1 Einleitung

Texturen sind ein fundamentales Mittel zur Erstellung von Modellen für die Darstellung in Bildern, Videos oder Spielen. Sie werden benötigt um die Oberflächendetails eines Modells darzustellen ohne diese explizit über Geometrie oder Materialeigenschaften modellieren zu müssen [WLKT09]. Damit wird der Detailgrad eines Modells erhöht, ohne dabei den Detailgrad der zu Grunde liegenden Geometrie zu erhöhen. Das führt unmittelbar zu einer schnelleren Berechnung des darzustellenden Bildes und einer Einsparung von Speicherplatz.

Texturen beschreiben die visuelle Oberfläche eines Modells in einem (meist) rechteckigem Bild. Mittels eines Mapping-Verfahrens wird dieses Bild anschließend auf das entsprechende Modell projiziert [KNL+15]. Oft werden Texturen strikter definiert, nämlich als eine Darstellung einer Oberfläche mit sich wiederholenden Mustern [WLKT09]. Der Grad der Zufälligkeit dieser Wiederholung kann dabei üblicherweise für unterschiedliche Texturen variieren. Naturgegebene Texturen besitzen in der Regel auf Grund der Unvollkommenheit der Natur eine komplett zufällige Wiederholung von sich ähnlichen Mustern, wo hingegen Menschengeschaffenes oft auch eine deterministische Komponente besitzt. Ein Fliesenmuster besitzt z.B. eine zufällige Wiederholung in den Mikrodetails seiner Fliesen, die Platzierung der Fliesen bzw. deren anliegende Kanten unterliegen dagegen aber einer deterministischem Komponente [WLKT09].

Es kann äußerst schwierig sein, eine Textur zu generieren, dessen Dimensionen und Auflösung an eine bestimmte Modellaufgabe angepasst sind [KNL+15]. Das zeichnet sich insbesondere dadurch wieder, dass es einen speziellen Berufszweig in der Film- und Spielebranche namens *Texture Artist* gibt, der sich ausschließlich mit der Aufgabe befasst, qualitativ hochwertige Texturen zu kreieren [WLKT09]. Diese werden meistens über Handzeichnungen oder über handbearbeitete Fotografien erstellt. Handzeichnungen können zwar ästhetisch ansprechend sein, aber sie garantieren keinerlei Fotorealismus. Fotografien hingegen werden in einem beliebigem Grafikprogramm solange bearbeitet, bis sie den Anforderungen der Textur gerecht werden. Das führt in der Regel dazu, dass die Erstellung einer Textur sehr zeitaufwändig ist und sich dennoch auffällige Wiederholungen oder Unsauberkeiten in der Textur nicht vermeiden lassen. Ferner ist nicht jeder Mensch ein Künstler und es ist äußerst schwierig wenn nicht gar unmöglich ohne Vorkenntnisse oder künstlerische Fähigkeiten qualitativ hochwertige Texturen zu erzeugen [WLKT09]. Das Verfahren der Textursynthese widmet sich diesem Problem.

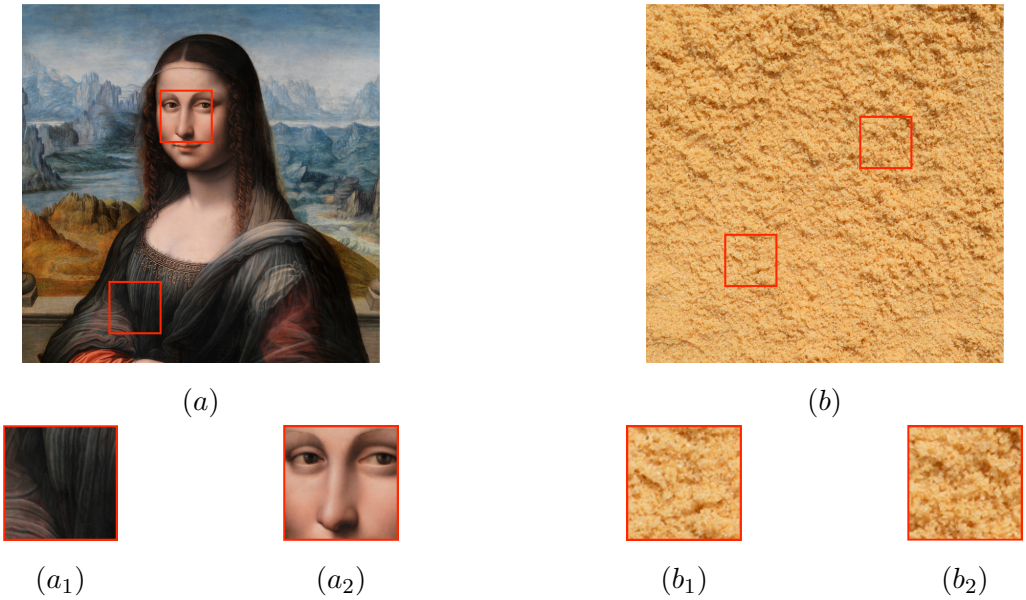
2 Textursynthese

Die *Textursynthese* ist ein alternativer Ansatz für die Erstellung von Texturen, der es erlaubt, auch ohne spezielle Vorkenntnisse, qualitativ hochwertige Ergebnisse zu erzielen. Der Benutzer muss lediglich ein Beispielmuster und eventuell benötigte Konfigurationsparameter an die Textursynthese übergeben, die aus diesen Daten eine Textur synthetisiert [WLKT09]. Die resultierende Textur zeichnet sich dadurch aus, dass sie eine beliebige Größe annehmen kann, aber weiterhin sichtbare Ähnlichkeiten zu dem Beispielmuster aufweist ohne identisch zu ihm zu sein. Die Textursynthese kümmert sich dabei automatisiert und im Idealfall ohne technische Benutzereingaben um die Schwierigkeiten bei der Erstellung von Texturen. Sie berücksichtigt die visuellen Charakteristiken des Beispielmusters und vermeidet dabei zeitgleich auffällige Wiederholungen oder Unnatürlichkeiten in der synthetisierten Textur.

2.1 „Markov Random Fields“-Eigenschaft

Die „Markov Random Field“-Eigenschaft ist eine der populärsten Eigenschaften, die der Textursynthese zu Grunde liegt [WLKT09]. Sie motiviert die Metrik, die Verwendung findet, um die Ähnlichkeit zwischen dem Beispielmuster und der zu synthetisierenden Textur zu beschreiben [KEBK05]. Die „Markov Random Field“-Eigenschaft beschreibt dabei die Synthese als eine Aneinanderreihung von *lokalen* und stationären Prozessen [WLKT09]. Das bedeutet, dass jede Farbe eines Pixels in der Textur ausschließlich über die Pixel in seiner direkten räumlichen Nachbarschaft charakterisiert werden kann (lokal). Diese Charakterisierung ist dabei unabhängig von der Position des betrachteten Pixels (stationär) [KEBK05]. Die Intuition dieser Eigenschaft lässt sich anhand von einem Beispiel verdeutlichen (vgl. Abbildung 1). Einem Betrachter liegt ein Bild vor, welches er aber nur durch ein kleines bewegliches Fenster betrachten kann. Er sieht demnach nie das komplette Bild auf einmal, kann aber durch Bewegungen seines Fensters einzelne Bereiche des Bildes entdecken und erschließen. Das Bild ist dann stationär, falls es unter verschiedenen Ausschnitten immer ähnlich erscheint (eine geeignete Fenstergröße vorausgesetzt). Das Bild ist lokal, falls jeder Pixel in der Mitte eines Fensters über die umliegenden Pixel in seinem Fenster bestimmt werden kann [WLKT09].

Basierend auf dieser Eigenschaft kann der Prozess der Textursynthese spezifiziert werden. Sei ein Beispielmuster gegeben. Dann lässt sich daraus eine Textur synthetisieren, sodass für jeden synthetisierten Pixel dessen räumliche Nachbarschaft zu mindestens einer Nachbarschaft im Beispielmuster ähnlich ist [WLKT09]. Die



(a) (b)

(a_1) (a_2) (b_1) (b_2)

Abbildung 1: (a) ist ein generelles Bild während (b) eine Textur ist. Ein bewegliches Fenster an zwei unterschiedlichen Positionen ist als rotes Quadrat in (a) und (b) gekennzeichnet. Unterschiedliche Aussschnitte einer Textur sind sichtbar ähnlich zueinander sein (b_1, b_2). Dies ist nicht der Fall für ein generelles Bild (a_1, a_2).

Größe der betrachteten Nachbarschaften ist dabei in der Regel ein benutzerdefinierter Parameter. Die Nachbarschaftssuche basiert im Allgemeinen auf der kleinsten quadrierten farblichen Abweichung

$$\min d(\mathbf{t}_i, \mathbf{x}_j) = \|\mathbf{t}_i - \mathbf{x}_j\|^2. \tag{1}$$

$\mathbf{t}_i$ beziehungsweise $\mathbf{x}_j$ beschreiben dabei die Farben einer Nachbarschaft der Größe $N \times N$ als Vektor um den Pixel i in der Textur T respektive um den Pixel j in dem Beispielmuster X. Für jede Nachbarschaft $\mathbf{t}_i$ ist dann ihre ähnlichste Nachbarschaft $\mathbf{x}_j$ im Beispielmuster gefunden, wenn $d(\mathbf{t}_i, \mathbf{x}_j)$ minimal ist [KEBK05].

Aufgrund der Ähnlichkeiten zwischen lokalen Nachbarschaften im Beispielmuster und der Textur wird garantiert, dass die synthetisierte Textur Gemeinsamkeiten mit dem Beispielmuster aufweist [WLKT09].

2.2 Verfahren

Der Großteil an veröffentlichten Algorithmen zur Textursynthese basiert auf der „Markov Random Fields“-Eigenschaft [WLKT09]. Diese Verfahren lassen sich in der Regel einer von zwei Kategorien zuordnen: den *lokal wachsenden Verfahren* und

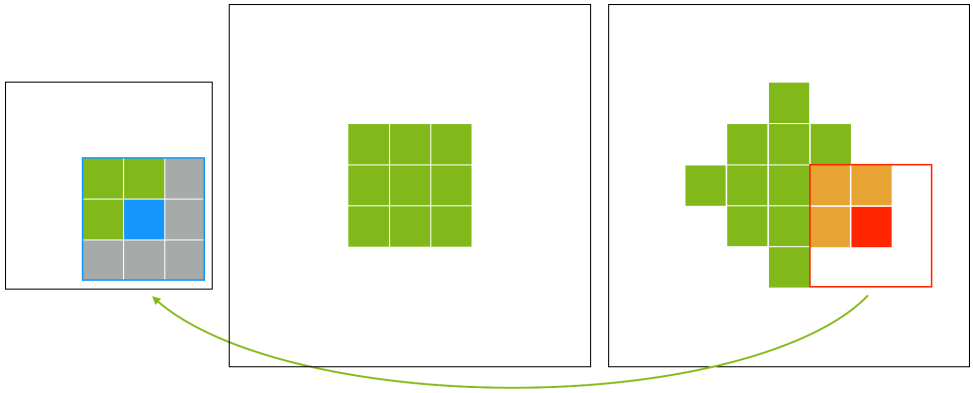


Abbildung 2: Der Algorithmus von [EL99]. Aus einem Beispielmuster (links) wird zuerst eine initiale Textur erstellt, indem eine kleine Region aus dem Beispielmuster in diese kopiert wird (Mitte). Der Algorithmus berechnet die Pixel dann sukzessive um die initiale Region auf Basis der Nachbarschaftssuche (rechts).

den *global optimierenden Verfahren*. Lokal wachsende Verfahren synthetisieren die Textur nach und nach über einzelne Pixel oder Regionen [KEBK05]. Global optimierende Verfahren hingegen synthetisieren und optimieren die Textur iterativ als Ganzes auf Basis einer Zielfunktion [KNL+15]. Im Folgenden sollen drei Verfahren der beiden Kategorien näher betrachtet werden.

2.2.1 Pixelbasierte Textursynthese

Einer der ersten Ansätze der Textursynthese ist die *pixelbasierte Textursynthese* (vgl. [EL99]). Er fällt in die Kategorie der lokal wachsenden Verfahren. Sei ein Beispielmuster und eine Nachbarschaftsgröße gegeben, dann funktioniert die grundlegende Idee hinter diesem Algorithmus wie folgt.

Die zu synthetisierende Textur wird zuerst initialisiert, indem eine kleine beliebige Region des Beispielmusters in die Mitte der Textur kopiert wird. Der Algorithmus berechnet dann sukzessive die einzelnen Pixelfarben der Textur kreisförmig um die initiale Region von innen nach außen. Abbildung 2 illustriert dieses Verfahren. Die Farbe eines aktuell betrachteten Pixels p (rot) in der Textur wird dann über eine Nachbarschaftssuche im Beispielmuster ermittelt. Dazu wird zuerst ein Rahmen mit der Größe der Nachbarschaft um den Pixel p gelegt (hier 3×3) und alle bereits gesetzten Pixel in diesem Rahmen gefunden (orange). Auf Basis dieser gesetzten Pixel wird im Beispielmuster die ähnlichste Nachbarschaft auf Grundlage von (1) gefunden. Der Pixel p erhält dann die Farbe des Pixels in der Mitte der ermittelten ähnlichsten Nachbarschaft im Beispielmuster (blau). Dieser Vorgang wird solange

wiederholt, bis alle Pixel der Textur gesetzt sind.

Der Algorithmus [EL99] ist relativ einfach zu verstehen sowie zu implementieren [WLKT09]. Er ist außerdem benutzerfreundlich, da dieser nur einen Konfigurationsparameter, die Nachbarschaftsgröße, übergeben muss [EL99]. Die Wahl der Nachbarschaftsgröße ist jedoch nicht trivial. Fällt die Wahl der Nachbarschaftsgröße zu klein aus, so kann das Ergebnis zu zufällig wirken. Ist sie auf der anderen Seite zu groß, können sichtbare Wiederholungen entstehen oder es kommt zu Unnatürlichkeiten, da die Nachbarschaftssuche auf Grund ihrer Größe wohlmöglich keine geeigneten Kandidaten finden kann [WLKT09].

2.2.2 Regionsbasierte Textursynthese

Ein weiterer Vertreter der lokal wachsenden Verfahren und eine Erweiterung bzw. ein Nachfolger der pixelbasierten Textursynthese ist die *regionsbasierte Textursynthese* (vgl. [WLKT09]). Die regionsbasierte Synthese ist sehr ähnlich zu der pixelbasierten Synthese, aber anstatt einzelne Pixel zu kopieren, werden stattdessen ganze Regionen in die Textur kopiert. Damit kann die Qualität der Textursynthese verbessert werden, denn es ist sichergestellt, dass Pixel innerhalb einer Region auch zueinander passen. Der wesentliche Unterschied der beiden Verfahren besteht letztendlich im Kopierungsprozess. In pixelbasierten Algorithmen ist die Kopie eines Pixels fest und kann nachträglich nicht mehr verändert werden. Bei den regionsbasierten Verfahren verhält es sich anders, da die Kopie einer Region üblicherweise dazu führt, dass bereits synthetisierte Bereiche der Textur durch die neue Region überdeckt werden. Der Algorithmus muss folglich eine Entscheidung treffen, wie er mit den im Konflikt stehenden Pixeln umgeht. Mehrere mögliche Szenarien sind die Folge. Neue Regionen können alte Regionen einfach überschreiben (vgl. Abbildung 3a). Das führt in der Regel aber zu sichtbaren Kanten der einzelnen Regionen [WLKT09]. Eine weitere Möglichkeit ist, neue Regionen mit den bereits überlappenden Bereichen der Textur zu überblenden (vgl. Abbildung 3b). In manchen Situationen kann dies jedoch zu verwaschenen Artefakten führen [WLKT09]. Die wohl populärste Methode ist das *GraphCut*-Verfahren (vgl. [KSE+03, WLKT09]). Der GraphCut-Algorithmus sucht einen optimalen Pfad zwischen zwei Regionen, der sie trennt (vgl. Abbildung 3c).

Regionsbasierte Verfahren sind in der Regel erfolgreicher als pixelbasierte Verfahren, da sie die globale Struktur des Beispielmusters besser einfangen können. Pixelbasierte Verfahren erlauben dagegen mehr Kontrolle über einzelne Pixel [KEBK05]. Ihnen beiden ist aber eine gemeinsame Schwäche zuteil. Auf Grund ihres lokalen Wachstums um eine initiale Region können sich kleine Fehler in den Anfängen der

(a) Überlagern (b) Überblenden (c) GraphCut

Abbildung 3: Verschiedene Methoden, wie mit den im Konflikt stehenden Pixeln bei der Kopie einer neuen Region (grün) und den bereits synthetisierten Regionen (rot) umgegangen wird.

Synthese schnell anhäufen und zu Inkonsistenzen führen [KEBK05]. Global wachsende Verfahren setzen dort an und versuchen, diesen Fehler so gering wie möglich zu halten.

2.2.3 Texturoptimierung

Literatur

[EL99] Efros, Alexei A. ; Leung, Thomas K.: Texture Synthesis by Non-Parametric Sampling. In: *IEEE International Conference on Computer Vision*, 1999, S. 1033–1038

[KEBK05] Kwatra, Vivek ; Essa, Irfan ; Bobick, Aaron ; Kwatra, Nipun: Texture Optimization for Example-based Synthesis. In: *SIGGRAPH '05*, 2005, S. 795–802

[KNL+15] Kaspar, Alexandre ; Neubert, Boris ; Lischinski, Dani ; Pauly, Mark ; Kopf, Johannes: Self Tuning Texture Optimization. In: *Computer Graphics Forum* 34 (2015), Nr. 2, S. 349–359

[KSE+03] Kwatra, Vivek ; Schödl, Arno ; Essa, Irfan ; Turk, Greg ; Bobick, Aaron: Graphcut Textures: Image and Video Synthesis Using Graph Cuts. In: *ACM SIGGRAPH 2003 Papers*, 2003, S. 277–286

[WLKT09] Wei, Li-Yi ; Lefebvre, Sylvain ; Kwatra, Vivek ; Turk, Greg: State of the Art in Example-based Texture Synthesis. In: *Eurographics 2009, State of the Art Report, EG-STAR*, 2009