Bachelorarbeit

Wave-Function-Collapse

Funktionsweise und Anwendungsfälle

TH-Nürnberg Georg-Simon-Ohm

Davoud Tavakol

29.12.2022

Abstract

zum schluss..

Inhaltsverzeichnis

1. Abkürzungsverzeichnis

WFC . . . . . Wave-Function-Collapse
PCG . . . . . Procedural-Content-Generation
CSP . . . . . Constraint-Satisfaction-Problem

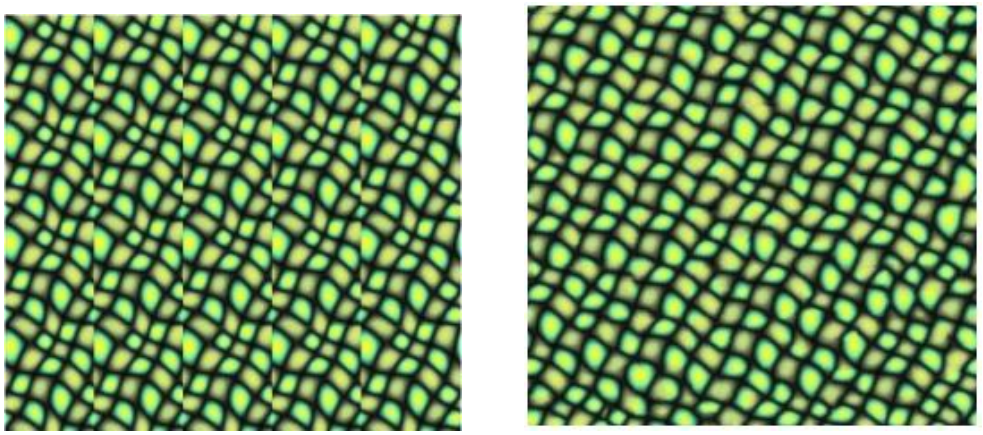
2. Einleitung

TODO AM ENDE. Die automatische Generierung von Inhalten wie Texte, Images oder Modellen ist heutzutage Standard in vielen Bereichen der Industrie. Um solche Inhalte vordefinierten Parametern zu erstellen werden vor allem zwei Methoden zur Generierung verwendend. AI's (Künstliche Intelligenzen) wie ChatGPT und Algorithmen. Der logische Vorteil von solchen Tools ist es, das diese in kürzester Zeit qualitative Resultate Generieren können und auch wie oben erwähnt vordefinierte Parameter als Input erhalten können, um die Ergebnisse für ihren gebrauch anzupassen. In dieser Bachelorarbeit werde ich mich auf den Wave-Function-Collapse Algorithmus, dessen Funktionsweise und Anwendungsfälle fokussieren.

3. Textursynthesen im Vergleich

Es gibt viele Möglichkeiten Textursynthese mit Algorithmen zu erzielen. Die meisten dieser Methoden basieren auf demselben Grundprinzip aus kleineren Input-Images größere oder gleich große Output-Images zu generieren. Nach D.Gomathi und Rajvi Shah ([8], S.1), kann die Textursynthese wie folgt definiert werden.

Ziel einer Textursynthese ist es aus einer Texturprobe eine neue Textur zu generieren die, "wenn sie von einem menschlichen Beobachter wahrgenommen wird, durch denselben zugrundeliegenden Prozess erzeugt wird."Das Ergebnis muss der Texturprobe ähnlich sein aber dennoch in der Wahrnehmung genügend Variation enthalten. So eine Synthese ist nicht möglich, wenn man die Eingabetextur einfach mehrfach kachelt. Dadurch erhält man keine ßauberen"Übergänge, und die einzelnen Blöcke sind klar erkennbar. (siehe Abb. 3.1)



(a) (b)

Abbildung 3.1: (a) Blockartige Textur, (b) Textursynthese

Dadurch ergeben sich folgende Metriken die erfüllt werden müssen.

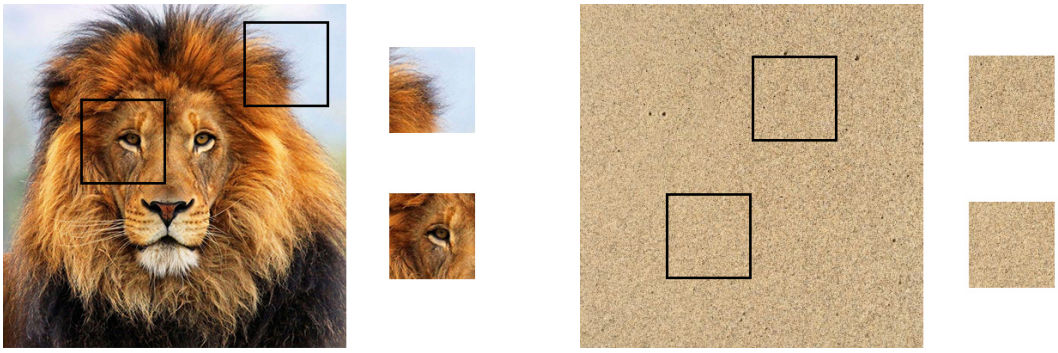
- Wenn Texturprobe gegeben, generiere eine neue Textur, die der Probe gleicht.
- Die neue Textur kann eine beliebige Größe haben die vom Benutzer festgelegt wird.
- Es sollen keine sichtbaren Übergänge, Artefakte oder fehlerhafte Kanten sichtbar sein.

- Dasselbe Muster soll nicht mehrfach in der neuen Textur vorkommen. ([8], S.2)

Im Folgenden werde ich auf drei verschiedene Textursynthese verfahren eingehen da diese sich fundamental voneinander unterscheiden. Es wird nicht in Detail auf alle verfahren eingegangen da dieser Abschnitt nur Hintergrundinformationen darlegen soll.

3.1 Pixel basierende Textursynthese

Bei dieser Methode werden neue Texturen Pixel für Pixel generiert. Jeder neue Pixelwert wird von seinen lokalen Nachbarn festgelegt. Dieser verfahren verwenden meistens Markow-Netzwerke *(Markov-Random-Fields)* die relativ gute Resultate liefern mit wenig Rechenlast. Markov-Random-Fields Methoden beurteilen jeden Pixel nach einer kleinen Menge von Nachbarn. Voraussetzung hierfür ist, dass das Input-Image stationär und lokal ist. Ein Image wird als stationär bezeichnet wenn, unter korrekter Fenstergröße, jeder betrachtete Bereich ähnlich zueinander aussehen. Lokal ist ein Image dann, wenn jeder Pixel allein von seinen Nachbarn bestimmt werden kann.[8]

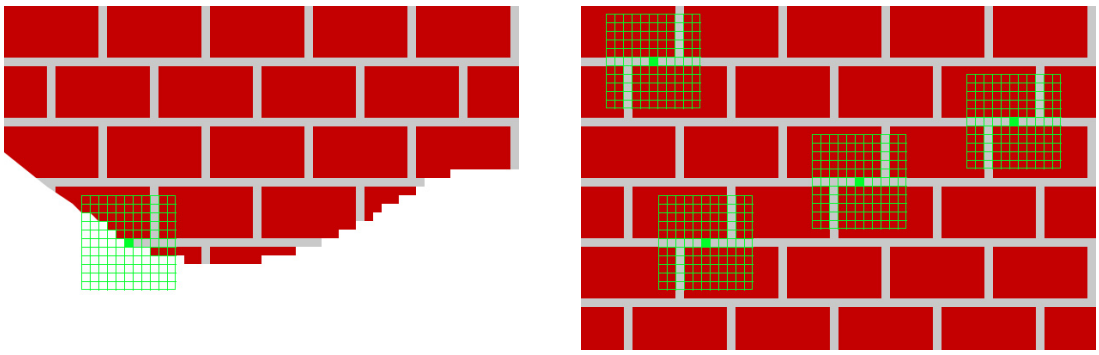


(a) (b)

Abbildung 3.2: (a) Nicht stationär und lokal, (b) stationär und lokal

Unterschiedliche Bereiche einer Textur sehen sich immer ähnlich (siehe Abb. 3.2 (b)). Dies ist nicht der Fall für normale Images wie wir bei Abb. 3.2 (a) erkennen können. Zudem ist es möglich jeden Pixel in (b) allein durch seine benachbarten Pixel zu bestimmen. Diese Attribute bezeichnet man als Stationär und Lokal.[8]

Funktionsweise eines Algorithmus basierend auf Markov-Random-Fields nach Efros und T. Leung: [9]



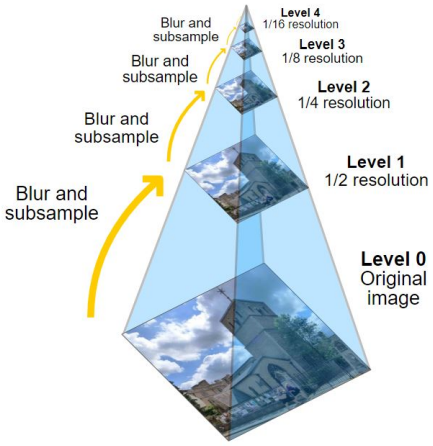
(a) (b)

Abbildung 3.3: (a) Pixelsynthese, (b) Sampling des Input-Images

- Zuerst wird vom Input-Image eine Teiltextur einer bestimmten Größe (z.B. 5 x 5 Pixel Fenster) ausgewählt. Von diesem Feld aus werden Spiralförmig neue Pixel generiert.
- Für jeden Pixel der betrachtet wird, wird ein Fenster einer selbst bestimmten Größe zentral über das Pixel gelegt. Die Größe des Fensters muss nach Größe der einzelnen Elemente der Textur gewählt werden.
- Mit dieser Gruppe von Pixel (der Zentrale Pixel und alle seiner Nachbarn im Fenster) werden nun alle im Input-Image ähnlichen N Kandidaten gesucht.
- Danach wird zufällig von einer dieser Kandidaten ausgewählt und der betrachtete Pixel im Output wird aus diesem Kandidaten kopiert.
- Dieser Prozess wiederholt sich so lange, bis alle nicht bekannten Pixel generiert wurden.([8], S.4)

3.2 Pyramid basierende Textur Analyse / Synthese

Bei der Pyramid-Methode wird das Verfahren der Bildpyramide verwendet. Hierbei werden aus dem Input-Image mehrere Output-Images in verschiedenen Auflösungen mithilfe von Glättung und Downsampling generiert (siehe Abb. 3.4 (a)). [8][10]
Zudem wird ein Bildrauschen, *(Noise-Image)* der in der regel uniform Weiß ist, verwendet. Das Noise-Image wir dann mithilfe von Histogram-Matching und der Image-Pyramid so verändert, das es dem Input-Image ähnlich ist.



(a)

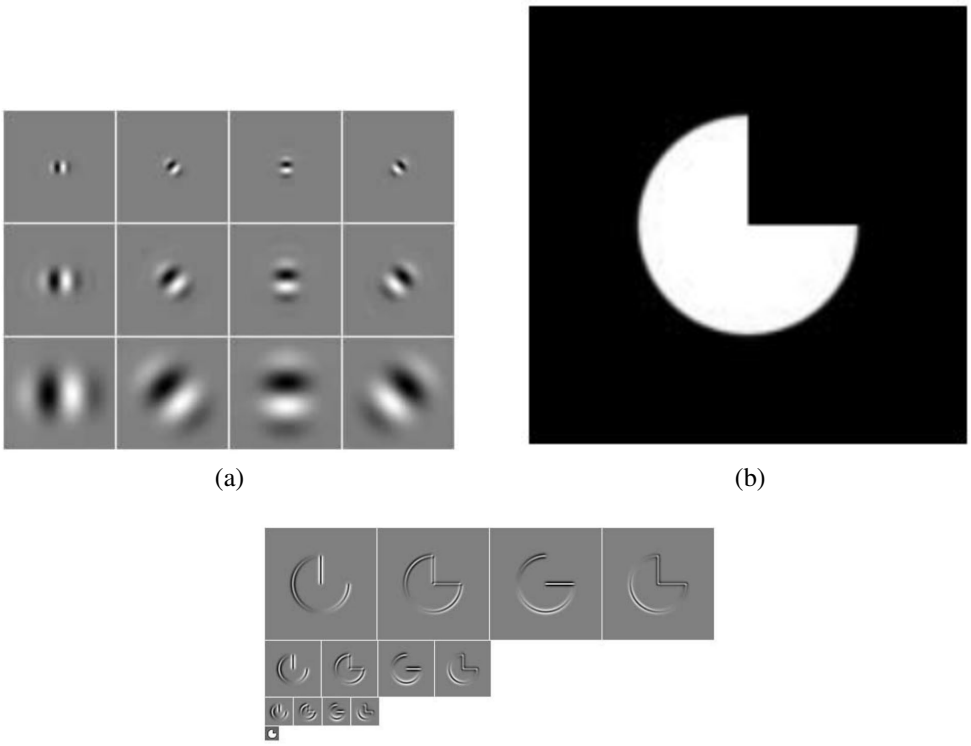
Abbildung 3.4: (a) Bildpyramide

Histogram-Matching ist die Generalisierung des Punktoperators, mehr spezifisch der Histogrammäqualisation. Bei der Histogrammäqualisation (auch Histogrammausgleich, Histogrammeinebnung, Histogrammegalisierung oder Histogrammequalisierung genannt) werden die Kontraste von Grauwertbildern derart verbessert, sodass sie über eine bloße Kontrastverstärkung hinausgeht. Dabei wird die Gleichverteilung mithilfe der Grauwertverteilung berechnet, damit der gesamte zur Verfügung stehende Wertebereich optimal ausgenutzt wird.[11] Bei dem Fall von der Pyramid-Methode nimmt der Algorithmus ein Input-Image und zwingt es über ein Paar von Nachschlagetabellen, ein bestimmtes Histogramm zu haben. Die beiden Nachschlagetabellen Tabellen sind:

1. die kumulative Verteilungsfunktion *(cumulative distribution function (CDF))* eines Bildes und

2. die inverse kumulative Verteilungsfunktion eines Bildes.

Die CDF ist eine Nachschlagetabelle, die das Intervall [0,256] auf das Intervall [0,1] abbildet. Die inverse CDF ist eine Nachschlagetabelle, die von [0, 1] auf [0, 256] zurückführt. Sie wird (mit linearer Interpolation) neu abgetastet, sodass die Stichproben gleichmäßig auf dem Intervall [0, 1] verteilt sind.[8][12]
Während der Algorithmus weiter Iteriert, beginnt das Noise-Image dem Input-Image zu ähneln. Wir Stoppen die den Prozess, wenn wir eine ausreichende Ähnlichkeit erreicht haben oder wir eine von uns festgelegte Anzahl von Iterationen erreicht haben.



(a) (b)

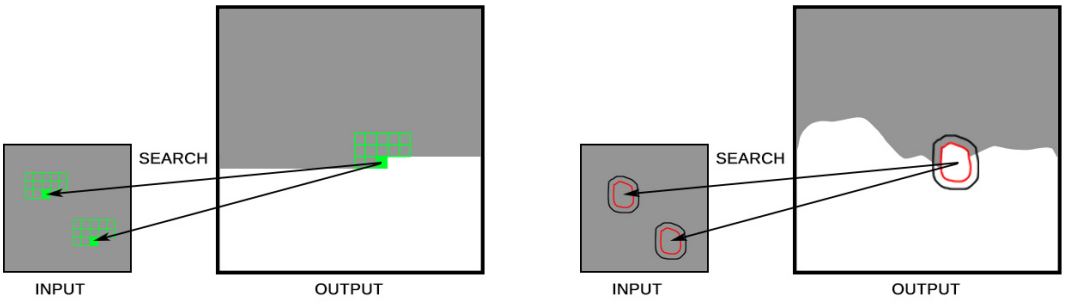
(c)

Abbildung 3.5: (a) Projektion der Bildpyramide, (b) Input-Image, (c) Teilband-Bilder vom Input-Image

Obwohl die Pyramid-Methode ist im Detail viel komplizierter ist, werde ich sie hier nicht weiter Behandeln da sie für das weitere Verständnis dieser Arbeit keine Relevanz hat und nur eine weitere Synthesemethode darstellen soll.

3.3 Patch basierende Textursynthese

Die Patch-Based Textursynthese *(auch Quilten genannt)* ist gewissermaßen eine Erweiterung der Pixel basierende Textursynthese. Hier werden statt einzelne Pixel gleich direkt ganze Felder *(Patches)* verglichen und generiert (siehe Abb. 3.6). Auch hier werden die Patches mithilfe ihrer benachbarten Pixel bestimmt und ausgewertet. Dadruch erhöht sich die Qualität und die Geschwindigkeit des Algorithmus.



(a) (b)

Abbildung 3.6: (a) Pixel-Based, (b) Patch-Based

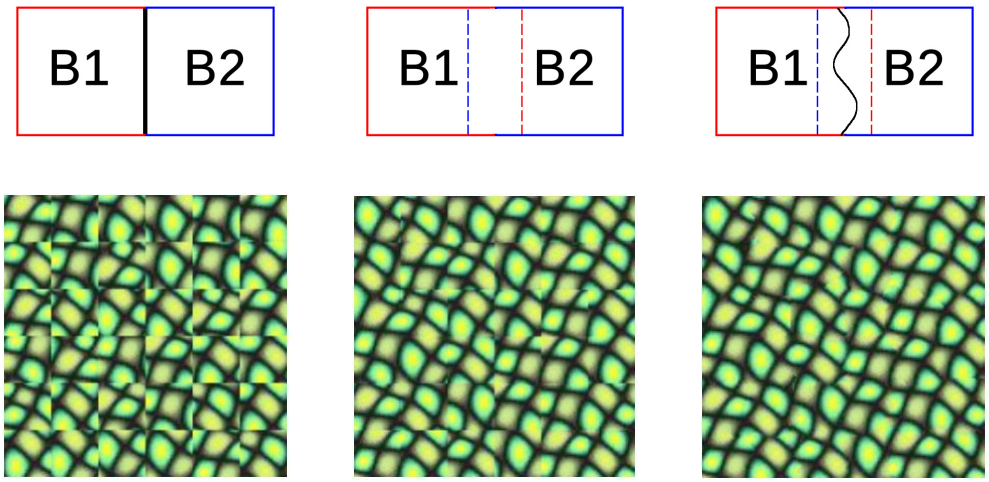
Ein Problem dieser Synthese im Vergleich zur Pixel-Based Synthese ist, das sich hier die neuen Patches mit bereits vorhandenen Patches überschneiden. Es gibt viele Methoden, um dieses Problem zu lösen. Eine davon ist es Patches mit verschiedenen Größen zu verwenden, damit die Konfliktbereiche zwischen den Patches durch das Phänomen der visuellen Maskierung *(Visual Masking)* reduziert werden. Gerade bei stationären Texturen erreicht diese Methode gute Ergebnisse.[8]

Die Funktionsweise des Algorithmus nach D.Gomathi und Rajvi Shah:

1. Generierung des ersten Patches in der oberen-linken Ecke des Output-Images. Der Patch wird zufällig aus dem Input-Image ausgewählt.

2. Von links-nach-rechts und von oben-nach-unten werden folgende Aufgaben im Output-Image ausgeführt.

    - Auswahl des nächsten hinzuzufügenden Feldes aus dem Input-Image aus den am besten passenden Patches.
    - Berechnen der Fehlerfläche zwischen diesem neuen Patch und seinem Überlappungsbereich mit bereits verarbeiteten Patches.
    - Berechnen des Pfades mit den geringsten Kosten durch die Fehlerfläche, um die Patch grenze zu bestimmen, und fügen Sie dann den neuen Patch zum Output-Image hinzu.



(a) (b) (c)

Abbildung 3.7: (a) Zufällige Patch Anordnung, (b) Patches eingeschränkt durch Nachbarn, (c) Minimal Fehler Randschnitt. Nach Alexei A. Efros and Thomas K. Leung [9]

4. Wave-Function-Collapse im Vergleich

Wichtig bei allen Textursynthesen ist, das die Muster des Output-Images immer lokal ähnlich oder gleich dem des Input-Images sind. Das wird größtenteils dadurch erzielt das aus dem Input-Image kleinere Subimages, Patches oder Pixel extrahiert werden. Bei den verfahren, wo die lokale Ähnlichkeit nicht 1-zu-1 bzw. pixelgenau stattfindet, werden die Pixel und deren Farbwert oft nach Grundlage der Abstandsmetrik (z.B. dem euklidischen Abstand von Pixelfarbvektoren) beurteilt. Solche Verfahren finden meistens in der rein visuellen Computergrafik Anwendung. Diese Methodiken haben große Nachteile im Gegensatz zu Algorithmen wo das lokale Muster des Outputs pixelgenau dem Input-Image gleicht. Gerade bei PCG (Procedural-Content-Generation) kann die Pixelgenauigkeit von großen Nutzen sein da dadurch Abgrenzungen der Pixel innerhalb des Output-Images klar definiert sein können. [1] Von allen oben beschriebenen Textursynthese Methoden ist WFC wahrscheinlich der Patch-Based Methode am ähnlichsten.

Gumin hat sich von der Arbeit von Paul Merell Inspirieren lassen, obwohl dieser sich Hauptsächlich mit der Generierung von 3D-Modellen befasst hatte. Bei seinem Verfahren werden die Modelle mithilfe von bereits erstellten Bausteinen zusammengesetzt. Das ist dahingehen wichtig da in vielen Textursynthesen gerade bei den Übergängen die Pixel sich Mischen und somit sich Artefakte bilden können. Dieser Verhalten ist bei WFC und dem Verfahren von Paul Merell nicht möglich da es sich um eine diskrete Synthese handelt. Jedes lokale Muster ist immer im Input wiederzufinden (siehe Abb. 4.1). [1][3][4]

(a)

Abbildung 4.1: (a) WFC generiertes Muster

4.1 Constraint-Satisfaction-Problem

Der WFC von Gumin ist lose an der Quantenmechanik angelehnt. Das liegt daran, dass bei der Synthese von WFC in jeder Zelle des $N \times N$ Output-Images theoretisch jedes Muster / Pixelwert vorkommen kann bevor sie final festgelegt werden. Dieser Zustand nennt sich *Superposition*. Jede Zelle hat mehrere Eigenwerte *(eigenstates)* und somit auch eine maximale Entropie bzw. einen maximalen Informationsgehalt. Auch hat jede Zelle bestimmte Regeln, bestimmt durch ihre Nachbarn, die erfüllt werden müssen. Denn nicht jede Zelle kann an jeder Position auftauchen, sobald bereits eine Zelle ein Eigenwert erhält. (Siehe Patch-Based Textursynthese. Nicht jeder Patch kann an einer beliebigen Position sein. Sie müssen sich schließlich an den Nachbarn anpassen damit die Synthese nahtlos ist.) Sobald eine Zelle bekannt wird *(Observation)* und damit nur einen Eigenwert besitzt, dann wird die Entropie aller anderen Zellen angepasst. [2] Da der WFC Algorithmus genau auf diese Art und Weise seine Synthese durchführt, dann kann man den WFC auch als Löser für solche Bedingungserfüllungsprobleme *(Constraint-Satisfaction-Problem)* verwenden.

Was ist ein Constraint-Satisfaction-Problem (CSP)? Grundsätzlich beschreiben CSP's Gruppen von Objekten denen Variablen zugeteilt sind. Diesen Variablen sind Regeln, sogenannte *(constraints)*, auferlegt die erfüllt werden müssen. Jeder dieser Variablen hat zu Beginn eine Superposition und kann damit jeden wert enthalten. Die Aufgabe von Algorithmen zum Lösen von CSP's *(solver)* ist es einen Zustand *(State)* zu finden in denen alle constraints erfüllt sind und jeder Variable nur noch ein wert zugeordnet ist. Für solche Probleme finden sich oft bei der Künstlichen Intelligenz und aus dem Operations Research. [5] Im Fall von WFC sind die Objekte, denen die Variablen zugeteilt sind, die einzelnen Bereiche im Output-Image. Jeder diese Bereiche muss ein lokales Muster aus dem Input zugeordnet werden. Immer, wenn einem Bereich ein wert zugeordnet wird, dann werden auch die benachbarten Bereiche damit beeinflusst *(Propagation)*. Der Gesamtprozess, wenn sich eine Gruppe aus Superpositionen mit mehreren Eigenwerten zu einem einzelnen Eigenwert aufgrund von Interaktion mit der Außenwelt *(Observation)* reduziert, nennt sich Wave-Function-Collapse. [6] Während dem Prozess einen gültigen State für das CSP zu finden, dann gibt es immer Situationen in dem es mehrere gültige Optionen für eine Variable gibt. Wenn so eine Situation auftritt, dann haben verschiedene Solver verschiedene Ansätze. Einige Algorithmen wähle zufällig eine der möglichen Werte von momentan zulässigen Optionen. Bei so einem Ansatz kann es sein das der Algorithmus nicht auf einen Zustand kommen kann, in dem alle constraints erfüllt werden können. In so einem Fall gibt es Rücksetzverfahren *(Backtracking)* bei dem der Algorithmus zu seinem letzten Ergebnis zurückfällt und ein anderen Wert für die Variable setzt, um aus dem ungültigen Zustand zu kommen. Andere Algorithmen verwenden zusätzliche Heuristiken abgesehen von den bereits bekannten constraints, um die Möglichkeit eines ungültigen Zustandes zu reduzieren. [1]

4.2 Wave-Function-Collapse in Game Development

In Abschnitt 4.0 wurde bereits auf mögliche Vorteile einer pixelgenauen Textursynthese hingewiesen. Der große Vorteil einer solchen Synthese ist, das dadurch der generierte Content auch Interaktiv genutzt werden kann, da wir vollen Kontrolle über die lokalen Muster des Outputs haben. Dadurch bietet sich WFC gerade für die Spiele-Entwicklung an, da keine unvorhersehbaren Artefakte entstehen können. Viele PCG Methoden Garantieren häufig Ergebnisse, wobei die Kosten für diese Garantie in Form einer unvorhersehbaren Gesamtlaufzeit einhergeht. Diese Backtracking Löser liefern bei ihren Aufgaben für die Lösung Ihrer CSP's oft gute Leistungen, wobei diese Ergebnisse in Theorie aber nur schwer zu charakterisieren sind. Eine ßchnelle"Methode von Horswill und Foged [7] um eine spielbares Leveldesign zu generieren ist der (AC3) Algorithmus. Dieser Algorithmus basierend auf Backtrackingsearch mit Constraint Propagation stellt zwar geringe Anforderungen an Prozessor- und Speicherressourcen, allerdings wird erwartet, dass er von Programmierern verwendet werden die sich wenigstens mäßig mit Suchalgorithmen und Design auskennen. Bereits ein Tag nach der Veröffentlichung von Gumin's WFC am 30. September 2016 haben viele Entwickler bereits begonnen mit diesem Algorithmus zu experimentieren. Grund für die große Beliebtheit dafür ist nicht nur die bereits erwähnten Pixelgenauigkeit, sondern auch die live Generierung des Outputs selbst. [1]

Viele PCG's Methoden variieren in ihrer Laufzeit bei der Generierung ihres Outputs. Dies führt dazu das manchmal bereits Großteile des Outputs sofort generiert werden dafür aber der Abschluss aufgrund von komplexen constraint solving Algorithmen nicht gleich-

mäßig entsteht. [1]

5. Begriffserklärung

6. Theorie

7. Stand der Forschung

8. Ergebnisse

9. Diskussion der Ergebnisse

10. Fazit

11. Literaturverzeichnis

[1] https://adamsmith.as/papers/wfc_is_constraint_solving_in_the_wild.

pdf
[2] https://en.wikipedia.org/wiki/Wave_function_collapse
[3] https://paulmerrell.org/wp-content/uploads/2021/06/thesis.pdf
[4] https://github.com/mxgmn/WaveFunctionCollapse
[5] https://en.wikipedia.org/wiki/Constraint_satisfaction_problem
[6] https://en.wikipedia.org/wiki/Wave_function_collapse
[7] https://ojs.aaai.org/index.php/AIIDE/article/view/12511/12364
[8] http://rajvishah.weebly.com/uploads/6/3/0/9/6309814/texture_synthesis_final_report.pdf
[9] https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/papers/efros-iccv99.pdf
[10] https://de.wikipedia.org/wiki/Bildpyramide
[11] https://de.wikipedia.org/wiki/Punktoperator_(Bildverarbeitung)
[12] https://www.cns.nyu.edu/heegerlab/content/publications/Heeger-siggraph95.pdf
https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_2016_paper.pdf
https://www.th-nuernberg.de/fileadmin/global/Gelenkte_Doks/Fak/SW/SW_0600_HR_Leitfaden_WA_public.pdf
https://www.ghost-writing.net/wissenschaftliche-arbeit-auf-englisch-verfassen/
https://www.youtube.com/watch?v=rI_y2GAlQFM&t=1135s&ab_channel=TheCodingTrain
https://users.informatik.haw-hamburg.de/~abo781/abschlussarbeiten/ba_westfalen.pdf
https://users.informatik.haw-hamburg.de/~abo781/abschlussarbeiten/ba_dzaebel.pdf
http://people.csail.mit.edu/celiu/Patch-based%20Texture%20Synthesis/Index.htm
https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/papers/efros-iccv99.pdf

12. Anhang

13. Eidesstattliche Erklärung