Koordinatensystem.

1.5.3 Feature-Punkte

Als Feature-Punkte werden im allgemeinen Bild-Punkte mit herausstechenden Merkmalen bezeichnet. Es gibt viele Verfahren zur Feature-Detektion, welche besondere Bild-Punkte finden. Zu diesen Verfahren gehören u.a. [10], [11] und [12]. Ein Beispiel so detektierter Feature-Punkte ist in Abbildung 5 (Mitte) zu sehen.

Die Bild-Koordinaten der Feature-Punkte werden in dieser Arbeit als gegeben angenommen.

Die Feature-Punkte werden in dieser Arbeit als Vektoren dargestellt. Damit ist, je nach Kontext, der Positions-Vektor des Feature-Punktes in Bildkoordinaten oder topologischen Koordinaten gemeint.

1.5.4 Cluster

Man kann den Feature-Punkten sogenannte Deskriptoren zuordnen, welche die Eigenschaften des Feature-Punktes quantitativ in Form eines Vektors beschreiben. Es gibt einige Verfahren, welche Feature-Punkten Deskriptoren zuordnen. Zu den bekanntesten gehören SIFT ([13]), SURF ([14]), GLOH ([15]) und HOG ([16]). Diese Deskriptoren sind je nach Verfahren invariant gegenüber bestimmten Eigenschaften sein. So ist z.B. SIFT³ invariant gegenüber Skalierung und Rotation, d.h. die Deskriptoren bestimmter Punkte bleiben gleich, solange man ein Bild nur skaliert oder rotiert.

Gefundene Feature-Punkte kann man dann nach ihren Deskriptoren gruppieren, indem man Feature-Punkte zusammen mit ähnlichen Feature-Punkten einer Menge zuordnet. So kann man z.B. mittels Mean Shift Clustering [17] die Feature-Punkte in verschiedene Mengen gruppieren. Die so entstehenden Mengen werden im folgenden Cluster genannt.

Ein Beispiel für Feature-Punkte welche in zwei verschiedene Cluster gruppiert wurden, ist in Abbildung 5 (Mitte) zu sehen. Die Feature-Punkte im Bild sind ihren Clustern entsprechend gefärbt.

Die Cluster gehören in dieser Arbeit zu den gegebenen Daten, auf die der entwickelte Algorithmus aufbaut. Die Deskriptoren der Feature-Punkte werden in dieser Arbeit nicht benutzt, sondern lediglich die Position der Feature-Punkte.

1.5.5 Quad & Texel

Ein Quad bezeichnet ein Rechteck der zugrundeliegenden Topologie der Textur. In topologischen Koordinaten hat es eine Höhe und Breite von jeweils 1.

Ein Texel⁴ bezeichnet die Fläche innerhalb des Quads. Das Quad beschreibt also die Topologie (d.h. die Form und Anordnung), während durch das Texel das Aussehen (d.h. Farbwerte) der vom Quad umschlossenen Rechtecks beschreibt. Ein Beispiel ist in Abbildung 11 zu sehen. Ist ein Texel bekannt, so ist durch seine Position implizit auch das zugehörige Quad definiert. Umgekehrt definiert ein gegebenes Quad auch implizit ein zugehöriges Texel.

³ Kurz für Scale-invariant feature transform, [13]

⁴ Kurz für Texturelement, analog zum Begriff Pixel.

Wird in dieser Arbeit also davon gesprochen, dass ein Quad ermittelt wird, so wird stets gleichzeitig der zugehörige Texel ermittelt und umgekehrt.

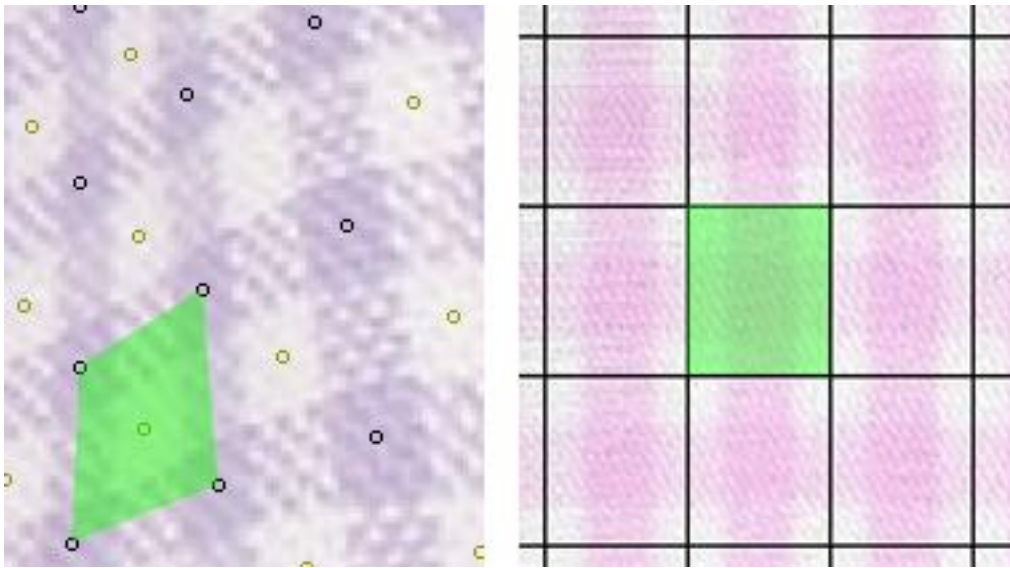


Abbildung 11: Links: Ein Texel (grün unterlegt) im Bild-Koordinatensystem. Rechts: Ein Texel (grün unterlegt) im topologischen Koordinatensystem.

1.5.6 Mapping

Als Mapping wird in dieser Arbeit die Menge von Abbildungen/Funktionen bezeichnet, welche einen Punkt vom Bildraum auf einen Punkt im topologischen Raum abbilden oder umgekehrt.

Die im Folgenden vorgestellten Mappings benutzen sogenannte Kontroll-Punkte, um die Abbildungen zu definieren. Diese Kontroll-Punkte definieren jeweils eine Bildkoordinate \vec{b} und eine topologische Koordinate \vec{t} als Tupel. Das Mapping m sei nun so definiert, dass es die beiden Koordinaten aufeinander abbildet: $m := \{m_{bild \rightarrow topo}, m_{topo \rightarrow bild}\}$ mit $m_{bild \rightarrow topo}(\vec{b}) := \vec{t}$ und $m_{topo \rightarrow bild}(\vec{t}) := \vec{b}$. Als Kurzschreibweise wird dafür im Folgenden auch einfach nur $m(\vec{b})$ und $m(\vec{t})$ geschrieben. Bei der Kurzschreibweise ist aus dem Kontext und der Art des Parameters zu folgern, welche Funktion aus m genutzt wird:

$$m(\vec{x}) := \begin{cases} m_{bild \rightarrow topo}(\vec{x}), & \text{falls } \vec{x} \text{ ist eine Bildkoordinate} \\ m_{topo \rightarrow bild}(\vec{x}), & \text{falls } \vec{x} \text{ ist eine topologische Koordinate} \end{cases}$$

Funktionen, welche durch Kontroll-Punkte definiert werden und dazwischen kontinuierlich interpolieren sind z.B. Splines ([18]).

Hat ein Mapping genügend Kontrollpunkte, so kann man mit ihm das topologische Koordinatensystem im Bildraum darstellen (wie in Abbildung 12), oder das gegebene Foto entzerrt im topologischen Raum darstellen (wie in Abbildung 10).

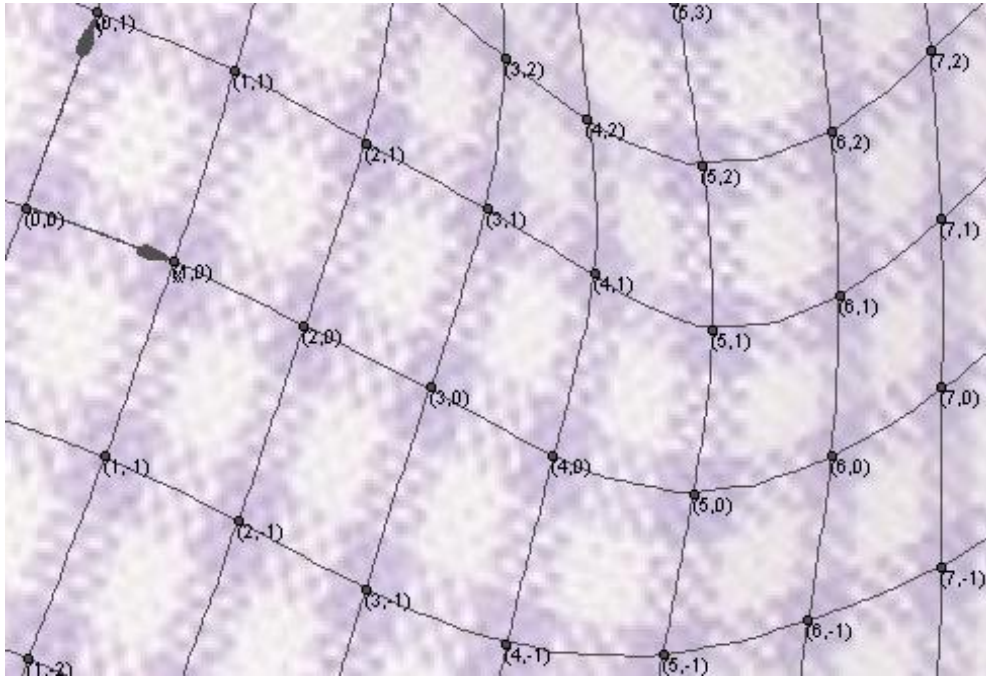


Abbildung 12: Ein topologisches Koordinatensystem kann mithilfe eines Mappings im Bildraum dargestellt werden.

In dieser Arbeit wird das Mapping mittels Thin-Plate-Splines (TPS) [19] realisiert⁵. Thin-Plate-Splines sind eine Interpolations-Methode, welche für gegebene Menge mehrdimensionaler Kontroll-Punkte eine glatte Fläche ermittelt, welche durch jeden der Kontroll-Punkte verläuft (Vgl. Abbildung 13).

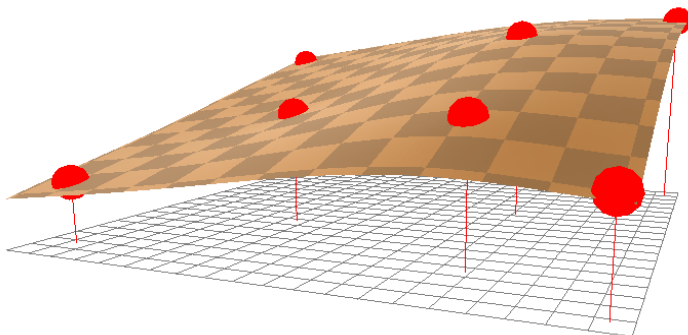


Abbildung 13: Thin-Plate-Splines interpolieren Werte zwischen gegebenen Kontroll-Punkten. Quelle: [20].

⁵ Auch [4] und [2] nutzen Thin-Plate-Splines auf ähnliche Weise.

Um das Mapping $m_{bild \rightarrow topo}$ umzusetzen, werden zwei TPS benutzt. Das erste TPS nutzt als Kontroll-Punkte (b_x, b_y, t_x) und interpoliert dadurch für gegebene Bildkoordinaten $\vec{b} = (b_x, b_y)$ die x -Komponente der topologischen Koordinate $\vec{t} = (t_x, t_y)$. Analog nutzt das zweite TPS (b_x, b_y, t_y) als Kontroll-Punkte. Mit beiden TPS kann nun für gegebene Bildkoordinaten \vec{b} eine topologische Koordinate \vec{t} ermittelt werden.

Analog werden für das Mapping $m_{topo \rightarrow bild}$ zwei weitere TPS benutzt, welche (t_x, t_y, b_x) und (t_x, t_y, b_y) als Kontrollpunkte verwenden.

1.6 Überblick der folgenden Kapitel

Hier soll ein kurzer Überblick über den Aufbau der weiteren Arbeit gegeben werden. Die Abschnitte 2 und 3 beschreiben das im Rahmen der Arbeit entwickelte Verfahren zur Lösung des zuvor beschriebenen Problems. Abschnitt 4 gibt einen Überblick über die konkrete Implementation des Verfahrens in C++. In Abschnitt 5 wird das entwickelte Verfahren an Testdaten evaluiert, Vorteile und Probleme aufgezeigt. Abschnitt 6 rundet die Arbeit mit einer Zusammenfassung ab und gibt einen Ausblick wie man das vorgestellte Verfahren weiter verbessern kann.

2. Lösungsansatz

Dieser Abschnitt gibt einen Überblick über die grobe Herangehensweise und die Grundkonzepte des Algorithmus. Details werden erst im nächsten Abschnitt genauer beschrieben.

2.1 Iterative Arbeitsweise

Der Algorithmus arbeitet iterativ.

Er bearbeitet nacheinander alle Cluster, anstatt alle Cluster auf einmal zu bearbeiten und beginnt nach dem letzten Cluster wieder mit der Bearbeitung des ersten, bis kein Cluster mehr etwas beisteuert. Ein kompletter Durchlauf des Algorithmus auf einem Cluster wird im Folgenden Cluster-Iteration genannt.

Während der Bearbeitung eines Clusters arbeitet der Algorithmus auch iterativ. Dabei propagiert er von bereits erkannten Quads aus, um noch nicht erkannte Quads und deren Texel zu erkennen. Dies passiert solange, bis ein Cluster in der aktuellen Cluster-Iteration nichts mehr beiträgt. Ein kompletter Durchlauf des Algorithmus zur Detektion neuer Quads innerhalb eines Clusters, wird im Folgenden Quad-Iteration genannt.

Der Algorithmus arbeitet also in mehreren Cluster-Iterationen, welche sich wiederum in mehrere Quad-Iterationen unterteilen lassen.

2.2 Ständiges globales Mapping

Während des Algorithmus wird kontinuierlich ein Mapping zwischen dem Bildraum und dem Topologischen Raum berechnet. Dieses Mapping wird mit jeder Quad-Iteration aktualisiert, indem die Eckpunkte der neuen Quads als Kontrollpunkte verwendet werden und damit ein neues Mapping berechnet wird. In diesem Zusammenhang wird später oft davon gesprochen „Feature-Punkte“ als „Kontrollpunkte“ zu benutzen, oder den Feature-Punkten eine topologische Koordinate zuzuordnen. Damit ist gemeint, dass ein Kontrollpunkt erzeugt wird, welcher die Pixel-Koordinaten des Feature-Punktes mit den zugeordneten Topologischen Koordinaten assoziiert und das Mapping mit diesem Kontroll-Punkt verfeinert.

Dieses Mapping gilt stets für das gesamte Bild, auch wenn es nur in der Nähe seiner Kontrollpunkte wirklich gut ist. Es wird in den Quad-Iterationen dazu benutzt, um neue Quads zu finden.

2.3 Quads und Texel als elementare Größe

Ziel ist es, mittels des globalen Mappings Bildpunkte und topologische Koordinaten einander zuzuordnen. Da das Mapping nicht an allen Punkten ausreichend gut ist, werden einige Punkte als „korrekt zugeordnet“ und andere als „falsch zugeordnet“ behandelt. Diese Zuordnung kann sich mit jeder Iteration des Mappings verändern.

Die Zuordnung erfolgt pro Quad bzw. Texel und nicht etwa pro Feature-Punkt, pro Kante oder gar pro Pixel. Es werden also stets *alle* Pixel und Eckpunkte eines Quads entweder als korrekt zugeordnet oder als falsch zugeordnet behandelt.

2.4 Durchschnitts-Textel

Wann immer ein Textel für korrekt befunden wird, wird ein durchschnittliches Textel aus allen korrekten Texteln berechnet. Dieses *Durchschnitts-Textel* wird benutzt, um andere potentielle Textel damit zu vergleichen und festzustellen, ob ein potentielles Textel korrekt oder falsch ist.

2.5 Relativer Fehler

Der Algorithmus vergleicht Textel mit dem Durchschnitts-Textel. Dabei wird zuerst, je nach Vergleichsmethode und je nach zu vergleichendem Textel, ein *absoluter Fehler* ermittelt. Dieser sollte im Intervall $[0, \infty[$ liegen und kleiner sein, je ähnlicher sich die Textel sind.

Um nicht für jedes Bild und jede Vergleichsmethode die Fehler anders behandeln und normieren zu müssen wird ein *relativer Fehler* genutzt. Dazu wird aus den absoluten Fehlern aller korrekten Textel der durchschnittliche absolute Fehler berechnet. Der relative Fehler eines Textels q_i ist dann:

$$\text{relativer Fehler von } q_i = \frac{\text{absoluter Fehler von } q_i}{\text{durchschnittlicher absoluter Fehler}}$$

So haben wir ein normiertes Fehlermaß, welches für durchschnittliche Fehler genau den Wert „1“ annimmt, für geringere Fehler Werte zwischen „0“ und „1“ und für größere Fehler Werte über „1“. Ist der Fehler doppelt so groß wie der durchschnittliche, so nimmt der relative Fehler z.B. den Wert „2“ an.

2.6 Cluster-Offsets

Alle Cluster werden im selben topologischen Koordinaten-System betrachtet. Daher kann jedem Cluster C_i ein Cluster-Offset \vec{d}_i relativ zum ersten Cluster C_0 zugeordnet werden. Dieser gibt an, wie weit die Punkte aus C_i gegenüber den Punkten aus C_0 verschoben sind. Der Cluster-Offset stellt also einen geometrischen Zusammenhang zwischen unterschiedlichen Clustern her.

Natürlich ist so nicht jedem Punkt aus C_0 ein Punkt in C_i zuzuordnen und umgekehrt. Jedoch lassen sich mithilfe dieses Zusammenhangs fehlende Punkte eines Clusters aus vorhandenen Punkten eines anderen Clusters bestimmen.

3. Theorie

In diesem Abschnitt werden einzelnen Schritte des Algorithmus vorgestellt und erläutert. Der Algorithmus lässt sich grob in folgende Schritte einteilen:

- 1) Cluster initialisieren: Der Algorithmus vergleicht die Größe der Cluster miteinander und verwirft Cluster mit stark abweichender Feature-Punkt Anzahl.
- 2) Initialisierung: Der Algorithmus sucht und bestimmt die zwei ersten Quads, um daraus ein Mapping, einen Durchschnitts-Textel und einen relativen Fehler ableiten zu können.
- 3) Cluster-Iterationen: Der Algorithmus versucht für einen Cluster nach dem anderen neue Quads und Textel zu finden. Dabei wird solange versucht für alle Cluster neue Quads und Textel zu finden, bis es auf keinem Cluster mehr gelingt.
- 4) Quad-Iterationen: Die Quad-Iterationen sind Teilschritte der Cluster-Iterationen. In jeder Quad-Iteration wird versucht, neue Quads in der Nachbarschaft der bereits ermittelten herzuleiten. Die Quad-Iterationen werden solange wiederholt, bis sie kein neues Quad mehr ermitteln, dann endet die aktuelle Cluster-Iteration.
- 5) Textel-Vergleichs-Methode: Während der vorherigen Schritte müssen immer wieder Textel miteinander verglichen werden um einen Fehler zu ermitteln. Dazu wird eine eigene Textel-Vergleichs-Methode verwendet.

Diese Teil-Schritte werden in den nachfolgenden Unterabschnitten detaillierter erklärt.

3.1 Cluster initialisieren

Der restliche Algorithmus setzt voraus, dass die gegebenen Feature-Punkte den gegebenen Clustern annähernd richtig zugeordnet wurden. D.h. insbesondere auch das gleiche Punkte eines Quads auch dem gleichen Cluster angehören.

Um sicherzustellen dass der Algorithmus auf den gegebenen Daten gute Ergebnisse erzielt, werden zuerst Cluster verworfen, welche zu stark vom größten Cluster abweichen.

Dazu wird die Größe (d.h. die Anzahl der enthaltenen Feature-Punkte) N_{max} des größten Clusters ermittelt und ein konstanter Faktor $C_{relClusterSize}$ festgelegt. Alle Cluster C_i die weniger Feature-Punkte enthalten als $N_{max} \cdot C_{relClusterSize}$ werden verworfen.

Die verbliebenen Cluster werden neu nummeriert. D.h. bei verbleibenden n Clustern, werden die Cluster im Folgenden C_0, \dots, C_n genannt.

3.2 Initialisierung

Zur Initialisierung werden zuerst zwei initiale Quads gesucht. Sobald die zwei Quads als korrekt festgelegt wurden, kann mit deren Hilfe das Mapping, das Durchschnitts-Textel und der durchschnittliche absolute Fehler initialisiert werden.

Dazu werden die folgenden Aktionen für jeden Feature-Punkt p_i des ersten Clusters ausgeführt:

- 1) Bestimmen des ersten Quad-Kandidaten: Der gewählte Feature-Punkt p_i wird benutzt, um daraus einen Kandidaten für ein erstes Quad zu konstruieren, welches p_i als Eckpunkt benutzt. Sobald drei Eckpunkte festgelegt wurden, wird ein Mapping mit diesen drei Kontrollpunkten erzeugt. Das Mapping wird mit dem vierten Eckpunkt noch verfeinert.
- 2) Bestimmen des zweiten Quad-Kandidaten: Das zuvor erstellte Mapping wird nun benutzt, um einen Kandidaten für ein zweites Quad zu bestimmen, welches auch p_i als Eckpunkt benutzt. Auch hier wird das Mapping verfeinert. Außerdem werden nun mit diesen zwei Kandidaten ein Durchschnitts-Textel und ein durchschnittlicher absoluter Fehler bestimmt.
- 3) Plausibilität der Quad-Kandidaten überprüfen: Es wird mittels einiger Heuristiken überprüft, ob die zuvor ermittelten zwei Quads, überhaupt korrekt sein können.
- 4) Bewertung der Ebenheit: Wurden die zwei Quads zuvor als korrekt bewertet, werden sie nun danach bewertet wie „eben“ ihre Umgebung ist.

Das Quad-Paar, welches als korrekt bewertet wurde und die ebenste Umgebung hat, wird endgültig als initiales Quad-Paar festgelegt und das zugehörige Mapping, Durchschnitts-Textel und durchschnittlicher absoluter Fehler werden beibehalten.

Wie genau die obigen Schritte ablaufen, wird in den folgenden Unterabschnitten erläutert.

3.2.1 Bestimmen des ersten Quad-Kandidaten

Es soll nun für jeden Feature- Punkt \vec{p}_i ein potentiell erstes Quad $q_{1,i}$ konstruiert werden und ein zugehöriges potentiell Mapping m_i ermittelt werden. Dazu wird wie folgt vorgegangen.

Dem Feature-Punkt \vec{p}_i , aus Cluster C_0 , aus dem das erste Quad konstruiert werden soll werden die topologischen Koordinaten $(0,0)$ zugeordnet⁶, weswegen er im Folgenden auch $\vec{p}_{i,(0,0)}$ genannt wird.

Es wird nun aus $C_0 \setminus \{p_{i,(0,0)}\}$ der Feature-Punkt $\vec{p}_{i,(1,0)}$ gesucht, welcher $\vec{p}_{i,(0,0)}$ am nächsten⁷ ist. Ihm werden die topologischen Koordinaten $(1,0)$ zugeordnet. Es sei $\vec{v}_{i,1}$ der erste Stützvektor: $\vec{v}_{i,1} := \vec{p}_{i,(1,0)} - \vec{p}_{i,(0,0)}$.

Aus $C_0 \setminus \{p_{i,(0,0)}, p_{i,(1,0)}\}$ werden nun die sieben Feature-Punkte $M_{i,potential(0,1)}$ gesucht, welche am nächsten an $p_{i,(0,0)}$ liegen. Für alle sieben Punkte $\vec{p}_{i,j,(0,1)} \in M_{i,potential(0,1)}$ wird nun jeweils ein potentieller zweiter Stützvektor $\vec{v}_{i,j,2}$ definiert: $\vec{v}_{i,j,2} := \vec{p}_{i,j,(0,1)} - \vec{p}_{i,(0,0)}$. Aus den sieben Punkten wird jenem Feature-Punkt $\vec{p}_{i,(0,1)}$ die topologische Koordinate $(0,1)$ zugeordnet, dessen potentieller zweiter Stützvektor am rechtwinkligsten zu $\vec{v}_{i,1}$ steht.⁸

Die ermittelten drei Kontroll-Punkte definieren nun ein Mapping. Dieses wird benutzt, um den vierten Eckpunkt $p_{i,(1,1)}$ aus $C_0 \setminus \{p_{i,(0,0)}, p_{i,(1,0)}, p_{i,(0,1)}\}$ zu finden, welcher der topologischen Koordinate $(1,1)$ am nächsten⁹ liegt.

⁶ Zur Erinnerung: „Einem Feature-Punkt p die Koordinaten t zuordnen“ heißt: Das Mapping um einen Kontrollpunkt, welcher die Pixel-Koordinaten von p mit t assoziiert zu ergänzen. Details sind in Abschnit 2.2 zu finden.

⁷ Zu diesem Zeitpunkt wird der Abstand noch im Pixel-Koordinaten-System gemessen, da noch nicht genügend Kontrollpunkte vorhanden sind, um ein topologisches Koordinatensystem aufzuspannen.

⁸ In [2] werden auf ähnliche Art und Weise zwei Stützvektoren konstruiert. Dort werden jedoch sowohl $p_{(1,0)}$ als auch $p_{(0,1)}$ aus den nächsten acht Punkten gewählt und die rechtwinkligste Kombination benutzt.

⁹ Hier wird der Abstand erstmals im topologischen Koordinaten-System gemessen, da nun ein Mapping vorliegt.

Mit diesem vierten Kontrollpunkt wird nun das Mapping m_i verfeinert und mit dem neuen Mapping ein potentielles erstes Quad q_i definiert.

3.2.2 Bestimmen des zweiten Quad-Kandidaten

Es soll nun noch für jeden Feature-Punkt \vec{p}_i ein potentielles zweites Quad $q_{2,i}$ konstruiert werden und das zugehörige potentielle Mapping m_i verfeinert werden. Dazu wird wie folgt vorgegangen.

Es wird das Mapping m_i benutzt, um den Feature-Punkt $p_{i,(-1,0)}$ zu finden, welcher am nächsten an den topologischen Koordinaten $(-1, 0)$ ist. Der Feature-Punkt wird $(-1, 0)$ zugeordnet und das Mapping entsprechend aktualisiert.

Analog wird nun für die topologische Koordinate $(0, -1)$ und dann für $(-1, -1)$ vorgegangen, um $p_{i,(0,-1)}$ und $p_{i,(-1,-1)}$ zu ermitteln.

Mithilfe des daraus resultierenden Mappings m_i wird nun das zweite Quad konstruiert. Mittels einer beliebigen Texel-Vergleichs-Methode¹⁰ wird nun das Texel des zweiten Quads mit dem Texel des ersten Quads verglichen und ein absoluter Fehler ermittelt. Dieser absolute Fehler dient als initialer durchschnittlicher absoluter Fehler. Der relative Fehler des zweiten Texels ist daher immer „1“. Das Durchschnitts-Texel ist jetzt das Mittel der ersten beiden Texel.

Ein Beispiel, wie die ersten beiden Quads aussehen könnten, ist in Abbildung 14 zu sehen.

¹⁰ Eine konkrete Texel-Vergleichs-Methode wird später im Abschnitt 3.5 vorgestellt.

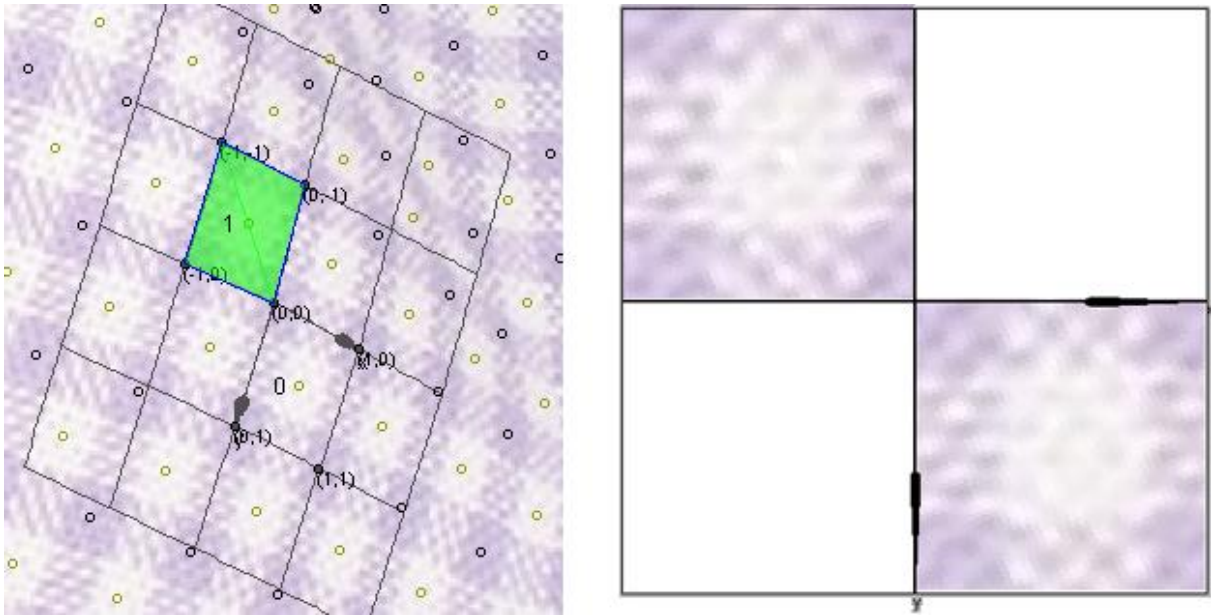


Abbildung 14: Die potentiellen ersten zwei Quads: Links: Die potentiellen ersten zwei Quads im Pixel-Koordinaten-System. Das zweite Texel ist grün unterlegt. Das Mapping ist als graue Linien dargestellt. Rechts: Die ersten zwei Texel im topologischen Koordinaten-System.

3.2.3 Plausibilität der Quad-Kandidaten überprüfen

Da es keine Vergleichswerte gibt, ob die ermittelten zwei Quad-Kandidaten jedes Feature-Punktes p_i korrekt sind oder nicht, werden einige Heuristiken angewendet, um eine Vermutung darüber anzustellen, ob es sich um korrekte Quads handeln kann. Diese Heuristiken wurden aus Beobachtungen und Erfahrungswerten hergeleitet, um falsche initiale Quads zu vermeiden. Diese sollen im Nachfolgenden erklärt und an einigen Bildern illustriert werden.

Annahme 1: Die zugeordneten Feature-Punkte sollten paarweise verschieden sein. Es ist möglich, dass die Eckpunkte des zweiten Quads gleichzeitig Eckpunkte des ersten Quads sind, oder untereinander identisch. Ist dies bei einem anderen Punkt als $p_{1,(0,0)}$ der Fall, so handelt es sich definitiv um ein falsches Quad-Paar.

Annahme 2: In den Quad-Kandidaten sollte kein weiterer Feature-Punkt desselben Clusters liegen. Liegt innerhalb des Kandidaten fürs erste Quad oder innerhalb des Kandidaten fürs zweite Quad ein weiterer Feature-Punkt des gleichen Clusters, so kann dies zwei Ursachen haben. Entweder ist Feature-Punkte verkehrt, oder der Quad-Kandidat. In diesem Fall wird der Einfachheit halber der Quad-Kandidat als falsch angenommen, um Inkonsistenzen zu vermeiden. Ein Quad-Paar, welches diese Annahme verletzt, ist in Abbildung 15 dargestellt.

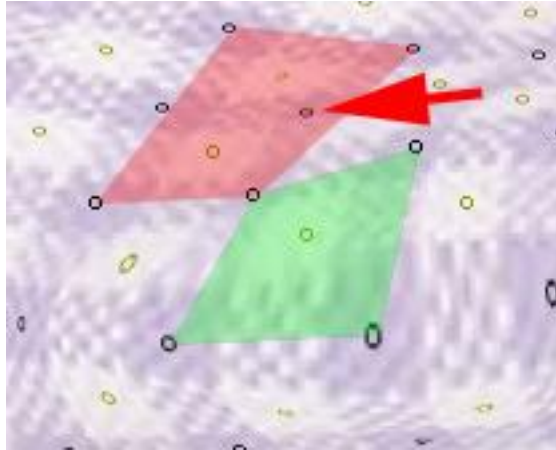


Abbildung 15: Hier wurde zwar der Kandidat fürs erste Quad (grün unterlegt) korrekt erkannt, der Kandidat fürs zweite Quad (rot) ist jedoch falsch, weil er zu groß ist. Der Algorithmus erkennt das, da ein Feature-Punkt desselben Clusters (roter Pfeil) innerhalb des zweiten Kandidaten liegt (Annahme 2 ist verletzt).

Annahme 3: Exakt ein Feature-Punkt jedes anderen Clusters muss in jedem Quad-Kandidaten liegen. Angenommen die Feature-Punkte sind korrekt und vollständig, dann muss jedes Quad exakt einen Feature-Punkt aller anderen Cluster enthalten. Quad-Kandidaten welche diese Annahme nicht erfüllen werden als falsch angenommen. Siehe auch Abbildung 16.

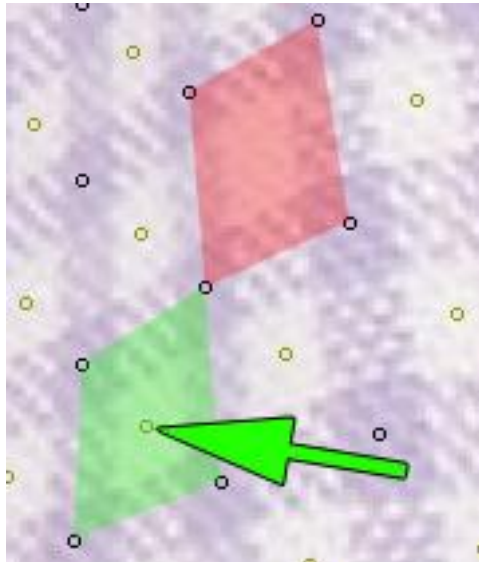


Abbildung 16: Der erste Quad-Kandidat (grün unterlegt) ist in Ordnung. Der zweite Quad-Kandidat ist zwar eigentlich auch korrekt, jedoch fehlt ein Punkt des zweiten Clusters darin (Annahme 3 ist verletzt). Daher wird dieses Quad-Paar als initiales Paar abgelehnt.

Annahme 4: Cluster-Offsets beider Quad-Kandidaten sind etwa gleich. Zum besseren Verständnis siehe Abbildung 17.

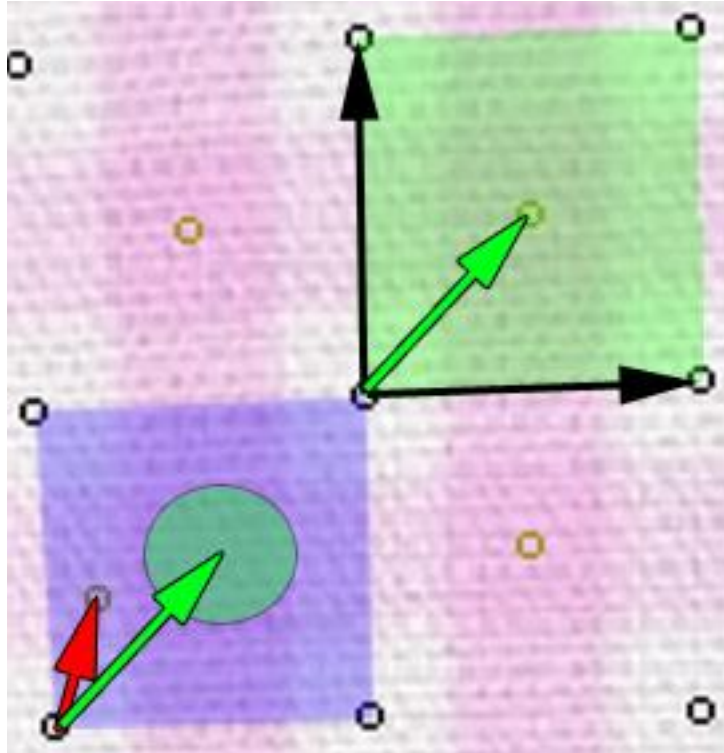


Abbildung 17: Im grünen Quad-Kandidaten hat der zweite Cluster einen Offset \vec{v}_1 (grüner Pfeil). Im blauen Quad-Kandidaten hat der zweite Cluster einen Offset \vec{v}_2 (roter Pfeil). Ist die Differenz $|\vec{v}_1 - \vec{v}_2|$ im topologischen Koordinatensystem zu groß, so gilt das Quad-Paar als falsch. Der grüne Kreis stellt die Fehler-Toleranz dar. Läge der Feature-Punkt des blauen Quad-Kandidaten darin, wäre die Differenz $|\vec{v}_1 - \vec{v}_2|$ klein genug und das Quad-Paar würde akzeptiert werden. Da der Feature-Punkt außerhalb liegt, wird das Quad-Paar nicht akzeptiert.

Nur Quad-Paare welche alle Annahmen erfüllen, kommen in die engere Auswahl. Alle anderen werden als initiale Quad-Paare verworfen.

3.2.4 Bewertung der Ebenheit

Wird ein kandidierendes Quad-Paar als richtig kategorisiert, so wird bewertet wie „eben“ seine Umgebung ist. Dazu wird das zum Quad-Paar gehörige Mapping benutzt, um festzustellen wie viele Feature-Punkte des ersten Clusters im topologischen Koordinatensystem in einer Umgebung um ganzzahlige Koordinaten liegen. Je mehr Feature-Punkt in der Nähe ganzzahliger Koordinaten liegen, desto „ebener“ wird die Umgebung des Quad-Paars bewertet. Siehe Abbildung 18. Diese Methode wird in ähnlicher Weise auch von [8] zur Initialisierung eines Gitters genutzt.

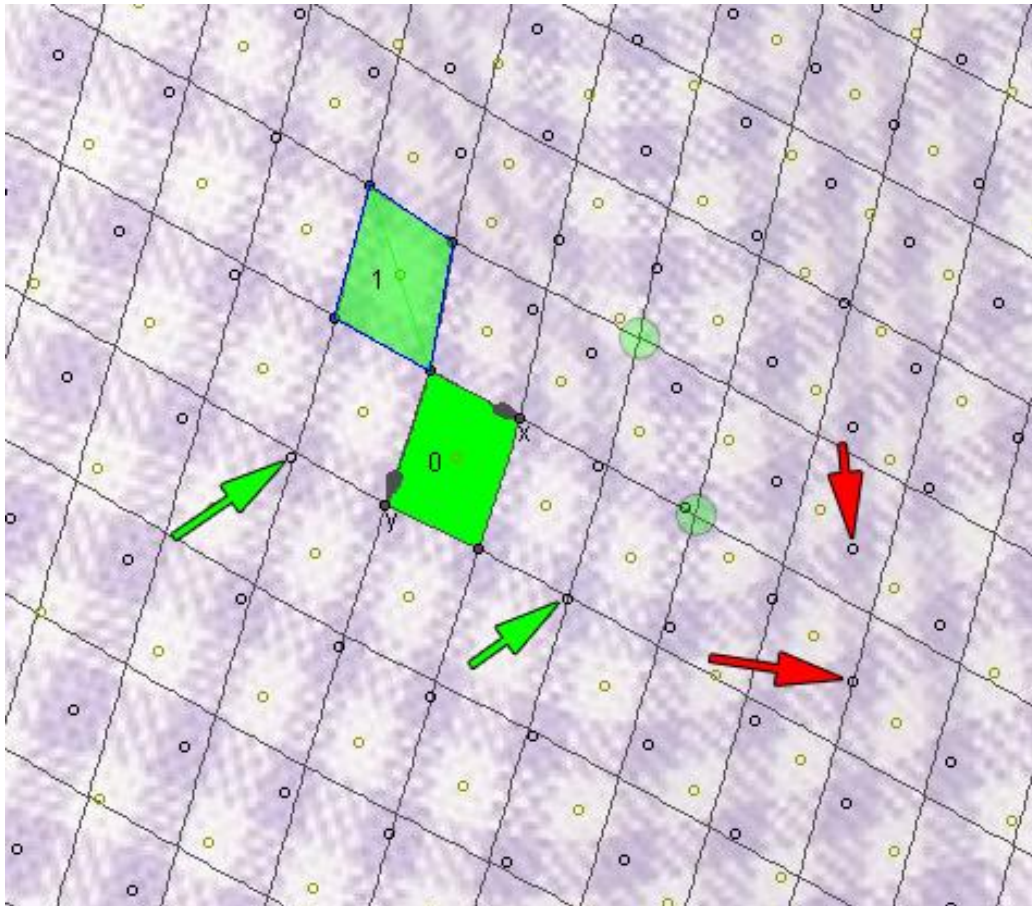


Abbildung 18: Durch das Quad-Paar (grüne Vierecke) wird ein Mapping bestimmt. Man kann nun jeder Bildschirmkoordinate eine topologische Koordinate zuordnen. Ganzzahlige topologische Koordinaten sind als graues Gitter dargestellt. Liegen Feature-Punkte in der Nähe (z.B. grüne Kreise) von ganzzahligen topologischen Koordinaten, so wie z.B. die mit grünen Pfeilen markierten, dann erhöhen sie die Ebenheits-Bewertung des Quad-Paares. Liegen Feature-Punkte zu weit von ganzzahligen Koordinaten entfernt (z.B. rote Pfeile), so erhöhen sie die Ebenheits-Bewertung nicht.

3.2.5 Auswahl eines Quad-Paares

Nachdem alle kandidierenden korrekten Quad-Paare eine Ebenheits-Bewertung haben, wird das Quad-Paar mit der höchsten Bewertung als initiales Paar benutzt. Diese Methode wird in ähnlicher Weise auch von [8] zur Initialisierung eines Gitters genutzt. Die ermittelten Informationen über alle anderen Paare, wie Mappings, Fehler und Durchschnitts-Textel, werden verworfen. Siehe auch Abbildung 19.

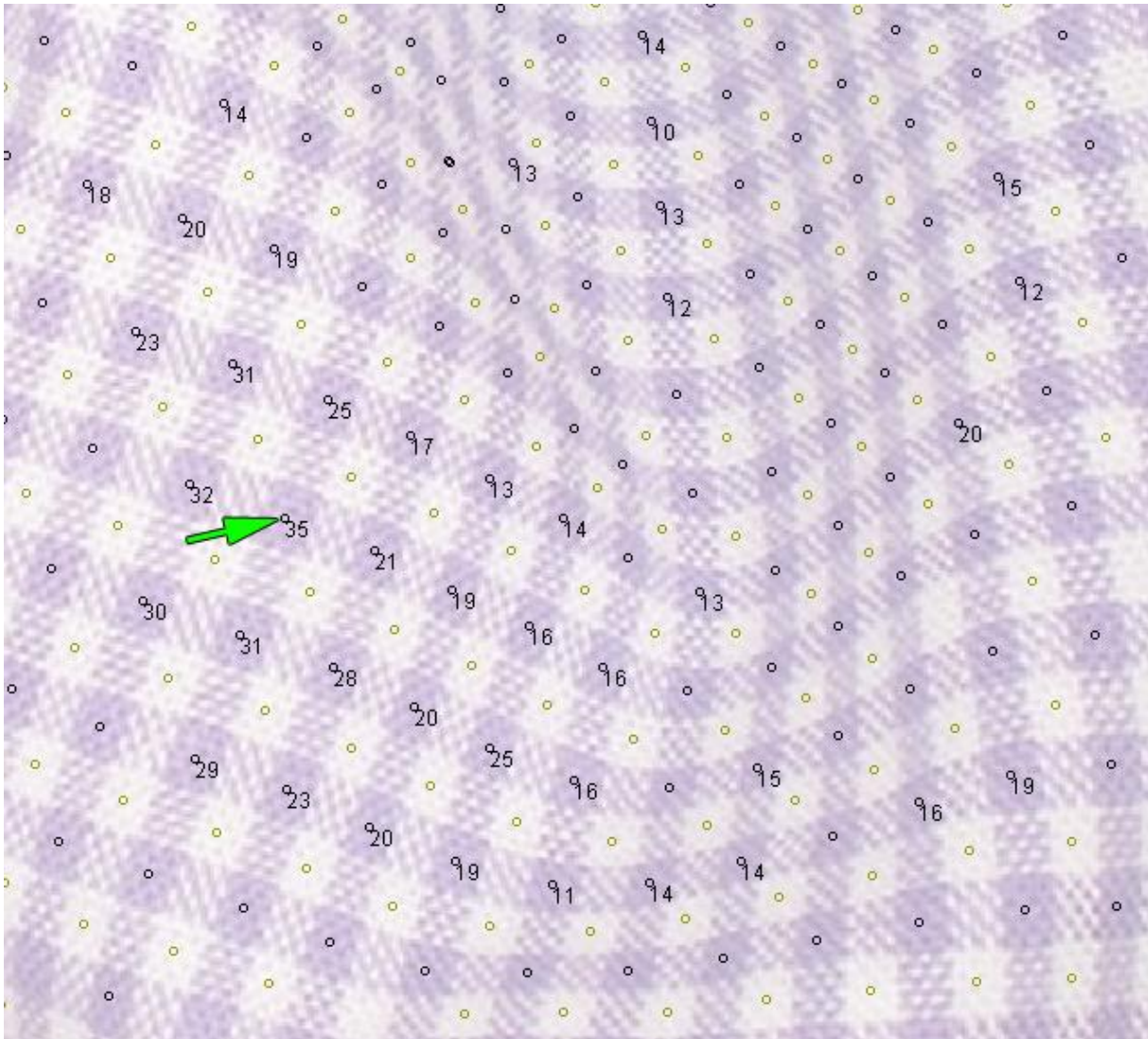


Abbildung 19: Im Beispielbild wurden alle Feature-Punkte mit der Ebenheits-Bewertung beschriftet, welche ihr Quad-Paar hatte, als es mit den beschrifteten Feature-Punkten als Punkt bei (0,0) generiert wurde. Das zum markierten Feature-Punkt gehörige Quad-Paar wird in diesem Beispiel das initiale Quad-Paar.

3.3 Cluster-Iteration

Nachdem der Algorithmus in der Initialisierung die ersten zwei Quads bestimmt hat, beginnt er mit den Cluster-Iterationen. Jede Cluster-Iteration versucht, neue Quads auf einem Cluster zu finden.

Dabei werden zuerst einige Aktionen einmalig pro Cluster-Iteration ausgeführt. Diese Aktionen werden in diesem Abschnitt erklärt. Danach werden für jede Cluster-Iteration solange Quad-Iterationen ausgeführt, bis die Quad-Iterationen keine neuen Quads mehr hinzufügen. Die Details der Quad-Iterationen werden erst im Abschnitt 3.4 genauer erklärt.

Da einige der hier erklärten Aktionen bei der ersten Iteration keinen Effekt haben, oder nicht besonders intuitiv sind, stelle man sich in diesen Fällen beispielhaft vor, dass es sich bereits um die zweite Cluster-Iteration handelt. Zum besseren Verständnis wurden also bereits einige Quads des ersten Clusters akzeptiert und wir befinden uns in der zweiten Cluster-Iteration die auf dem zweiten Cluster durchgeführt wird.

3.3.1 Cluster-Offset initialisieren

Dieser Teilschritt ordnet dem Cluster C_i der aktuellen Cluster-Iteration einmalig einen Cluster-Offset \vec{o}_i relativ zum ersten Cluster C_0 zu (vgl. Abschnitt 2.6). Dieser Teilschritt wird nur einmal für jeden Cluster ausgeführt und bei folgenden Cluster-Iterationen auf Clustern mit einem Offset übersprungen.

Dem ersten Cluster C_0 wird der Nullvektor als Offset zugeordnet: $\vec{o}_0 := \vec{0} = (0, 0)$. Punkten aus diesem Cluster werden stets ganzzahlige topologische Koordinaten zugeordnet.

Für einen anderen Cluster C_i wird die Menge der validierten Quads aller anderen Cluster $Q_{All \setminus i}$ hergenommen. Für jedes dieser Quads $q_j \in Q_{All \setminus i}$, welches genau einen Feature-Punkt $\vec{p}_j = (p_{j,x}, p_{j,y})$ aus C_i enthält wird ein „Teil-Offset“ $\vec{o}_{j,i}$ ermittelt. Da Cluster C_0 stets ganzzahlige topologische Koordinaten hat, wird dieser „Teil-Offset“ definiert als die Verschiebung zur abgerundeten ganzzahligen Koordinate des Feature-Punktes \vec{p}_j :

$$\vec{o}_{j,i} := (p_{j,x} - \lfloor p_{j,x} \rfloor, p_{j,y} - \lfloor p_{j,y} \rfloor)$$

So berechnet sich beispielsweise der „Teil-Offset“ in topologischen Koordinaten für den Feature-Punktes $\vec{p}_{42} = (5.673, 8.322)$ aus Cluster C_3 folgendermaßen:

$$\begin{aligned} \vec{o}_{42,3} &:= (5.673 - \lfloor 5.673 \rfloor, 8.322 - \lfloor 8.322 \rfloor) \\ &= (5.673 - 5, 8.322 - 8) \\ &= (0.673, 0.322) \end{aligned}$$

Der Offset \vec{o}_i von Cluster C_i wird letztlich definiert als das arithmetische Mittel aller „Teil-Offsets“.

3.3.2 Akzeptiere Feature-Punkte in fertigen Quads

In diesem Schritt werden bereits akzeptierte Quads anderer Cluster benutzt, um Feature-Punkte des aktuellen Clusters zu validieren oder hinzuzufügen.

Dazu wird für alle akzeptierten Quads anderer Cluster geprüft, ob diese Feature-Punkte des aktuellen Clusters enthalten. Ist dabei in einem Quad kein Feature-Punkt enthalten, wird dieser mithilfe des Cluster-Offsets an der korrekten Koordinate erzeugt und akzeptiert (vgl. Abbildung 20). Ist dabei in einem Quad genau ein Feature-Punkt enthalten, so wird dieser als korrekt akzeptiert. Befindet sich dabei in einem Quad mehr als ein Feature-Punkt, so werden diese verworfen und ein neuer erzeugt und akzeptiert, als wäre keiner enthalten gewesen.

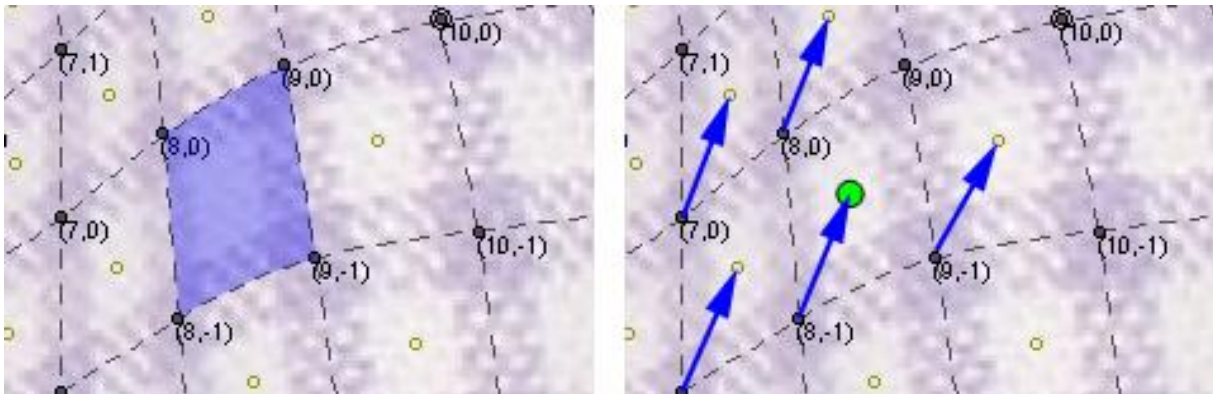


Abbildung 20: Links ist ein akzeptiertes Quad (blau) des ersten Clusters, in dem jedoch ein Feature-Punkt des zweiten Clusters fehlt. Rechts wird der Cluster-Offset des zweiten Clusters (blaue Pfeile) benutzt, um einen neuen Feature-Punkt (grüner Kreis) innerhalb des akzeptierten Quads zu erzeugen.

3.3.3 Akzeptiere neu entstandener Quads

Wurden alle 4 Eckpunkte eines Quads des aktuellen Clusters akzeptiert, das Quad selbst jedoch noch nicht akzeptiert, so wird dieses Quad in diesem Schritt akzeptiert. Vgl. Abbildung 21.

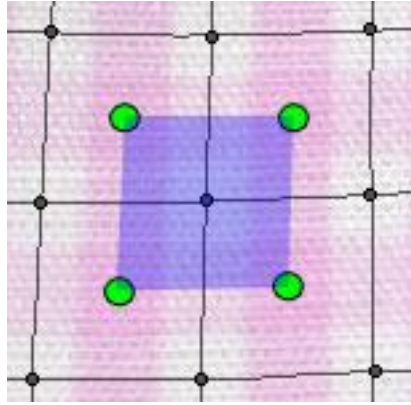


Abbildung 21: Wurden alle vier Eckpunkte (Grün) eines Quads als richtig akzeptiert, so wird auch das Quad selbst (blau) als richtig akzeptiert.

3.3.4 Führe Quad-Iterationen aus

Nachdem eine Cluster-Iteration durch die (zuvor erklärten) Aktionen initialisiert wurde, werden solange Quad-Iterationen durchgeführt, bis eine Quad-Iteration kein weiteres Quad als richtig akzeptiert. Erst dann endet die Cluster-Iteration und die Cluster-Iteration auf dem nächsten Cluster kann beginnen. Die Quad-Iterationen werden im nächsten Abschnitt detailliert erklärt.

3.4 Quad-Iteration

Die Quad-Iterationen sind Teilschritte der Cluster-Iterationen. In jeder Quad-Iteration wird versucht neue Quads in der Nachbarschaft der bereits ermittelten herzuleiten.

In einer Quad-Iteration werden ausschließlich die Feature-Punkte des aktuellen Clusters betrachtet. Auf diesen werden die nachfolgend erklärten Teilschritte ausgeführt, um neue Quads zu finden.

3.4.1 Bestimme Kontroll-Punkt-Kandidaten

Für jeden bereits validierten Feature-Punkt \vec{p}_i wird in der direkten Nachbarschaft nach neuen Kontroll-Punkt-Kandidaten gesucht.

Dazu werden die topologischen Koordinaten \vec{t}_i von \vec{p}_i mittels des Mappings bestimmt: $\vec{t}_i = m_{\text{bild} \rightarrow \text{topo}}(\vec{p}_i)$. Zu diesen topologischen Koordinaten \vec{t}_i wird einer der vier Vektoren $(1, 0)$, $(0, 1)$, $(-1, 0)$ oder $(0, -1)$ addiert. Die vier daraus entstehenden Vektoren $\vec{a}_{i,j}$ geben

die erwarteten Positionen der benachbarten vier Feature-Punkte in topologischen Koordinaten an. Nun werden im topologischen Koordinaten-System die jeweils nächstgelegenen Feature-Punkte als Kandidaten für die vier Koordinaten $\vec{a}_{i,j}$ gewählt.

3.4.2 Verwerfen ungeeigneter Kontroll-Punkt-Kandidaten

In diesem Schritt werden einige der Kontroll-Punkt-Kandidaten für die aktuelle Quad-Iteration wieder verworfen. Dabei werden sie nicht komplett gelöscht, sondern lediglich für diese Quad-Iteration aus der Menge der Kandidaten entfernt. Dazu werden nacheinander folgende Prüf-Schritte für alle Kandidaten durchgeführt.

Falls ein Kandidat bereits zuvor als richtig akzeptiert wurde, so wird er nicht als Kandidat benötigt und wird daher verworfen.

Falls ein Kandidat zu weit von seiner erwarteten topologischen Koordinate $\vec{a}_{i,j}$ entfernt ist, die Distanz also einen Schwellwert überschreitet, dann wird der Kandidat verworfen.

Falls einem Kandidaten mehrere verschiedene topologische Koordinaten zugewiesen wurden dann wird der Kandidat verworfen.

Falls mehreren verschiedenen Kandidaten die gleiche topologische Koordinate zugewiesen wurde, dann werden beide Kandidaten verworfen.

Ein Beispiel-Ergebnis der ersten beiden Schritte der Quad-Iteration ist in Abbildung 22 zu sehen.

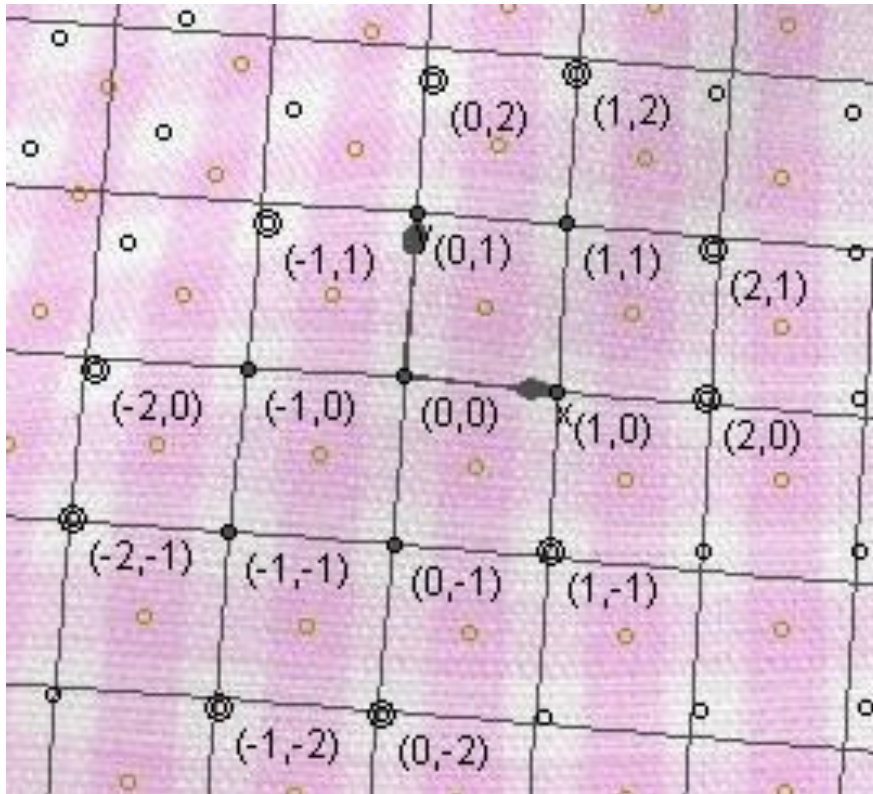


Abbildung 22: Aus bereits akzeptierten Kontroll-Punkten (ausgefüllte kleine Kreise, z.B. (1,1)), werden mittels des Mappings im topologischen Koordinatensystem (graues Gitter) neue Kontroll-Punkt-Kandidaten hergeleitet (doppelte konzentrische Kreise). Ungeeignete Kontroll-Punkt-Kandidaten werden dabei verworfen. So ist z.B. der Punkt (0,0) ein Kandidat, welcher wieder verworfen wird, da er bereits zuvor als richtig akzeptiert wurde. Im Bild ist die erste Quad-Iteration zu sehen, welche nur die bereits akzeptierten Punkte des initialen Quad-Paares nutzt.

3.4.3 Validierung der Quad-Kandidaten

In diesem Schritt werden aus den akzeptierten Kontroll-Punkten und den zuvor ermittelten Kontroll-Punkt-Kandidaten neue Quad-Kandidaten ermittelt. Diese werden dann validiert und entweder akzeptiert oder verworfen.

Dazu wird ein temporäres Mapping m_{temp} verwendet, welches sowohl die schon akzeptierten Kontroll-Punkte, als auch die übrigen Kontroll-Punkt-Kandidaten als Kontroll-Punkte benutzt. Vgl. Abbildung 23.

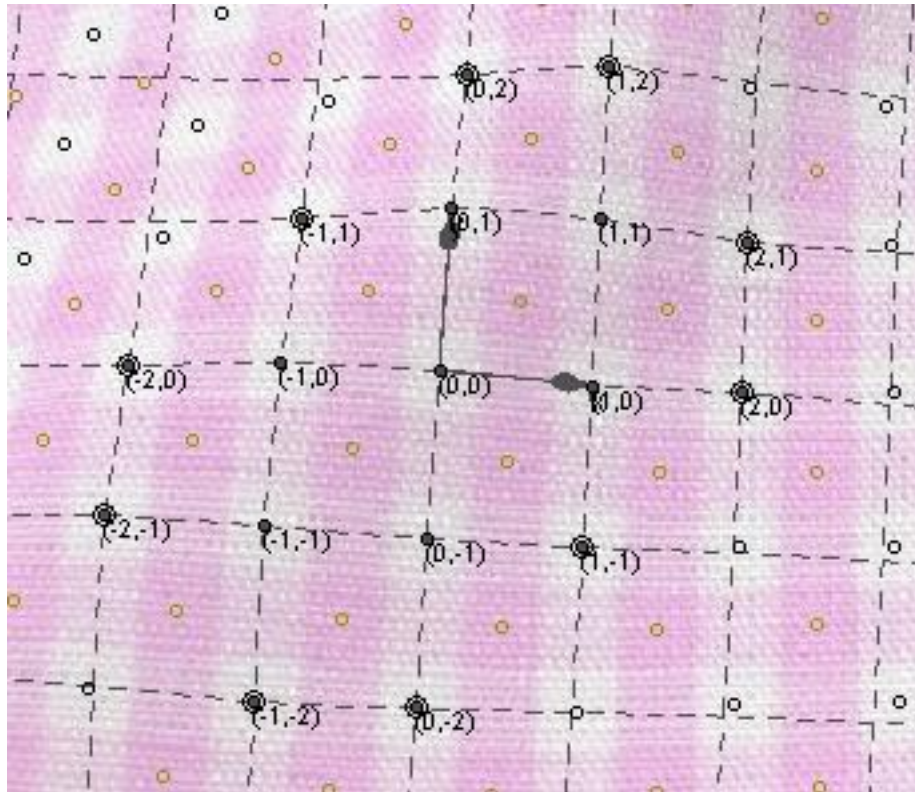


Abbildung 23: Zur Validierung neuer Quads wird ein temporäres Mapping (graues gestricheltes Gitter) verwendet, welches bereits akzeptierte Kontroll-Punkte (ausgefüllte kleine Kreise, z.B. (1,1)) und übrige Kontroll-Punkt-Kandidaten (doppelte konzentrische Kreise) als Kontroll-Punkte benutzt.

Nun werden Quads mittels des temporären Mappings m_{temp} konstruiert, welche die Kontroll-Punkte von m_{temp} als Eckpunkte verwenden. Die so konstruierten Quads, welche mindestens einen Kontroll-Punkt-Kandidaten als Eckpunkt haben, nennen wir Quad-Kandidaten.

Die Texel dieser Quad-Kandidaten werden mit dem Durchschnitts-Textel verglichen und mittels einer beliebigen Texel-Vergleichs-Methode¹¹ wird ein relativer Fehler für jeden Quad-Kandidaten bestimmt. Die Quads, deren relativer Fehler einen Schwellwert unterschreitet, werden als korrekte Quad-Kandidaten angenommen.

Alle als korrekt angenommenen Quad-Kandidaten werden als richtig akzeptiert. Das gleiche gilt für ihre Eckpunkte, welche bisher nur Kontroll-Punkt-Kandidaten sind. Das globale Mapping, das Durchschnitts-Textel und der durchschnittliche absolute Fehler werden aktualisiert. Das temporäre Mapping wird verworfen.

¹¹ Eine konkrete Texel-Vergleichs-Methode wird später im Abschnitt 3.5 vorgestellt.

3.4.4 Löschen von Feature-Punkten in neuen Quads

Innerhalb der neu akzeptierten Quads sollten sich nun keine Feature-Punkte aus dem aktuellen Cluster, mehr befinden. Ist dies doch der Fall, werden diese endgültig gelöscht.

Damit ist eine Quad-Iteration abgeschlossen, und die nächste kann beginnen. Falls die Quad-Iteration kein neues Quad akzeptiert hat, endet auch die Cluster-Iteration.

3.5 Texel-Vergleichs-Methode

Der zuvor beschriebene Algorithmus erfordert eine Texel-Vergleichs-Methode, welche zwei Texel miteinander vergleicht und dem Paar einen absoluten Fehler zuordnet, welcher umso größer ist, je verschiedener die Texel sind.

Im Folgenden soll die verwendete Texel-Vergleichs-Methode beschrieben werden, ihre Vor- und Nachteile beleuchtet werden und Verbesserungsvorschläge gemacht werden, wie man diese Methode noch weiter verbessern kann.

3.5.1 Beschreibung der Methode

Angenommen zwei Texel q_1 und q_2 sollen verglichen werden (Vgl. Abbildung 24).



Abbildung 24: Beispiel zweier zu vergleichender Texel (im topologischen Koordinaten-System). Links das Durchschnitts-Texel. Rechts ein Texel mit verschiedenen lokalen Beleuchtungsstärken und Verzerrung.

Dann werden zuerst beide Texel auf die gleiche Größe skaliert. Für jeden Pixel $p_i := (r_i, g_i, b_i)$ werden drei Werte $w_{i,1}$, $w_{i,2}$ und $w_{i,3}$ bestimmt:

$$w_{i,1} := \begin{cases} \frac{r_i}{g_i}, & \text{falls } g_i \neq 0 \\ 300, & \text{sonst} \end{cases}, w_{i,2} := \begin{cases} \frac{g_i}{b_i}, & \text{falls } b_i \neq 0 \\ 300, & \text{sonst} \end{cases} \quad \text{und } w_{i,3} := \begin{cases} \frac{b_i}{r_i}, & \text{falls } r_i \neq 0 \\ 300, & \text{sonst} \end{cases}$$

Angenommen durch Beleuchtung einer Originalfarbe $o_i = (o_{i,r}, o_{i,g}, o_{i,b})$ mit einem Beleuchtungsfaktor l wird jede Pixelfarbe nach folgender Formel bestimmt:

$$p_i := (r_i, g_i, b_i) = (l \cdot o_{i,r}, l \cdot o_{i,g}, l \cdot o_{i,b})$$

Dann sind die Werte $w_{i,1}$, $w_{i,2}$ und $w_{i,3}$ unabhängig vom Beleuchtungsfaktor und hängen nur noch von der Originalfarbe o_i ab. Eine Visualisierung dieser drei Werte ist in Abbildung 25 zu sehen.

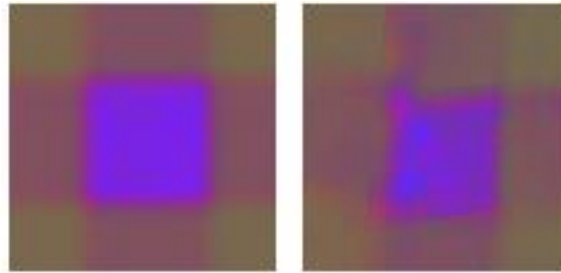


Abbildung 25: Darstellung der normierten, beleuchtungsunabhängigen Farbwerte (zur besseren Darstellung skalar multipliziert mit 100) der Texel aus **Abbildung 24**. Dabei wurde $100 \cdot w_{i,1}$ als Rotwert, $100 \cdot w_{i,2}$ als Grünwert und $100 \cdot w_{i,3}$ als Blauwert jedes Pixels verwendet.

Um nun einen absoluten Fehler zwischen den Texteln zu bestimmen, bestimmen wir zuerst einen Fehler e_i für jeden Pixel $p_{1,i}$ aus Texel q_1 und den zugehörigen Pixel $p_{2,i}$ aus Texel q_2 . Dazu werden die normierten Farbwerte $w_{1,i,1}$, $w_{1,i,2}$ und $w_{1,i,3}$ von Pixel $p_{1,i}$ mit den normierten Farbwerten $w_{2,i,1}$, $w_{2,i,2}$ und $w_{2,i,3}$ von Pixel $p_{2,i}$ verglichen:

$$e_i := \frac{|w_{1,i,1} - w_{2,i,1}| + |w_{1,i,2} - w_{2,i,2}| + |w_{1,i,3} - w_{2,i,3}|}{3}$$

Die Pixelfehler e_i sind in Abbildung 26 visualisiert.

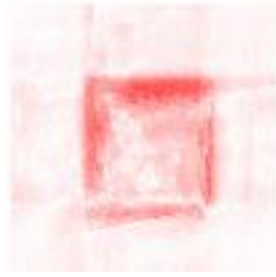


Abbildung 26: Die Pixelfehler der Beispiel-Textel aus **Abbildung 24**. An den verzerrten Stellen ist der Fehler besonders hoch (rot), an anderen Stellen ist der Fehler trotz unterschiedlicher Beleuchtung, eher niedrig (weiß).

Der absolute Fehler der zwei Textel berechnet sich dann aus dem arithmetischen Mittel aller Pixelfehler e_i .

3.5.2 Vor- und Nachteile

Ein Vorteil dieser Textel-Vergleichs-Methode ist, dass der Beleuchtungsfaktor keine Auswirkung auf das Ergebnis hat. Durch die Normierung pro Pixel, anstatt einer Normierung pro Textel, haben auch Beleuchtungsunterschiede innerhalb eines Textels keine Auswirkungen. So ist beispielsweise ein „Knick“ im Stoff im rechten Textel in **Abbildung 24**, durch welchen die linke Seite des Textels dunkler ist als die rechte. In **Abbildung 25** und **Abbildung 26** sieht man, dass dieser Helligkeitsunterschied nahezu keine Auswirkungen auf den ermittelten Fehler hat.

Ein Nachteil der Textel-Vergleichs-Methode ist, dass die Fehler für verschiedene Farben verschieden skalieren. So erzeugt beispielsweise ein schwaches Rauschen auf dunklen Farben einen höheren Fehler als auf hellen Farben. Dieser Nachteil hatte in den durchgeführten Experimenten (vgl. nächster Abschnitt) so gut wie keine Auswirkungen und wurde daher als vernachlässigbar betrachtet.

Ein weiterer Nachteil ist, dass diese Methode nicht nur den Beleuchtungsfaktor ignoriert, sondern gleichermaßen auch helle Originalfarben nicht von dunklen Originalfarben unterscheiden kann. So sind zwei Textel welche nur aus hellen und dunklen Blautönen bestehen, für diese Textel-Vergleichs-Methode an allen Pixeln gleich. Dieses Problem ist jedoch nicht algorithmisch lösbar, da man auf einem Bild unter Umständen nicht unterscheiden kann, ob ein Muster ein Resultat der Originalfarben ist (es wurden z.B. unterschiedliche Stoffe beim Weben benutzt), oder ob es nur ein Resultat der Beleuchtung ist (z.B. ein Maschendrahtzaun, welcher Schatten auf eine Betonstraße wirft).

3.5.3 Mögliche Verbesserung

Wie man im Beispiel aus Abbildung 24 bis Abbildung 26 sieht, werden für zwei korrekte Texel, welche jedoch in sich verzerrt sind höhere Fehler berechnet als für unverzerrte Texel. Um das zu umgehen müsste man den Texel noch einmal entzerren. Dazu könnte man genauso vorgehen wie im großen Gesamtbild. Man sucht in beiden Texeln Feature-Punkte, korreliert diese miteinander und benutzt dann ein Mapping, um das eine Texel so zu entzerren, dass es dem anderen Texel entspricht.

4. Umsetzung

Der in Abschnitt 3 vorgestellte Algorithmus wurde in C++ implementiert. Dieser Abschnitt soll einen kurzen Überblick über die Implementierung des Algorithmus geben.

4.1 Eingabe- und Ausgabedaten

Der Algorithmus selbst ist nicht interaktiv. Die Implementation bietet jedoch die Möglichkeit zur Laufzeit durch Interaktion mit der GUI die Bildschirm-Visualisierung anzupassen, um bestimmte Teilaspekte der einzelnen Schritte oder des Ergebnisses genauer zu analysieren.

Es werden zur Ausführung folgende Eingabedaten benötigt:

- Bild der Textur, z.B. im jpg-Format.
- Eine txt-Datei pro Cluster, welche die Positionen der Feature-Punkte Zeilenweise auflistet.
- Eine Datei namens `config.settings`, welche die Dateipfade zu Bild- und Cluster-Dateien definiert und welche die Einstellungen zur Ausführung und Darstellung des Algorithmus festlegt.
- Verschiedene DLL- und TTF-Dateien, zur Darstellung des Algorithmus auf dem Bildschirm.

Der Algorithmus erzeugt folgende Ausgabedaten:

- Eine txt-Datei pro Cluster, welche zeilenweise die bestätigten Kontrollpunkte, also Tupel von Bildschirmkoordinaten und topologischen Koordinaten, enthält.

- Eine Bildschirm-Visualisierung während der Ausführung, auf der Details der Ausführung sichtbar sind.
- Eine txt-Datei pro Eingabe-Bild mit statistischen Daten des Programmdurchlaufs.

4.2 Programm Architektur

Der C++ Programm Code ist in zwei Projekte aufgeteilt.

- Studienarbeit05Lib: Bibliothek, welche die technologieunabhängige Umsetzung des Algorithmus enthält. Sie enthält auch die technologieunabhängigen Klassen zur Visualisierung des Algorithmus. Konkrete Visualisierungsimplementierungen müssen jedoch vom Nutzer der Bibliothek bereitgestellt werden.
- Studienarbeit05Proj: C++ Applikation, welche die technologieabhängige Visualisierung des Algorithmus und die main()-Funktion implementiert. Sie benutzt die Bibliothek, zur Ausführung des Algorithmus.

Die zwei Projekte sind über das abstract factory Entwurfsmuster¹² miteinander gekoppelt, um eine leicht wartbare Trennung zwischen technologieabhängigem Projekt und technologieunabhängigem Projekt zu erreichen. Des Weiteren sorgt das Entwurfsmuster dafür, dass das technologieabhängige Projekt vom technologieunabhängigen abhängt und nicht umgekehrt, wie es das Dependency-Inversion-Prinzip fordert.

Das Projekt Studienarbeit05Proj ist sehr klein gehalten und stellt nur die nötigste Funktionalität zur Verfügung. Es benutzt SDL¹³ und einige DLL- und TTF-Dateien zur Visualisierung. Das Projekt stellt der Bibliothek die Implementationen primitiver Bildschirmzeichenoperationen (Linien, Kreise usw.), die Implementationen einfacher GUI Elemente und die Implementationen von Bitmap-Daten bereit.

Das Projekt Studienarbeit05Lib unterteilt sich selbst wieder in 5 Pakete und 2 lose Klassen:

- control-Paket: Enthält die Klassen zur Verwaltung des Kontrollflusses.

¹² Vgl. [22]

¹³ Simple DirectMedia Layer [23]

- lib-Paket: Enthält die Implementierung¹⁴ der Mapping Funktionen als Thin-Plate-Splines.
- model-Paket: Enthält die Daten, auf denen der Algorithmus arbeitet und welche visualisiert werden.
- utils-Paket: Enthält Klassen mit Hilfsfunktionen.
- view-Paket: Enthält Klassen zur Visualisierung der Klassen aus dem Model-Paket. Die Visualisierung benötigt eine externe Implementierung von primitiven Anzeige-Funktionen (wie z.B. Linien und Boxen).
- Const-Klasse: Enthält die C++-Repräsentation der Einstellungen aus der `config.settings` Datei.
- Studienarbeit05Lib-Klasse: Initialisiert Klassen aus den Paketen und startet den Kontrollfluss. Ist die Hauptschnittstelle für Benutzer der Bibliothek.

Das Projekt Studienarbeit05Lib enthält unter anderem die Implementationen der Texel-Vergleichs-Methode auf welche im nächsten Abschnitt genauer eingegangen werden soll.

5. Ergebnis

Der implementierte Algorithmus wurde auf 14 verschiedenen Bildern von Stoffen und Kleidung getestet. Dabei wurden bei allen 14 Bildern die gleichen Parameter benutzt, da in der Anwendung keine Anpassung pro Bild vorgenommen werden soll, sondern der Algorithmus auf allen Bildern gleich arbeiten soll.

In diesem Abschnitt soll gezeigt werden welche der zuvor vorgestellten Ideen in der Praxis wirklich funktionieren und was für Probleme noch auftreten. Dazu wird in Abschnitt 5.1 der verwendete Parametersatz spezifiziert. In Abschnitt 5.2 werden einige exemplarische Ergebnisse des Algorithmus gezeigt¹⁵. In den Abschnitten 5.3 findet eine erste quantitative Auswertung statt. In Abschnitt 5.4 werden die Ergebnisse dann qualitativ ausgewertet und die Ursachen der aufgetretenen Probleme genauer untersucht.

¹⁴ Quellcode von [24] wiederverwendet. Das zugehörige Paper ist: [25].

¹⁵ Ausführlichere Ergebnisse aller 14 Test-Bilder sind im Anhang zu finden.

5.1 Eingestellte Parameter und Schwellwerte

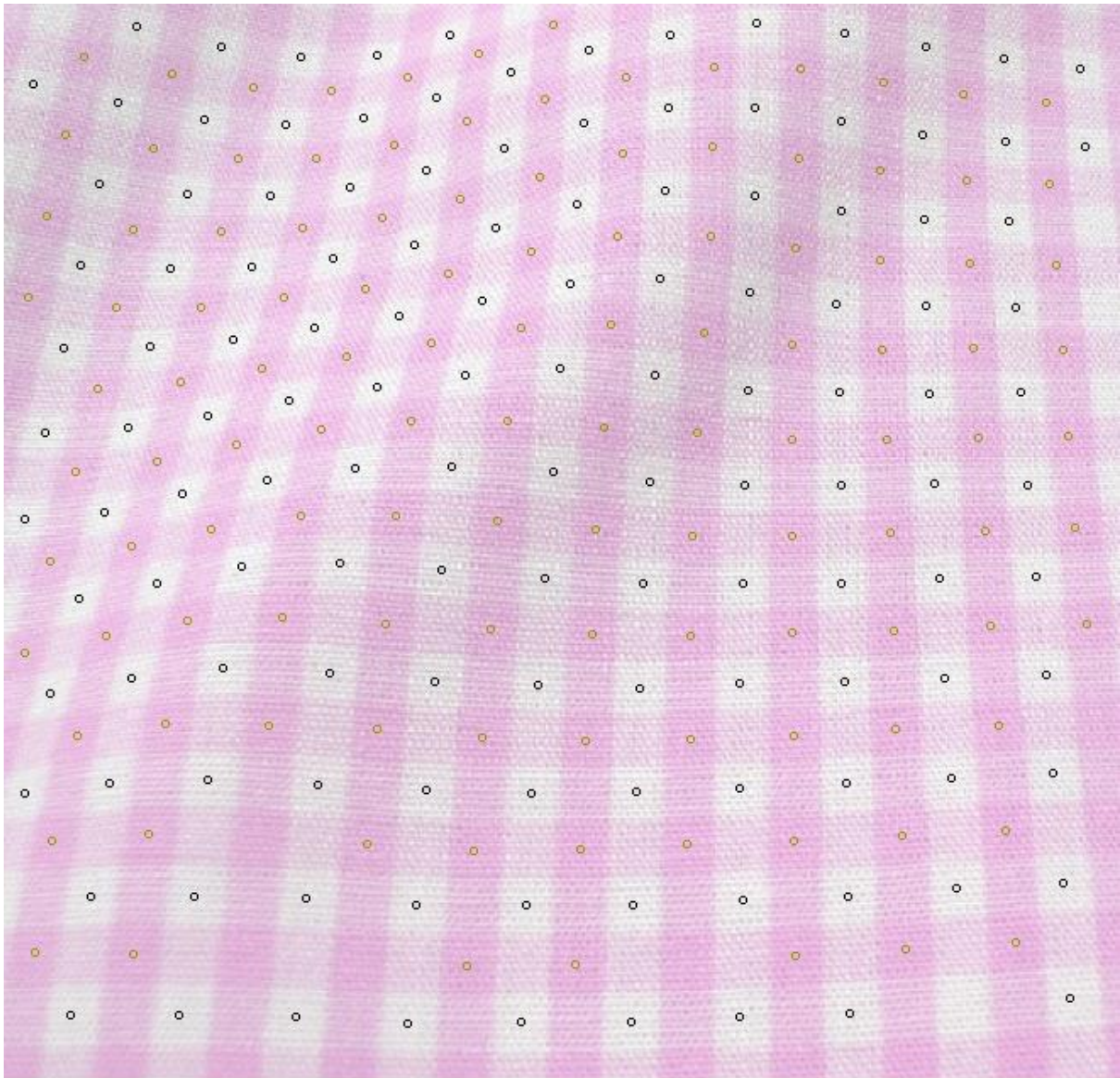
Für die Evaluierung des Algorithmus wurde folgender Parametersatz gewählt:

- Maximal akzeptierter relativer Quad-Fehler (Quads bzw. Texel mit größerem Fehler werden nicht akzeptiert): 1.7
- Benötigte relative Clustergröße (kleinere Cluster werden zu Beginn verworfen): 0.6
- Differenz in topologischen Koordinaten, welche die Cluster-Offsets im ersten und zweiten Quad haben dürfen: 0.2
- Differenz in topologischen Koordinaten, welche die Gefundene Feature-Punkte zur vermuteten Position haben dürfen: 0.5
- Genauigkeit, mit der die entzerrten Texel berechnet und verglichen werden: 50 x 50 Pixel

5.2 Beispielergebnisse

In diesem Abschnitt werden Ergebnisse von zwei der 14 Test-Bilder vorgestellt. Die Ergebnisse aller 14 Bilder sind im Anhang zu finden.

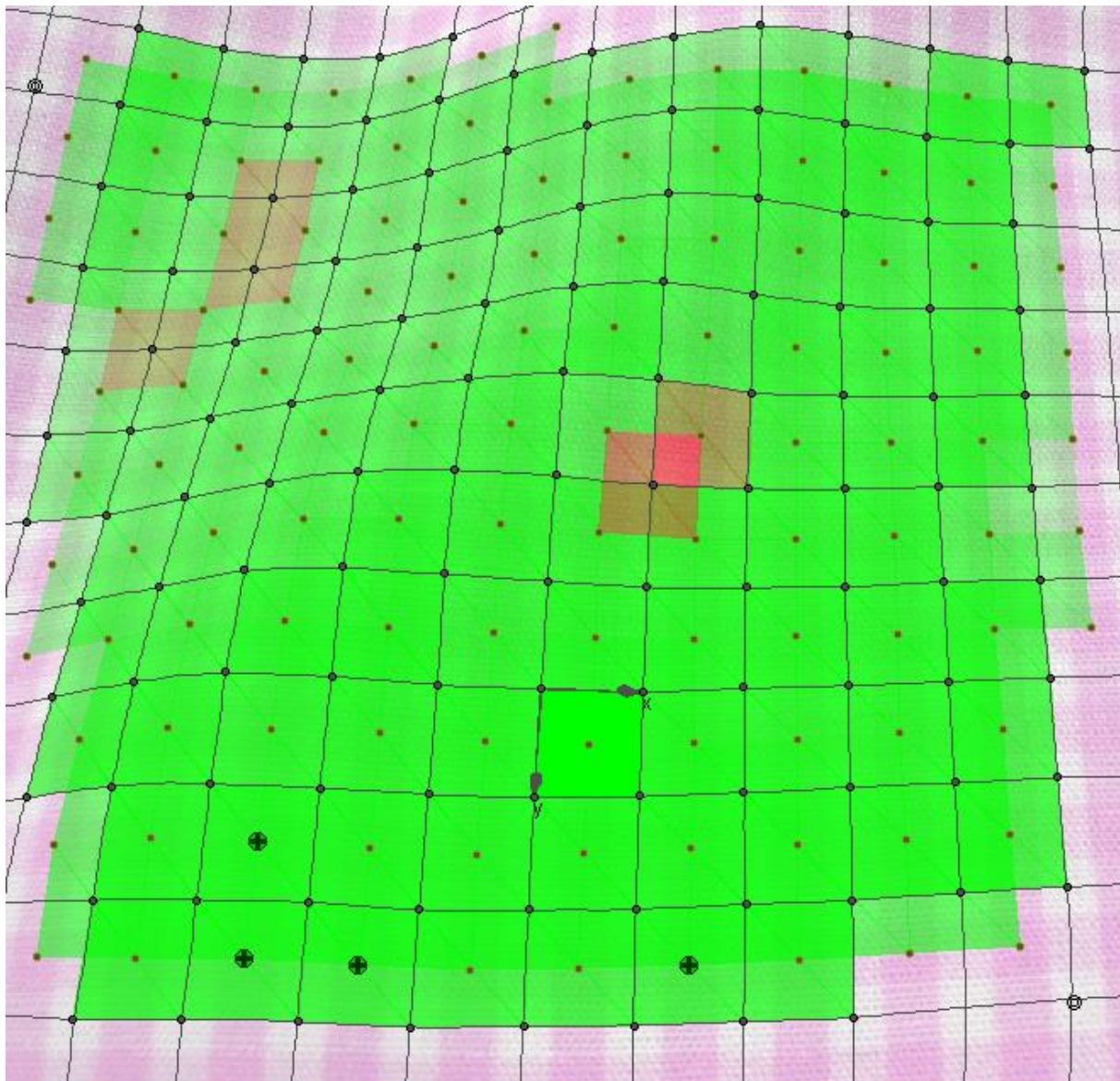
Die Ergebnisse der zwei exemplarischen Bilder dieses Abschnittes sind in Abbildung 27 bis Abbildung 32 zu sehen. Zum besseren Verständnis werden diese Ergebnisse direkt in den Beschriftungen der Bilder beschrieben.



Legende

o o gegebene Feature-Punkte
(eine Farbe pro Cluster)

Abbildung 27: Gegebenes Test-Bild (Tx1) mit Feature-Punkten aus zwei Clustern. Zur besseren Darstellung ist das Bild leicht aufgehellt und an den Rändern beschnitten, so dass nur noch die Bereiche nahe der gegebenen Feature-Punkte dargestellt werden.



Legende

○ ○ gegebene Feature-Punkte
(eine Farbe pro Cluster)

● ● verwendete Kontroll-Punkte
(eine Farbe pro Cluster)

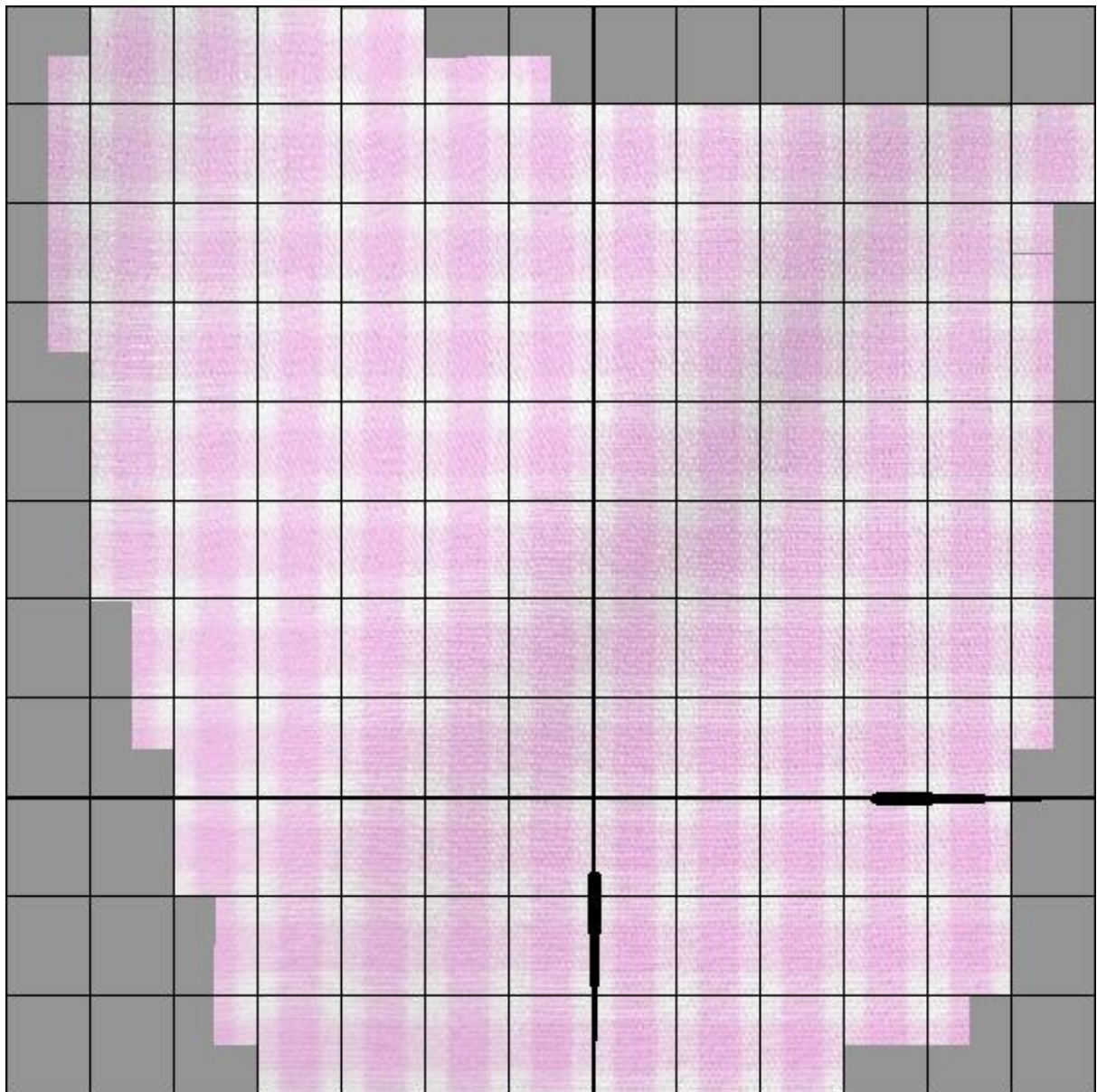
⊕ neu erstellte Kontroll-Punkte

⌌ topologische Koordinaten mit
ganzahliger x- oder y-Koordinate

akzeptierte Quads. Farbe und Transparenz variieren je nach relativem Fehler der Quads:

- rot, deckend: Fehler > Schwellwert, höherer Fehler
- rot, transparent: Fehler > Schwellwert, niedrigerer Fehler
- grün, transparent: Fehler ≤ Schwellwert, höherer Fehler
- grün, deckend: Fehler ≤ Schwellwert, niedrigerer Fehler

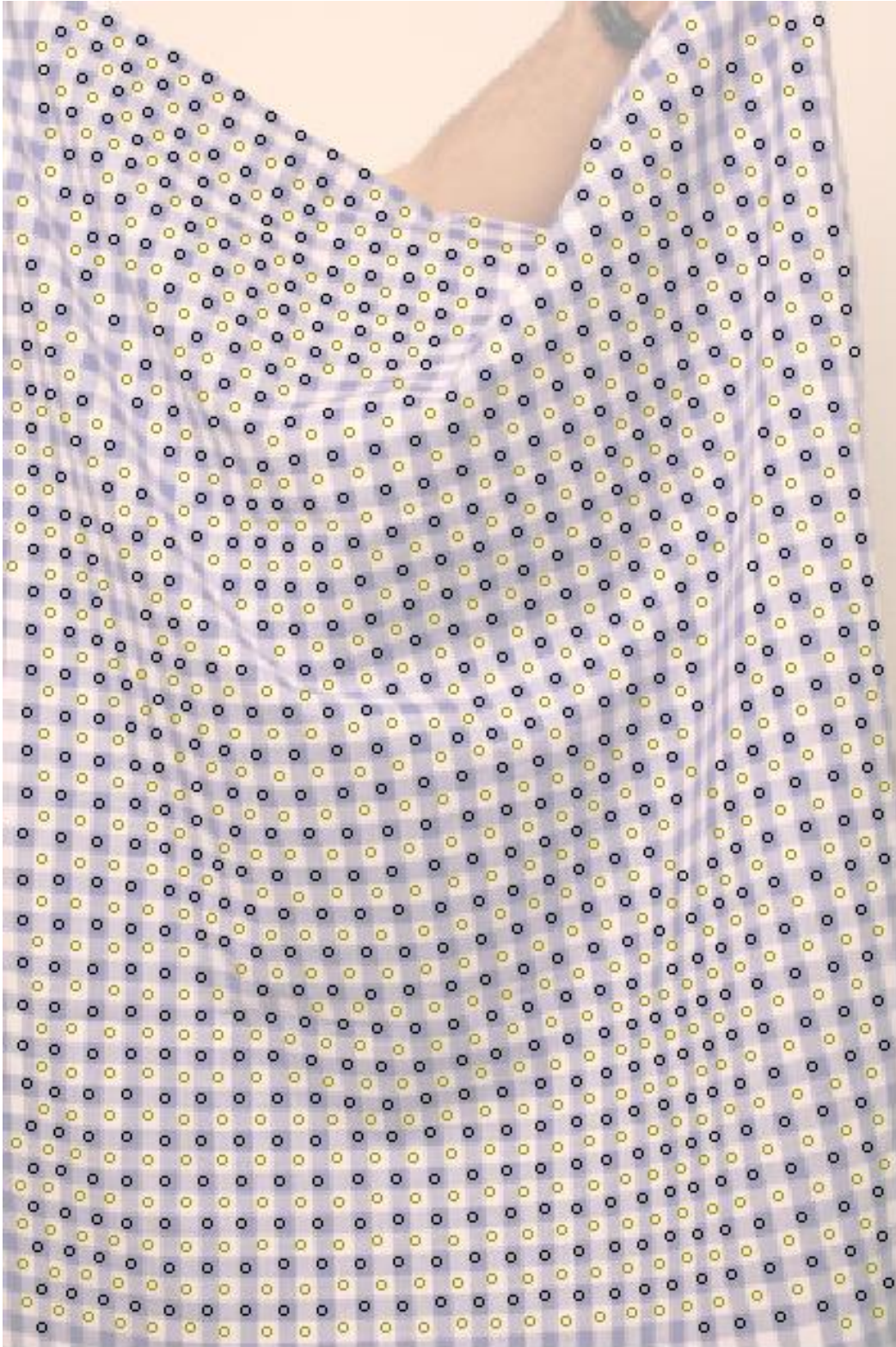
Abbildung 28: Gefundene Kontroll-Punkte und Quads in Test-Bild (Tx1). Zur besseren Darstellung ist das Bild leicht aufgehellt und an den Rändern beschnitten, so dass nur noch die Bereiche nahe der gegebenen Feature-Punkte dargestellt werden. Der Koordinatenursprung der topologischen Koordinaten (und damit implizit die zwei initialen Quads) sind mit grauen Pfeilen dargestellt und die x- und y-Richtung beschriftet.



Legende


 topologische Koordinaten
 mit ganzzahliger x- oder y-
 Koordinate

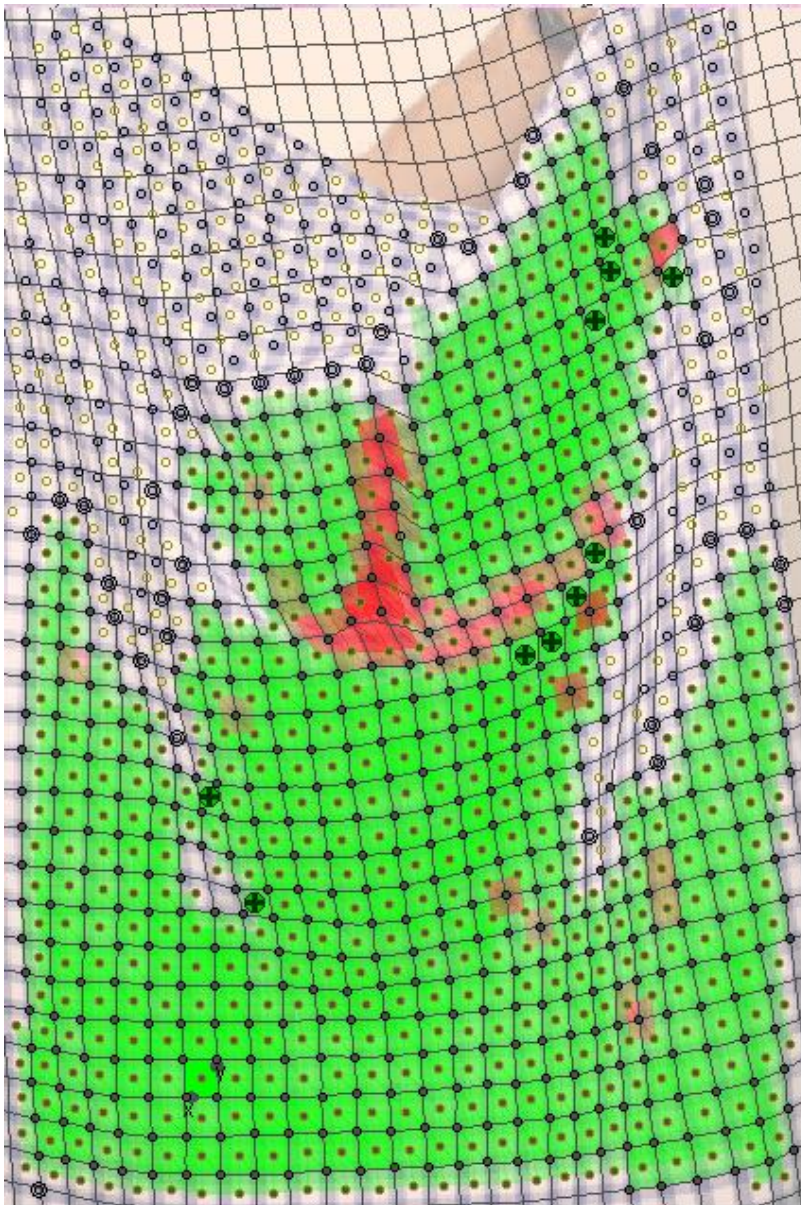
Abbildung 29: Darstellung aller zugeordneten Quads bzw. Texel des Test-Bildes (Tx1) im topologischen Koordinatensystem. Zur besseren Darstellung sind die Farben leicht aufgehellt.




Legende

○ ○ gegebene Feature-Punkte
(eine Farbe pro Cluster)

Abbildung 30: Gegebenes Test-Bild (Tx31) mit Feature-Punkten aus zwei Clustern. Zur besseren Darstellung ist das Bild leicht aufgehellt und an den Rändern beschnitten, so dass nur noch die Bereiche nahe der gegebenen Feature-Punkte dargestellt werden.



Legende

- ○ gegebene Feature-Punkte
(eine Farbe pro Cluster)
 - ● verwendete Kontroll-Punkte
(eine Farbe pro Cluster)
 - + neu erstellte Kontroll-Punkte
-  topologische Koordinaten mit
ganzzahliger x- oder y-Koordinate


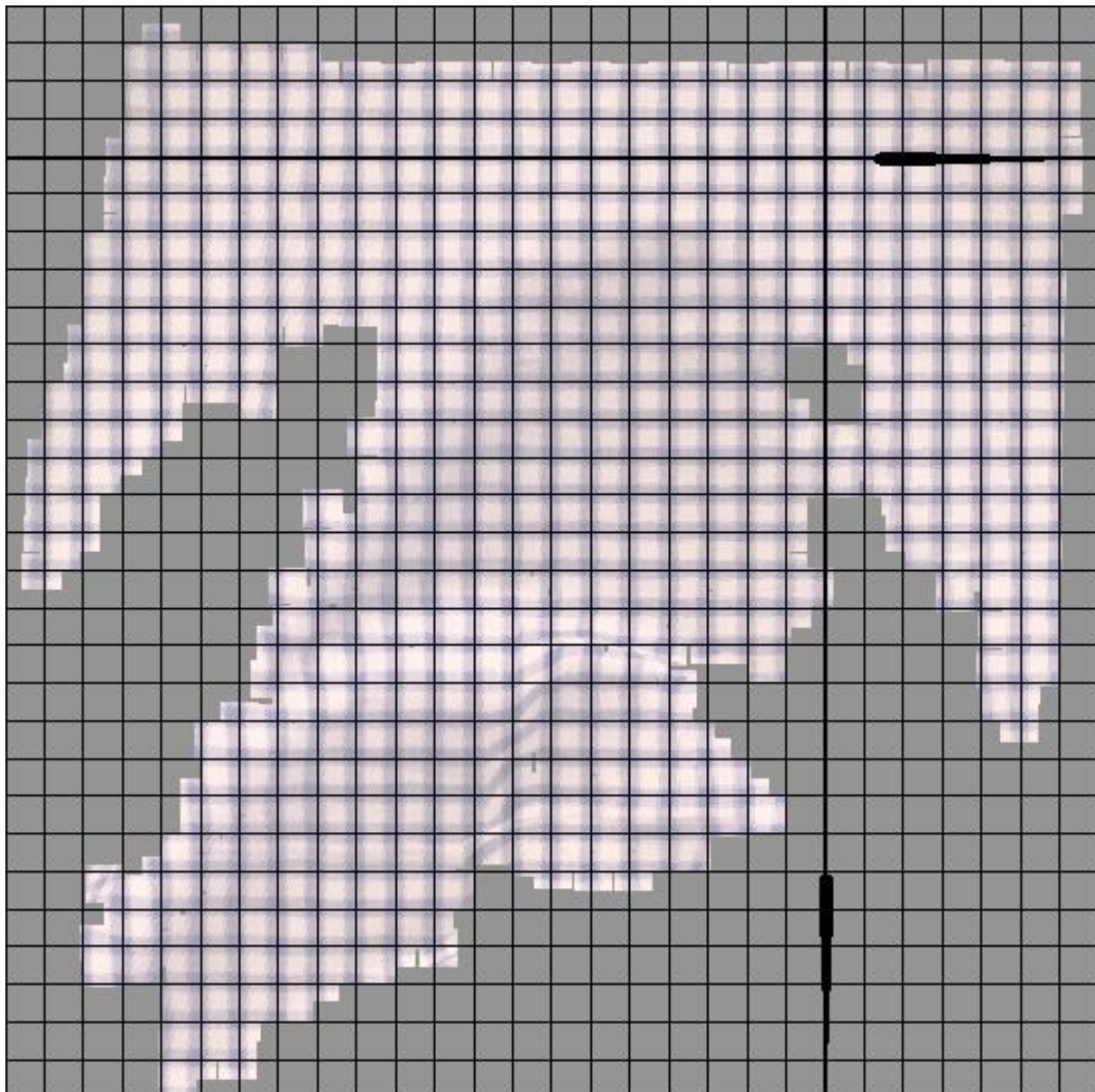
-  akzeptierte Quads. Farbe und Transparenz variieren je
nach relativem Fehler der Quads:
 - rot, deckend: Fehler > Schwellwert,
höherer Fehler
 - rot, transparent: Fehler > Schwellwert,
niedrigerer Fehler
 - grün, transparent: Fehler ≤ Schwellwert,
höherer Fehler
 - grün, deckend: Fehler ≤ Schwellwert,
niedrigerer Fehler

Abbildung 31: Gefundene Kontroll-Punkte und Quads in Test-Bild (Tx31). Zur besseren Darstellung ist das Bild leicht aufgehellt und an den Rändern beschnitten, so dass nur noch die Bereiche nahe der gegebenen Feature-Punkte dargestellt werden. Der Koordinatenursprung der topologischen Koordinaten (und damit implizit die zwei initialen Quads) sind mit grauen Pfeilen dargestellt und die x- und y-Richtung beschriftet.



Legende

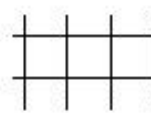

 topologische Koordinaten
 mit ganzzahliger x- oder y-
 Koordinate

Abbildung 32: Darstellung aller zugeordneten Quads bzw. Texel des Test-Bildes (Tx31) im topologischen Koordinatensystem. Zur besseren Darstellung sind die Farben leicht aufgehellt.

5.3 Quantitative Bewertung

Eine objektive quantitative Bewertung des Algorithmus fällt schwer, da die genauen Zahlen der Test-Ergebnisse stark von den verwendeten Test-Daten abhängen und im Rahmen dieser Arbeit nur eine Auswahl von 14 Bildern getestet wurden. Daher soll dieser Abschnitt dazu

dienen erste Hinweise auf Probleme zu bekommen, welche dann in den folgenden Abschnitten genauer untersucht werden können.

5.3.1 Hinzufügen neuer Kontroll-Punkte

In Abschnitt 3.3.2 und Abbildung 20 wird gezeigt, wie der Algorithmus neue Feature-Punkte bzw. Kontroll-Punkte zu den bereits gegebenen hinzufügt. Abbildung 33 zeigt, dass im Durchschnitt nur etwa 1% der verwendeten Kontroll-Punkte neu hinzugefügte Punkte sind, während der Rest bereits gegebene Punkte sind. Die hinzugefügten Punkte sind zudem teils falsch oder bringen kaum einen Mehrwert, da sie inmitten gegebener Feature-Punkte liegen.

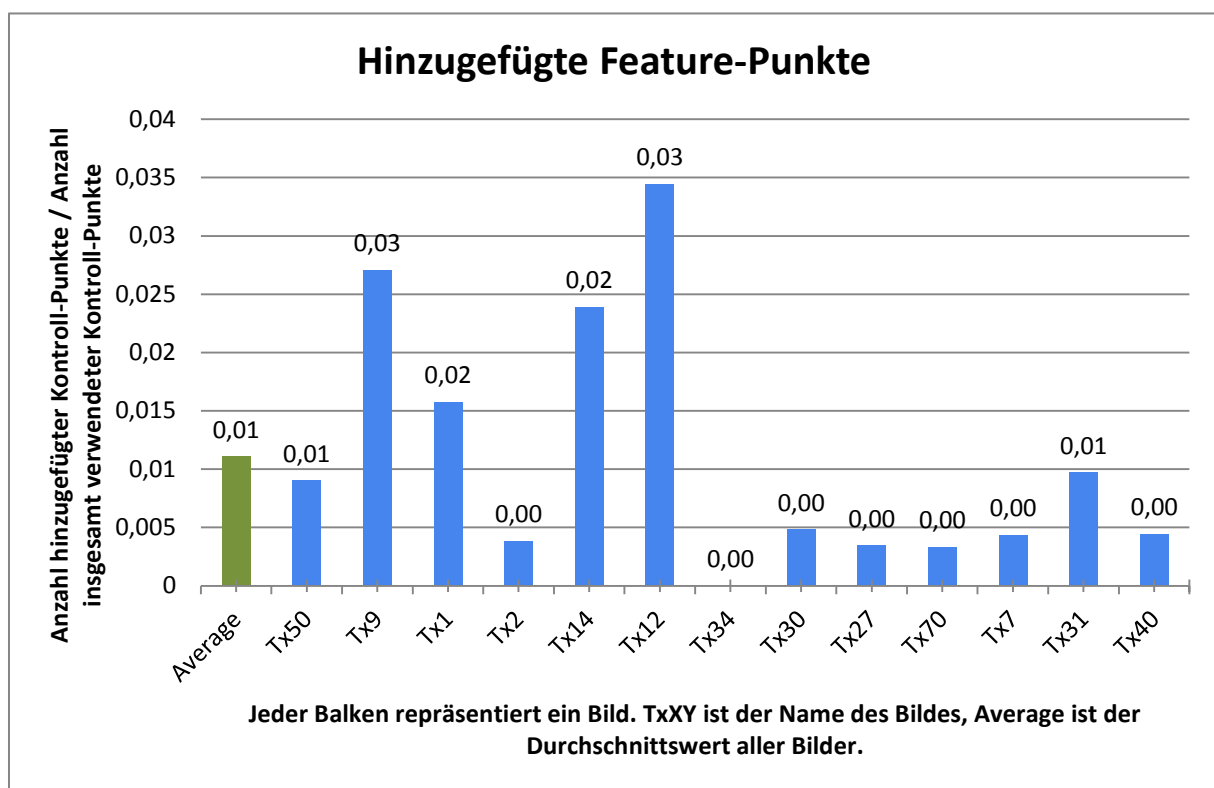


Abbildung 33: Anzahl hinzugefügter Feature-Punkte

Auf die Ursachen der geringen Anzahl neuer Kontroll-Punkte und die teils falsch hinzugefügten Kontroll-Punkte wird in Abschnitt 5.4 genauer eingegangen.

5.3.2 Verwendung gegebener Feature-Punkte

In Abbildung 34 ist zu sehen, wie viele der initial gegebenen Feature-Punkte am Ende als Kontroll-Punkte verwendet wurden. Im Durchschnitt wurden 69% der gegebenen Feature-Punkte als Kontroll-Punkte verwendet, jedoch weichen die Einzelwerte der Test-Bilder sehr

stark ab. So werden auf fünf der Test-Bilder über 90% der Punkte zugeordnet und während auf drei der Test-Bilder unter 30% zugeordnet werden. Auf die Ursachen der teils geringen Zuordnung wird in Abschnitt 5.4 genauer eingegangen.

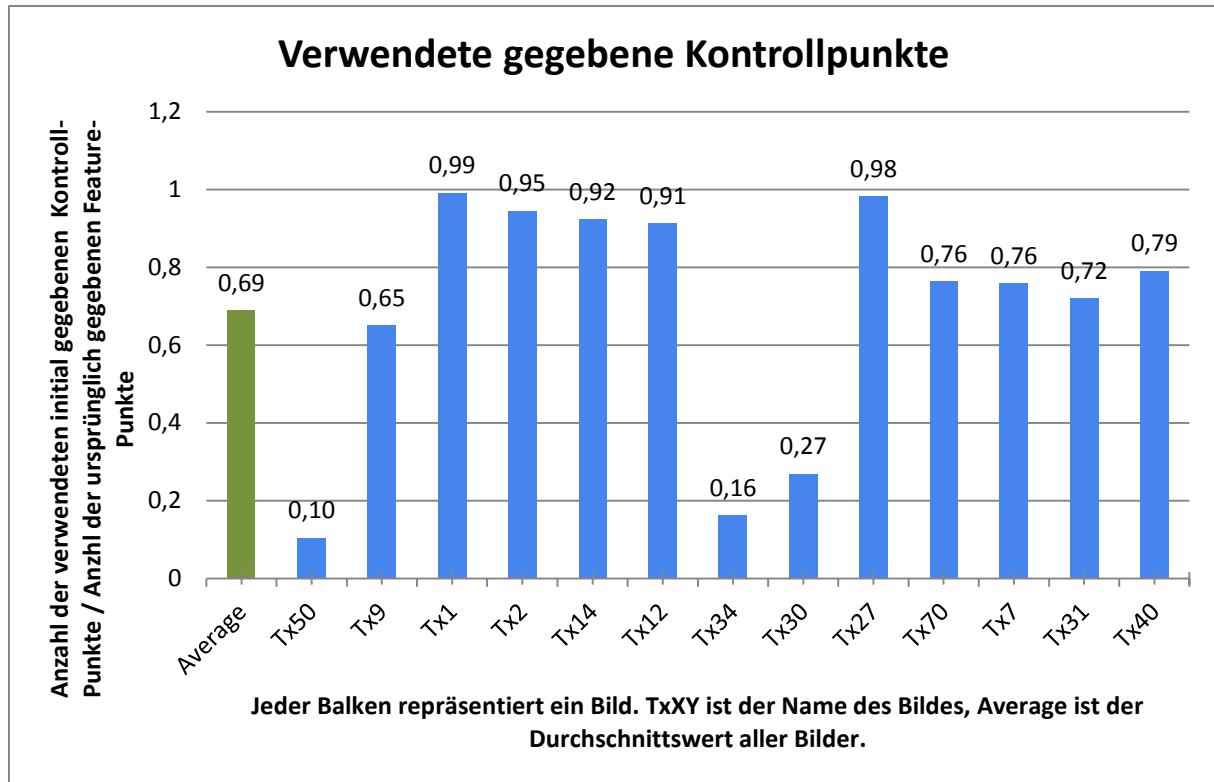


Abbildung 34: Anzahl der verwendeten Kontroll-Punkte

5.3.3 False Positives

Die Anzahl der zugeordneten Feature-Punkte (positives) muss man gegen die Anzahl der falschen Zuordnungen (false positives) abwägen. Automatisch ist nur schwer auszuwerten, ob ein Kontroll-Punkt korrekt oder inkorrekt zugeordnet wurde. Daher wurde ein manuelles Auswertungs-Verfahren gewählt, welches leicht zu handhabenden ist und in kurzer Zeit durchgeführt werden kann.

Dazu wurden am Ende die zugeordneten Quads aus den Bild-Koordinaten in topologische Koordinaten projiziert (Vgl. Beispiel in Abbildung 35). Im topologischen Koordinatensystem kann nun leicht manuell ausgezählt werden wie viele der zugeordneten Quads falsch sind. Die gesamte Anzahl der zugeordneten Quads kann leicht automatisch ausgezählt werden. Wann genau ein Quad als falsch bewertet wird, ist in Abbildung 36 zu sehen.

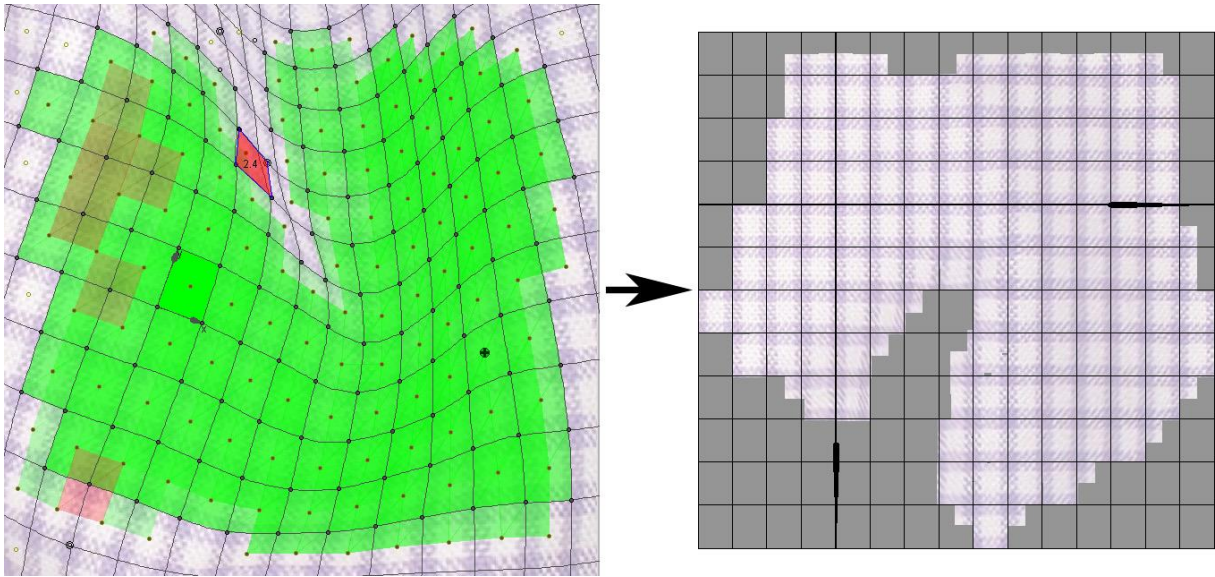


Abbildung 35: Zugeordnete Quads in Bildschirm-Koordinaten (grüne und rote Flächen links) werden in topologische Koordinaten projiziert, um leichtes Auszählen falscher Quads zu ermöglichen.

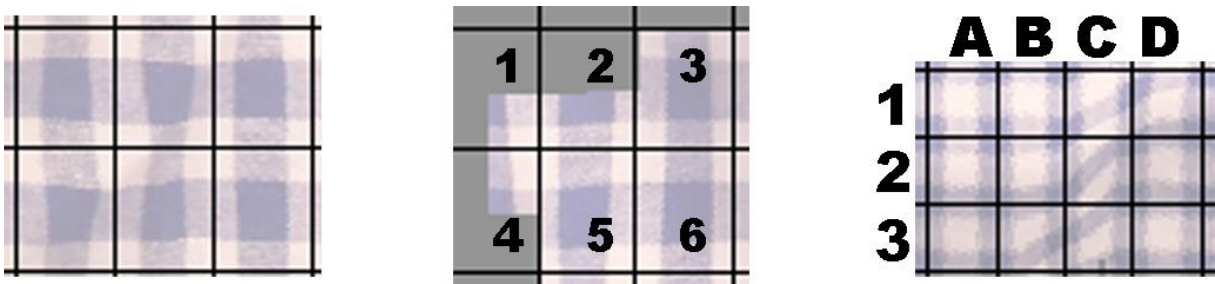


Abbildung 36: Ob ein Quad korrekt oder falsch ist muss manuell bewertet werden. Hier einige Beispiele wie dabei Bewertet wurde. In den Beispielen stellen die schwarzen Linien ganzzahlige topologische Koordinaten dar. Alle 6 Quads im linken Beispiel-Bild wurden als korrekt bewertet, denn auch wenn manche verzerrt sind, so sind die vier Eck-Punkte korrekte Kontroll-Punkte und die Verzerrung der Texel hängt ausschließlich vom gewählten Mapping ab. Im mittleren Beispiel-Bild wurden nur die Quads 3, 5 und 6 als korrekt bewertet, während die Quads 1, 2 und 4 komplett aus der Wertung fallen, egal ob sie falsch oder richtig sind. Es gehen also nur Quads aus Cluster C_0 in die manuelle Wertung ein. Im rechten Beispiel-Bild wurden die Quads C1, D1, C2, B3 und C3 als falsch bewertet, der Rest als korrekt. Rechts sieht man auch, dass die manuelle Bewertung nur lokal stattfindet, denn beispielsweise sind auch entweder die Koordinaten von B2 oder die Koordinaten von D2 falsch zugeordnet, dennoch werden beide als korrekt angenommen, da die Eckpunkte ein korrektes Quad bilden.

Teilt man nun für jedes Bild die absolute Anzahl der false positive Quads aus Cluster C_0 durch die Gesamtzahl der Quads aus Cluster C_0 , so erhält man einen relativen Wert für die false positives. Die so ermittelten Werte sind in Abbildung 37 zu sehen.

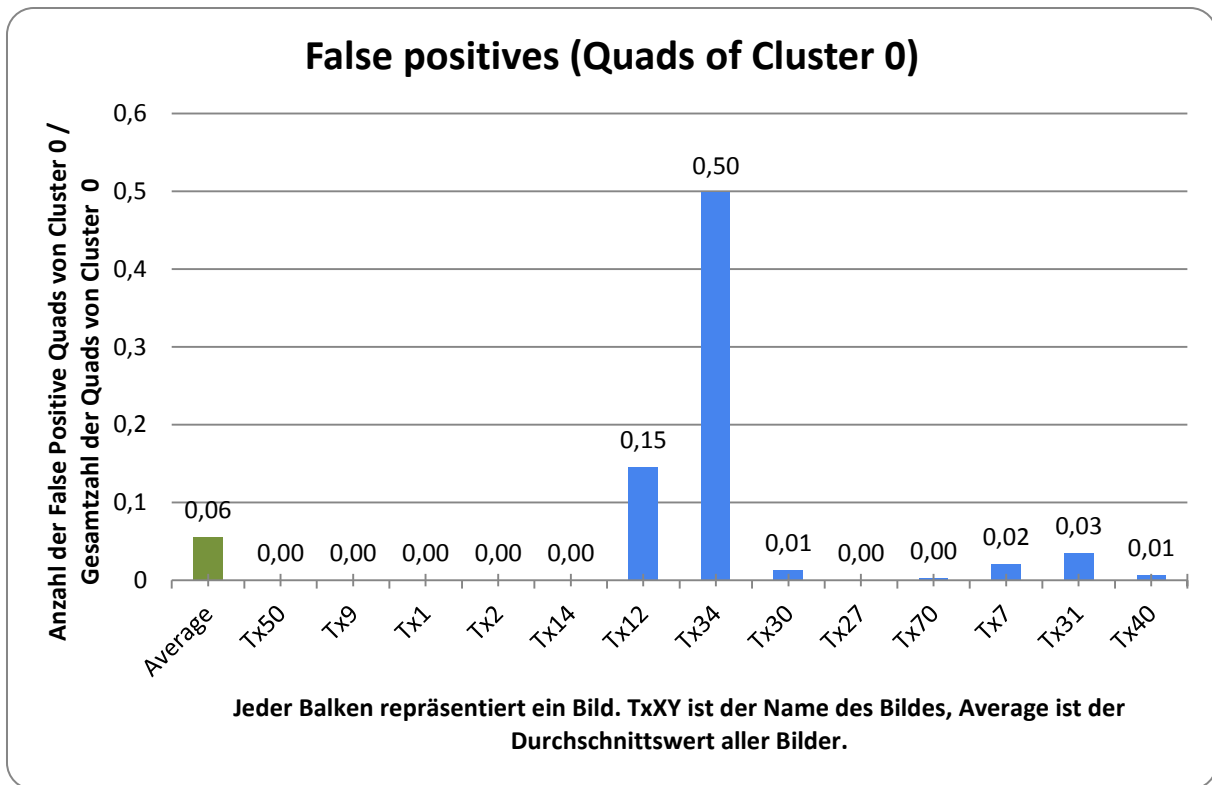


Abbildung 37: Relative false positives der einzelnen Test-Bilder.

Im Durchschnitt sind 6% der ermittelten Quads false positives, jedoch weichen die Werte der einzelnen Bilder stark vom Durchschnittswert ab. Die genauen Ursachen dafür werden in Abschnitt 5.4 genauer erklärt.

5.3.4 Verwerfen von Clustern

In Abschnitt 3.1 wird beschrieben, wie zu Beginn des Algorithmus komplette Cluster verworfen werden, falls ihre Größe zu verschieden vom Cluster mit den meisten Feature-Punkten ist. Das Verwerfen von Feature-Punkten bedeutet auch das Verwerfen von Informationen, welche vielleicht sinnvoll genutzt werden könnten. Abbildung 38 zeigt, wie viele Feature-Punkte nach dem Verwerfen erhalten bleiben. Man sieht, dass durchschnittlich 97% der gegebenen Punkte erhalten bleiben und nur in einem der Test-Fotos mehr als 5% der Punkte verworfen werden. Es wird also im Allgemeinen nur ein geringer Teil der gegebenen Informationen verworfen. Warum in einem Fall so viele Punkte verworfen werden, wird in Abschnitt 3.5 erklärt.

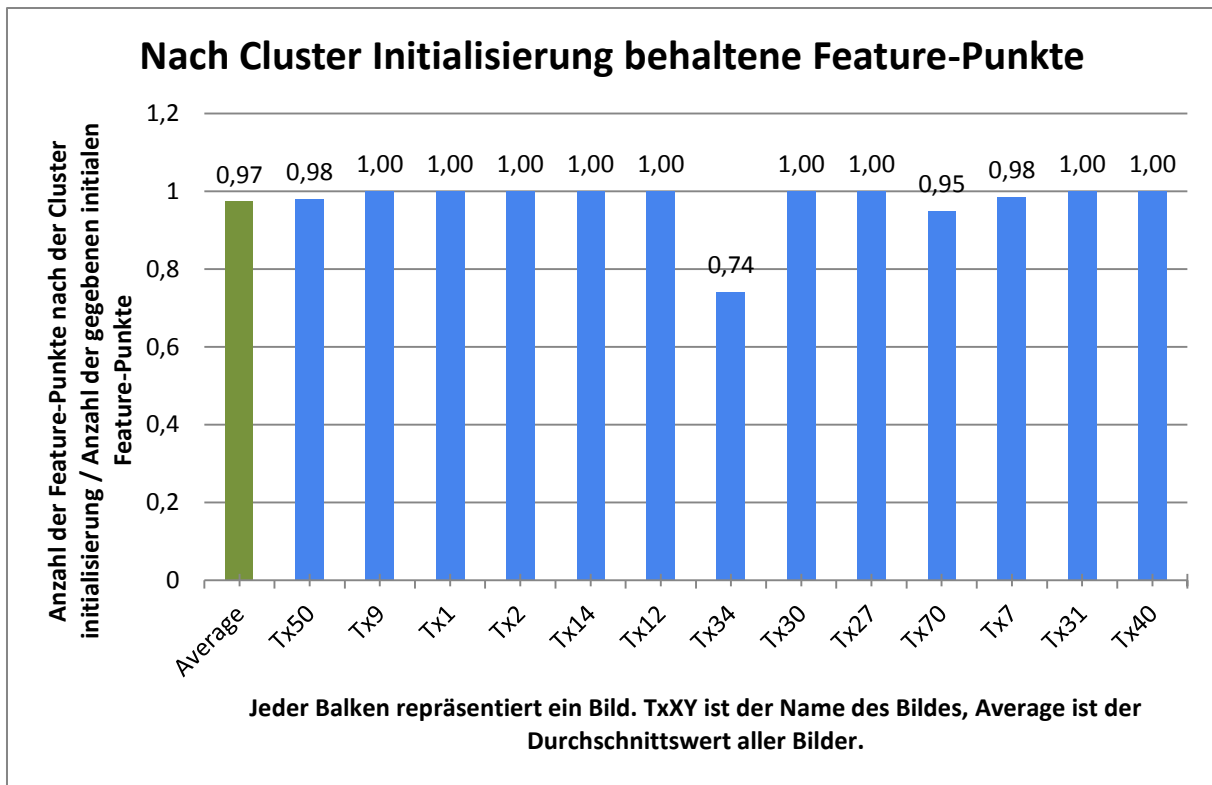


Abbildung 38: Zu Beginn des Algorithmus können ganze Cluster verworfen werden. Die Grafik zeigt, wie viele der gegebenen Feature-Punkte nach dem Verwerfen noch erhalten bleiben.

5.3.5 Laufzeit

Da der Algorithmus in einem Datenbank basierten Ansatz benutzt werden soll, wurde die Implementation nicht auf schnelle Laufzeit optimiert und ist entsprechend langsam. Trotzdem soll hier kurz die Laufzeit analysiert werden und Ursachen für die langsame Laufzeit benannt werden. In Abschnitt „6.2 Ausblick“ werden dann erste Ansätze zur Laufzeit Optimierung vorgestellt.

In Abbildung 39 sind die absoluten Laufzeiten des Algorithmus dargestellt. Sie schwanken bei den Testdaten zwischen 83 und 4481 Sekunden pro Bild. Im Durchschnitt werden 2 Sekunden pro Kontroll-Punkt benötigt.

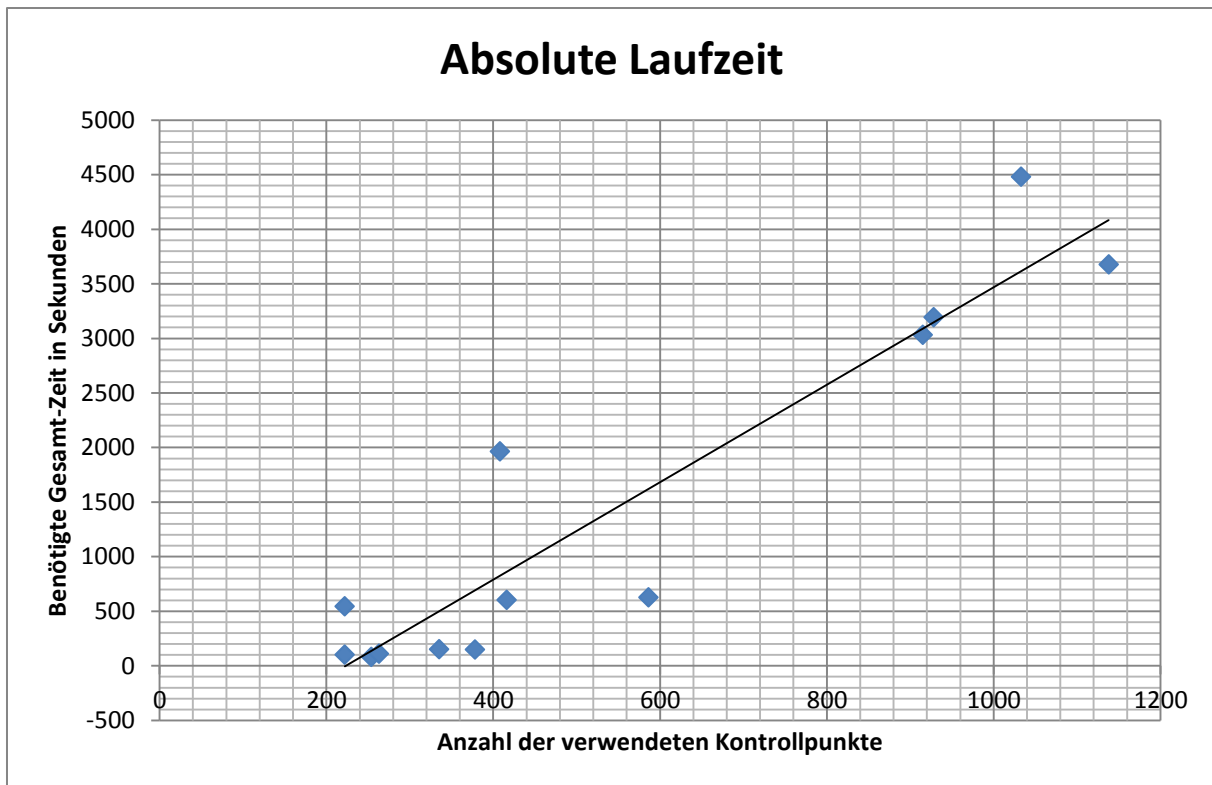


Abbildung 39: Absolute Laufzeit des Algorithmus in Abhängigkeit von der Anzahl der Kontroll-Punkte. In schwarz eine lineare Trendlinie.

Es gibt natürlich viele Ursachen für die Laufzeit, von denen viele nur schwer messbar sind. Hier die Punkte, welche scheinbar für die größten Performance Probleme sorgen:

- Das Mapping durch Thin-Plate-Splines wird bei jedem hinzugefügten Kontroll-Punkt komplett neu berechnet, ohne bereits berechnete Daten wiederzuverwenden. Je mehr Kontroll-Punkte verwendet werden, desto länger dauert diese neue Berechnung. Wie stark sich die Anzahl der Kontroll-Punkte auf die Geschwindigkeit auswirkt, ist in Abbildung 40 zu sehen. Dieser Punkt hat die stärkste Auswirkung auf die Performance.
- Es werden `shared_pointer` verwendet, deren Speicher automatisch verwaltet wird, anstatt einfache Pointer zu verwenden, deren Speicher gezielt freigegeben werden müsste. Diese werden auch für jeden einzelnen Pixel viermal verwendet.
- Es werden virtuelle Funktionen verwendet, um eine saubere Trennung zwischen technologieunabhängigem Code und technologieabhängigem Code zu schaffen. Durch diese Trennschicht hindurch, werden zulasten der Performance sogar Pixeldaten transferiert.

- Es wird komplett sequentiell gearbeitet und kausal unabhängige Prozesse werden nicht parallel ausgeführt, weder in parallelen Threads, noch wird die GPU benutzt.
- Die Genauigkeit mit der Texel berechnet und verglichen werden. Für diese Evaluation war diese Genauigkeit auf 50 x 50 Pixel eingestellt.
- Das Rendering, welches keinerlei Einfluss auf das Ergebnis hat und nur dem Nachvollziehen und Debuggen dient, läuft derzeit im gleichen Thread wie die Berechnung. Für die Auswertung der Laufzeit wurde es auf ein Minimum gestellt, hatte jedoch noch immer Auswirkungen.

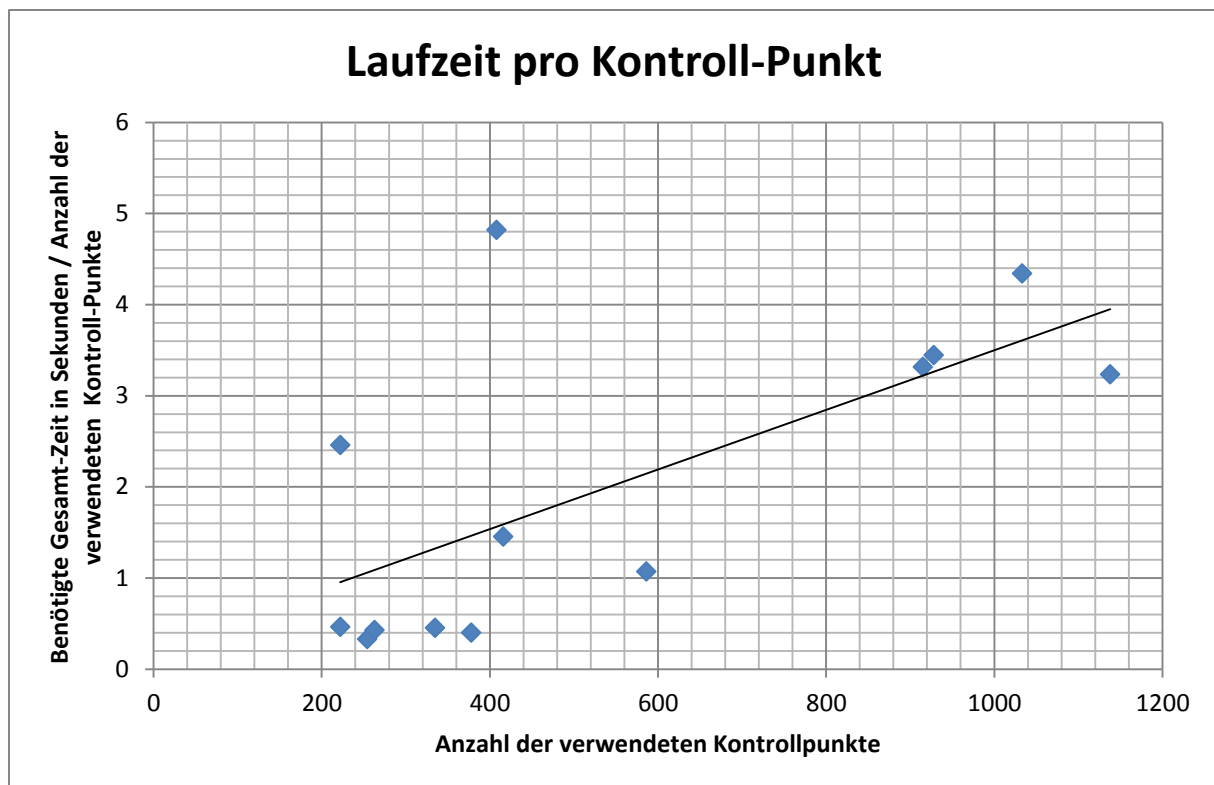


Abbildung 40: Die Laufzeit pro Kontroll-Punkt in Abhängigkeit von der Anzahl der Kontroll-Punkte. Je mehr Kontroll-Punkte benutzt werden, desto langsamer wird der Algorithmus. In schwarz eine lineare Trendlinie.

5.4 Qualitative Bewertung und Ursachen der Probleme

In Abschnitt 5.3 wurden die Ergebnisse quantitativ ausgewertet und erste Hinweise auf Probleme ermittelt. In diesem Abschnitt wird genauer auf die Ursachen dieser Probleme eingegangen und auch sonstige Beobachtungen zum Verhalten des Algorithmus werden hier beschrieben.

5.4.1 Fehlerhafte Eingabedaten

Der Algorithmus basiert und verlässt sich sehr darauf, dass die Mehrheit der gegebenen Feature-Punkt-Cluster korrekt ist. Mit den Test-Daten mit vielen Feature-Punkten an falschen Stellen, oder falschen Clustern, konnte der Algorithmus nicht umgehen und lieferte entsprechend schlechte Ergebnisse.

Beispielsweise hat eines der gegebenen Evaluierungs-Bilder (Tx34) die Feature-Punkte welche eigentlich zu zwei verschiedenen Clustern gehören in einem einzigen Cluster vereint. Ein Ausschnitt der gegebenen Feature-Punkte und der daraus ermittelten Quads ist in Abbildung 41 zu sehen. Der Algorithmus gleicht diesen Fehler nicht aus, und es wurde pauschal die Hälfte der ermittelten Quads als fehlerhaft bewertet, ein Cluster mit vielen Feature-Punkten verworfen und nur ein geringer Teil der gegebenen Feature-Punkte als Kontroll-Punkte benutzt. Dies erklärt das größte der in der in Abschnitt 5.3 gezeigten Probleme.



Abbildung 41: Gehören die gegebenen Feature-Punkte welche eigentlich zu zwei verschiedenen Clustern gehören sollten, nur einem einzigen Cluster an, so gleicht der Algorithmus dieses Problem nicht aus und erzeugt falsche Quads. Das ermittelte Durchschnittstexel und der relative durchschnittliche Quad-Fehler sind in diesem Fall entsprechend unbrauchbar.

5.4.2 Fehlende Extrapolation oder unvollständige Eingabe-Daten

Fehlende Feature-Punkte ergänzt der Algorithmus nur durch Interpolation innerhalb validierter Quads. Durch Extrapolation werden jedoch keine neuen Feature-Punkte zum Rand hinzugefügt, weshalb die Ergebnisse des Algorithmus durch die Vollständigkeit der gegebenen Feature-Punkte begrenzt sind.

So wurden in einem der Test-Bilder (Tx50) nur 10% der gegebenen Feature-Punkte als Kontroll-Punkte verwendet. Dies ist in Abbildung 42 zu sehen.

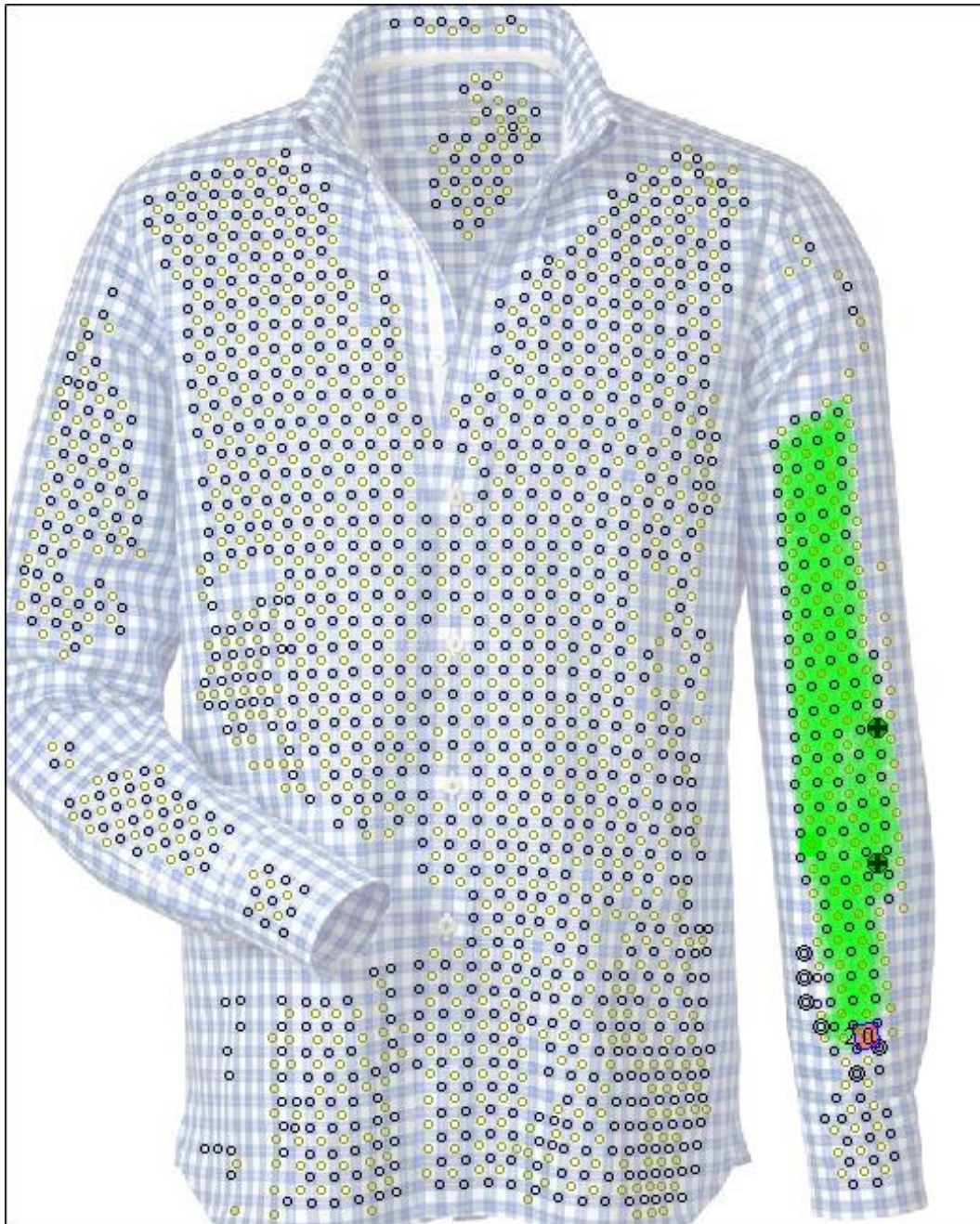


Abbildung 42: Der Algorithmus extrapoliert nicht, um neue Feature-Punkte bzw. Kontroll-Punkte zu ermitteln und kann auch nicht mit Diskontinuitäten umgehen. Daher kann es passieren, dass nur ein geringer Teil der gegebenen Feature-Punkte tatsächlich als Kontroll-Punkte verwendet wird. Dies hängt vor allem vom ermittelten ersten Quad-Paar ab.

5.4.3 Diskontinuitäten in der Topologie

Mit Diskontinuitäten in der Topologie, z.B. Nähten, kann der Algorithmus nicht umgehen. Durch Diskontinuitäten getrennte Flächen können nicht korrekt in einem einzigen Koordinaten-System verbunden sein. Daher benötigt jede der durch Diskontinuitäten

getrennten Flächen ein eigenes topologisches Koordinatensystem und damit ein eigenes Mapping.

Der Algorithmus leistet dies im Moment nicht (Vgl. Abbildung 42). Man könnte ihn jedoch nochmals auf die verbleibenden (nicht zugeordneten) Feature-Punkte eines Bildes anwenden, um auch die Punkte hinter einer Diskontinuität zuzuordnen. Dadurch würden die Feature-Punkte dann verschiedenen topologischen Koordinatensystemen zugeordnet werden.

5.4.4 Verwerfen von Clustern

Die in Abschnitt 3.1 vorgestellte Methode zum initialisieren und aussortieren der Cluster funktionierte auf den 14 Testbildern ausgezeichnet. Es wurden ausschließlich die Cluster verworfen, welche nicht gleichmäßig in allen Quads verteilt sind, sondern welche nur einzelne Punkte am Rand beinhalten. Einer der verworfenen Cluster war sogar komplett falsch.

5.4.5 Wahl des initialen Quad-Paars

Die in Abschnitt 3.2 vorgestellte Methode zum Initialisieren der ersten beiden Quads funktionierte auch sehr gut. Es wurden stets zwei Quads in einem ebenen Bereich gewählt, welche korrekt und minimal waren, also nicht weiter in kleinere Quads zerlegbar.

5.4.6 Nutzung mehrerer Cluster

Auch die Nutzung mehrerer Cluster mittels der Cluster-Offsets (Vgl. Abschnitte 2.6 und 3.3.1) funktionierte problemlos und stellt eine gute Möglichkeit dar fehlende Informationen durch redundante Daten auszugleichen. Abbildung 43 zeigt, wie die Informationen verschiedener Cluster auf einem der Test-Bilder mit einander wechselwirken.

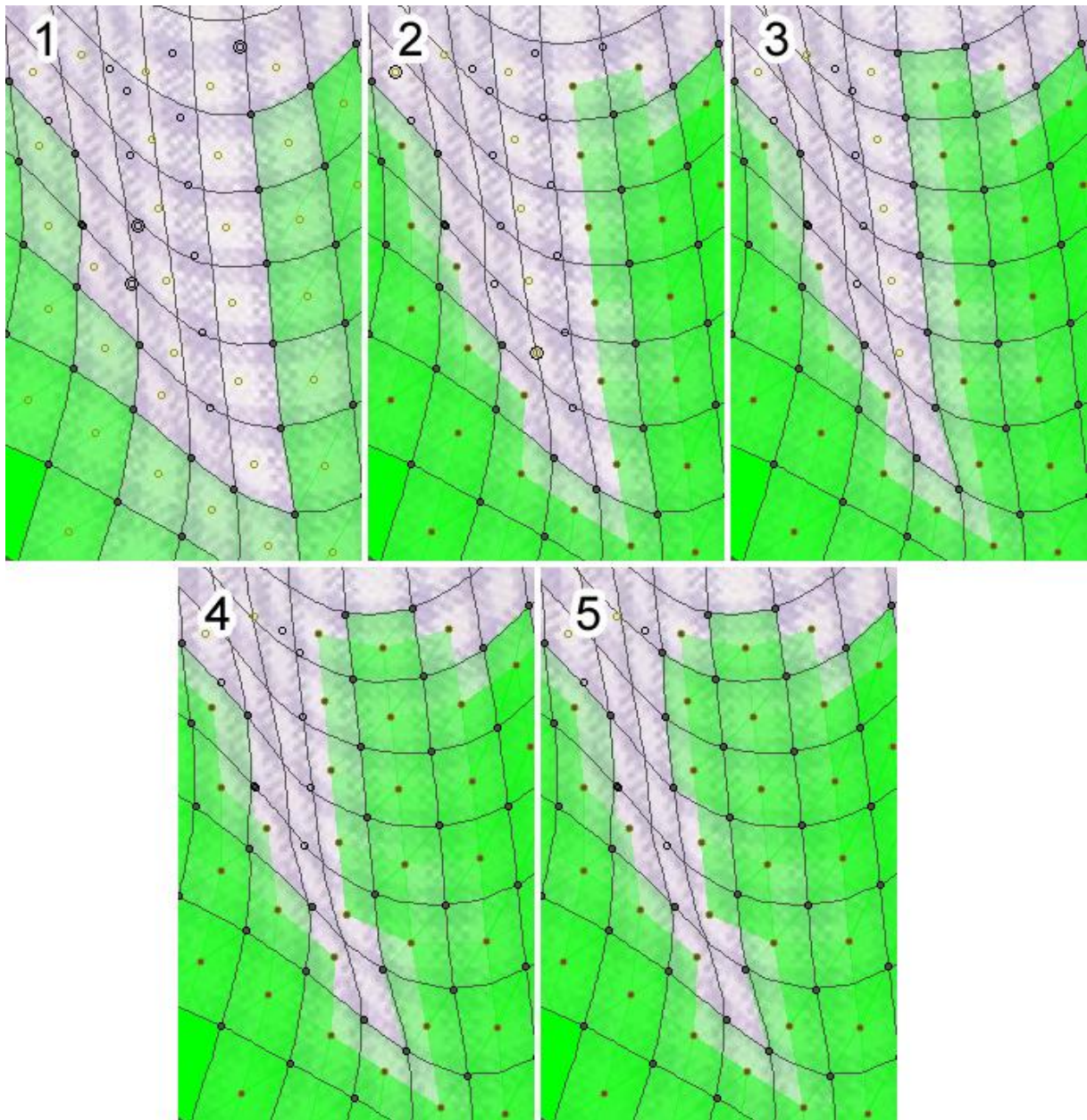


Abbildung 43: Die Cluster-Iteration auf Cluster 1 findet keine weiteren Quads, da diese zu weit vom Erwartungswert abweichen (Bild 1). Die Cluster-Iteration auf Cluster 2 fügt jedoch eine weitere Reihe Quads aus Cluster 2 hinzu, bevor auch hier keine weiteren Quads akzeptiert werden (Bild 2). Nun fügt wieder eine Cluster-Iteration auf Cluster 1 Quads hinzu (Bild 3). Diese Wechselwirkung wiederholt sich auf dem Test-Bild noch mehrmals (Bild 4 und 5).

5.4.7 Globales Mapping

Das globale Mapping (Vgl. Abschnitt 2.2), welches kontinuierlich mit neuen Kontrollpunkten aktualisiert wird, erfüllt seinen Zweck relativ gut. Der Hauptvorteil liegt hierbei in der einheitlichen Behandlung vieler verschiedener Teilaufgaben. So wird ein einziger Mechanismus benutzt, um damit zu interpolieren, zu extrapolieren, Quads vom Bildraum in

den topologischen Raum zu transformieren und umgekehrt. Dabei werden alle Abstände und Schwellwerte im topologischen Raum, anstatt im Bildraum gemessen, wodurch der Algorithmus auf allen Bildern mit den gleichen Schwellwerten arbeiten kann. Das Arbeiten im topologischen Raum funktionierte auf allen Test-Bildern gut.

5.4.8 Quads und Texel als elementare Größe

Die in den Abschnitten 2.3 und 2.4 vorgestellte Idee Feature-Punkte erst als valid zu akzeptieren, wenn das umschlossene Quad anhand eines Durchschnitts-Texels validiert ist, scheint auch zielführend zu sein, jedoch nur wenn die Texel-Vergleichs-Methode entsprechend gute Ergebnisse liefert. Es wurden auch Vorgehensweisen getestet, welche nicht jedes Mal Bilddaten miteinander vergleichen, sondern welche ausschließlich geometrisch oder topologisch versuchen zu bestimmen, ob ein Feature-Punkt valid ist oder nicht. Die Texel der Quads während der Suche nach neuen Quads zu vergleichen, lieferte jedoch deutlich bessere Ergebnisse.¹⁶

5.4.9 Texel-Vergleichs-Methode

Im Abschnitt 3.5 wurden bereits einige Vor- und Nachteile der hier vorgestellten Texel-Vergleichs-Methode genannt. Insgesamt liefert sie durchwachsene Ergebnisse. Der Stoff in einem der Testbilder hatte ein Muster mit gleichen Farben, jedoch unterschiedlichen Helligkeiten. Die verwendete Texel-Vergleichs-Methode konnte die falschen und richtigen Quads in diesem Bild nahezu nicht unterscheiden. Des Weiteren hatte die Texel-Vergleichs-Methode oft Probleme, wenn das Quad zwar korrekt war, jedoch in sich verzerrt. So wurden korrekte verzerrte Quads (Vgl. Abbildung 36), oft als invalid angesehen.

5.4.10 Thin-Plate-Splines als Mapping-Methode

Die benutzten Thin-Plate-Splines als Mapping-Methode hatten Vor- und Nachteile. Sie approximieren die reale Verformung von Stoffen eher mittelmäßig. So approximierten die TPS vor allem an den Stellen an denen der Stoff stark gebogen ist, die reale Biegung im Bild recht schlecht. Dies ist auch eine der Ursachen für die verzerrten Quads (Vgl. Abbildung 36), an welchen oft die Texel-Vergleichs-Methode scheiterte. Auch beim Extrapolieren lieferten die Thin-Plate-Splines nur mäßige Ergebnisse, wie in Abbildung 44 gezeigt.

¹⁶ Auch in [2] werden Bilddaten und nicht nur geometrische Daten verwendet. Jedoch nicht während der Suche nach neuen Quads, sondern beim Initialisieren.

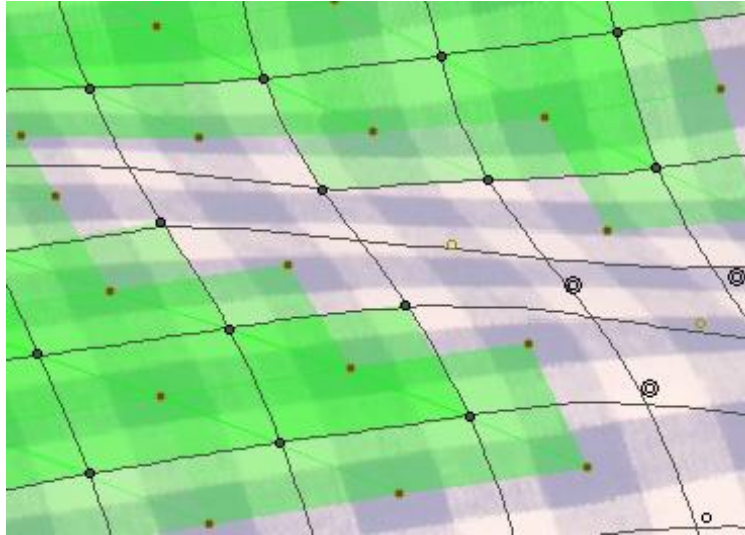


Abbildung 44: Obwohl viele Kontroll-Punkte (ausgefüllte Kreise) in der Nähe gegeben sind, wird die reale Topologie durch die Thin-Plate-Splines nur ungenau approximiert (schwarze Linien).

5.4.11 Ergänzen neuer Feature-Punkte und Quads

In Abschnitt 5.3.1 wurde festgestellt, dass das Hinzufügen neuer Kontroll-Punkte (vgl. Abschnitt 3.3.2) nur wenige neue Punkte und teils falsche Punkte ermittelt.

Dass nur ca. 1% der verwendeten Kontroll-Punkte neu hinzugefügte Kontroll-Punkte sind, hat zwei Gründe. Erstens extrapoliert der Algorithmus nicht, sondern interpoliert nur. Das heißt es werden nur Punkte zwischen bereits gegebenen Punkten ergänzt, jedoch nicht außerhalb. Zweitens sind die gegebenen Feature-Punkte in den verarbeiteten Bildregionen fast vollständig. Das heißt es fehlen sowieso kaum Punkte die durch Interpolation ergänzt werden könnten.

Die Ursache für die fehlerhaft hinzugefügten Feature-Punkte ist jedoch nicht das Hinzufügen selbst, sondern, dass zuvor bereits falsche Quads akzeptiert wurden. In diesen falschen Quads werden dann neue Kontroll-Punkte hinzugefügt, die selbst auch falsch sind. Für das Akzeptieren falscher Quads wurden wiederum zwei Hauptursachen beobachtet. Zum einen ist die verwendete Texel-Vergleichs-Methode (vgl. Abschnitt 3.5) nicht perfekt und akzeptiert deshalb in einigen Fällen auch falsche Quads. Zum anderen werden oft fehlerhafte Quads akzeptiert, wenn die vier Eckpunkte bereits als korrekt akzeptiert wurden (vgl. Abschnitt 3.3.3). Ein Beispiel dafür ist in Abbildung 45 zu sehen.

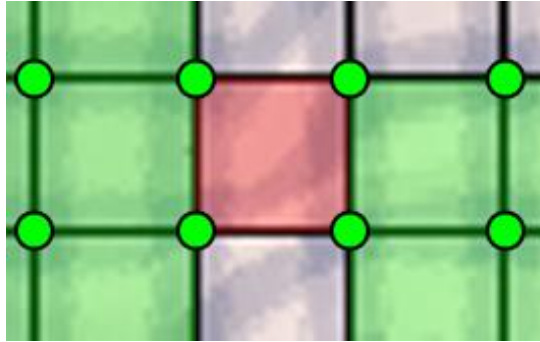


Abbildung 45: Wurden umliegende Feature-Punkte akzeptiert (grüne Kreise), so wird das dazwischenliegende Quad (rot) akzeptiert, auch wenn es falsch ist.

Daher kann man sagen, dass das Hinzufügen neuer Quads, wie in Abschnitt 3.3.3 beschrieben, zu wenig Nutzen und zu viele Fehler mit sich bringt und daher in Weiterentwicklungen des Algorithmus besser komplett weggelassen werden sollte.

Das Hinzufügen neuer Kontroll-Punkte funktionierte, für sich betrachtet, ohne Fehler. Es brachte jedoch auf den verwendeten Test-Daten wenig Nutzen.

6. Zusammenfassung und Ausblick

Dieser Abschnitt fasst die vorgestellte Problemlösung noch einmal zusammen, und zeigt Möglichkeiten für weitere Verbesserungen und Forschung auf.

6.1 Zusammenfassung

Es wurde ein Verfahren entwickelt, welches aus einem gegebenen Bild und gegebenen Clustern mit Feature-Punkten eine zugrundeliegende topologische Struktur ermittelt. Dieses Verfahren arbeitet iterativ, nutzt dabei jedoch globale, bereits ermittelte Informationen.

In ersten Tests erzielte es vielversprechende Ergebnisse, bedarf jedoch noch weiterer Verbesserung.

Ein besonderer Vorteil des Verfahrens liegt dabei in der Nutzung der Bild-Informationen bei der Suche nach neuen Quads, zusätzlich zu den geometrischen Informationen, welche durch die Feature-Punkte gegeben sind. Außerdem wird ein einheitliches Verfahren, zur Extrapolation, Interpolation und Transformation zwischen Bild- und topologischem Raum benutzt. Das Verfahren arbeitet vollautomatisch, stellt jedoch einige Parameter und

austauschbare Teilverfahren zur Optimierung bereit. Es findet auch eine Wechselwirkung zwischen Feature-Punkten verschiedener Cluster statt, welche in einigen Fällen positive Einflüsse auf Gesamtergebnis hatte.

Es gibt auch einige Nachteile, welche jedoch durch Weiterentwicklung gemindert oder eliminiert werden können, ohne das Grundprinzip des Verfahrens zu ändern. So hat die erste Implementierung des Algorithmus sehr lange Laufzeiten. Des Weiteren kann sie nicht mit Diskontinuitäten umgehen, hat Probleme mit einfarbigen Mustern (mit verschiedenen Helligkeitsstufen) und extrapoliert nur schlecht über die gegebenen Feature-Punkte hinaus. Auch fehlerhafte Eingabedaten werden nur teilweise wieder ausgeglichen.

Das Verfahren ist bereits in einem größeren Kontext nutzbar, jedoch wird empfohlen, erst die Schwachstellen zu verbessern, bevor es endgültig eingesetzt wird. Die Implementierung ist direkt ausführbar und durch externe Konfigurationsdatei und externe Eingabedaten muss der Programmcode nicht modifiziert oder verstanden werden, falls man das Verfahren benutzen möchte wie es ist.

6.2 Ausblick

6.2.1 Laufzeit

Um die Laufzeit zu verbessern, könnte eine iterative arbeitende Alternative zu den verwendeten Thin-Plate-Splines gesucht werden, welche Daten aus bereits bekannten Kontroll-Punkten wiederverwendet. Danach könnten eine parallele Bearbeitung einzelner Teilschritte durch Threads und GPU untersucht werden. Eine Auflösung der smart_pointer und virtuellen Methoden, wird erst als letzter Schritt empfohlen, da dieser zulasten der Lesbarkeit und Wartbarkeit des Codes ginge.

6.2.2 Texel-Vergleichs-Methode

Die Texel-Vergleichs-Methode ist essentiell für die Qualität des Ergebnisses. Daher führt eine bessere Texel-Vergleichs-Methode zwangsläufig zu besseren Ergebnissen. Zum einen sollte noch darüber nachgedacht werden, wie man mit gleichen Farben in unterschiedlichen Helligkeiten umgeht. Zum anderen könnte man die Texel-Vergleichs-Methode so verbessern, dass sie gegebene Texel noch weiter entzerrt, als es durch das Mapping bisher passiert. So könnte man auf den Texeln nochmals Feature-Punkte suchen und mittels eines feineren

Mappings für jeden Texel-Vergleich die verglichenen Texel entzerren. Man könnte auch untersuchen, welche bekannten Methoden der Bildregistrierung ([21]) diesem Zweck am besten dienen. So nutzen z.B. [8] die normalisierte Kreuzkorrelation um Texel zu vergleichen.

Die Texel-Vergleichs-Methode kann auch völlig unabhängig vom hier vorgestellten Algorithmus untersucht werden.

6.2.3 Extrapolation und Diskontinuitäten

Die hier benutzte Extrapolation und Ergänzung zusätzlicher Feature-Punkte ist sehr dürftig, jedoch essentiell, um ein Bild vollständig zu analysieren. Auch der genaue Umgang mit Diskontinuitäten wird bisher nicht weiter behandelt. Das Problem der Diskontinuitäten könnte man schnell dadurch lösen, dass nach einem Durchlauf nochmals auf den noch nicht zugeordneten Feature-Punkten ausgeführt wird, um getrennte Flächen einzeln zuzuordnen. Das Problem der Extrapolation sollte erst nach der Verbesserung der Texel-Vergleichs-Methode stattfinden, da man diese gut benutzen könnte, um neue korrekte Quads zu extrapolieren.

6.2.4 Fehlerhafte Eingabedaten

In wieweit fehlerhafte Eingabedaten im Rahmen dieses Verfahrens ausgeglichen werden sollten und welche Fehler bereits bei der Erstellung der Daten bereinigt werden sollten, ist im Einzelnen im Kontext zu klären. Jedoch sollte eine saubere Kommunikation zwischen dem Verfahren welches die Eingabe-Daten bereit stellt und dem hier vorgestellten Verfahren geklärt sein. Diese sollte nicht nur klären in welcher Form (Text-Datei, Zeilenumbrüche um Werte zu trennen, usw.) die Kommunikation stattfindet, sondern auch genau die Verantwortungen klären, welcher Algorithmus für welche Fehler und Probleme zuständig ist.

Anhang: Ergebnisse

Die Ergebnis-Bilder hier im Anhang sind zwar mit Legenden versehen, jedoch wird nicht jede einzelne Abbildung erklärt. Die Ergebnis-Bilder hier im Anhang werden analog zu den exemplarischen Abbildungen aus Abschnitt 5.2 dargestellt und können zum besseren Verständnis mit diesen verglichen werden.

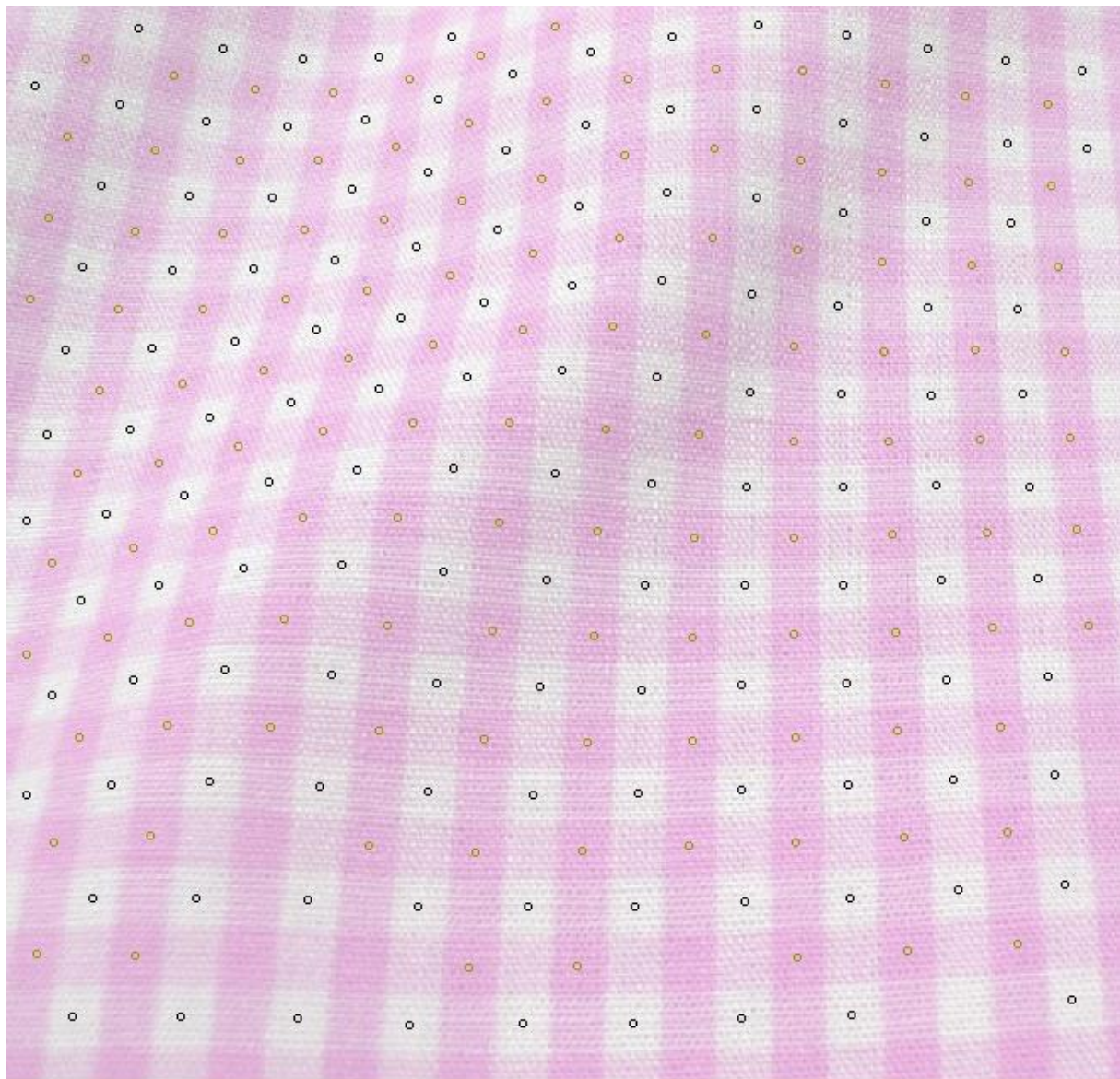
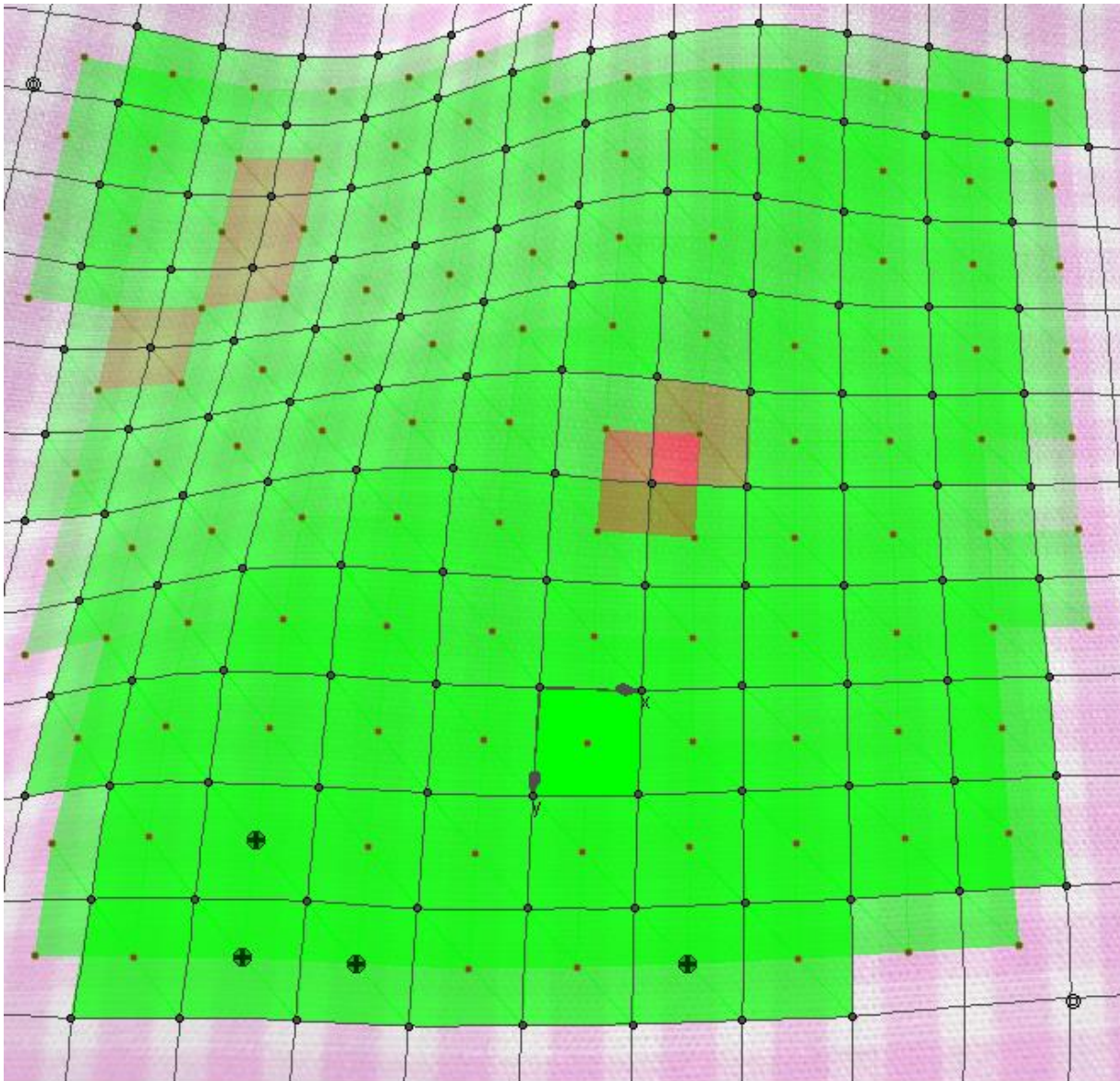



Abbildung 46: Bild „Tx1“



Legende

- ○ gegebene Feature-Punkte
(eine Farbe pro Cluster)
- ● verwendete Kontroll-Punkte
(eine Farbe pro Cluster)
- ⊕ neu erstellte Kontroll-Punkte
-  topologische Koordinaten mit
ganzzahliger x- oder y-Koordinate


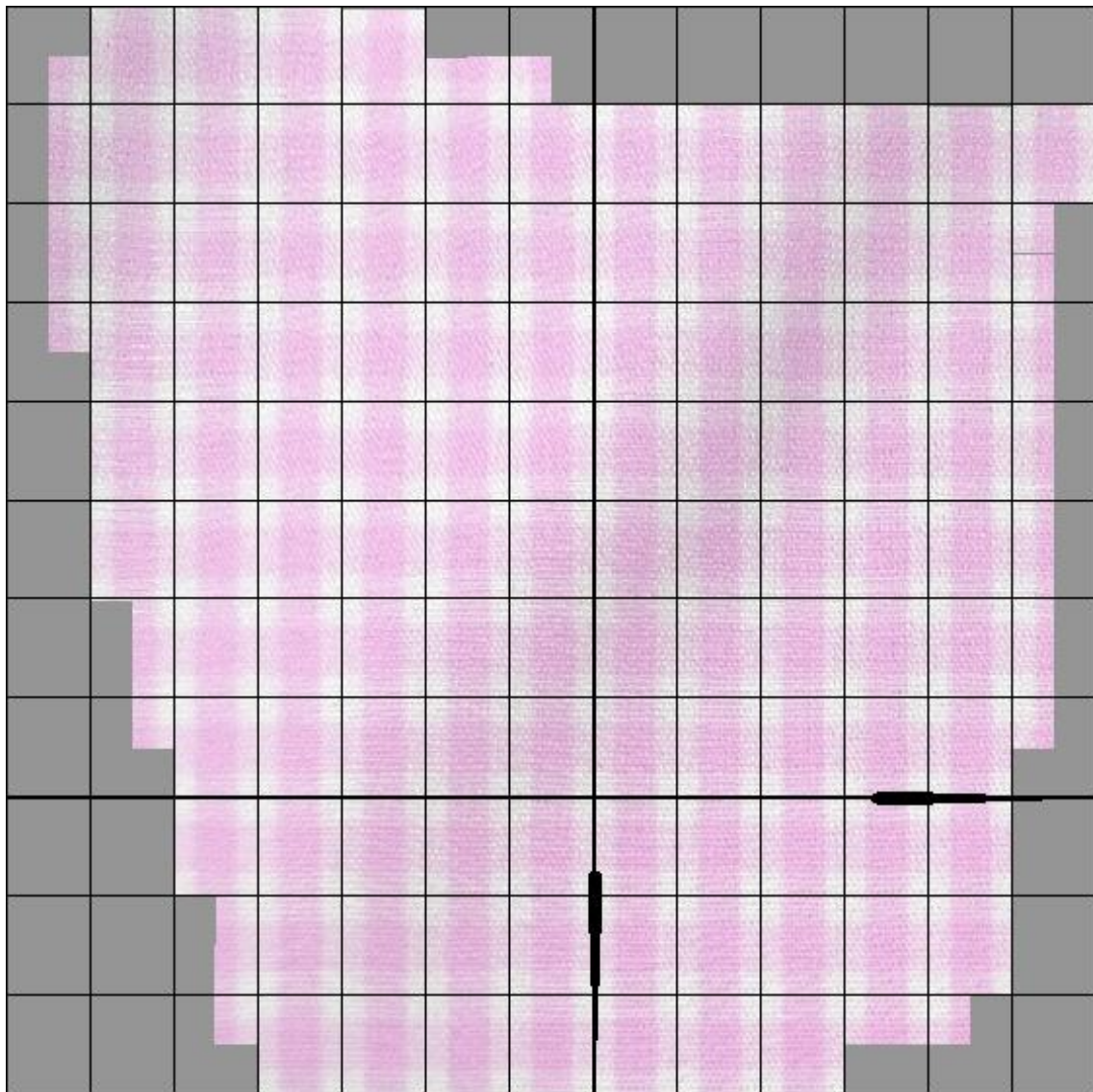
-  akzeptierte Quads. Farbe und Transparenz variieren je
nach relativem Fehler der Quads:
- rot, deckend: Fehler > Schwellwert,
höherer Fehler
- rot, transparent: Fehler > Schwellwert,
niedrigerer Fehler
- grün, transparent: Fehler ≤ Schwellwert,
höherer Fehler
- grün, deckend: Fehler ≤ Schwellwert,
niedrigerer Fehler

Abbildung 47: Bild „Tx1“



Legende

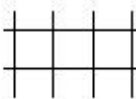
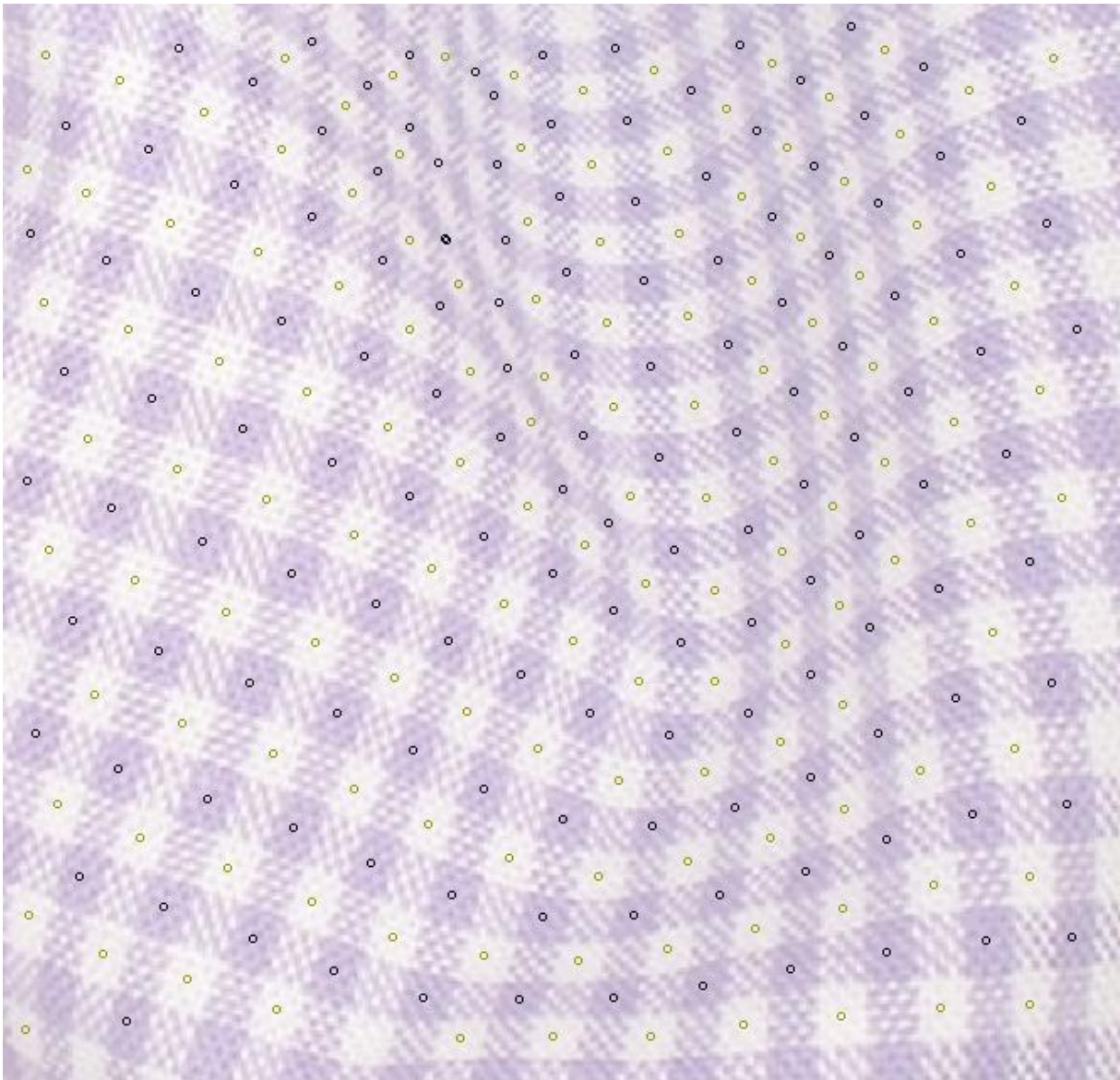

 topologische Koordinaten
 mit ganzzahliger x- oder y-
 Koordinate

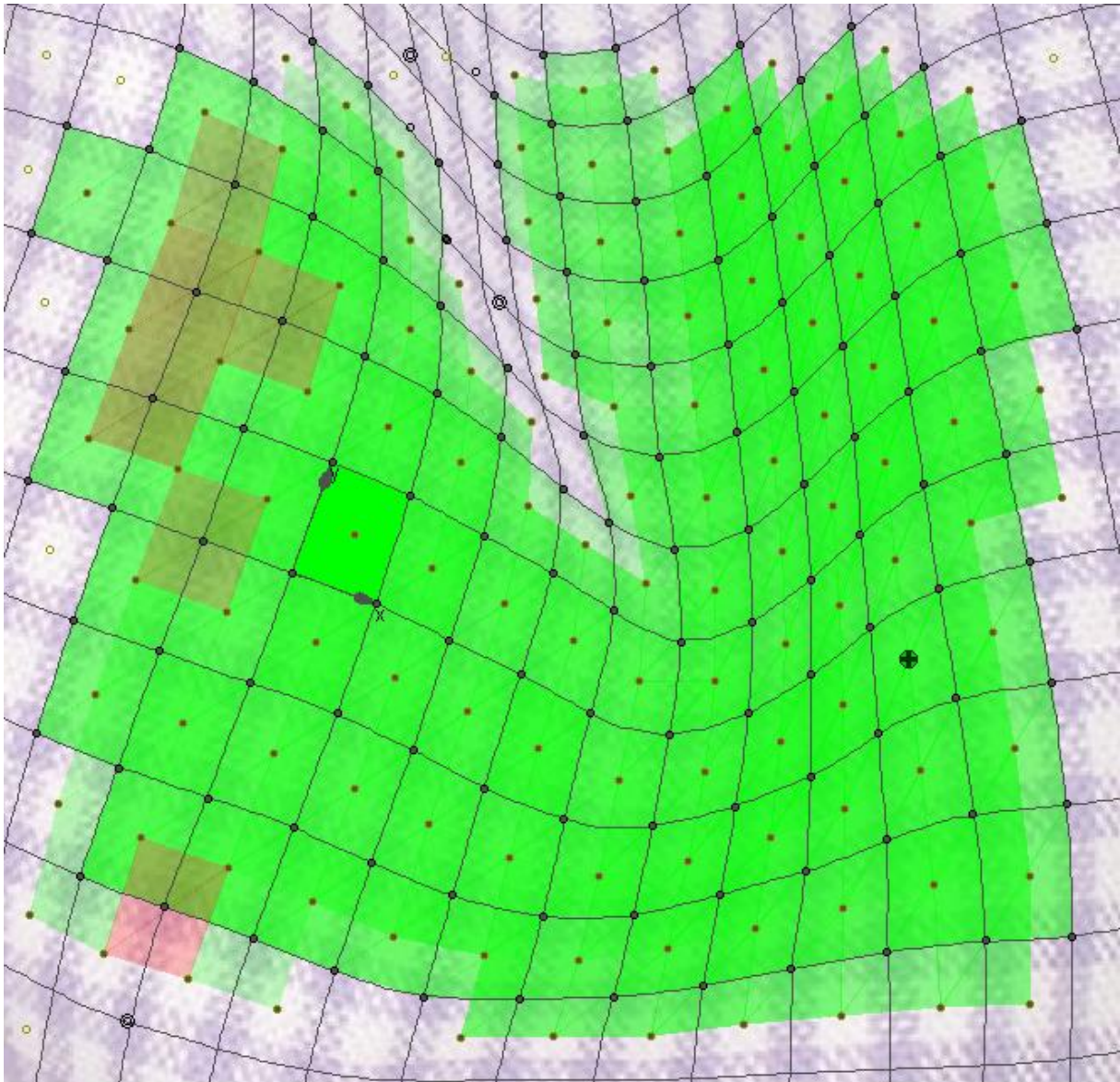
Abbildung 48: Bild „Tx1“



Legende

- ○ gegebene Feature-Punkte
(eine Farbe pro Cluster)

Abbildung 49: Bild „Tx2“



Legende

○ ○ gegebene Feature-Punkte
(eine Farbe pro Cluster)

● ● verwendete Kontroll-Punkte
(eine Farbe pro Cluster)

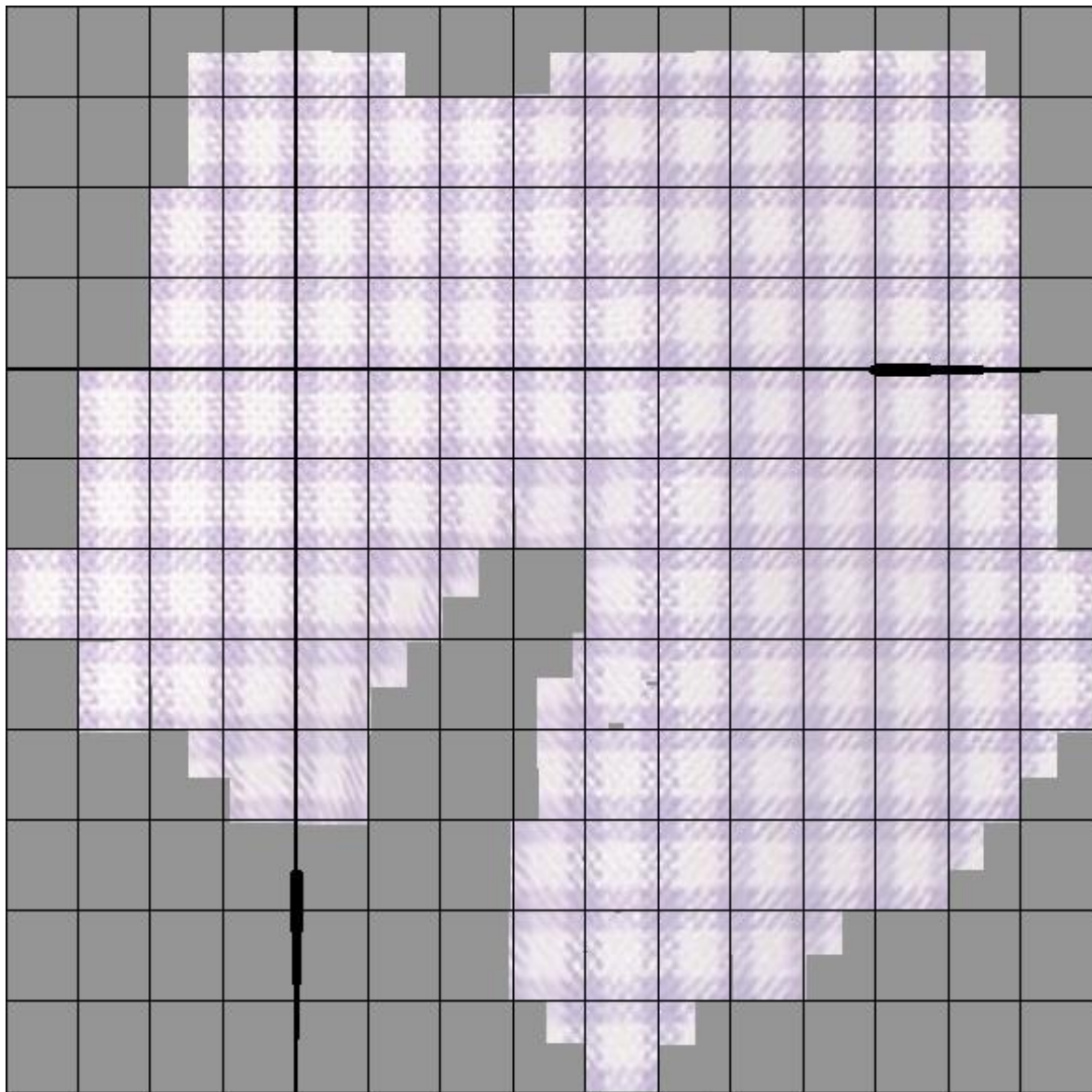
⊕ neu erstellte Kontroll-Punkte

⌌ topologische Koordinaten mit
ganzzahliger x- oder y-Koordinate

akzeptierte Quads. Farbe und Transparenz variieren je nach relativem Fehler der Quads:

- rot, deckend: Fehler > Schwellwert, höherer Fehler
- rot, transparent: Fehler > Schwellwert, niedrigerer Fehler
- grün, transparent: Fehler ≤ Schwellwert, höherer Fehler
- grün, deckend: Fehler ≤ Schwellwert, niedrigerer Fehler

Abbildung 50: Bild „Tx2“



Legende

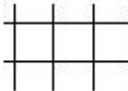
 topologische Koordinaten
 mit ganzzahliger x- oder y-
 Koordinate

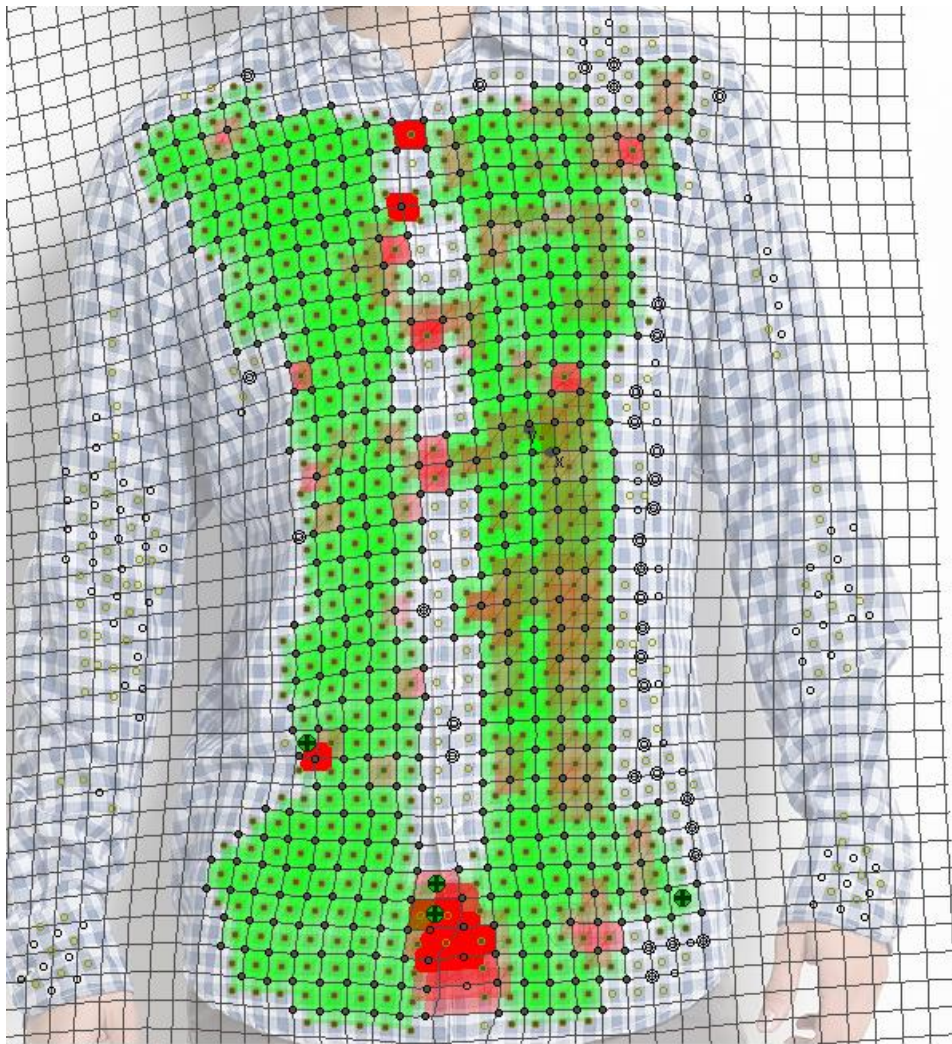
Abbildung 51: Bild „Tx2“



Legende

- o o gegebene Feature-Punkte
(eine Farbe pro Cluster)

Abbildung 52: Bild „Tx7“

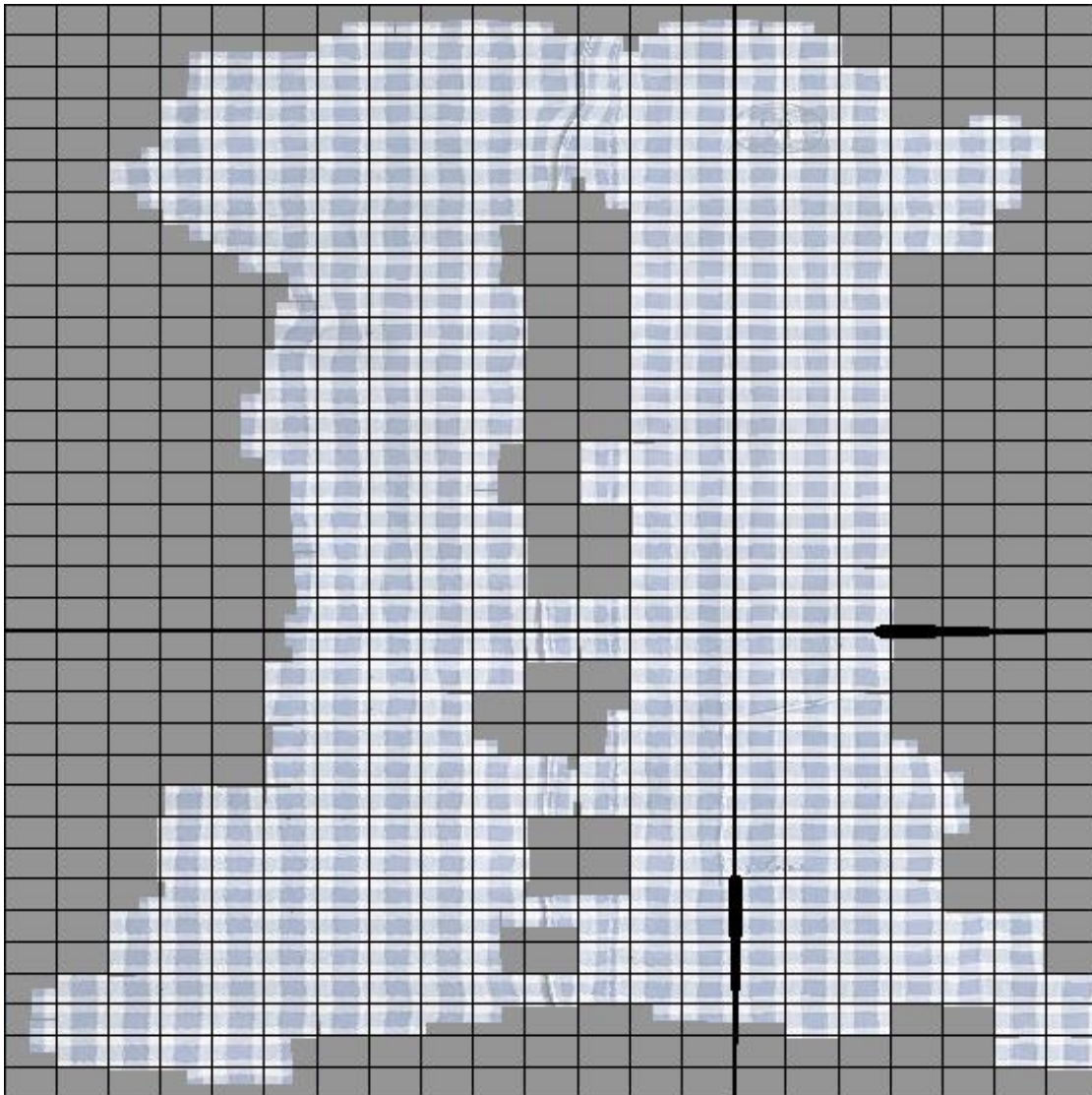


Legende

- ○ gegebene Feature-Punkte
(eine Farbe pro Cluster)
- ● verwendete Kontroll-Punkte
(eine Farbe pro Cluster)
- ⊕ neu erstellte Kontroll-Punkte
- ⌵ topologische Koordinaten mit
ganzzahliger x- oder y-Koordinate

- akzeptierte Quads. Farbe und Transparenz
variieren je nach relativem Fehler der Quads:
- rot, deckend: Fehler > Schwellwert,
höherer Fehler
 - rot, transparent: Fehler > Schwellwert,
niedrigerer Fehler
 - grün, transparent: Fehler ≤ Schwellwert,
höherer Fehler
 - grün, deckend: Fehler ≤ Schwellwert,
niedrigerer Fehler

Abbildung 53: Bild „Tx7“



Legende

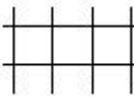
 topologische Koordinaten
mit ganzzahliger x- oder y-
Koordinate

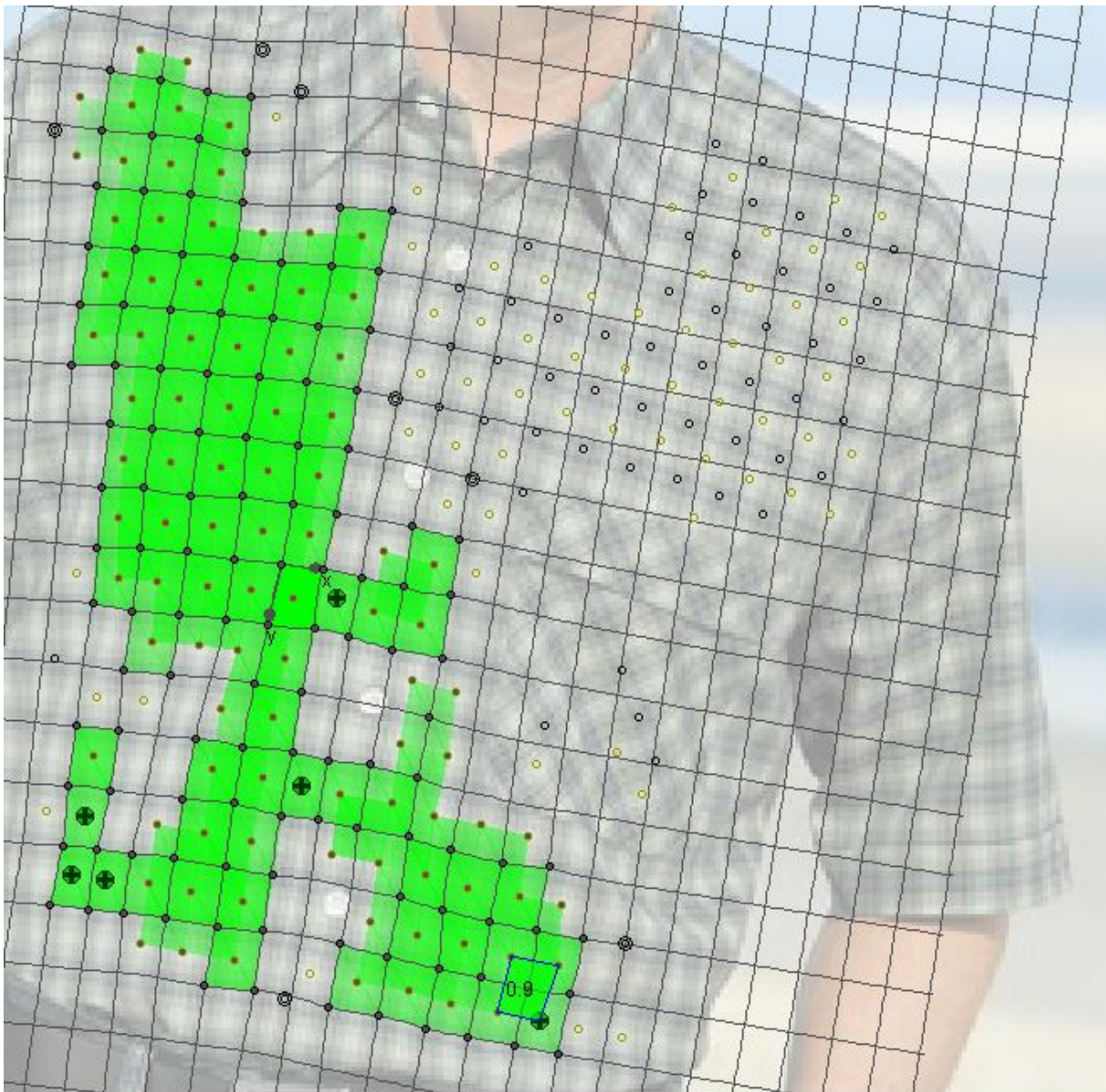
Abbildung 54: Bild „Tx7“



Legende

- ● gegebene Feature-Punkte
(eine Farbe pro Cluster)

Abbildung 55: Bild „Tx9“



Legende

○ gegebene Feature-Punkte
(eine Farbe pro Cluster)

● verwendete Kontroll-Punkte
(eine Farbe pro Cluster)

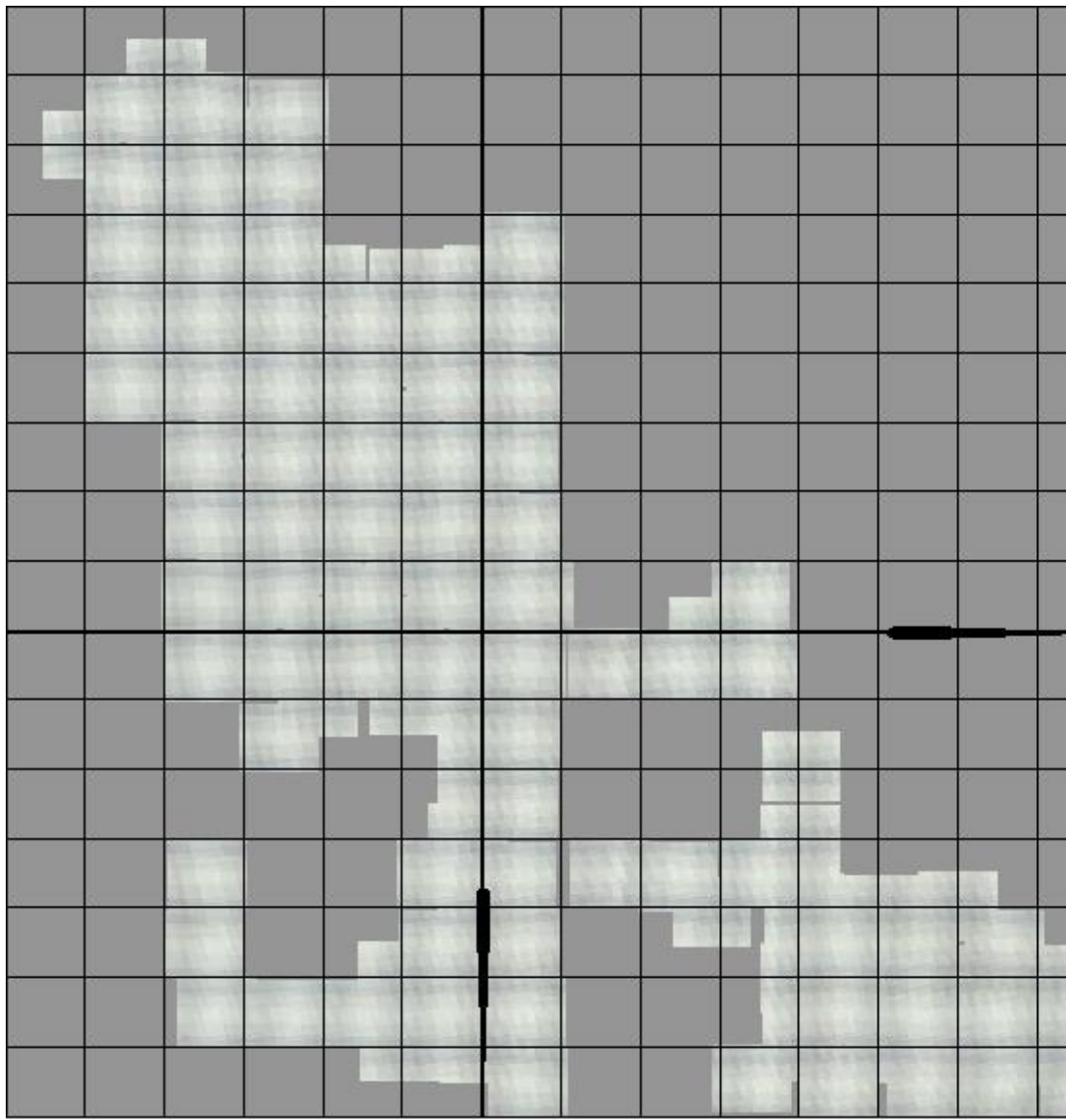
⊕ neu erstellte Kontroll-Punkte

⌘ topologische Koordinaten mit
ganzzahliger x- oder y-Koordinate

akzeptierte Quads. Farbe und Transparenz variieren je nach relativem Fehler der Quads:

- rot, deckend: Fehler > Schwellwert, höherer Fehler
- rot, transparent: Fehler > Schwellwert, niedrigerer Fehler
- grün, transparent: Fehler ≤ Schwellwert, höherer Fehler
- grün, deckend: Fehler ≤ Schwellwert, niedrigerer Fehler

Abbildung 56: Bild „Tx9“



Legende

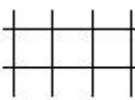

 topologische Koordinaten
 mit ganzzahliger x- oder y-
 Koordinate

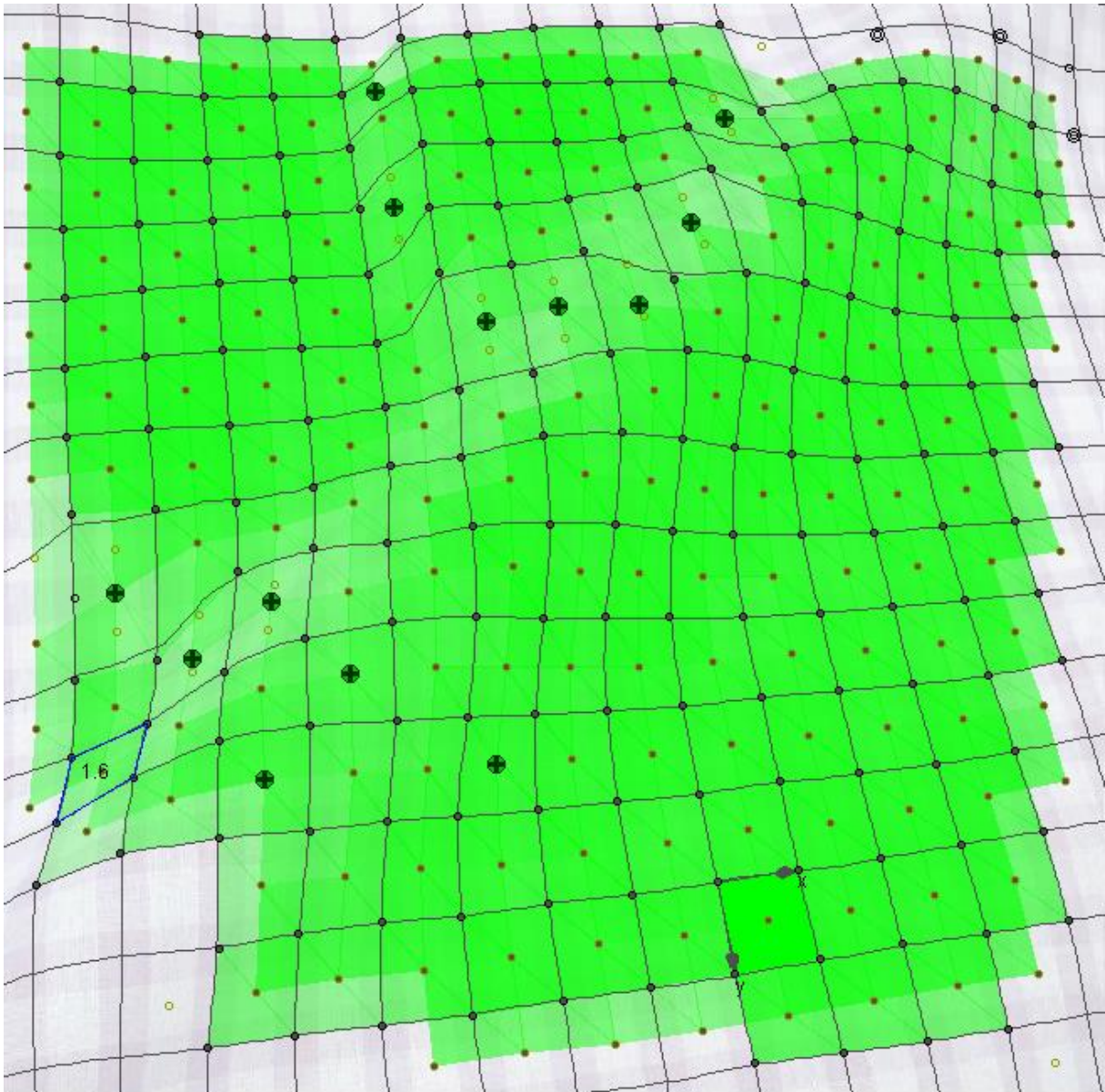
Abbildung 57: Bild „Tx9“



Legende

- gegebene Feature-Punkte
(eine Farbe pro Cluster)

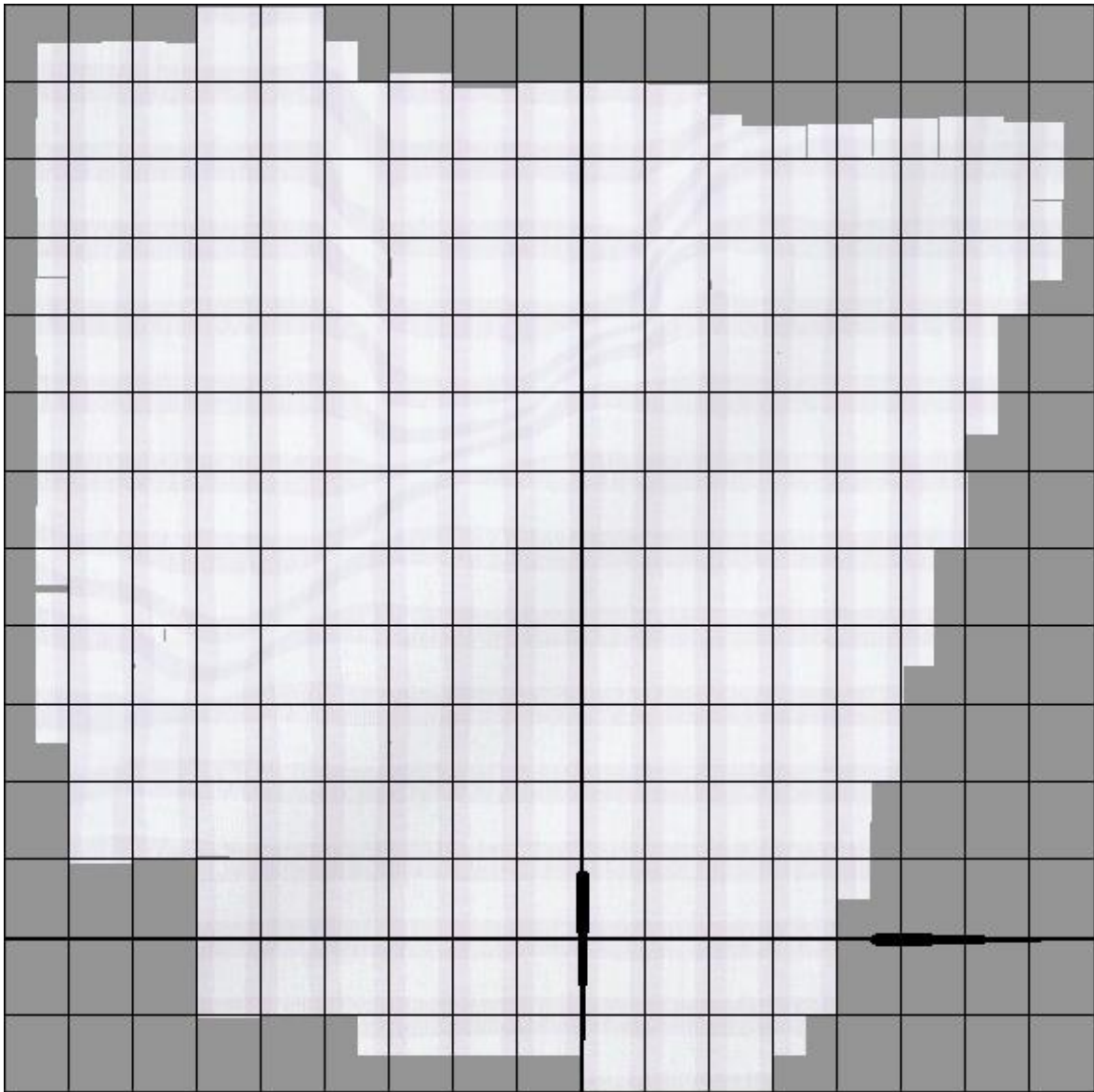
Abbildung 58: Bild „Tx12“



Legende

- ○ gegebene Feature-Punkte
(eine Farbe pro Cluster)
 - ● verwendete Kontroll-Punkte
(eine Farbe pro Cluster)
 - ⊕ neu erstellte Kontroll-Punkte
 - ⌌ topologische Koordinaten mit
ganzzahliger x- oder y-Koordinate
- | | |
|---|--|
|  | <p>akzeptierte Quads. Farbe und Transparenz variieren je nach relativem Fehler der Quads:</p> <ul style="list-style-type: none"> - rot, deckend: Fehler > Schwellwert, höherer Fehler - rot, transparent: Fehler > Schwellwert, niedrigerer Fehler - grün, transparent: Fehler ≤ Schwellwert, höherer Fehler - grün, deckend: Fehler ≤ Schwellwert, niedrigerer Fehler |
|---|--|

Abbildung 59: Bild „Tx12“



Legende

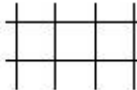

 topologische Koordinaten
 mit ganzzahliger x- oder y-
 Koordinate

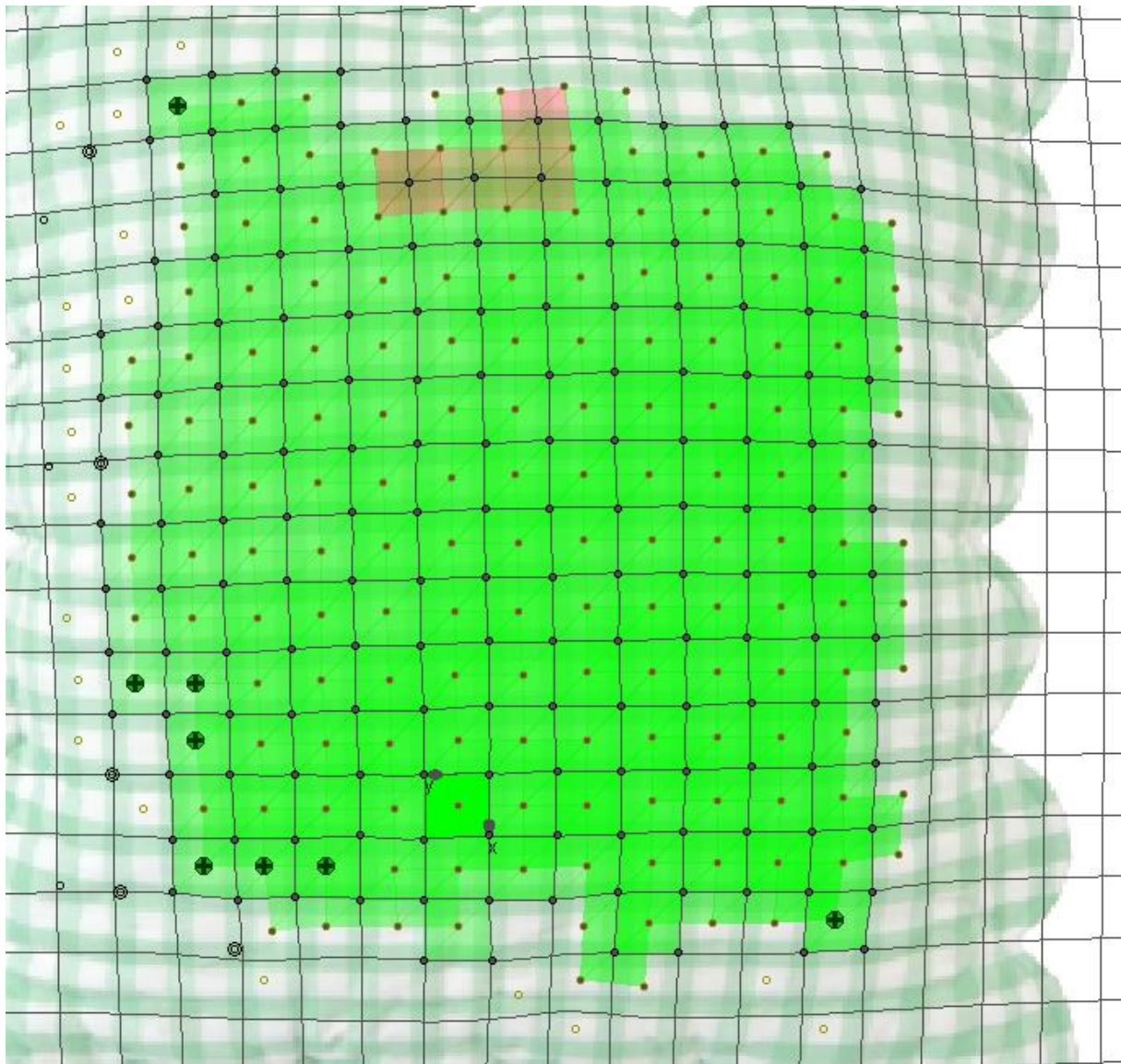
Abbildung 60: Bild „Tx12“



Legende

- o o gegebene Feature-Punkte
(eine Farbe pro Cluster)

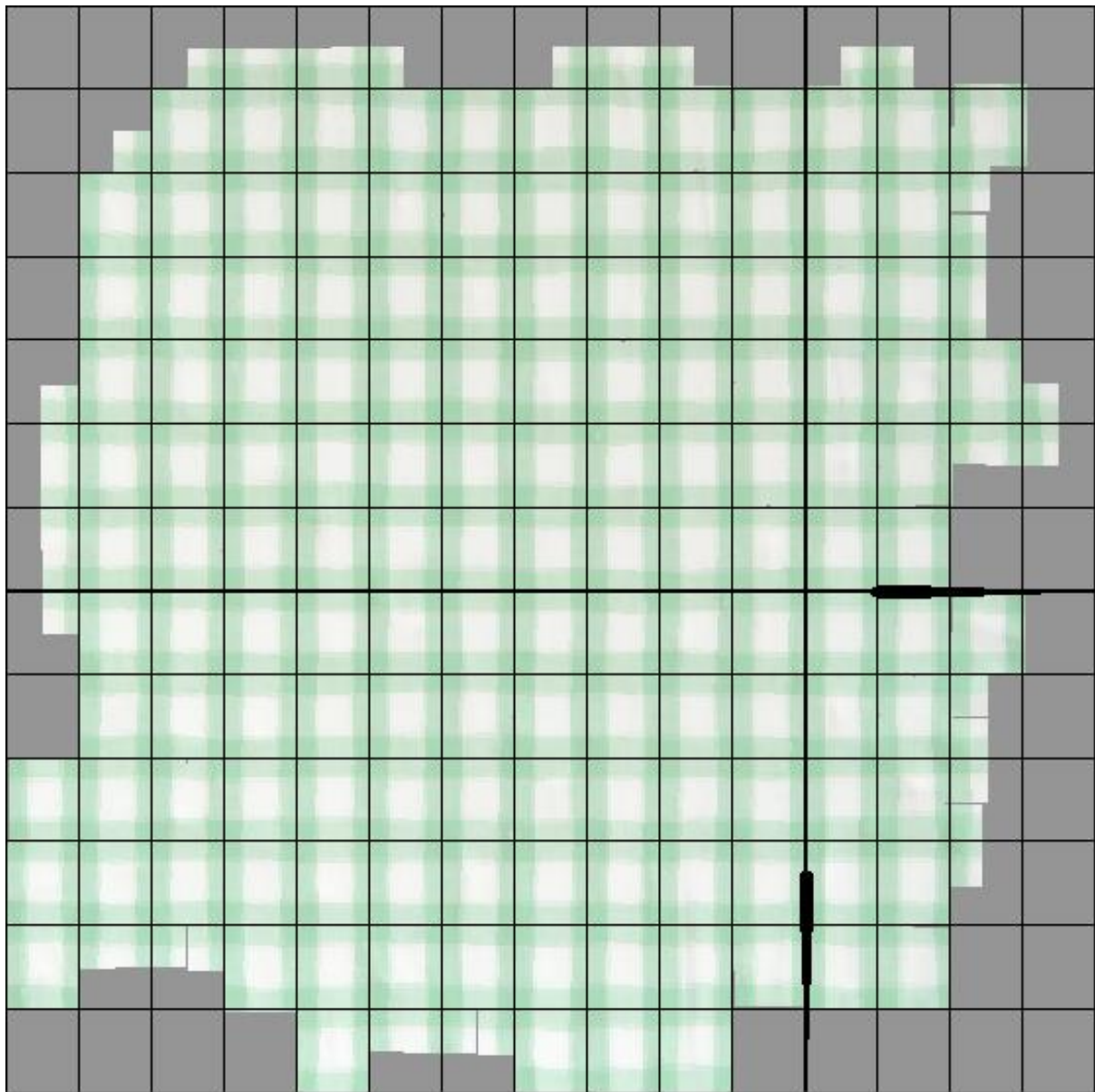
Abbildung 61: Bild „Tx14“



Legende

- ○ gegebene Feature-Punkte (eine Farbe pro Cluster)
 - ● verwendete Kontroll-Punkte (eine Farbe pro Cluster)
 - + neu erstellte Kontroll-Punkte
 -  topologische Koordinaten mit ganzzahliger x- oder y-Koordinate
- | | |
|---|--|
|  | akzeptierte Quads. Farbe und Transparenz variieren je nach relativem Fehler der Quads: |
| - rot, deckend: | Fehler > Schwellwert, höherer Fehler |
| - rot, transparent: | Fehler > Schwellwert, niedrigerer Fehler |
| - grün, transparent: | Fehler ≤ Schwellwert, höherer Fehler |
| - grün, deckend: | Fehler ≤ Schwellwert, niedrigerer Fehler |

Abbildung 62: Bild „Tx14“



Legende

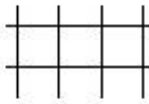
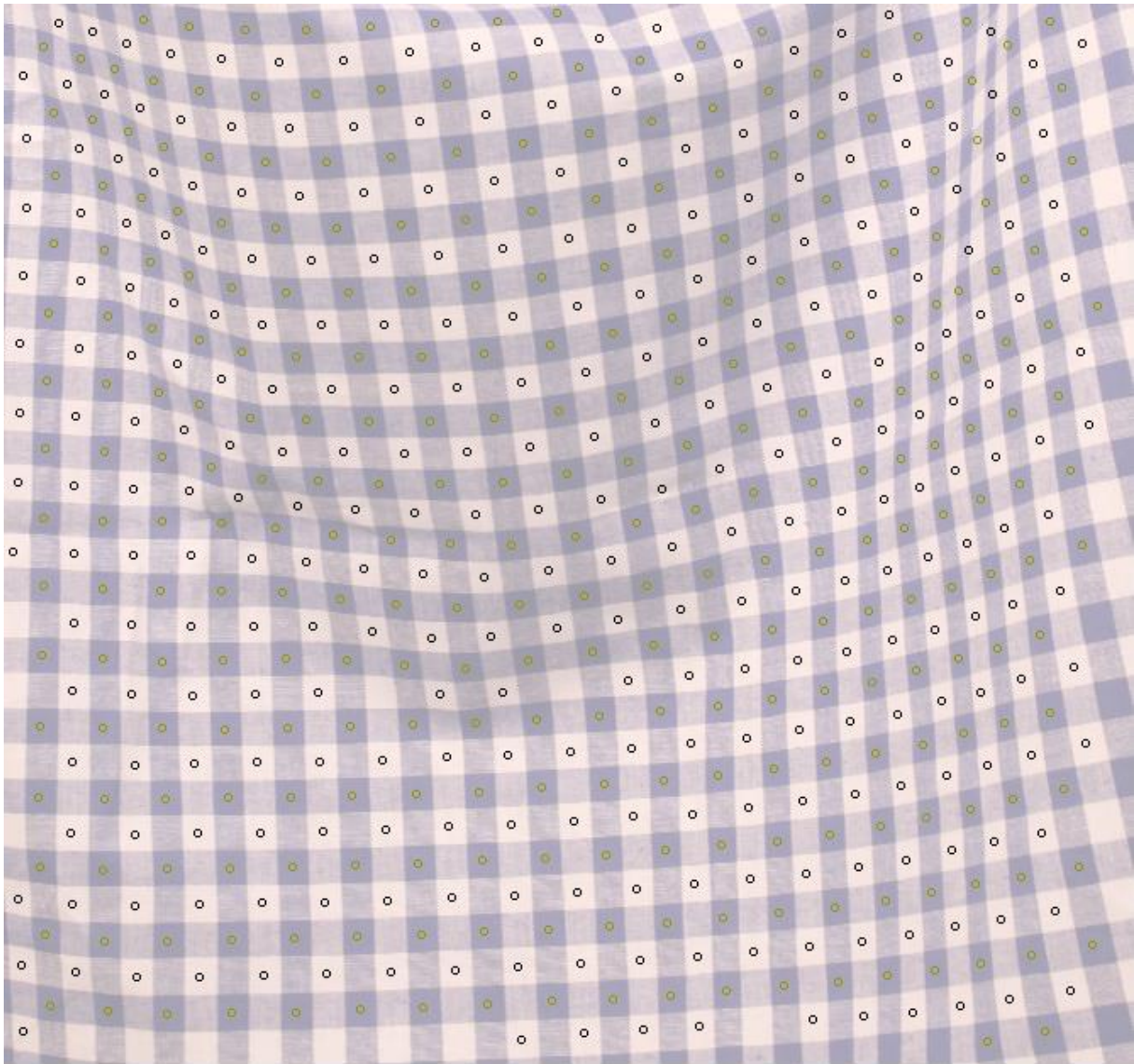

 topologische Koordinaten
 mit ganzzahliger x- oder y-
 Koordinate

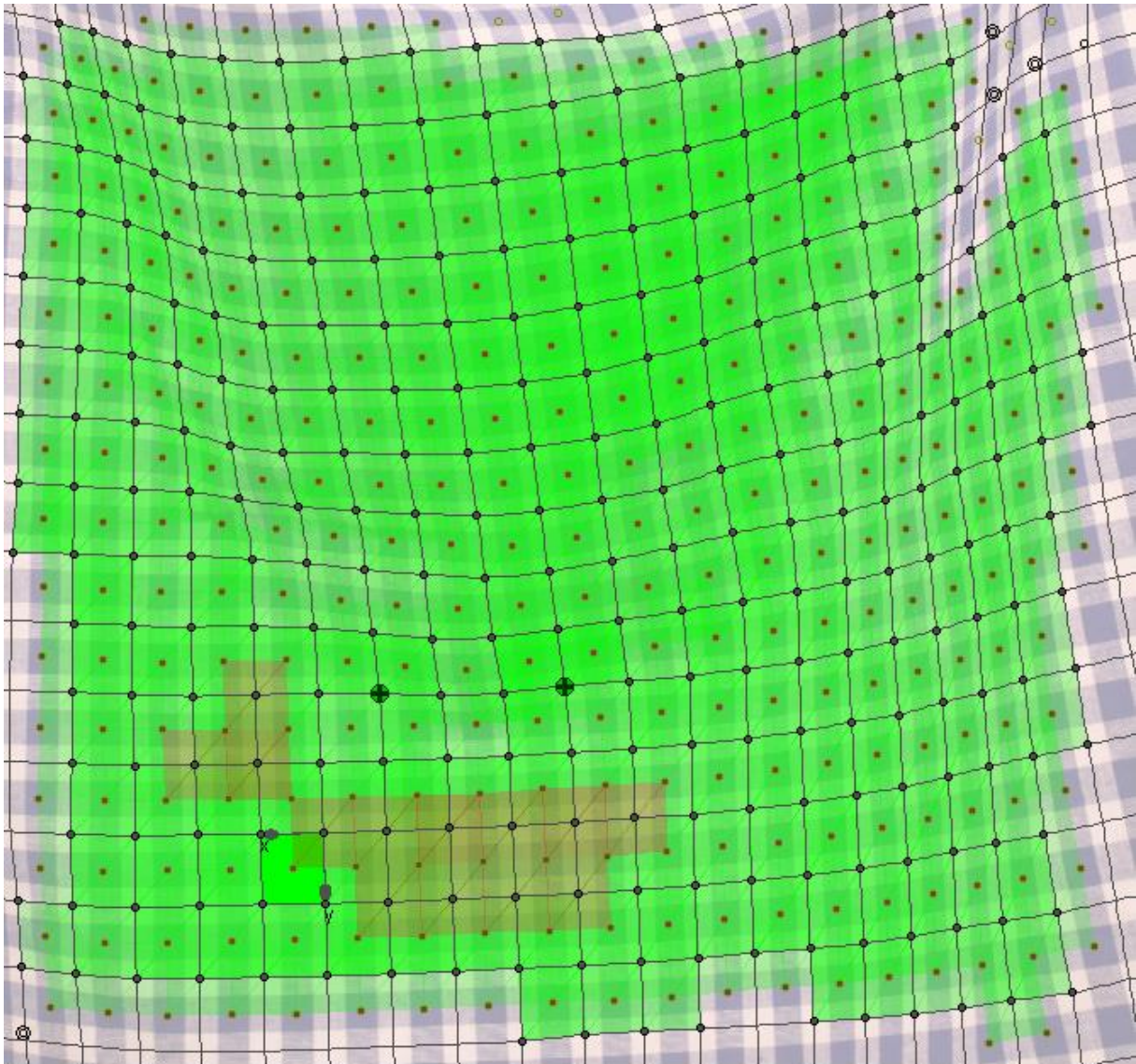
Abbildung 63: Bild „Tx14“



Legende

- ● gegebene Feature-Punkte
(eine Farbe pro Cluster)

Abbildung 64: Bild „Tx27“

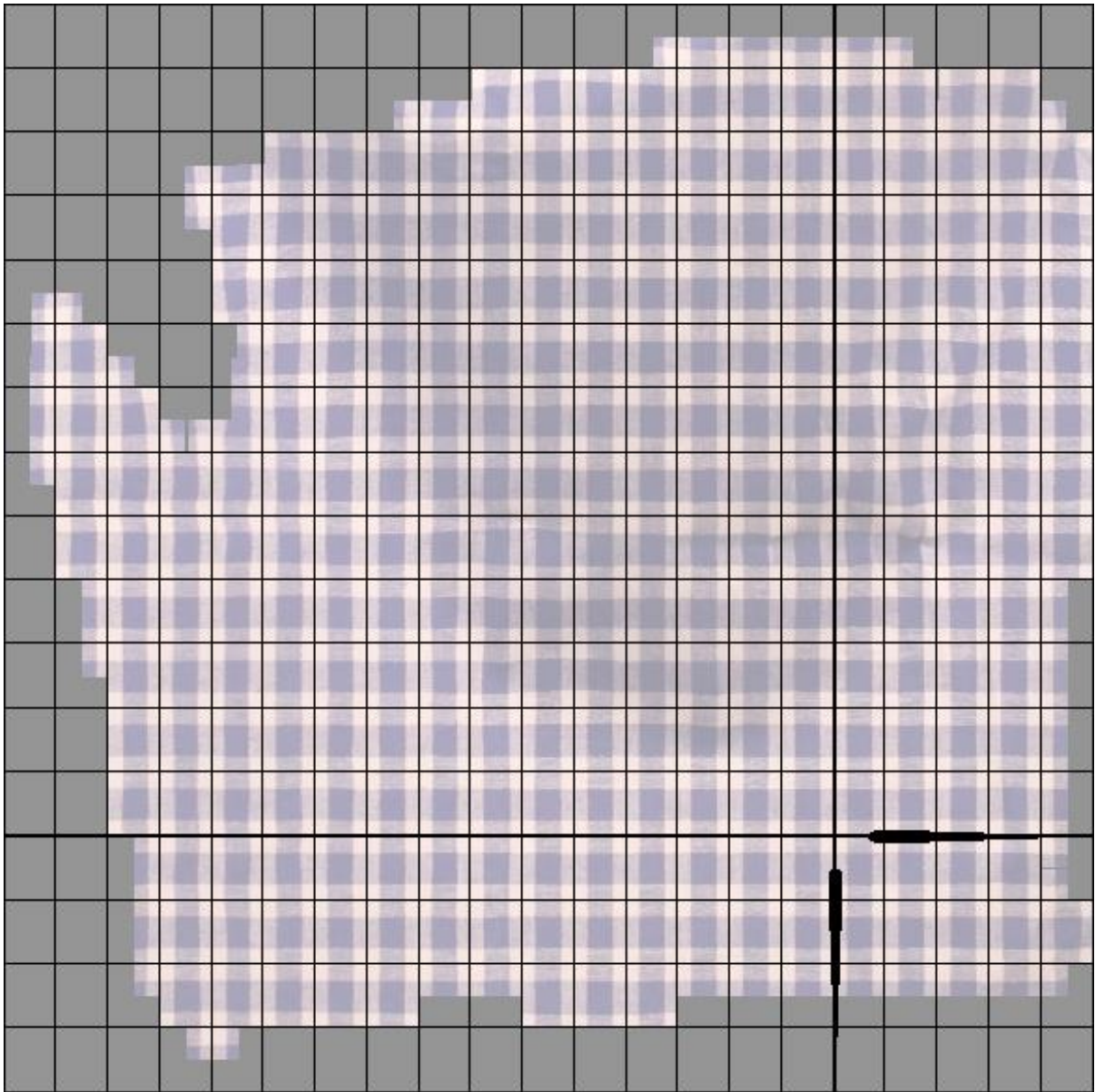


Legende

- ○ gegebene Feature-Punkte
(eine Farbe pro Cluster)
- ● verwendete Kontroll-Punkte
(eine Farbe pro Cluster)
- neu erstellte Kontroll-Punkte
- ⊥ ⊥ topologische Koordinaten mit
ganzzahliger x- oder y-Koordinate

- akzeptierte Quads. Farbe und Transparenz variieren je
nach relativem Fehler der Quads:
- rot, deckend: Fehler > Schwellwert,
höherer Fehler
- rot, transparent: Fehler > Schwellwert,
niedrigerer Fehler
- grün, transparent: Fehler ≤ Schwellwert,
höherer Fehler
- grün, deckend: Fehler ≤ Schwellwert,
niedrigerer Fehler

Abbildung 65: Bild „Tx27“



Legende

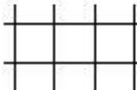

 topologische Koordinaten
 mit ganzzahliger x- oder y-
 Koordinate

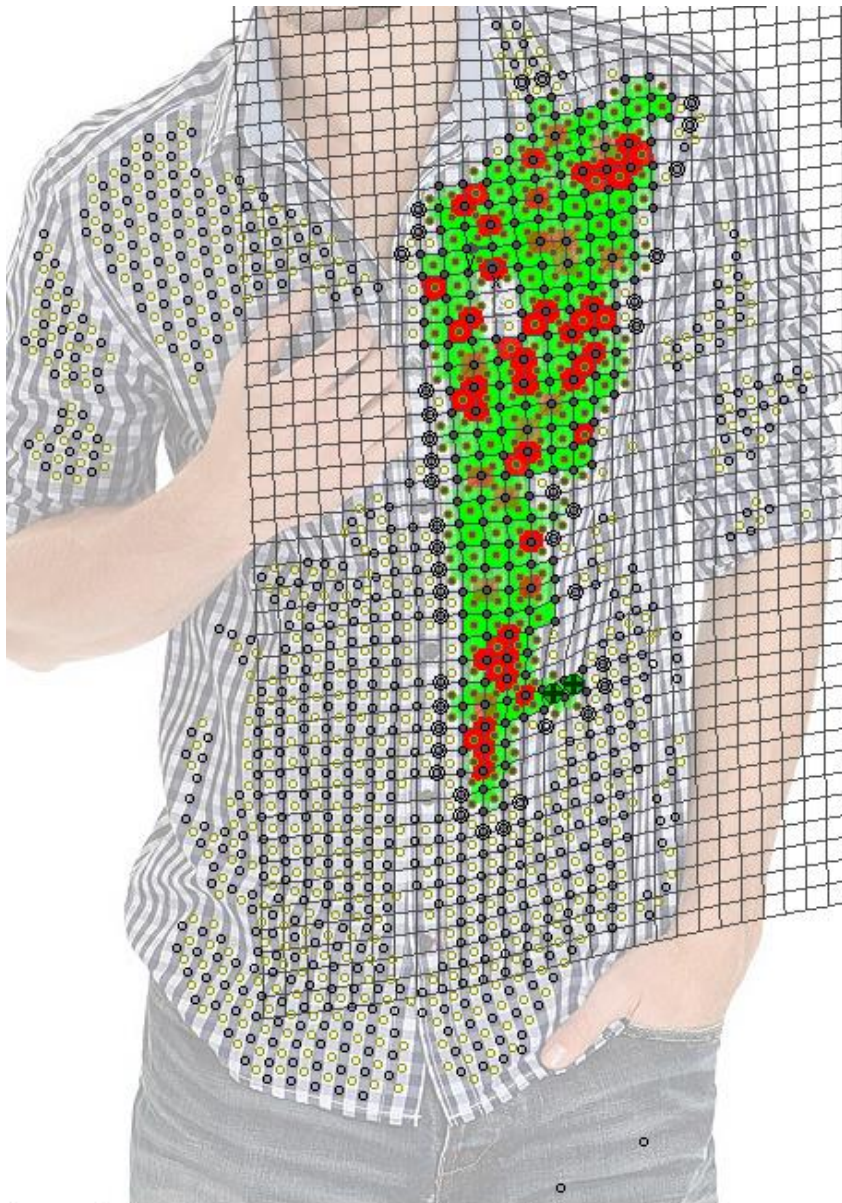
Abbildung 66: Bild „Tx27“



Legende

- ● gegebene Feature-Punkte
(eine Farbe pro Cluster)

Abbildung 67: Bild „Tx30“



Legende

○ gegebene Feature-Punkte
(eine Farbe pro Cluster)

● verwendete Kontroll-Punkte
(eine Farbe pro Cluster)

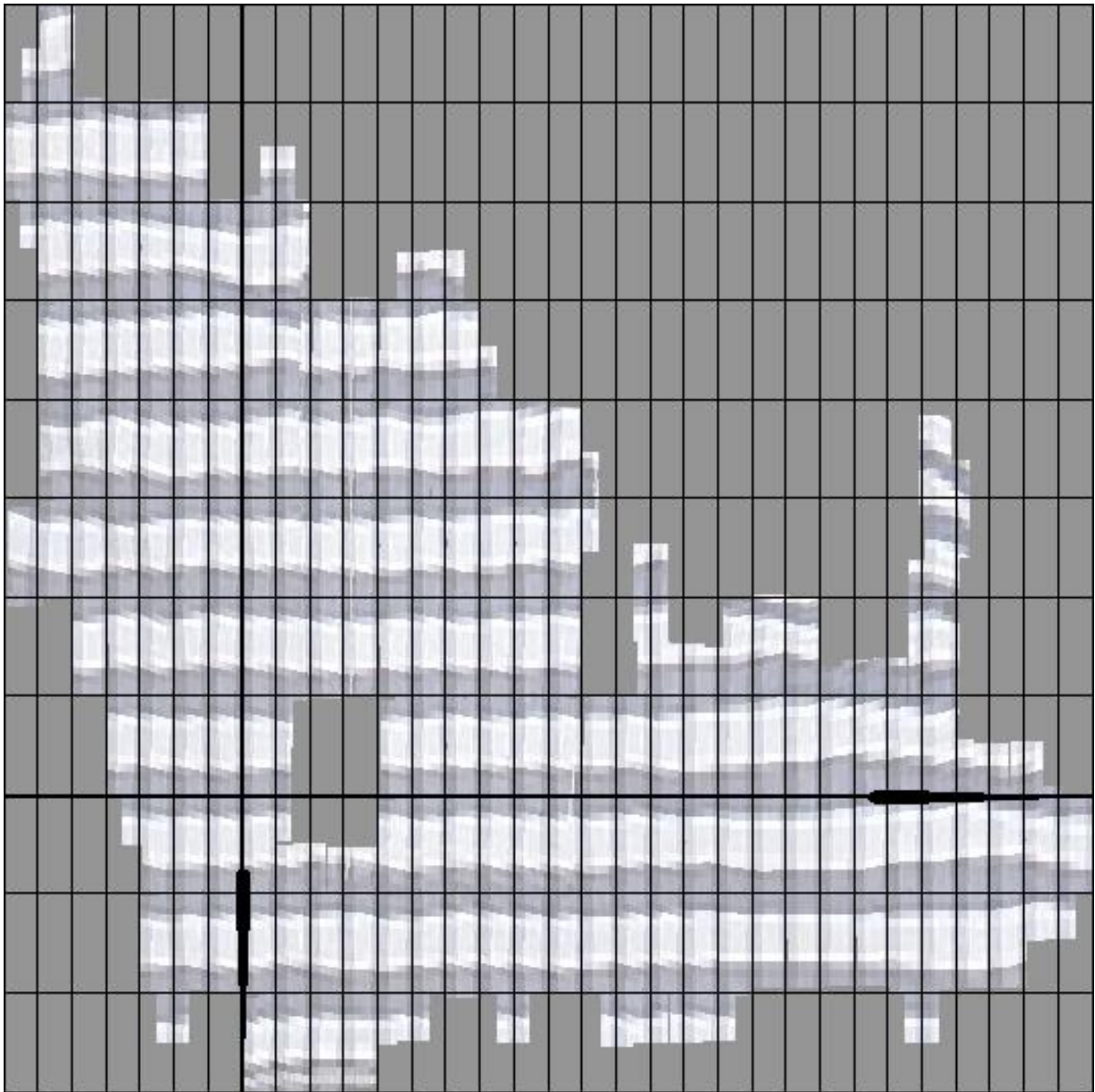
● neu erstellte Kontroll-Punkte

⌘ topologische Koordinaten mit
ganzzahliger x- oder y-Koordinate

akzeptierte Quads. Farbe und Transparenz variieren je nach relativem Fehler der Quads:

- rot, deckend: Fehler > Schwellwert, höherer Fehler
- rot, transparent: Fehler > Schwellwert, niedrigerer Fehler
- grün, transparent: Fehler ≤ Schwellwert, höherer Fehler
- grün, deckend: Fehler ≤ Schwellwert, niedrigerer Fehler

Abbildung 68: Bild „Tx30“



Legende

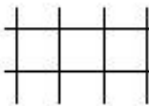
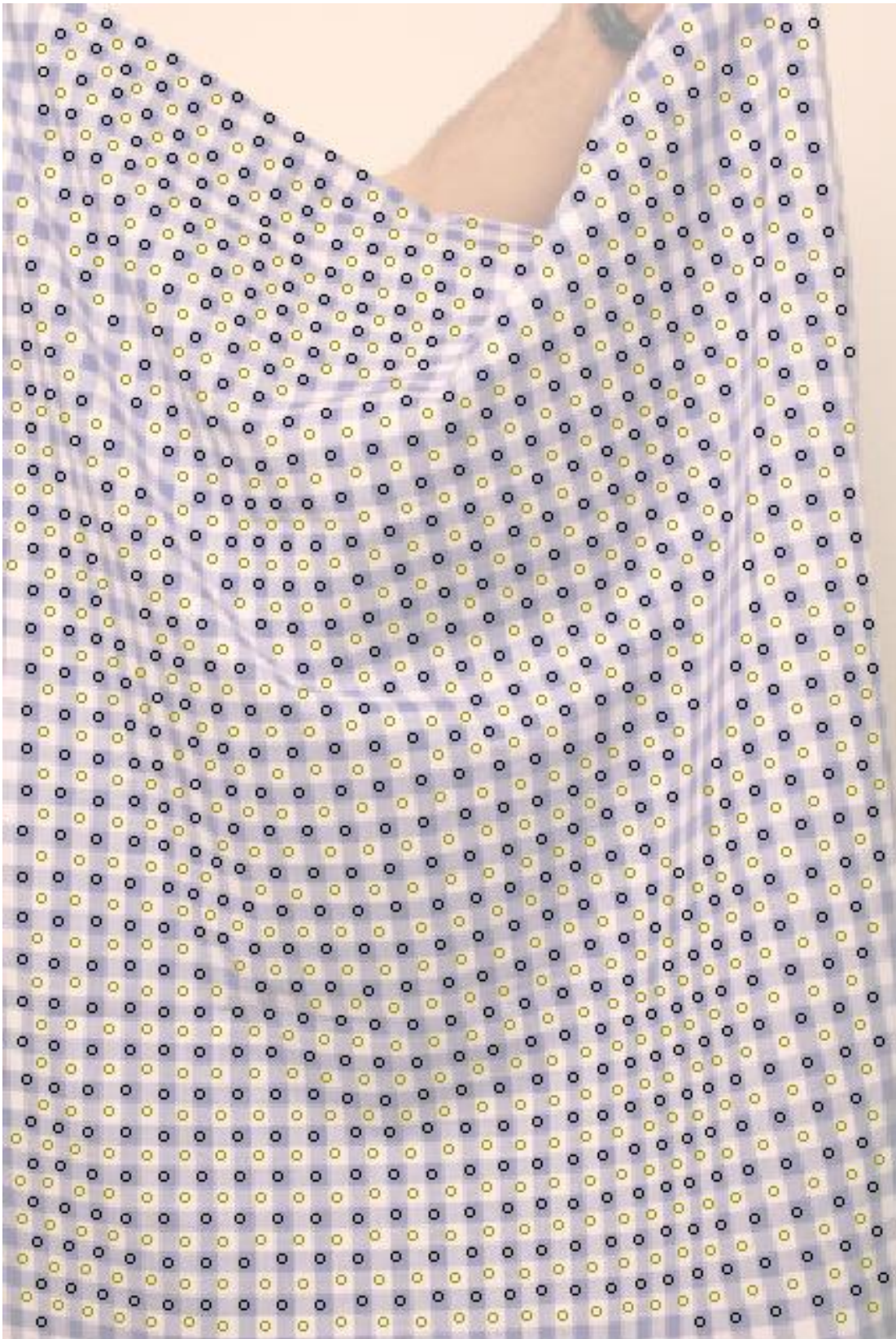

 topologische Koordinaten
 mit ganzzahliger x- oder y-
 Koordinate

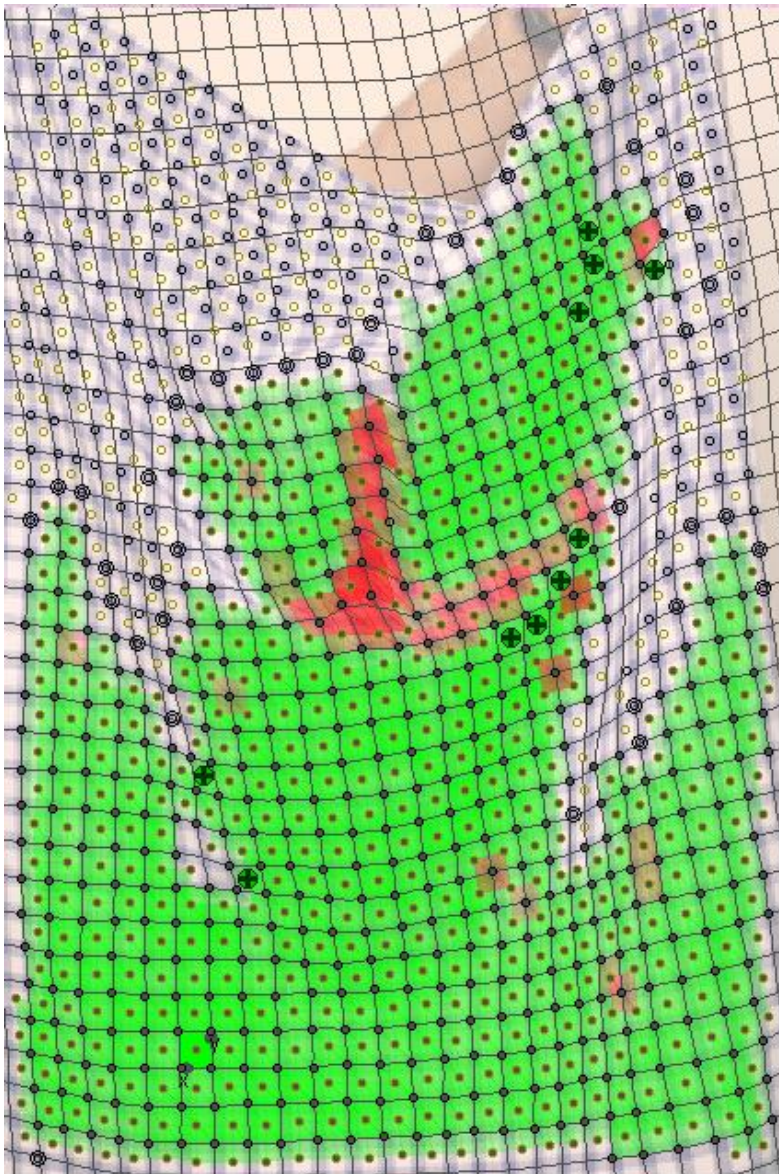
Abbildung 69: Bild „Tx30“



Legende

- ○ gegebene Feature-Punkte
(eine Farbe pro Cluster)

Abbildung 70: Bild „Tx31“



Legende

○ ○ gegebene Feature-Punkte
(eine Farbe pro Cluster)

● ● verwendete Kontroll-Punkte
(eine Farbe pro Cluster)

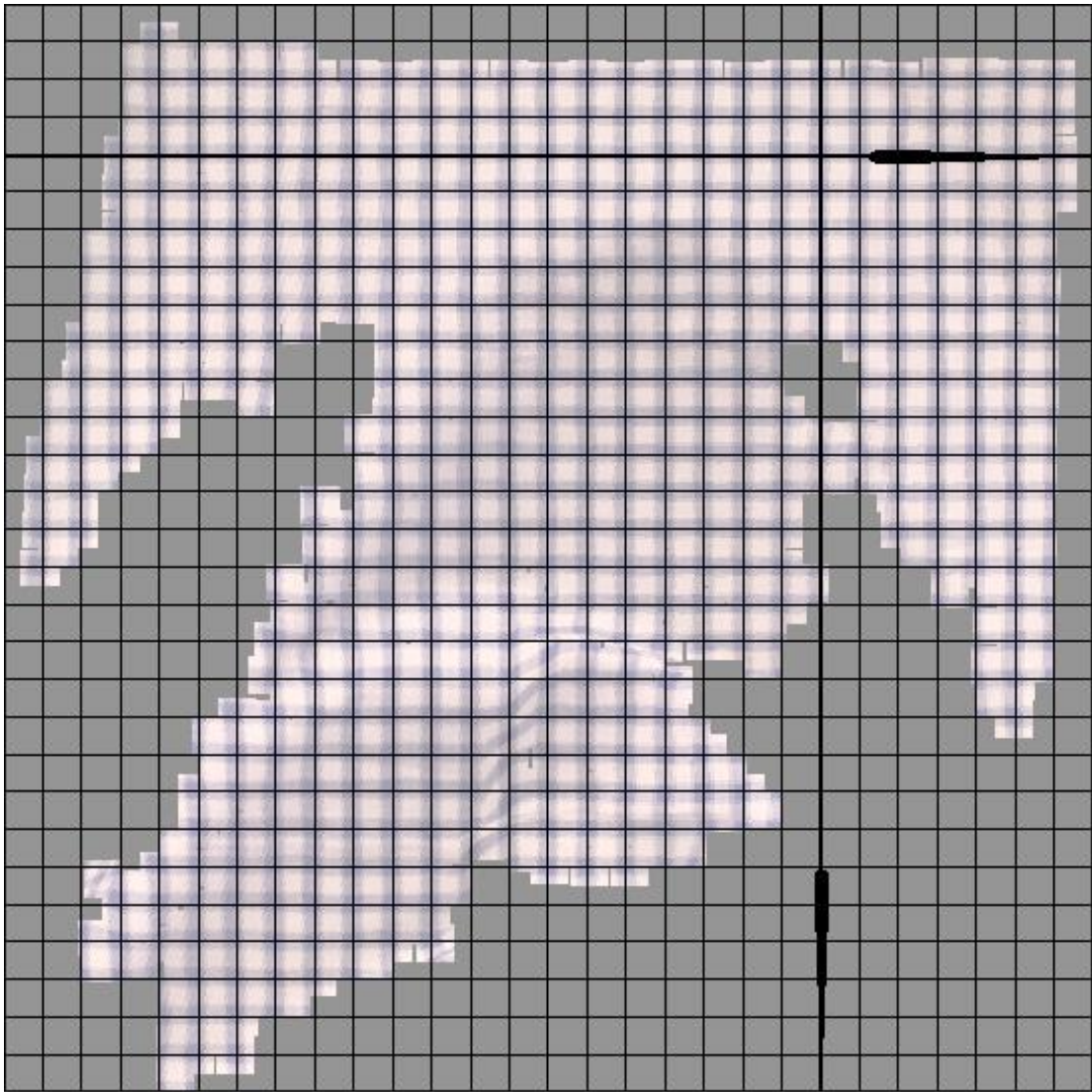
⊕ neu erstellte Kontroll-Punkte

⌌ topologische Koordinaten mit
ganzzahliger x- oder y-Koordinate

akzeptierte Quads. Farbe und Transparenz variieren je nach relativem Fehler der Quads:

- rot, deckend: Fehler > Schwellwert, höherer Fehler
- rot, transparent: Fehler > Schwellwert, niedrigerer Fehler
- grün, transparent: Fehler ≤ Schwellwert, höherer Fehler
- grün, deckend: Fehler ≤ Schwellwert, niedrigerer Fehler

Abbildung 71: Bild „Tx31“



Legende

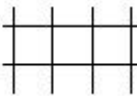
 topologische Koordinaten
mit ganzzahliger x- oder y-
Koordinate

Abbildung 72: Bild „Tx31“



Legende

- ○ gegebene Feature-Punkte
(eine Farbe pro Cluster)

Abbildung 73: Bild „Tx34“



Legende

○ ○ gegebene Feature-Punkte
(eine Farbe pro Cluster)

● ● verwendete Kontroll-Punkte
(eine Farbe pro Cluster)

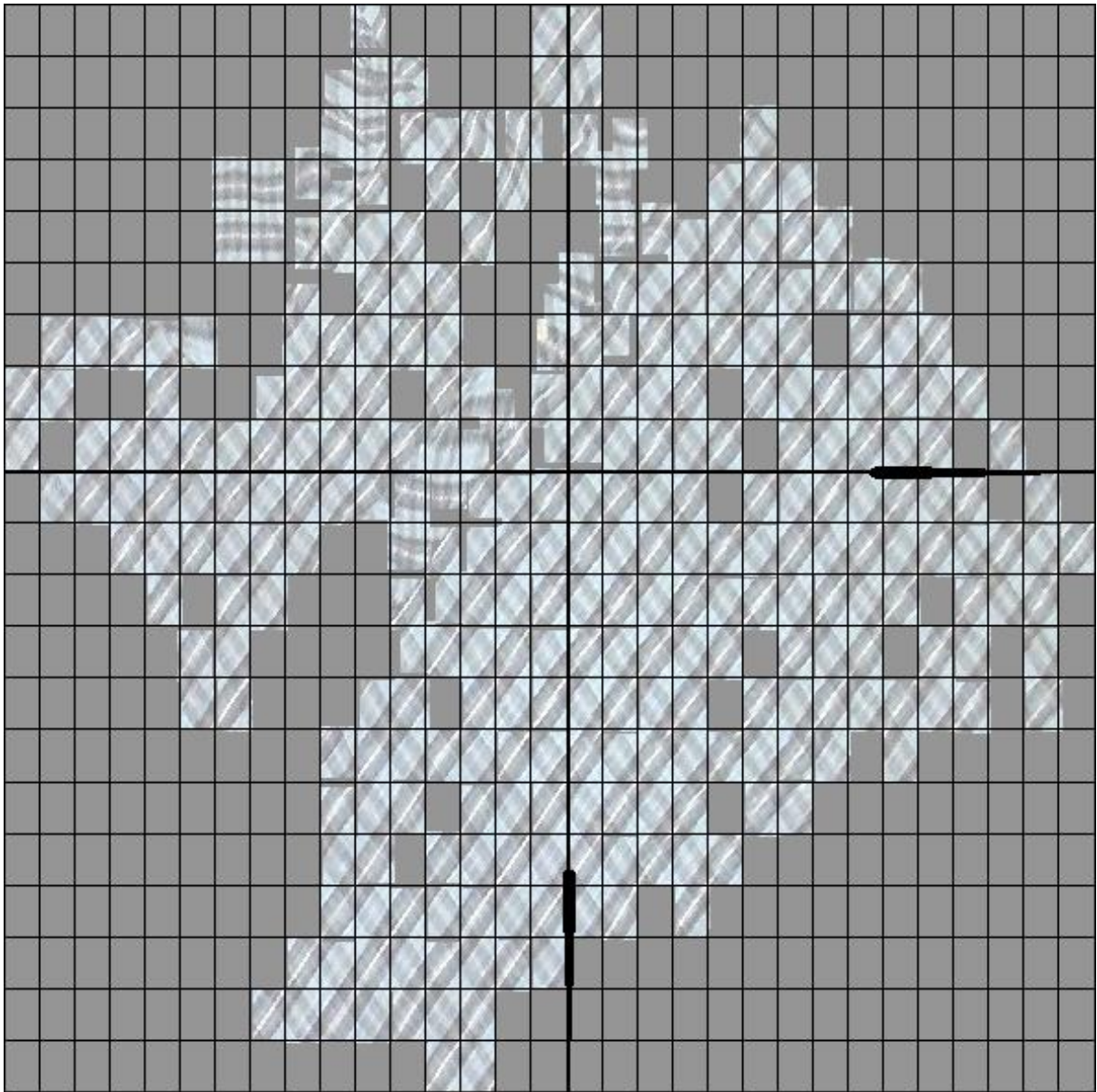
●●● neu erstellte Kontroll-Punkte

⌘ topologische Koordinaten mit
ganzzahliger x- oder y-Koordinate

akzeptierte Quads. Farbe und Transparenz variieren je nach relativem Fehler der Quads:

- rot, deckend: Fehler > Schwellwert, höherer Fehler
- rot, transparent: Fehler > Schwellwert, niedrigerer Fehler
- grün, transparent: Fehler ≤ Schwellwert, höherer Fehler
- grün, deckend: Fehler ≤ Schwellwert, niedrigerer Fehler

Abbildung 74: Bild „Tx34“



Legende

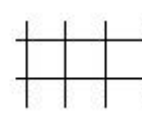
 topologische Koordinaten
mit ganzzahliger x- oder y-
Koordinate

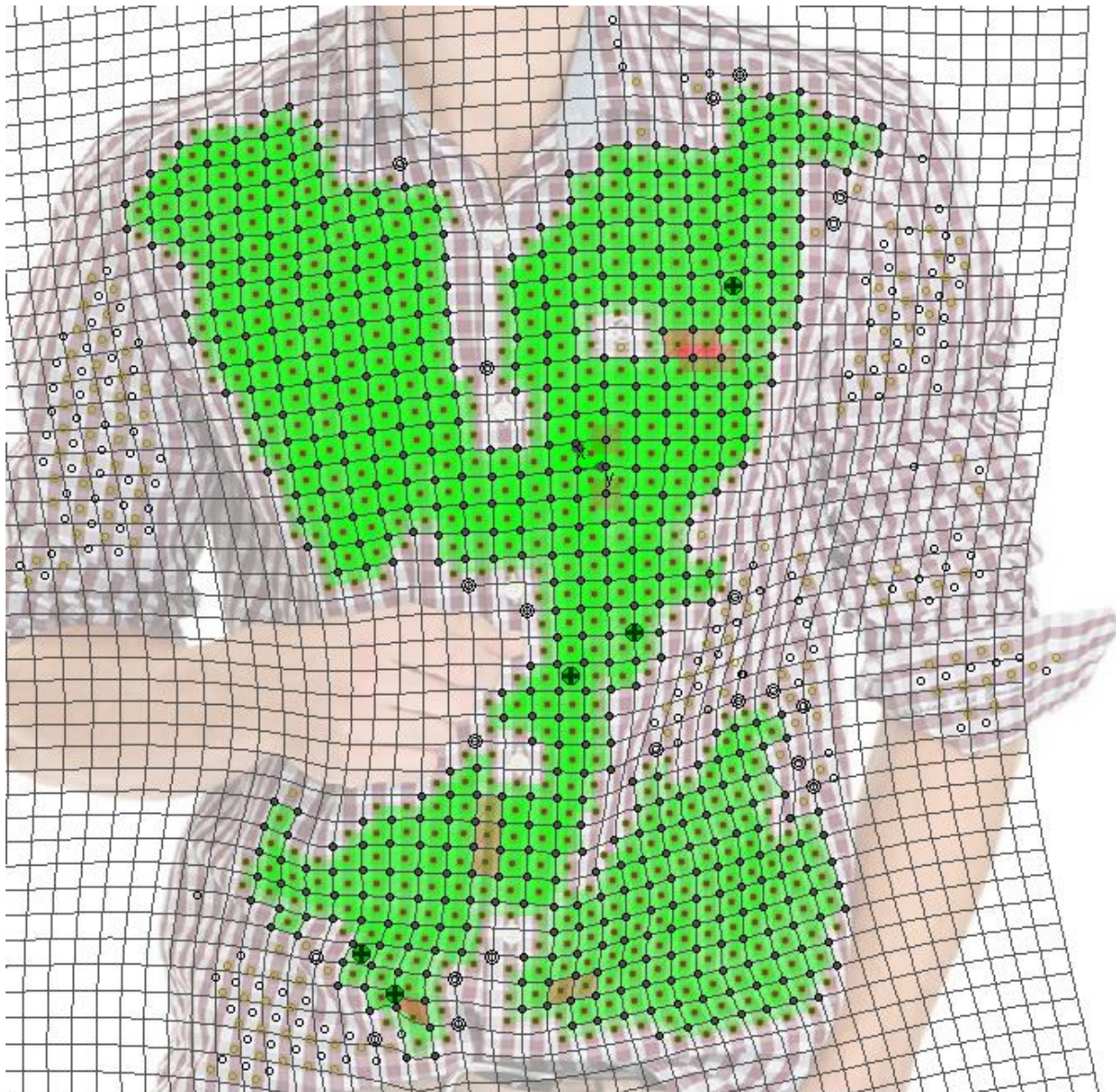
Abbildung 75: Bild „Tx34“



Legende


○ ● gegebene Feature-Punkte
(eine Farbe pro Cluster)

Abbildung 76: Bild „Tx40“



Legende

- ○ gegebene Feature-Punkte
(eine Farbe pro Cluster)
- ● verwendete Kontroll-Punkte
(eine Farbe pro Cluster)
- neu erstellte Kontroll-Punkte

 topologische Koordinaten mit
ganzzahliger x- oder y-Koordinate






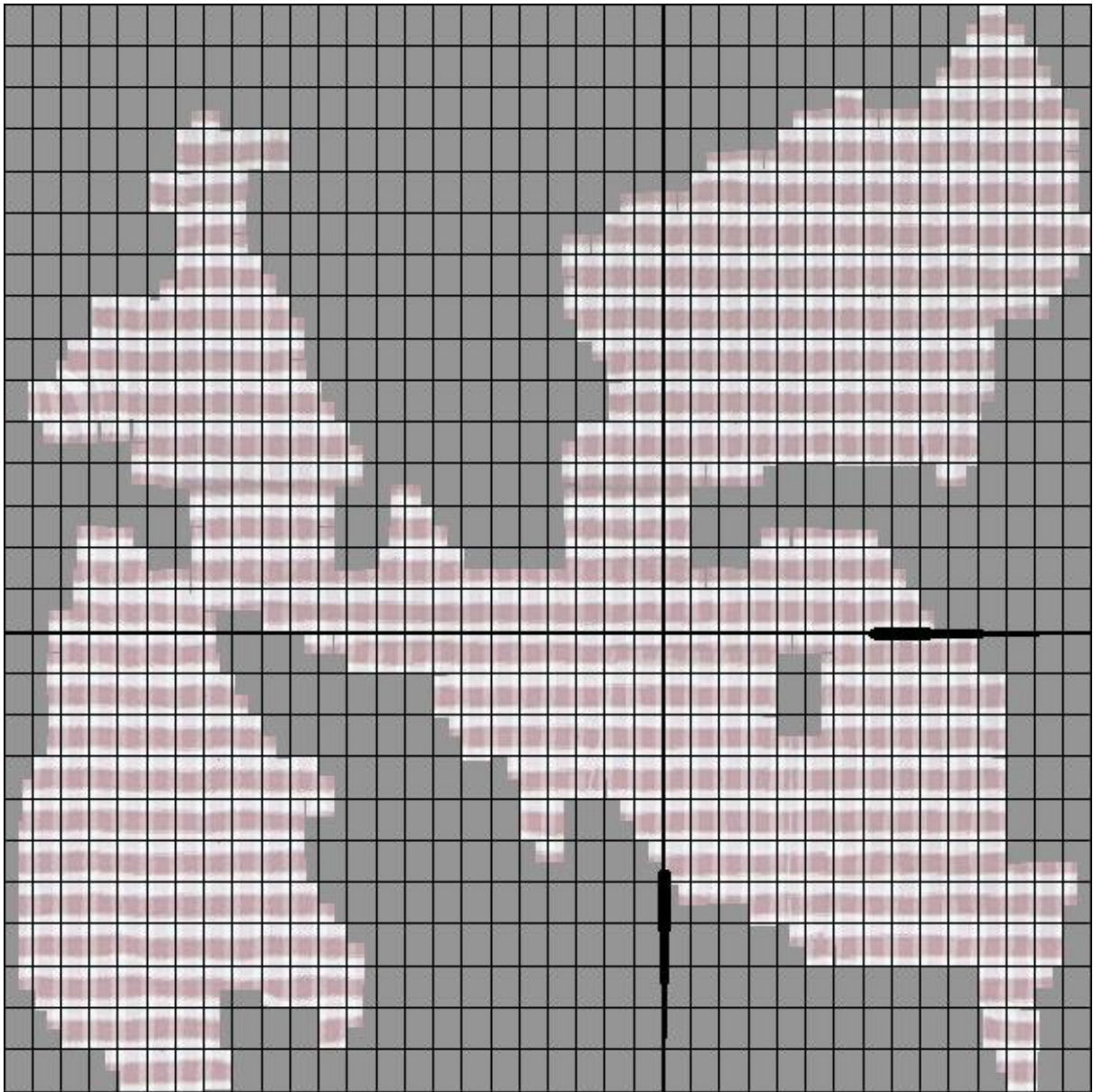
- | | |
|---|--|
|  | akzeptierte Quads. Farbe und Transparenz variieren je nach relativem Fehler der Quads: |
|  | - rot, deckend: Fehler > Schwellwert, höherer Fehler |
|  | - rot, transparent: Fehler > Schwellwert, niedrigerer Fehler |
|  | - grün, transparent: Fehler ≤ Schwellwert, höherer Fehler |
|  | - grün, deckend: Fehler ≤ Schwellwert, niedrigerer Fehler |

Abbildung 77: Bild „Tx40“



Legende

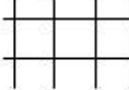
 topologische Koordinaten
mit ganzzahliger x- oder y-
Koordinate

Abbildung 78: Bild „Tx40“



Legende

- ● gegebene Feature-Punkte
(eine Farbe pro Cluster)

Abbildung 79: Bild „Tx50“



Legende

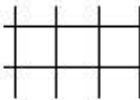

 topologische Koordinaten
 mit ganzzahliger x- oder y-
 Koordinate

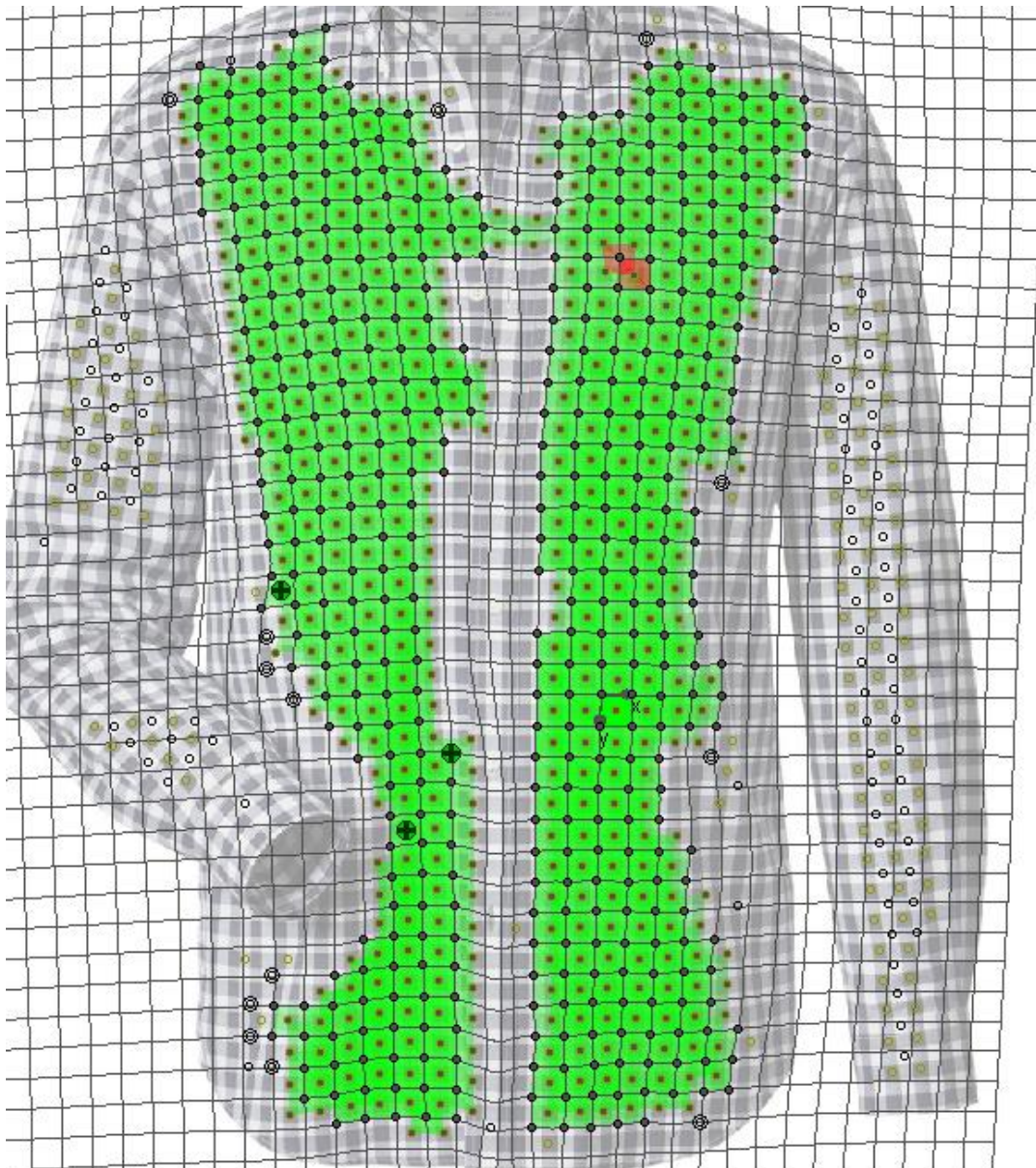
Abbildung 81: Bild „Tx50“



Legende

- ● gegebene Feature-Punkte
(eine Farbe pro Cluster)

Abbildung 82: Bild „Tx70“



Legende

○ ○ gegebene Feature-Punkte
(eine Farbe pro Cluster)

● ● verwendete Kontroll-Punkte
(eine Farbe pro Cluster)

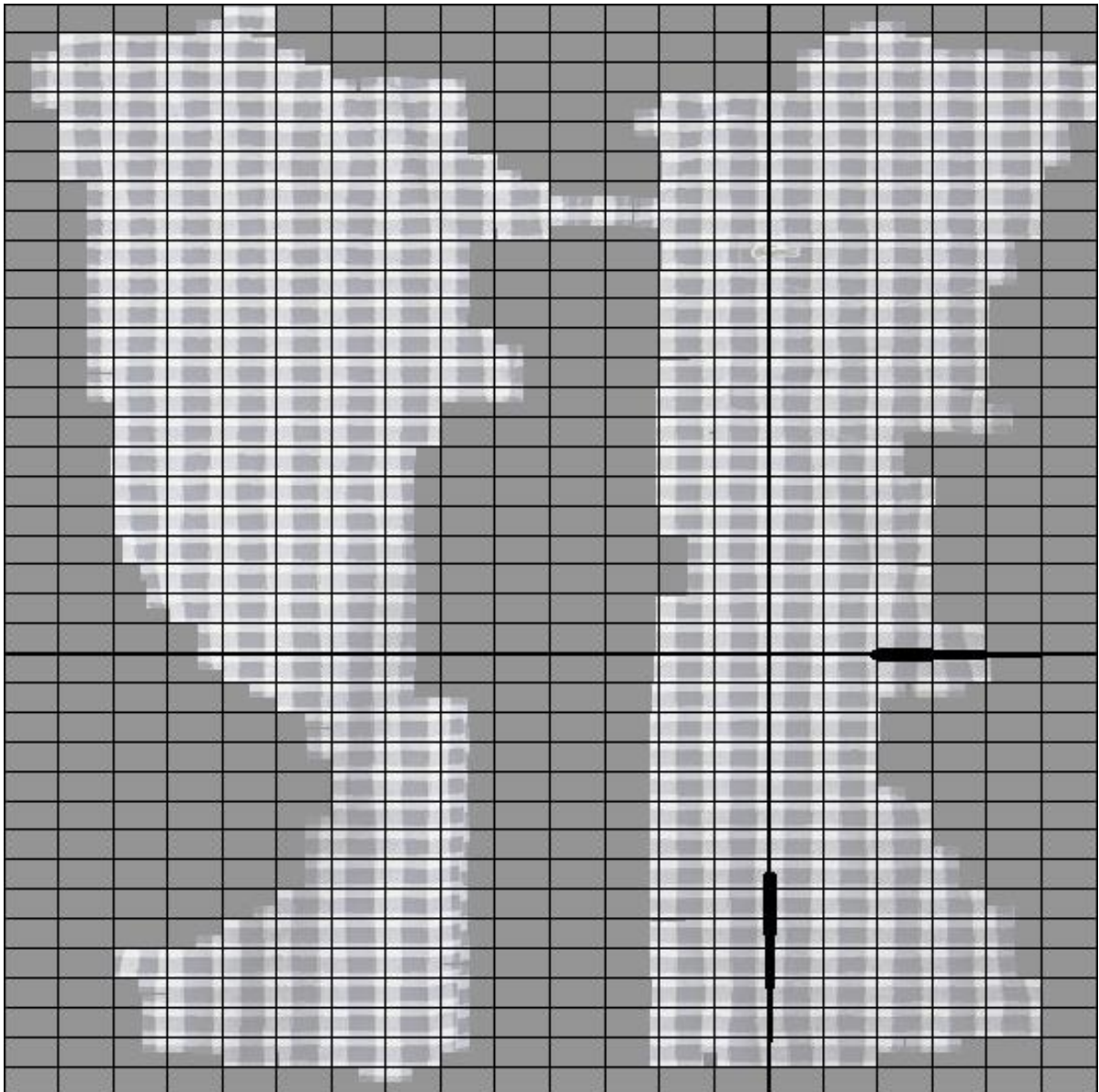
●● neu erstellte Kontroll-Punkte

⌌ topologische Koordinaten mit
ganzzahliger x- oder y-Koordinate

akzeptierte Quads. Farbe und Transparenz variieren je nach relativem Fehler der Quads:

- rot, deckend: Fehler > Schwellwert, höherer Fehler
- rot, transparent: Fehler > Schwellwert, niedrigerer Fehler
- grün, transparent: Fehler ≤ Schwellwert, höherer Fehler
- grün, deckend: Fehler ≤ Schwellwert, niedrigerer Fehler

Abbildung 83: Bild „Tx70“



Legende


 topologische Koordinaten
mit ganzzahliger x- oder y-
Koordinate

Abbildung 84: Bild „Tx70“

Literatur

- [1] Internet, “Image Processing - Virtual Mirror,” <http://www.hhi.fraunhofer.de/fields-of-competence/image-processing/research-groups/computer-vision-graphics/research/virtual-mirror.html>, 2013. .
- [2] A. Hilsmann, D. C. Schneider, and P. Eisert, “Warp-based Near-Regular Texture Analysis for Image-based Texture Overlay,” in *Vision, Modeling, and Visualization*, 2011.

- [3] Y. Liu, W.-C. Lin, and J. Hays, "Near-regular texture analysis and manipulation," *ACM Transactions on Graphics*, vol. 23, no. 3, p. 368, 2004.
- [4] J. Hays, M. Leordeanu, A. A. Efros, and Y. Liu, "Discovering Texture Regularity as a Higher-Order Correspondence Problem," in *Computer Vision – ECCV 2006*, 2006, p. pp 522–535.
- [5] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust wide baseline stereo from maximally stable extremal regions," in *British Machine Vision Conference*, 2002, pp. 384–393.
- [6] Y. Liu, R. Collins, and Y. Tsin, "A computational model for periodic pattern perception based on frieze and wallpaper groups," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 3, pp. 354 – 371, 2004.
- [7] M. Leordeanu and M. Hebert, "A Spectral Technique for Correspondence Problems Using Pairwise Constraints," *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, vol. Vol. 2, pp. 1482–1489, 2005.
- [8] M. Park, K. Broeklehurst, R. T. Collins, and Y. Liu, "Deformed lattice detection in real-world images using mean-shift belief propagation.," *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, no. 10, pp. 1804–16, Oct. 2009.
- [9] M. P. M. Park, Y. L. Y. Liu, and R. T. Collins, "Efficient mean shift belief propagation for vision tracking," *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 2008.
- [10] C. Harris and M. Stephens, "A Combined Corner and Edge Detector," *Proceedings of the Alvey Vision Conference 1988*, pp. 23.1–23.6, 1988.
- [11] W. Förstner and E. Gülch, "A fast operator for detection and precise location of distinct points, corners and centres of circular features," in *ISPRS Intercommission Workshop*, 1987, pp. 281–305.
- [12] H. Wang and M. Brady, "Real-time corner detection algorithm for motion estimation," *Image and Vision Computing*, vol. 13, no. 9, pp. 695–703, 1995.
- [13] D. G. Lowe, "Object recognition from local scale-invariant features," *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, no. [8, pp. 1150–1157 vol.2, 1999.
- [14] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded Up Robust Features," *Computer Vision–ECCV 2006*, vol. 3951, no. 3, pp. 404–417, 2006.
- [15] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 10, pp. 1615–1630, 2005.
- [16] N. Dalal and W. Triggs, "Histograms of Oriented Gradients for Human Detection," *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR05*, vol. 1, no. 3, pp. 886–893, 2004.

- [17] D. Comaniciu and P. Meer, “Mean shift: A robust approach toward feature space analysis,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, VOL. 24, NO. 5, pp. 603–619, 2002.
- [18] C. De Boor, *A Practical Guide to Splines*, vol. 27, no. 149. Springer-Verlag, 1978, p. 325.
- [19] F. Bookstein, “Principal warps: Thin-plate splines and the decomposition of deformations,” *Pattern Analysis and Machine Intelligence, IEEE ...*, vol. 11, pp. 567 – 585, 1989.
- [20] Internet, “Herve Lombaert,” <http://step.polymtl.ca/~rv101/thinplates/>, 2013. .
- [21] L. Gottesfeld, “A Survey of Image Registration,” *International Journal of Image Processing IJIP*, vol. 5, no. 4, pp. 245–269, 2011.
- [22] Internet, “Design Pattern,” <http://www.philippbauer.de/study/se/design-pattern.php>, 2013. .
- [23] Internet, “SDL - Simple DirectMedia Layer,” <http://www.libsdl.org/>, 2013. .
- [24] Internet, “Herve Lombaert,” <http://elonon.iki.fi/code/tpsdemo/index.html>, 2013. .
- [25] H. Lombaert and F. Cheriet, “Geodesic Thin Plate Splines for Image Segmentation,” *Pattern Recognition (ICPR), 2010 20th ...*, pp. 2234 – 2237, 2010.