Simulation biologischer Wachstumsprozesse mit Hilfe zellulärer Automaten

Diplomarbeit

von

Jürgen Dollinger

Abteilung für mathematische Physik
Universität Ulm

Inhaltsverzeichnis

Kapitel 1

Einleitung

Diese Diplomarbeit beschäftigt sich überwiegend mit zellulären Automaten. Zuerst war geplant das Wachstum von Tumoren, aber auch andere biologische Vorgänge zu simulieren und mit eingescannten Fotos zu vergleichen. Zu solchen Zwecken dienen eigens in unserer Abteilung entwickelte Bildverarbeitungsprogramme, mit deren Hilfe sich inbesondere die fraktale Dimension, als ein Charakteristikum für die Geometrie eines Gegenstandes, genau bestimmen läßt.

Zum einen erwies sich das Konzept des zellulären Automaten als sehr interessant aber auch umfangreich, zum andern standen keine geeigneten Fotos zur Verfügung. Daher beschränkt sich diese Arbeit auf die Simulation.

Im Zuge der theoretischen Analyse einer bestimmten Art zellulärer Automaten, den Lattice Gas Automaten, wurde ein anderes Teilgebiet, das in unserer Abteilung behandelt wird, interessant: Die Boltzmanngleichung.

Ansonsten bestimmten programmiertechnische Probleme den Alltag. Hierfür wurde die noch neue Programmiersprache *Cellang*, die ausschließlich der Programmierung zellulärer Automaten dient und dafür besonders geeignet ist, ausgewählt. Programme zur Initialisierung und Weiterverarbeitung der Daten wurden in C geschrieben.

Diese Arbeit wäre nicht möglich gewesen ohne die elektronische Kommunikation über das Internet, insbesondere mit dem *Cellang*-Erfinder Dana J. Eckart, den ich über den Netzwerkdienst News kennenlernte.

Kapitel 2

Zelluläre Automaten

Zelluläre Automaten haben in den letzten Jahren immer mehr Aufmerksamkeit erreicht. Das Interesse reicht von der theoretischen Informatik über die angewandte Physik bis hin zu Geschichtswissenschaften.

2.1 Definition

Ein zellulärer Automat ist ein diskretes dynamisches System. Sein Raum, seine Zeit und sein Wertevorrat sind diskret. Jeder Punkt $\vec{x} \epsilon \mathbb{Z}^d$ eines regelmäßigen räumlichen Gitters, im folgenden Zelle genannt, nimmt einen von endlich vielen Zuständen ein. Diese Zustände werden nach einer lokalen, deterministischen Regel synchron berechnet, d. h. der Zustand einer Zelle zu einem bestimmten Zeitpunkt hängt nur von seinem eigenen Zustand einen Zeitschritt vorher und von den Zuständen der benachbarten Zellen einen Zeitschritt vorher ab.

Lokale Regel bedeutet, daß der neue Wert nicht von den Werten irgendwelcher weit entfernter Zellen abhängt, sondern nur von dem Wert der Zelle selbst und dem Wert ihrer Nachbarn. Die Art der Nachbarschaft ist zwar nicht grundsätzlich eingeschränkt, aber wird meist sehr klein und symmetrisch gewählt. Außerdem hängt die Regel nicht vom absoluten Ort ab, ist also translationsinvariant.

Wichtig ist dabei, neben der Diskretheit und Lokalität, das synchrone updaten aller Gitterpunkte. Der Wert eines Gitterpunkts hängt nicht von dem seiner Nachbarn zum gleichen Zeitschritt, sondern nur von deren Wert zum vorhergehenden Zeitschritt ab.

In einer etwas mathematischeren Sichtweise kann man zelluläre Automaten über eine lokale oder eine globale Regel definieren:

Sei $a^t_{\vec{x}} \in S$, $S \subset \mathbb{Z}$ der Zustand des Automaten zur Zeit t am Ort $\vec{x} \in \mathbb{Z}^d$. Dann ist mit

$$a^{t+1}_{\vec{x}} = \Phi(\{a^t_{\vec{y}_i}\}) \tag{2.1}$$

eine lokale Regel definiert. Es gilt

$$||\vec{x} - \vec{y}_i|| = 1 \quad \forall i \tag{2.2}$$

mit einer geeigneten Norm $||.||$, die die Art der Nachbarschaft bestimmt.

Die globale Regel $F : S^{\mathbb{Z}^d} \to S^{\mathbb{Z}^d}$ ist durch $F(a)_{\vec{x}} = \Phi(\{a_{\vec{y}_i}\})$ gegeben. Zelluläre Automaten werden dann durch folgende Eigenschaften gekennzeichnet:

- der Zustandsraum $S^{\mathbb{Z}^d}$ ist kompakt.
- Die Funktion F ist stetig.
- Die Funktion F kommutiert mit dem Translationsoperator.

2.2 Game of Life

Das bekannteste Beispiel eines zelluläre Automaten ist das Game of Life [1]. Es wird nicht nur als Bildschirmschoner verwendet, sondern ist auch Gegenstand ernsthafter Forschung[2].

Die Zellen bilden ein zweidimensionales Gitter und können zwei verschiedene Werte annehmen. Der Wert einer Zelle ist eins falls im Zeitschritt vorher entweder 3 der 8 Nachbarn den Wert eins hatten oder die Zelle selbst den Wert eins hatte und 2 Nachbarn den Wert eins hatten. Sonst wird der Wert null. Eine Implementierung in der Sprache *Cellang* (siehe Abschnitt 3.3) zeigt Abb. 2.1.

```
2 dimensions of 0..1

sum := [0, 1] + [1, 1] + [1, 0] + [-1, 1] + [-1, 0] +
       [-1, -1] + [0, -1] + [1, -1]

cell := 1 when (sum = 2 & cell = 1) | sum = 3
     := 0 otherwise
```

Abbildung 2.1: Das Game of Life: Zunächst wird die Summe aller acht Nachbarn, die durch die relativen Koordinaten referenziert werden, berechnet, dann der neue Wert der Zelle bestimmt.

2.3 Eigenschaften und Erweiterungen

Zelluläre Automaten heißen ..., falls ...

autoplectic Auch aus einfachen Anfangsbedingungen ergeben sich komplizierte[1] Endzustände[2]. Man könnte autoplectic etwa mit "selbstrandomisierend" übersetzen. Ein Beispiel ist der Automat in Abb. 2.3 rechts unten.

homoplectic Nur aus komplizierten Anfangsbedingungen ergeben sich komplizierte Endzustände.

additiv Der Endzustand einer Summe von Anfangszuständen ergibt sich als Summe der Endzustände zu den einzelnen Anfangszuständen. Dabei ist die Summe jeweils modulo einer Maximalzahl auszuwerten. Der Zustand solcher Automaten läßt einfach vorhersagen; man berechnet den Zustand mit einer einfachen Anfangsbedingung und stellt die kompliziertere Anfangsbedingung als Überlagerung von solchen einfachen Anfangsbedingungen dar, wobei man die Translationsinvarianz ausnutzt.

---

[1] Die Begriffe "einfach" und "kompliziert" werden hier nicht näher definiert, was die ganze Definition etwas schwammig macht

[2] Falls der Automat nicht stationär wird so gibt es eigentlich keinen Endzustand. Gemeint ist dann der Zustand nach langer Zeit, wenn man aufhört zu beobachten.

homogen Die Regeln hängen nicht vom absoluten Ort im zellulären Automaten-Universum ab. Dies ist normalerweise stets der Fall.

totalistic Die Regeln hängen nur von der Summe der Nachbarn ab.

Typische Eigenschaften zellulärer Automaten sind:

- Einfache lokale Regeln.
- Abhängigkeit nur einen Zeitschritt zurück. (Analogie: Markov-Prozeß)
- Kompliziertes globales Verhalten. "Chaos" und "Selbstorganisation".
- Skalierbarkeit. Manche Automaten erzeugen selbstähnliche Muster.

Man kann zelluläre Automaten aber auch um bestimmte Features erweitern:

- Zwei oder mehr Werte pro Zelle. Dies braucht man für die Lattice Gas Automaten in Kapitel 4. Allerdings ist das keine echte Erweiterung, denn man kann natürlich mehrere diskrete Werte in *einem* diskreten Wert kodieren. In der Praxis wird man die niederwertigen bits als den einen, die höherwertigen bits als den andern Wert interpretieren. Analog für mehrere Werte.
- Räumlich erweiterte Nachbarschaft. Meist verwendet man nur die nächsten Nachbarn, man kann aber auch Nachbarschaften beliebiger fester Größe und Form nehmen. In der mathematischeren Definition ist dies durch die freie Wahl der Norm aus Gl. 2.2 bereits enthalten.
- Erweiterung der der zeitlichen Abhängigkeit, d. h. nicht nur der Zustand zum vorherigen Zeitschritt ist für die neue Konfiguration verantwortlich sondern auch der Zustand eine begrenzte Anzahl von Zeitschritten davor. Dies kann jedoch auch durch zusätzliche Werte pro Zelle implementiert werden die dann die "alten" Werte zwischenspeichern.
- Aufgabe des synchronen Updating [2].
- Regeln, die vom Zufall beeinflußt werden. Da man aber mit deterministischen Regeln Pseudozufallszahlen erzeugen kann und echte Zufallszahlen sowieso schwer aufzutreiben sind, ist diese Erweiterung eher theoretischer Natur.

2.4 Anwendungsmöglichkeiten

- Diskrete Approximation partieller Differentialgleichungen.
- Simulation biologischer Systeme. z. B. [3].
- Berechenbarkeitstheorie. Zelluläre Automaten können als Berechnungssyteme angesehen werden, wobei die gesamte Information über ihr künftiges Verhalten in ihrem Anfangszustand gespeichert ist. Manche Automaten sind äquivalent zu einer Turingmaschine und somit "fähig zur allgemeinen Berechnung"[3]. Das bedeutet, daß, zumindest theoretisch, jeder endliche Algorithmus mit ihrer Hilfe implementiert werden kann[8].
- Erzeugung von "Zufallszahlen". Vor allem die nicht selbstähnlichen "autoplectischen" Klasse 3 Automaten [4, 5].

[3] capable for universal computation

- Kryptographie [6]. Man interpretiert eine Nachricht als Anfangsbedingung eines geeigneten Automaten und rechnet mehrere Zeitschritte weiter um die verschlüsselte Nachricht zu erhalten.
- Thermodynamische Simulationen und Strömungsmechanik. Siehe Kapitel 4.

Es gibt einfach zu viele Anwendungen um sie hier alle aufzuzählen.

2.5 Eindimensionale Automaten

Eindimensionale Automaten können durch die Anzahl N der Zellen, die Art der Regel

$$a_i^{t+1} = \Phi(a_i^t, a_{i-1}^t, a_{i+1}^t, \ldots, a_{i+r}^t)$$

mit dem Radius der Nachbarschaft r, sowie den Wertebereich $0..k$ beschrieben werden. Die Regel wird häufig durch eine sog. Wolframzahl [7] beschrieben: Man schreibt alle k^{2r+1} möglichen Konfigurationen eines Gitterpunktes und seiner Nachbarn der Reihe nach auf, und bildet aus jeder mit Hilfe des Automaten den neuen Wert des mittleren Gitterplatzes. Die so entstandene Ziffernfolge interpretiert man im Zahlensystem zur Basis k und rechnet ins Dezimalsystem um. Regel 90 (binär 01011010 siehe Abb. 2.2) bedeutet also, daß aus $110, 100, 011$ und 001 jeweils eine Eins entsteht, aus anderen Konfigurationen eine Null.

$$\frac{111}{0} \quad \frac{110}{1} \quad \frac{101}{0} \quad \frac{100}{1} \quad \frac{011}{1} \quad \frac{010}{0} \quad \frac{001}{1} \quad \frac{000}{0}$$

Abbildung 2.2: Entstehung der Regel 90 ($k = 1$, $r = 1$).Die obere Zeile besteht aus den 8 möglichen Zuständen dreier benachbarter Zellen, die untere Zeile aus dem Wert des mittleren Gitterpunkts zum nächsten Zeitschritt.

Eindimensionale Automaten werden nach der Vorhersagbarkeit ihres Langzeitverhaltens in vier Klassen eingeteilt [8].

Klasse 1 Diese Automaten streben von fast allen Anfangszuständen *einem* festen Endzustand zu in dem alle Zellen denselben Wert haben. Es mag zwar Anfangszustände geben, die nicht in diesen festen Endzustand übergehen, jedoch geht deren relative Anzahl gegen Null für $N \to \infty$

Klasse 2 Diese Automaten streben festen Endzuständen zu oder gehen in kurze ($< k$) Zyklen. Es gibt aber beständige Strukturen, d. h. der Endzustand ist räumlich inhomogen und hängt vom Anfangszustand ab. Die relative Anzahl der Anfangszustände, die längere Zyklen bilden, geht, analog Klasse 1, gegen null.

Klasse 3 Fast alle Anfangszustände führen zu nicht periodischen, "chaotischen" Zuständen. Endliche Automaten sind natürlich immer periodisch, hier ist die Periode eben sehr groß d. h. fast alle k^N möglichen Zustände werden durchlaufen. Einige Automaten bilden selbstähnliche fraktale Strukturen, andere sind irregulärer. Für große Zeiten lassen sich statistische Aussagen über den Zustand machen.

Klasse 4 Je nach Anfangsbedingung stirbt der Automat aus (d. h. alle Zellen haben den selben Wert) oder es bilden sich beständige Strukturen. Solange ein Zustand nicht ausgestorben oder echt periodisch ist, läßt sich aber keinerlei Aussage über sein späteres Verhalten machen ohne explizit nachzurechnen. Diese

Automaten sind "fähig zur allgemeinen Berechnung" d. h. man kann deren Äquivalenz zu einer Turingmaschine nachweisen.

Die Aufteilung auf die Klassen ist selbst für eindimensionale Automaten umstritten, da obige Definition nicht ganz eindeutig ist. Trotzdem lassen sich auch für mehrdimensionale Automaten ähnliche Einteilungen machen. Einige Beispiele für eindimensionale Automaten sieht man in Abbildung 2.3, 2.4 und 2.5. Es ist jeweils der Zustand des Automaten nach rechts und die Zeit nach unten aufgetragen. Ein dunkler Pixel kodiert eine Eins, ein weißer eine Null.

Verteilung einiger Automaten auf die Klassen:

	$k = 2$ $r = 1$	$k = 2$ $r = 2$	$k = 3$ $r = 1$
Klasse 1	0.5	0.25	0.12
Klasse 2	0.25	0.16	0.19
Klasse 3	0.25	0.53	0.6
Klasse 4	0	0.06	0.07

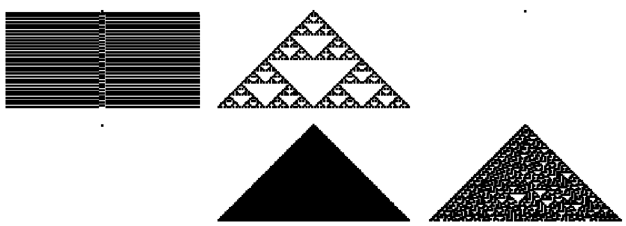

Abbildung 2.3: Eindimensionale Automaten mit einem einzelnen besetzten Gitterplatz als Anfangsbedingung. Obere Reihe von links nach rechts $k = 1, r = 1$ Regel 37, 90, 8. Untere Reihe $k = 1, r = 2$ totalistische Regel 20, $k = 1, r = 1$ Regel 254 und 30. Der Automat links unten und der rechts oben stirbt bereits nach dem ersten Zeitschritt aus.

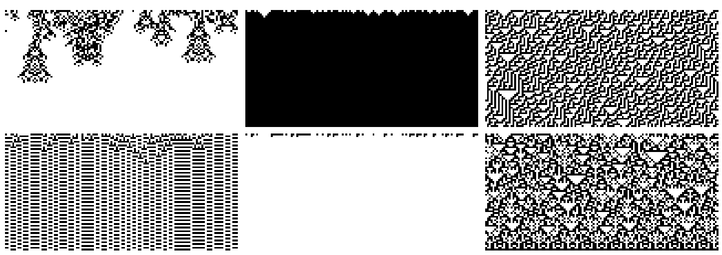

Abbildung 2.4: Dieselben Automaten wie in Abb. 2.3, nur mit zufälliger Anfangsbedingung. Die Aufteilung auf die Klassen ist leicht ersichtlich. Links oben Klasse 4, links unten Klasse 2, die beiden mittleren Klasse 1 und rechts Klasse 3

Abbildung 2.5: Eindimensionaler Automat der Klasse 4 (totalistic rule 20, $k = 1$, $r = 2$) mit verschiedenen zufälligen Anfangsbedingungen.

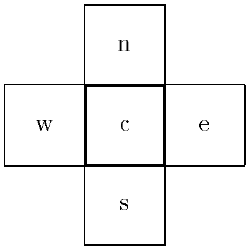

Abbildung 2.6: von Neumann Nachbarschaft

2.6 Nachbarschaften in $d = 2$

2.6.1 von Neumann Nachbarschaft

Dies ist die einfachste Art der Nachbarschaft, nur die vier Nachbarn an den Seiten jeder Zelle werden berücksichtigt. Häufig werden wie in Abbildung 2.6 die Zellen mit den Himmelsrichtungen identifiziert. Meist wird auch die Zelle selbst mit zur Nachbarschaft gerechnet.

2.6.2 Moore Nachbarschaft

Die Moore Nachbarschaft ist die logische Erweiterung der von Neumann Nachbarschaft um die Zellen in der Ecke. Sie besteht also aus allen 8 Nachbarn an den Kanten und Ecken jeder Zelle. Von Neumann und Moore Nachbarschaft sind auch auf d Dimensionen verallgemeinerbar. Die von Neumann Nachbarschaft besteht dann aus $2d + 1$ Zellen, die Moore Nachbarschaft aus 3^d Zellen.

2.6.3 Margolus Nachbarschaft

Diffusion von Gasen kann durch Implementierung eines random walk mit einer sog. Margolus Nachbarschaft simuliert werden [11]. Es handelt sich dabei um nicht überlappende Blöcke von je vier Zellen, die stets gleich auf sich selbst abgebildet werden. Um die Interaktion der einzelnen Blöcke zu erreichen, werden diese jeden

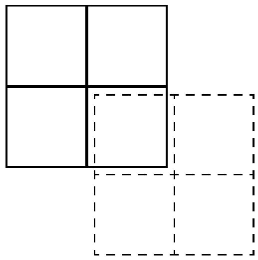

Abbildung 2.7: Margolus Nachbarschaft

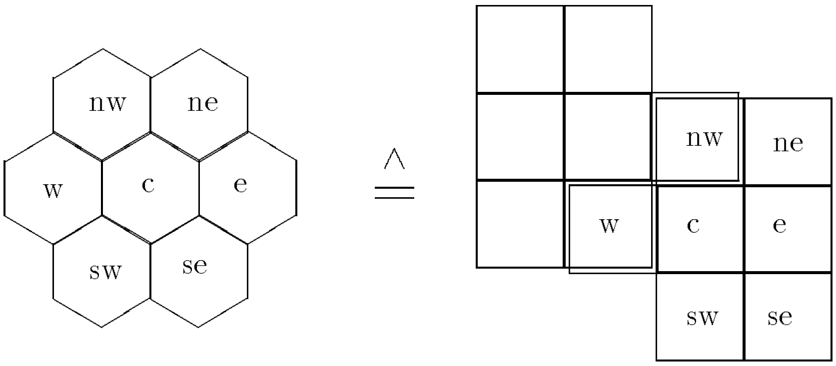

Abbildung 2.8: Hexagonale Nachbarschaft

Zeitschritt alternierend verschoben. Zur Simulation eines random walk läßt man einfach die Teilchen innerhalb eines solchen Blocks zufällig rotieren. Abbildung 2.7 demonstriert die Margolus Nachbarschaft. Der Block mit den durchgezogenen Linien wird z. B. zu geraden Zeiten, der mit den gestrichelten Linien zu ungeraden Zeiten benutzt. Jederzeit wird das gesamte Gitter von nicht überlappenden Blöcken bedeckt.

2.6.4 Hexagonale Nachbarschaft

Ein hexagonales Gitter braucht man für hexagonale Gittergasautomaten die im Kap. 4 näher behandelt werden. Um eine hexagonale Nachbarschaft mit einem normalen quadratischen Gitter, das leicht in jeder Programmiersprache, insbesondere aber in *Cellang* zu realisieren ist, zu implementieren, muß man zu einem kleinen Trick greifen [12]. Verschiebt man jede zweite Zeile des quadratischen Gitters um eine halbe Zellbreite, so erhält man ein hexagonales Gitter. In Praxis bedeutet das, daß in jeder zweiten Zeile die 6 Nachbarn so zu wählen sind, daß von der Moore Nachbarschaft die beiden linken Ecken unberücksichtigt bleiben, in der anderen Zeile die beiden rechten Ecken. Um die Zeilen voneinander zu unterscheiden wird ein Wert "which" eingeführt der dadurch, daß er gerade oder ungerade ist, die Nachbarschaft anzeigt.

2.7 Beispiel: Zufallszahlen in zwei Dimensionen

Um für die Lattice Gas Automaten in Kapitel 4 eine Zufallszahl für jede Zelle zur Verfügung zu stellen, muß man einen Automaten programmieren, der mit zufälliger Anfangsbedingung die Zufallszahlen immer wieder durchmischt. Dabei sollten weder zeitliche noch räumliche Korrelationen auftreten.

Die Idee einfach die Summe der Nachbarn modulo einer Maximalzahl als neue Zufallszahl für jede Zelle zu benutzen kann so schlecht nicht sein. Schließlich werden Zufallsgeneratoren auch so ähnlich programmiert. Zunächst betrachten wir den Fall, daß die Zufallszahl entweder Null oder Eins ist. Man hat also einen totalistischen, additiven, zweidimensionalen Automaten mit dem Wertevorrat $\{0, 1\}$.

Das erste Problem, auf das man stößt, ist die Initialisierung, die man sinnvollerweise mit einem kleinen C-Program erledigt:

```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 128
main()
{
int i,j;
printf("0\n");
for(i=0;i<SIZE;i++)
   for(j=0;j<SIZE;j++)
      printf("[%d, %d] = %d\n",i,j,rand()%2);
}
```

Steigert man die Größe des Automaten so stellt man fest, daß die Werte des Zufallsgenerators alles andere als zufällig sind. Die man page unter SUNOS gibt Auskunft:

```
The low bits of the numbers generated are not very random; use
the middle bits.
```

oder unter HPUX:

```
The spectral properties of rand() leave a great deal to be
desired. drand48() provides a much better, though more elaborate,
random-number generator (see drand48(3C)).
```

Allerdings stellt sich heraus daß die niederwertigen bits von `drand48` auch nicht gerade zufällig sind. Die mittleren bits sind aber in Ordnung.

Nachdem diese Klippe umschifft ist, schreibt man das zugehörige *Cellang* Programm:

```
2 dimensions of
   random of 0..1
end
cell.random := ([1,0].random + [-1,0].random +
                [0,1].random + [0,-1].random)%2
```

Das Resultat sieht man in Abb. 2.9. Ab einem gewissen Zeitpunkt nehmen alle Zellen den Wert Null an. Ganz prima!

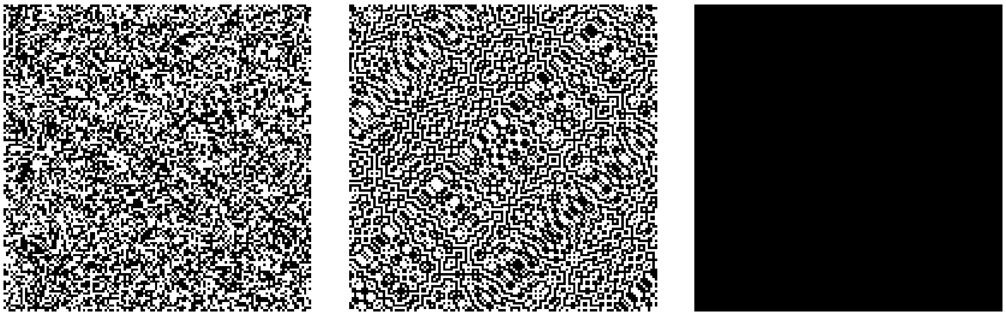

Abbildung 2.9: Automat der die eigene Zelle nicht mitsummiert, nach 1, 63 und 64 Zeitschritten. Ab diesem Zeitpunkt sind alle Zellen Null.

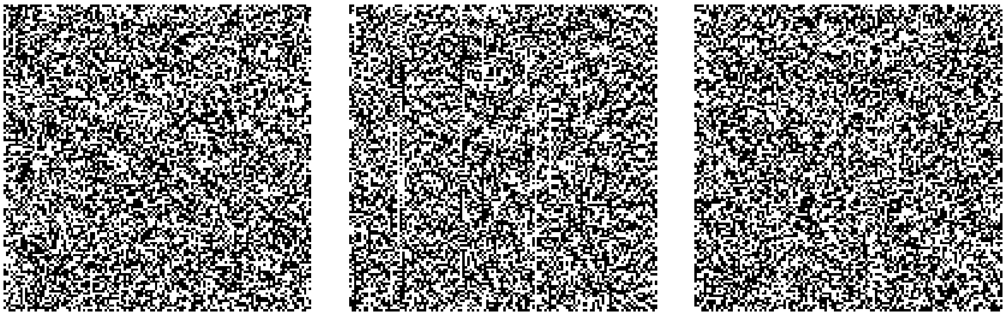

Abbildung 2.10: Automat aus Abb. 2.13 der die eigene Zelle mitsummiert, nach 1, 64 und 2000 Zeitschritten. Ein schwarzer Pixel kodiert eine Null, ein heller eine Eins.

Dieser Automat ist auch als "Fredkin rule" [13] bekannt. Ändert man das Programm so ab, daß auch die Zelle selbst in die Summe mit einbezogen wird (Abb. 2.13), sind die Ergebnisse schon etwas sinnvoller. Siehe Abb. 2.10.

Der Grund für ein solches Verhalten offenbart sich, wenn man von einer einfachen Anfangsbedingung ausgeht, nämlich von einer einzelnen Eins in der Mitte. Aufgrund der Additivität der Regeln läßt sich dann auf das Verhalten mit der zufälligen Anfangsbedingung schließen. Wie in Abb. 2.11 und 2.12 ersichtlich reproduziert sich die anfängliche Konfiguration mehrmals auf einer größeren Längenskala. Und zwar stets viermal nach außen. Nur falls die Zelle selbst in die Summe mit einbezogen wird, bleibt auch noch eine fünfte Konfiguration zusätzlich in der Mitte. Diese Reproduktion findet stets nach 2^n Zeitschritten um 2^n Zellen nach außen verschoben statt.

Nach 64 Zeitschritten, der halben Automatengröße, überlagern sich dann aufgrund der periodischen Randbedingungen jeweils zwei identische Konfigurationen auf den beiden Rändern. Diese löschen sich gegenseitig aus. Wird die Zelle selbst mitsummiert, bleibt die Anfangskonfiguration übrig. Im andern Fall löscht sich alles aus.

Der Effekt ist also ein Artefakt der periodischen Randbedingungen, zusammen mit der Verwendung einer Zweierpotenz als Automatengröße.

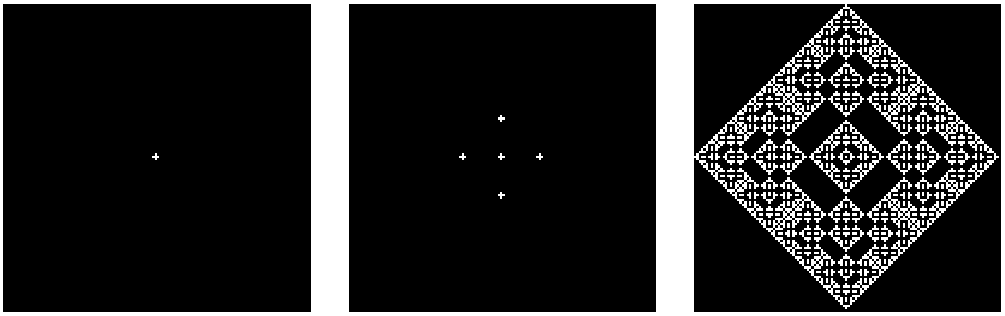

Abbildung 2.11: Automat aus Abb. 2.13 der die eigene Zelle mitsummiert, nach 1, 17, und 63 Zeitschritten. Im 65. Zeitschritt sieht der Automat wieder genau wie im ersten Bild aus.

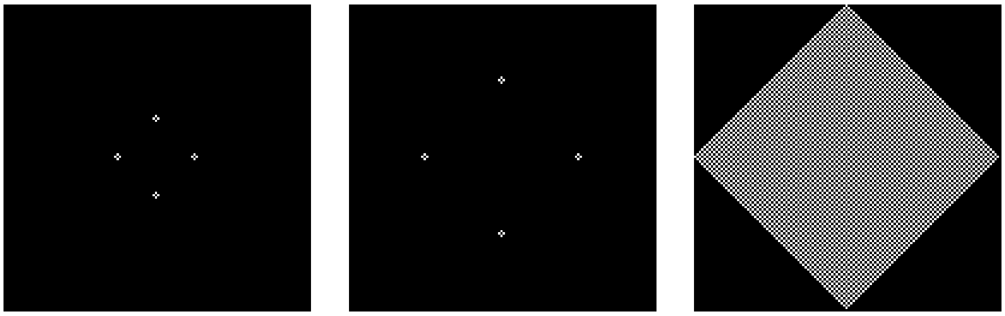

Abbildung 2.12: Automat der die eigene Zelle nicht mitsummiert, nach 17, 33 und 63 Zeitschritten. Ab dem 64. Zeitschritt sind alle Zellen Null.

```
2 dimensions of
random of 0..1
end
cell.random := (cell.random + [1,0].random + [-1,0].random
               + [0,1].random + [0,-1].random)%2
```

Abbildung 2.13: Programm das die eigene Zelle mitsummiert.

Kapitel 3

Implementierung zellulärer Automaten

Bisher war nur von einem abstrakten Konzept des zellulären Automten die Rede. Dieses läßt sich aber auf modernen Computern in gewisser Weise in die Tat umsetzen. Aufgrund der diskreten Werte sind die sonst in der Numerik gefürchteten Rundungsfehler von vornherein ausgeschlossen.

3.1 Hardwarelösungen

Eigentlich ist so ein Computer selbst einem Zellulären Automaten sehr ähnlich: Diskrete Bits befinden sich in diskreten Speicherstellen und werden zu diskreten Zeiten (Taktfrequenz!) verändert. Nur wird statt der lokalen Regel eine zentrale Steuerung durch eine CPU verwendet.

Zelluläre Automaten sind geeignet auf massiv parallelen Rechnern implementiert zu werden. Das sonst übliche Problem der Kommunikation der Prozessoren untereinander entfällt, da wegen der lokalen Regeln nur benachbarte Prozessoren Informationen austauschen müssen. Komplizierte Architekturen wie bei der n-Cube im Rechenzentrum unserer Universität sind nicht notwendig, man schaltet die Prozessoren einfach in einem Gitter zusammen, am besten für jede Zelle einen.

Bekannte Hardwareimplementierungen sind:

- CAM Cellular Automata Machine [35]
- CART Cellular Automata Research Tool [12]
- Connection Machine

3.2 Cellsim, Mathematica und andere

Einen Überblick über die verschiedenen Implementierungsmöglichkeiten (Hardware und Software) geben [29, 2]. Erwähnenswert sind Cellsim und Mathematica.

Cellsim ist ein CA-Simulationsprogramm mit eigener Programmiersprache, das aber

Compiler und Viewer[1] in einem Programm verbindet. Es ist ursprünglich für Sunview gemacht, einer SUN-spezifischen graphischen Oberfläche, inzwischen soll es aber auch eine X11-Version[2] geben.

Mathematica ist das bekannte Mathematikpaket von Stephen Wolfram, der bei dessen Design wohl auch schon Anwendungen für zelluläre Automaten im Sinn hatte. In [18] wird ein kleines Programm vorgestellt das in einer Art Vorgängersprache von Mathematica verfaßt ist. Mit Rücksicht auf die beschränkte Anzahl der Mathematica-Lizenzen und in Anbetracht des bekanntermaßen nicht gerade sparsamen Umgangs von Mathematica mit dem Hauptspeicher habe ich aber darauf verzichtet. Vielleicht war's ja ein Fehler?

3.3 Cellular

Cellular ist ein Programmpaket zur Implementierung zellulärer Automaten auf verschiedenen Hardwareplattformen. Es besteht aus einer Programmiersprache *Cellang* und dem zugehörigen Compiler `cellc`, einem Skript das die Ausführung steuert `avcam`[3] und einem Viewer `cellview`. Ein compiliertes *Cellang*-Programm hat ein festgelegtes I/O Format. Der Output kann über eine pipe entweder dem mitgelieferten Viewer übergeben werden oder von anderen Programmen weiterverarbeitet werden.

So wurden von mir Programme entwickelt, die den *Cellang*-Output meiner Lattice Gas Automaten in platzsparende Bildformate wie "gif" oder in LaTeX-Bilder wandeln und überdies noch Teilchenstatistik betreiben. Der *Cellang*-Output ist leicht zu verstehen, da er aus einfachen ASCII-Zahlenkolonnen besteht. Bei großen Automaten wird er aber schnell sehr groß: Ein einzelner Zeitschritt hat bei einem 1024×1024 Automaten mehrere Megabyte.

Dieses Baukastensystem ist eigentlich eine typische UNIX[4] Denkweise. Allerdings gibt es inzwischen auch eine DOS-Version.

Frühe Veröffentlichungen über *Cellular* [30, 31] bezogen sich auf die Version 1.0, die Version 2.0 findet man auf vielen ftp-Servern.

Die Aktuelle Version von *Cellular* bekommt man vom Author

J Dana Eckart
Box 6933
Computer Science Department
Radford University
Radford, VA 24142

email

`<dana@rucs.faculty.cs.runet.edu>`

oder via anonymous ftp: **rucs2.sunlab.cs.runet.edu** '`pub/ca/cellular.tar.gz`'

[1] Der Compiler übersetzt die Programme, mit dem Viewer kann man das Ergebnis am Bildschirm sichtbar machen.

[2] X11 ist die graphische Oberfläche die im UNIX-Bereich am verbreitetsten, und auch auf den Workstations in unserer Abteilung installiert ist. X11 wird oft fälschlich auch als X-Windows bezeichnet. (Siehe `man X`)

[3] abstract virtual cellular automata machine

[4] UNIX is a registered trademark of AT&T.

Ich begann mit der Version 3.0. Bald stellte sich heraus, daß der Sprache ein paar Kleinigkeiten fehlten. So konnte man als Index für "Arrayaliases"[5] nur Konstanten oder um Konstanten geshiftete `forall`-Indexvariablen verwenden. Auf meine Anregung hin wurden dann auch nichtkonstante Shifts zugelassen. Jetzt kann man auch die Modulosumme mehrerer `forall`-Indexvariablen verwenden [32].

Als dann die neue Version herauskam, die jetzt auch deutlich schnelleren Code produziert, mußten erst mal die dadurch zwangsläufig eingebauten Bugs beseitigt werden. Ständiger e-mail Austausch mit Dana[6] J. Eckart war hierfür nötig. Dies war auch ein nicht unwesentlicher Teil meiner Diplomarbeit und wohl auch der Teil mit dem größten "bleibenden Wert".

Auch die mitgelieferten Beispielprogramme wurden durch eines meiner Programme ergänzt. Dieses Beispielprogramm simuliert zwei nichtmischende Flüssigkeiten ohne Ruheteilchen. Als Dana J. Eckart über das World Wide Web einige meiner Ergebnisse zu sehen bekam, war er erstaunt, was mit "seiner" Sprache so alles möglich ist: "`I did look at your mosaic page and am very impressed. The fluid stuff looks VERY nice. ( I'm actually a little surprised, and quite pleased, that my system was able to do it ).`"

Aktuelle Features von Cellang sind u. a.:

- Mehrere Werte für jede Zelle.
- Beliebige endliche Nachbarschaften.
- Prinzipiell unbeschränkte Anzahl von Dimensionen.
- Diskrete Zellwerte und diskrete Zeitschritte.
- Synchrones Updating.
- Periodische Randbedingungen.
- Die Operatoren +, −, *, /,% (Rest), +% (Modulo Addition), −% (Modulo Subtraktion), ! (logisches nicht), & (logisches und), | (logisches oder), =, !=, <= und >=.
- Die vordefinierten Variablen `time` und `cell`. `time` ist eine ganze Zahl die den Zeitschritt bezeichnet. `cell` ist eine Struktur, die am Anfang jeden Cellang-Programms durch eine Zelldeklaration näher erklärt werden muß. Nur durch eine Zuweisung an diese Variable kann der Wert einer Zelle verändert werden. Der neue Wert wird aber erst mit dem nächsten Zeitschritt aktiv, wodurch das synchrone Updating unterstützt wird.
- `if` ... `then` ... `elsif` ... `else` ... `end` Abfragen.
- `forall` Schleifen, die ihren Indexbereich an vordefinierten Arrayaliases erkennen können. Diese sind die einzige Kontrollstruktur für Wiederholungen.
- Zuweisungen mit Nebenbedingungen (`when`).

Für näher Interessierte empfehle ich die Sprachreferenz [32] die man mit dem Paket *Cellular* erhält.

[5] Eine Art abgespeckte Version eines normalen Arrays, die eng mit den `forall`-Schleifen zusammenarbeitet. Näheres siehe [32].

[6] Dana ist in diesem Fall ein Männername.

Was noch nicht möglich ist, ist die Modularisierung von *Cellang*-Programmen. Es gibt weder Funktionen noch kann ein *Cellang*-Programm aus mehreren Dateien zusammengesetzt werden. Dies könnte durch die Verwendung von m4-Makros[7], die ich sowieso schon zur Festlegung globaler Konstanten verwende, zumindest teilweise erreicht werden.

Da *Cellang* optimal an die Programmierung zellulärer Automaten angepaßt ist sind normalerweise *Cellang*-Programme so klein und einfach, daß weder Funktionen noch Modularisierung notwendig wird. Im Fall kompliziertester Lattice Gas Automaten jedoch[8], die zwar vielleicht kompakter aber dafür unverständlicher programmiert werden könnten[9], wird eine Grenze erreicht, die zumindest einfache Makros erfordert.

[7] m4 ist eine unter UNIX übliche Makrosprache.
[8] siehe Abschnitt 6.3
[9] Zur Abschreckung hätte ich da zwei Fortranprogramme.

Kapitel 4

Gitter-Gas-Automaten

Gitter-Gas-Automaten[1] sind ein sehr einfaches, aber dennoch taugliches Modell einer fiktiven Mikrowelt zur Simulation makroskopischer strömungsmechanischer Systeme. Es besteht aus idealisierten Teilchen, die sich entlang diskreter Richtungen bewegen und Stöße machen, wenn sie sich an Gitterpunkten treffen. Lattice Gas Automaten sind spezielle zelluläre Automaten.

4.1 Geschichte der Lattice Gas Automaten

4.1.1 HPP Gas

Die ersten Lattice Gas Automaten wurden von J. Hardy, Y. Pomeau und O. de Pazzis [14, 15, 16] entwickelt. Auf einem quadratischen Gitter befinden sich identische Teilchen. Alle Teilchen haben betragsmäßig die gleiche Geschwindigkeit und die gleiche Masse. Wie das bei zellulären Automaten sein muß, hüpfen die Teilchen zu diskreten Zeiten von einer Zelle zur nächsten. Die Geschwindigkeit beträgt also eine Zelle pro Zeitschritt. Aufgrund der Symmetrie des Gitters gibt es vier mögliche Richtungen, die jeweils von maximal einem Teilchen besetzt werden. Es gibt also bis zu vier Teilchen auf jedem Gitterplatz.

Normalerweise laufen die Teilchen stets in diese eine Richtung weiter. Aufgrund der Synchronizität des Gitterupdates können auch nie zwei Teilchen gleichzeitig auf demselben Gitterplatz mit derselben Richtung zusammenkommen. Kommen jedoch zwei oder mehr Teilchen auf einem Gitterplatz zusammen, so ereignen sich — mit gewissen Wahrscheinlichkeiten — Stöße, die die Teilchenzahl- und die Impulserhaltung gewährleisten.

Sinnvoll sind nur Zweiteilchenstöße. Bei vier Teilchen würde sich nichts ändern, da alle Richtungen besetzt sind, bei nur einem Teilchen oder bei drei Teilchen (eines fehlt) kann sich nichts ändern, da sonst einer der Erhaltungssätze verletzt würde. Kommen aber genau zwei Teilchen aus entgegengesetzter Richtung in einer Zelle an, geschieht der Zweiteilchenstoß und die beiden Teilchen fliegen in die beiden anderen, vorher nicht besetzten Richtungen weg.

Natürlich hat das Gitter in einer realen Implementierung eine endliche Größe, daher müssen Randbedingungen vereinbart werden. Üblich sind periodische Randbedin-

[1] Das ist eine etwas künstliche Übersetzung des englischen "lattice gas automata". Im folgenden wird auch der Ausdruck "Lattice Gas Automaten" benutzt.

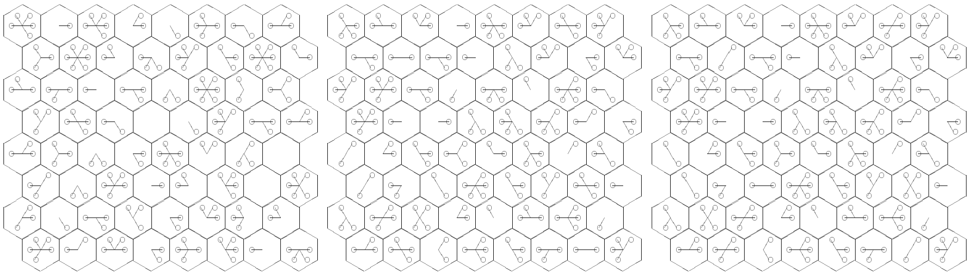

Abbildung 4.1: FHP-Gas vor der Bewegung (links), nach der Bewegung, vor dem Stoß (mitte) und nach dem Stoß (rechts)

gungen. Man kann aber auch "Wände" anbringen, an denen die Teilchen in die Richtung reflektiert werden aus der sie kommen. Wollte man ein Refexionsgesetz, mit Einfallswinkel gleich Ausfallswinkel, implementieren so müßte wegen der Lokalität der Regeln in jeder Wandzelle eine Information darüber gespeichert sein in welche Richtung die die Wand verläuft.

Dieses einfache Modell hat bereits einige bemerkenwerte Eigenschaften:

- Die typische Längenskala einer kollektiven Bewegung ist wesentlich größer als die mittlere freie Weglänge eines Teilchens.
- Es gibt lokale thermodynamische Gleichgewichte.

Das HPP Gas hat jedoch ein Problem: Es ist nicht isotrop. Der "Impulsflußtensor" reduziert sich sich unter der vierzähligen Symmetrie nicht zu einem Skalar [17].

4.1.2 FHP Gas

Um das Problem des HPP Gases zu lösen haben U. Frisch, B. Hasslacher und Y. Pomeau [17] ein ähnliches Modell auf einem hexagonalen Gitter entworfen. Das von den Autoren als HLG [2] bezeichnete Modell ist isotrop und makroskopisch galileiinvariant.

Wie beim HPP Gas gliedert sich die Bewegung der Teilchen in zwei Teilschritte. Zunächst bewegen sich die Teilchen in die Richtung die von ihrer Geschwindigkeit vorgegeben ist, und dann ereignen sich Stöße. Die Teilschritte sieht man in Abb. 4.1. Sinnvoll sind Zwei-, Drei- und Vierteilchenstöße, obwohl im Originalmodell nur Zwei- und Dreiteilchenstöße vorgesehen waren. Zusätzlich zu den in Abb. 4.2 gezeigten Stößen sind noch asymmetrische Dreiteilchenstöße mit Impuls- und Teilchenzahlerhaltung verträglich bei denen der Gesamtimpuls vor dem Stoß nicht verschwindet. Es handelt sich quasi um einen Zweiteilchenstoß bei dem das dritte Teilchen unberührt bleibt. Solche Stöße werden jedoch meist nicht berücksichtigt.

Folgende Eigenschaften sind grundlegend:

- Es liegt ein hexagonales Gitter zugrunde.
- Auf jeder Zelle können sich bis zu sechs Teilchen befinden, in jede Richtung eines.

[2] hexagonal lattice gas

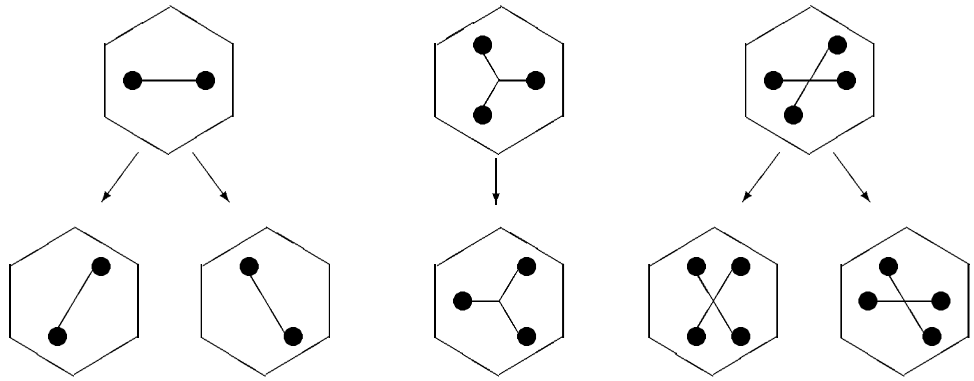

Abbildung 4.2: Stöße des FHP-Gases

- Die Teilchen hüpfen zu diskreten Zeiten zwischen den Zellen.
- Jeder Zeitschritt gliedert sich in einen Propagationsteil und einen Stoßteil.
- Bei Stößen gilt Impulserhaltung.
- Bei Stößen gilt Teilchenzahlerhaltung.
- Sind von einer Konfiguration ausgehend mehrere Stöße möglich, werden diese mit gleicher Wahrscheinlichkeit ausgeführt.

4.2 Ein wenig Theorie

Mikroskopisch gesehen bestehen Flüssigkeiten aus diskreten Teilchen, makroskopisch werden sie durch die Navier-Stokes-Gleichungen beschrieben. Änderungen in den Gesetzen zwischen den Molekülen wirken sich allenfalls auf Parameter wie die Viskosität, nicht aber auf die Form der Gleichungen aus. Man kann daher die Teilchen auch durch sehr einfache Modelle beschreiben, die dann auf makroskopischem Level den Navier Stokes Gleichungen äquivalent sind.

Im folgenden betrachten wir ein FHP Gas [18]. Das hexagonale Gitter wird durch die diskreten Vektoren $\vec{x}$, die jeweils auf eine Zelle zeigen beschrieben. Von einer Zelle aus erhält man deren Nachbarn, indem man die Einheitsvektoren

$$\vec{e}_a = (\cos(2\pi a/6), \sin(2\pi a/6))$$

hinzuaddiert. Zu jedem Zeitschritt bewegt sich ein Teilchen zunächst gemäß seiner Geschwindigkeit $\vec{e}_a$ zur nächsten Zelle. Dort werden die Teilchen neu angeordnet, und es finden Stöße statt, die Impuls und Teilchenzahl erhalten.

4.2.1 Diskrete Boltzmann Gleichung

Wir betrachten nun die Einteilchenverteilungsfunktion $f_a(\vec{x}, t)$ die die Wahrscheinlichkeit angibt am Ort $\vec{x}$ zur Zeit t ein Teilchen mit der Geschwindigkeit $\vec{e}_a$ zu finden. Die Verwendung einer solchen Einteilchenverteilungsfunktion stellt eine aus der statistischen Physik bekannte Näherung dar, die dann gut ist wenn die Teilchen gering miteinander korreliert sind.

Betrachtet man keine Stöße, so gilt für die zeitliche Entwicklung von f

$$f_a(\vec{x}+\vec{e}_a, t+1) = f_a(\vec{x}, t). \tag{4.1}$$

In makroskopischen Maßstäben ist der Abstand zwischen den Zellen und der zeitliche Abstand zwischen den Zeitschritten (hier diskret 1) sehr klein. Eine Taylorentwicklung liefert die stoßfreie Boltzmanngleichung

$$\partial_t f_a(\vec{x}, t) + \vec{e}_a \cdot \nabla f_a(\vec{x}, t) = 0. \tag{4.2}$$

Man beachte den Unterschied in der Bedeutung von $\vec{x}$ bzw t in den beiden letzten Gleichungen. In Gleichung 4.1 ist $\vec{x}$ ein diskreter Vektor zu einer bestimmten Zelle. In 4.2 ist $\vec{x}$ eine makroskopische Variable, der man gar nicht mehr anmerkt, daß sie nur diskrete Werte annehmen kann.

Betrachtet man nun auch Stöße so erhält man zusätzlich einen Stoßterm Ω_a, also

$$\partial_t f_a(\vec{x}, t) + \vec{e}_a \cdot \nabla f_a(\vec{x}, t) = \Omega_a. \tag{4.3}$$

Betrachten wir der Einfachheit halber nur Zweiteilchenstöße, die stets und mit gleicher Wahrscheinlichkeit ausgeführt werden sollen falls zwei Teilchen head–on–head aufeinanderstossen, so ist

$$\Omega_a = \frac{1}{2}\Lambda_a(1,4) + \frac{1}{2}\Lambda_a(2,5) - \Lambda_a(0,3)$$

mit

$$\Lambda_a(i,k) = \frac{f_{a+i}}{1-f_{a+i}} \frac{f_{a+j}}{1-f_{a+j}} \prod_{j=1}^{6}(1-f_{a+j}),$$

wobei alle Indices der f_b modulo 6 ausgewertet werden. Betrachtet man auch andere Stöße, so kommen entsprechend kompliziertere Terme dazu. Falls Stöße nicht immer oder mit verschiedener Wahrscheinlichkeit ausgeführt werden, so sind die $\Lambda_a(i,k)$, noch mit den zugehörigen Wahrscheinlichkeiten γ_{ik} zu gewichten.

Stets müssen jedoch die Erhaltungssätze für Impuls und Teilchenzahl gelten

$$\sum_a \vec{e}_a \Omega_a = 0 \tag{4.4}$$

$$\sum_a \Omega_a = 0. \tag{4.5}$$

Führt man die makroskopische Geschwindigkeit $\vec{u}$ und die Teilchenzahldichte n mittels

$$\sum_a f_a = n \tag{4.6}$$

$$\sum_a \vec{e}_a f_a = n\vec{u} \tag{4.7}$$

ein, so kann man die Boltzmanngleichung 4.3 über a summieren und erhält

$$\partial_t \sum_a f_a + \sum_a \vec{e}_a \cdot \nabla f_a = 0. \tag{4.8}$$

Nach Umschreiben des zweiten Terms und Benutzung der makroskopischen Variablen erhält man die Kontinuitätsgleichung

$$\partial_t n + \nabla \cdot (n\vec{u}) = 0. \tag{4.9}$$

Die Impulserhaltung führt nach Multiplikation von 4.3 mit $\vec{e}_a$ und Summation zu einer etwas komplizierteren Gleichung

$$\partial_t \sum_a \vec{e}_a f_a + \sum_a \vec{e}_a (\vec{e}_a \cdot \nabla f_a) = 0. \tag{4.10}$$

Mit der Einführung des Impulsflußtensors

$$\Pi_{ij} = \sum_a (\vec{e}_a)_i (\vec{e}_a)_j f_a \tag{4.11}$$

erhält man

$$\partial_t (n u_i) + \partial_{x_j} \Pi_{ij} = 0. \tag{4.12}$$

4.2.2 Diverse Parameter

Die weitere Rechnung geht folgendermaßen: Man entwickelt die Verteilungsfunktion f_a nach den makroskopischen Größen n und $\vec{u}$ sowie deren Ableitungen. Man nennt dies eine Chapman-Enskog-Entwicklung.

$$\begin{aligned} f_a &= c^{(0)} n \left\{ 1 + c^{(1)} \vec{e}_a \cdot \vec{u} + c^{(2)} \left[(\vec{e}_a \cdot \vec{u})^2 - \frac{1}{2} |\vec{u}|^2 \right] \right. \\ & \left. + c_\nabla^{(2)} \left[(\vec{e}_a \cdot \nabla)(\vec{e}_a \cdot \vec{u}) - \frac{1}{2} \nabla \cdot \vec{u} \right] + \ldots \right\} \end{aligned} \tag{4.13}$$

Aus 4.4 und 4.5 folgt $c^{(0)} = 1/6$ und $c^{(1)} = 2$. Die beiden anderen Koeffizienten $c^{(2)}, c_\nabla^{(2)}$ hängen von der Art der Stöße ab und bestimmen die Viskositäten. Diese erhält man durch Vergleich der mit der Chapman-Enskog-Entwicklung erhaltenen Gleichung, mit der makroskopischen Navier-Stokes Gleichung.

Diese lautet in d Dimensionen

$$\frac{\partial}{\partial t} n\vec{u} + \mu n (\vec{u} \cdot \nabla) \vec{u} = -\nabla p + \eta \nabla^2 \vec{u} + \left(\zeta + \frac{\eta}{d} \right) \nabla (\nabla \cdot \vec{u}) \tag{4.14}$$

mit dem Druck p, der Scherviskosität η und der Volumenviskosität ζ. Der Vorfaktor μ ist normalerweise wegen der Galileiinvarianz Eins. Bei Lattice Gasen läßt sich dies nur durch eine Umnormierung der Geschwindigkeit $\vec{u}$ erreichen.

Man erhält für das FHP-Gas:

$$\eta = n\{12\frac{n}{6}(1 - \frac{n}{6})[\gamma_2 (1 - \frac{n}{6})^2 + 4\gamma_{3A} \frac{n}{6}(1 - \frac{n}{6}) + \gamma_4 (\frac{n}{6})^2]\}^{-1}, \tag{4.15}$$

$$\zeta = 0, \tag{4.16}$$

$$p = n/2. \tag{4.17}$$

Dabei sind die γ_i die Wahrscheinlichkeiten für Zweiteilchenstöße, antisymmetrische Dreiteilchenstöße und Vierteilchenstöße.

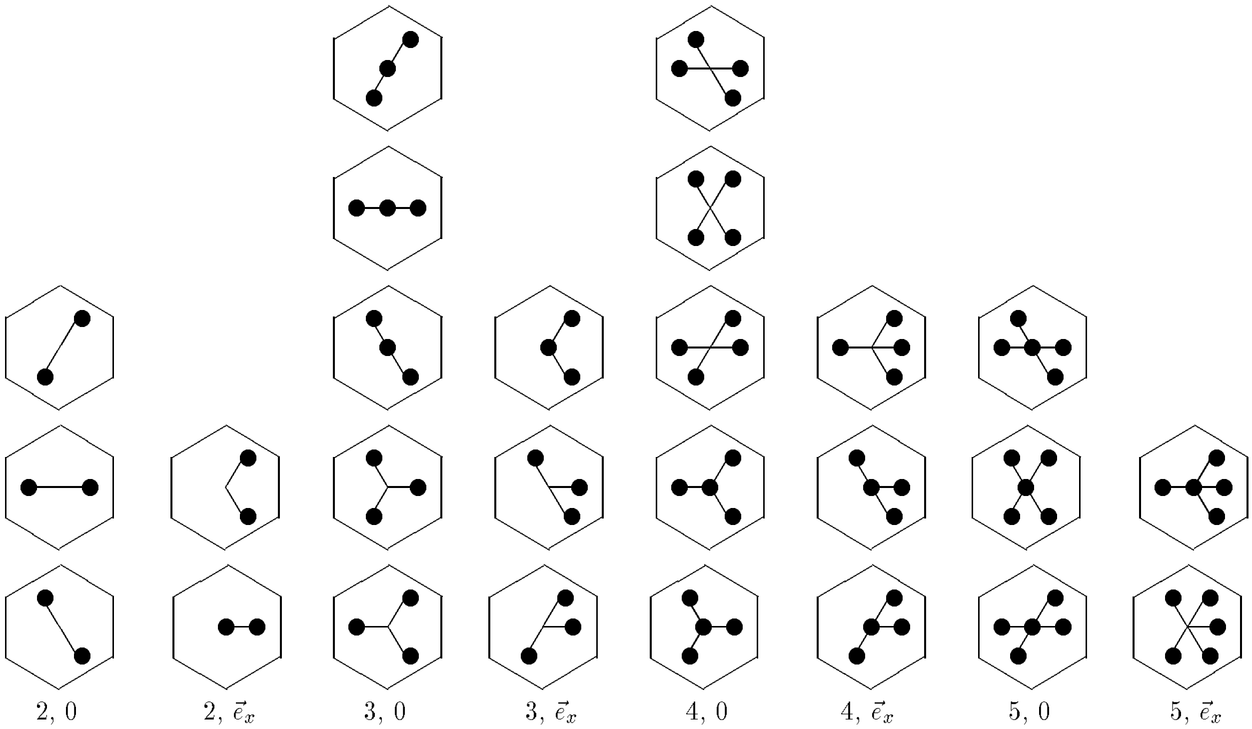

Abbildung 4.3: Konfigurationen des erweiterten FHP Modells. Alle Konfigurationen jeder Spalte haben die selbe Teilchenzahl und den selben Gesamt-Impuls. Die Konfigurationen, die aus den obigen mit Gesamt-Impuls $\vec{e}_x$ durch Drehung hervorgehen, und diejenigen die keine Stösse machen können sind nicht gezeigt. Die unterste Zeile gibt Teilchenzahl und Gesamt-Impuls an.

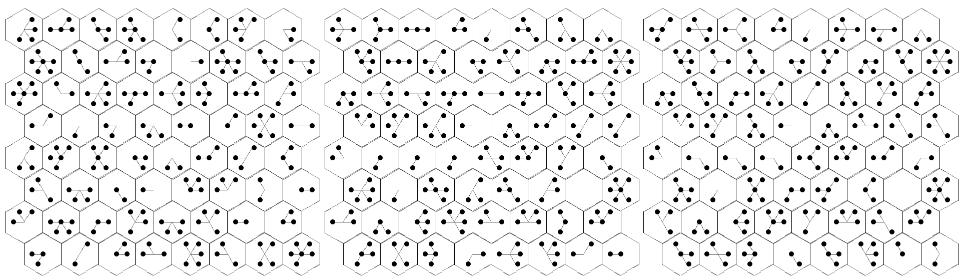

Abbildung 4.4: FHP-Gas mit Ruheteilchen vor der Bewegung (links), nach der Bewegung, vor dem Stoß (mitte) und nach dem Stoß (rechts)

4.3 Erweiterung mit ruhenden Teilchen

Um mehr Möglichkeiten bei den Stoßregeln zu haben, kann man zusätzlich noch ruhende Teilchen einführen. So hat man mehr Spielraum zur Einstellung makroskopischer Parameter. Anders als es bei Modellen der Fall wäre, die zwei von Null verschiedene Geschwindigkeiten benutzen, hat man hier keine Probleme damit, daß sich Teilchen gegenseitig überholen.

Man hat nach Einführung der Ruheteilchen also bis zu sieben Teilchen pro Zelle: Maximal eines in jede Richtung und maximal ein Ruheteilchen.

Da auch Stöße ausgeführt werden, die das Ruheteilchen in Bewegung setzen oder ein sich bewegendes Teilchen zum Ruheteilchen machen, gilt lokal keine Energieerhaltung mehr wenn man nicht den Ruheteilchen eine "intrinsische Energie" zuschreibt. Da aber beide Prozesse mit gleicher Wahrscheinlichkeit auftreten ist die makroskopische Energieerhaltung gesichert.

Alle möglichen Stöße zeigt Abb. 4.3. Je nachdem welche Stöße tatsächlich ausgeführt werden, unterscheidet man die einzelnen Modelle [20].

- **FHP I** Ohne Ruheteilchen, Zwei- und symmetrische Dreiteilchenstöße.
- **FHP II** Mit Ruheteilchen, Zweiteilchenstöße, alle symmetrischen Drei- und Vierteilchenstöße die das Ruheteilchen nicht verändern.
- **FHP III** Mit Ruheteilchen. Alle Stöße mit weniger als fünf Teilchen außer den symmetrischen Drei- und Vierteilchenstößen die das Ruheteilchen verändern.
- **Mein Modell** Mit Ruheteilchen. Alle Stöße, aber eventuell mit kontinuierlich variierbarer Wahrscheinlichkeit, diese doch nicht auszuführen.

Prinzipiell sind noch andere Regeln denkbar, sogar Modelle mit mehr als einem Ruheteilchen wurden schon probiert.

4.4 Zur Viskosität

Man kann zwar nach Vorbild von Abschnitt 4.2.2 für jedes der FHP Modelle die makroskopischen Parameter angeben [19, 20], und erreicht mit den Ruheteilchenmodellen z. B. eine nichtverschwindende Volumenviskosität, allerdings gelten solche Formeln nicht für beliebige Stoßwahrscheinlichkeiten. Sonst würde nämlich mit verschwindender Stoßwahrscheinlichkeit die Viskosität divergieren.

Die Chapman-Enskog-Entwicklung ist eigentlich eine Entwicklung für kleine Knudsenzahlen. Die Knudsenzahl ist das Verhältnis von mittlerer freier Weglänge zu einer typischen makroskopischen Größe z. B. der Gesamtgröße des Automaten.

Die mittlere freie Weglänge ist aber umgekehrt proportional der Stoßwahrscheinlichkeit oder bei realen Gasen dem Streuquerschnitt.

Im Bereich niedriger Stoßwahrscheinlichkeiten gilt zuerst die Chapman-Enskog-Entwicklung nicht mehr, dann aber wird die mittlere freie Weglänge makroskopisch und man kommt in Bereiche, in denen es überhaupt keinen Sinn mehr ergibt, eine Viskosität anzugeben, da die makroskopischen Größen nicht mehr für eine Beschreibung ausreichen [21, 22].

Es überrascht zunächst (mich zumindest), daß die Viskosität mit der Stoßwahrscheinlichkeit abnimmt. Tatsächlich ging ich auch lange Zeit davon aus, daß, falls mehr Stöße passieren, auch mehr Energie dissipiert wird und daher die Viskosität zunimmt. Dies ist aber sowohl für Lattice Gase als auch für reale Gase falsch, und beruht auf einer falschen Vorstellung von "Viskosität".

Vikosität ist der Diffusionkoeffizient für den Transport von Impuls über ein Geschwindigkeitsgefälle. Steigt beispielsweise die y-Komponente der Geschwindigkeit in x-Richtung an, so kann ein Teilchen mit kleinem x einen Schritt von der Größenordnung der mittleren freien Weglänge in Richtung steigender x-Werte zurücklegen. Dabei nimmt es *seinen* Impuls mit, bis es einen Stoß mit einem Teilchen, das eine größere y-Komponente der Geschwindigkeit hat, macht und dabei seinen Impuls überträgt.

Ist die mittlere freie Weglänge länger so kann das Teilchen zu höheren Geschwindigkeiten vordringen bevor es einen Stoß macht, wodurch der Impulstransport verstärkt und daher die Viskosität erhöht wird.

4.5 Nichtmischbare Flüssigkeiten

Man kann das Konzept der FHP Gase auch auf Flüssigkeiten mit verschiedenen Teilchensorten erweitern. Der Propagationschritt ist der selbe wie beim einphasigen Gas. Um die Teilchen auch sprachlich leichter unterscheiden zu können werden ihnen Farbeigenschaften zugewiesen. Z. B. "rote" und "grüne" Teilchen.

Im Kollisionsschritt unterscheidet man rein rote, rein grüne und gemischte Stöße. Die Regeln für einfarbige Stöße bestimmen die Eigenschaften der jeweiligen Flüssigkeit. Mit den Regeln für gemischte Stöße kann man Eigenschaften wie Oberflächenspannung implementieren.

Regeln mit hoher Oberflächenspannung bewirken den Effekt der Unmischbarkeit und sogar Entmischung [25, 26].

Die Implementierung einer Oberflächenspannung beruht auf der Berechnung einer Oberflächenarbeit und deren Minimierung für gemischtfarbige Stöße. Sei $r_j(\vec{x}) = 1$ falls am Ort $\vec{x}$ in Richtung j sich ein rotes Teilchen befindet. Analog zeigt $g_j(\vec{x})$ die An– oder Abwesenheit eines grünen Teilchens an. Man definiert ein Farbfeld (eine Art Farbgradient) durch

$$\vec{f} = \sum_a \vec{e_a} \sum_j [r_j(\vec{x} + \vec{e_a}) - g_j(\vec{x} + \vec{e_a})] \tag{4.18}$$

und einen Farbfluß durch

$$\vec{q} = \sum_k \vec{e_k}(r_k - g_k). \tag{4.19}$$

Die zu leistende Arbeit ergibt sich dann aus dem Skalarprodukt aus Feld und Fluß

$$W = -\vec{f} \cdot \vec{q} = -\sum_{ajk} \vec{e_a} \cdot \vec{e_k}(r_k - g_k)[r_j(\vec{x} + \vec{e_a}) - g_j(\vec{x} + \vec{e_a})]. \tag{4.20}$$

Das Skalarprodukt der beiden Einheitsvektoren kann noch berechnet werden

$$\vec{e_a} \cdot \vec{e_k} = \begin{pmatrix} \cos(\frac{\pi a}{3}) \\ \sin(\frac{\pi a}{3}) \end{pmatrix} \begin{pmatrix} \cos(\frac{\pi k}{3}) \\ \sin(\frac{\pi k}{3}) \end{pmatrix} = \cos\left(\frac{\pi}{3}(a-k)\right). \tag{4.21}$$

Es finden sowohl gewöhnliche Stöße statt, wenn die Bedingungen für einen "einfarbigen" Stoß gegeben sind, als auch "Umfärbeprozesse". Dabei bleiben alle vorher besetzten Positionen erhalten, wodurch die Impulserhaltung automatisch gewährleistet wird, nur die Färbung der einzelnen Teilchen verschiebt sich. Bei einem gemischtfarbigen Stoß finden so alle Prozesse statt, die neben der farbblinden Impulserhaltung

$$\sum_a \vec{e_a}(r'_a + g'_a) = \sum_a \vec{e_a}(r_a + g_a) \tag{4.22}$$

die getrennte Teilchenzahlerhaltung

$$\sum_a r'_a = \sum_a r_a \qquad \sum_a g'_a = \sum_a g_a \tag{4.23}$$

gewährleisten und die Oberflächenarbeit W minimieren.

Berücksichtigt man auch ruhende Teilchen so ändert sich bei dieser Rechnung nichts. Nur die Anzahl der möglichen Endformationen bei einem Stoß wird größer. Ein *Cellang*-Programm, das dieses realisiert findet man im Abschnitt 6.3. Allerdings werden für gemischtfarbige Stöße nicht alle Stöße implementiert sondern nur die symmetrischen Stöße, die das Ruheteilchen unberührt lassen. Drei aufeinanderfolgende Zeitschritte dieses Programms kann man in Abb. 4.5 ganz genau nachverfolgen.

Eine andere Methode bei der auch die Löcher (Stellen an denen sich keine Teilchen befinden) Farbinformation tragen [24] konnte sich nicht durchsetzen.

4.6 Lattice Boltzmann Modelle

Lattice Boltzmann Modelle sind der neue Trend, der die Lattice Gase zunehmend verdrängt [28]. Sie sind nicht diskret in der Teilchenzahl sondern operieren auf Grundlage der diskreten Boltzmann Gleichung 4.1 mit Verteilungsfunktionen.

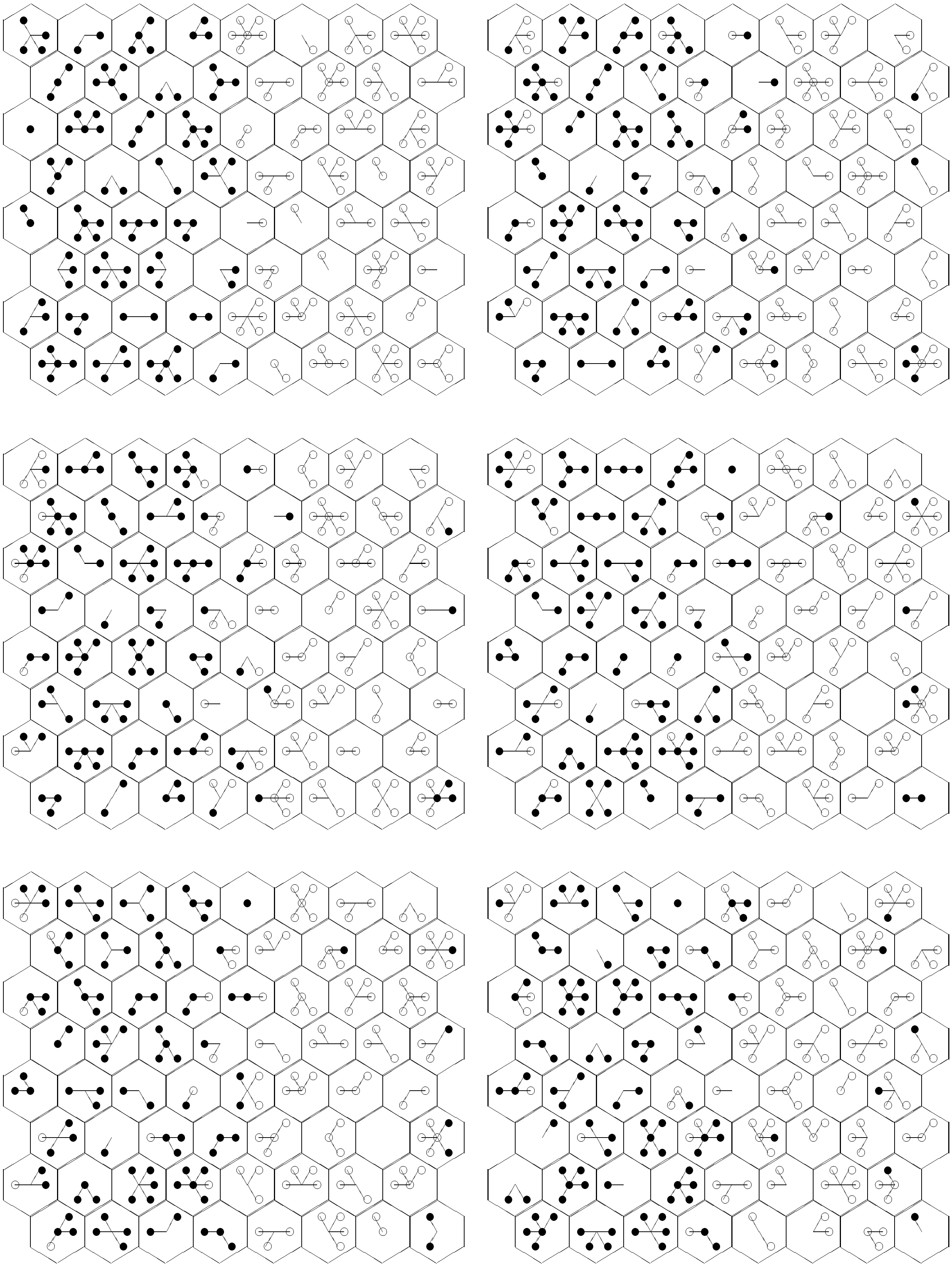

Abbildung 4.5: Drei Zeitschritte eines zweiphasigen FHP-Gases mit Ruheteilchen und minimierter Oberflächenarbeit. Zwischen den linken und den rechten Bildern ist jeweils ein Propagationsschritt, zwischen den Zeilenumbrüchen ein Stoßschritt.

Kapitel 5

Modelle für Tumorwachstum

5.1 Richardson Modell

Das Richardson Modell [38] ist ein Wachstumsmodell mit diskreten Zeitschritten und diskretem Gitter. Es ist eng verwandt mit dem Eden Modell [39].

Die Regeln sind relativ einfach:

- Zur Zeit 0 sind alle Zellen rot gefärbt, bis auf eine in der Mitte, die grün gefärbt ist.
- Ist eine Zelle grün gefärbt, so bleibt sie für alle Zeiten grün.
- Ist eine Zelle rot und alle ihre Nachbarn der von Neumann Nachbarschaft rot, so bleibt sie rot. Ist eine Zelle rot und mindestens einer ihrer Nachbarn grün, so wird sie mit Wahrscheinlichkeit p grün und bleibt mit Wahrscheinlichkeit $(1-p)$ rot.
- Die Wahrscheinlichkeit p ist für alle Zeiten und alle Zellen unabhängig.

Das Ergebnis ist allerdings auch nicht gerade überraschend. Der Cluster wächst "im Wesentlichen kreisförmig" wenn man eine geeignete Norm zugrunde legt. Die relativen Abweichungen zur "Kreisform" in dieser Norm verschwinden mit zunehmender Zeit. In der Praxis heißt das für große p (≈ 1) "diamantförmig" [1] für kleine p kreisförmig. Dies wird in Abb. 5.1 illustriert.

5.2 Williams Bjerkness Modell

Das Williams Bjerkness Modell beschreibt zwei konkurrierende Zellpopulationen [40, 41]. Normale und kanzerogene (meinetwegen auch rote und grüne) Zellen befinden sich auf einem ebenen Gitter. Mit jeder Zellteilung bleibt die eine Tochterzelle auf ihrem Platz, und die andere nimmt einen benachbarten Platz ein. Die Teilungsrate für die kanzerogenen Zellen ist aufgrund eines "kanzerogenen Vorteils" um einen Faktor κ größer als die normaler Zellen.

Mathematisch ausgedrückt liest sich das so: Sei A (endliche Teilmenge des $\mathbb{Z}^2$) die Menge aller ursprünglich von kanzerogenen Zellen besetzten Punkte, und ξ_t^A die

[1] Ein auf der Spitze stehendes Quadrat. Das ist in der Norm $||x|| = \sum |x_i|$ ein Kreis.

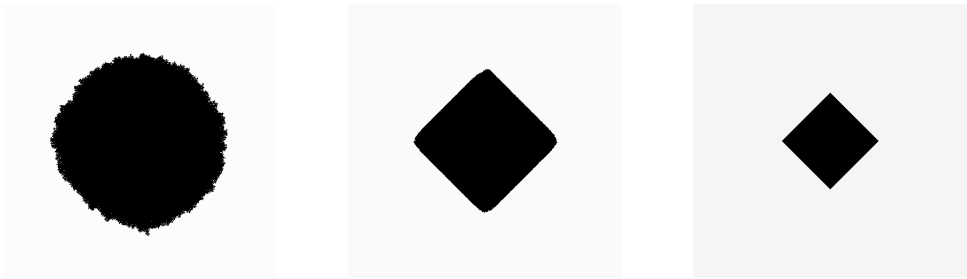

Abbildung 5.1: Richardson Modell mit $p = 0.1$ nach 880 Zeitschritten, mit $p = 0.9$ nach 140 Zeitschritten, und mit $p = 1$ nach 89 Zeitschritten.

Menge aller von kanzerogenen Zellen besetzten Punkte zur Zeit t. Dann sind die Prozesse $(\xi_t^A)_{t \geq 0}$ strenge Markovprozesse mit den Übergangsraten

$$A \to A \cup \{x\}(x \notin A) \quad \text{mit der Rate} \quad \kappa|\{y \in A : \|y - x\| = 1\}| \tag{5.1}$$
$$A \to A - \{x\}(x \notin A) \quad \text{mit der Rate} \quad |\{y \in A^c : \|y - x\| = 1\}|. \tag{5.2}$$

Dabei ist $|\Lambda|$ die Mächtigkeit der Menge Λ und damit die Anzahl der gleichartigen Nachbarn.

Für $\kappa \to \infty$ geht das Williams Bjerkness Modell in das Richardson Modell über.

In [40, 41] zeigen aber die Autoren, daß auch das Williams Bjerkness Modell entgegen ursprünglicher Annahmen, die fraktales Wachstum postulierten, im Wesentlichen kreisförmig wächst.

5.3 Modelle ohne Ortsauflösung

Andere Modelle berücksichtigen nicht die räumliche Verteilung sondern nur die Anzahl N der Zellen [37]. Ein einfaches Modell, das auch eine Beschränkung des Wachstums beinhaltet, ist die logistische Gleichung

$$\frac{dN}{dt} = GN(1 - N/N_{\max}). \tag{5.3}$$

Diese hat eine Lösung der Form

$$N(t) = \frac{N_{\max}}{1 + e^{u-Gt}}. \tag{5.4}$$

Solche Modelle beschreiben zwar die zeitliche Entwicklung der Zellzahl recht brauchbar, machen aber keinerlei Aussage wie sich die Zellen räumlich verteilen. Will man aber aus dem Vergleich eines solchen Modells mit der Wirklichkeit Rückschlüsse auf Mechanismen und Ursachen des Wachstums ziehen, so ist die Zahl der Zellen einfach kein ausreichender Parameter.

5.4 Mein Modell

Wenn man das Wachstum eines Tumors simulieren will, muß man erst einmal wissen, welche Faktoren überhaupt dieses Wachstum beeinflussen können.

Das sind unter anderem [36, 37]:

- Platzbedarf.
- Konkurrenz zu umgebendem Gewebe.
- Nahrungsmittelversorgung.
- Vermehrung durch Zellteilung.
- Vergrößerung der Zellen durch Wachstum.
- Hormonelle Einflüsse.

Das Richardson und das Williams Bjerkness Modell sind praktisch reine Zellteilungsmodelle. Hormonelle Einflüsse wären zwar interessant, aber sind von einem Simulationsprogramm praktisch nicht in den Griff zu bekommen.

Da experimentell beobachtet wird, daß Tumore, je bösartiger sie sind (also je schneller sie wachsen) um so bizarrere Strukturen aufweisen, könnte der Platzbedarf und die rein mechanische Konkurrenz zum umgebenden Gewebe eine wichtige Rolle spielen.

Das aus der Physik bekannte Analogon ist "viscous fingering" in einer Hele-Shaw Zelle [42, 43]. Eine Hele-Shaw Zelle besteht aus zwei planparallelen Platten deren Zwischenraum mit einer hochviskosen Flüssigkeit gefüllt ist. Durch ein kleines Loch in der Mitte einer Platte wird eine andere Flüssigkeit mit niedriger Viskosität eingefüllt. Es bilden sich dann fingerartige Strukturen, die aber nicht stabil weiterwachsen, sondern ab einer bestimmten Größe sich weiter teilen. Das so entstehende Muster ist ein Fraktal.

Die Hele-Shaw Zelle wird oft in einem Atemzug zusammen mit "Strömung durch poröse Medien" genannt, da wohl die Haftung an den Glasplatten einen ähnlichen Effekt hat.

Man braucht also einen Lattice Gas Automaten, der nicht nur zweierlei Teilchen behandeln kann, sondern auch Teilchen der einen Sorte einsetzt. Um die Gesamtdichte und damit den Druck nicht zu sehr ansteigen zu lassen, werden am Rand wieder Teilchen entnommen. Die Einsetzrate wird dabei durch die Größe des festen Einsetzgebietes bestimmt. Wird im Einsetzgebiet ein Platz frei, so wird dieser sofort wieder aufgefüllt. Da das Ergebnis nicht von der genauen Form des Einsetzgebietes abhängen sollte, wird dieses quadratisch gewählt. Auch die Stellen an denen Teilchen entnommen werden, werden am Anfang festgelegt.

Jeder Zeitschritt besteht dann aus den drei Teilen: Propagation, Stöße und Einsetzen. Diese werden als unterschiedliche Zeitschritte implementiert, was die Zählung etwas verwirrend macht, da man stets angeben muss, ob es sich um komplette Zeitschritte oder um die (dreifache) Anzahl der Teilschritte handelt.

Um poröse Medien zu simulieren können zufällig verteilt Wände eingebracht werden. Wände zeichnen sich durch eine besondere Art der Stoßregel aus: Einfallende Teilchen werden reflektiert. Die Dichte und Lage dieser Wandsprenkel wird vor Start des Automaten festgelegt. Allerdings wachsen Tumore auch ohne diesen Effekt, es wäre also erstrebenswert mit der Dichte Null auszukommen.

Da sich fingering im Wechselspiel zwischen stabilisierender Oberflächenspannung und destabilisierenden Kräften abspielt, wurde die Oberflächenspannung variabel gemacht indem die Stoßregeln, oder vielmehr Umfärberegeln, die diese implementieren nur zu einem bestimmten Prozentsatz angewandt werden. Ganz ähnlich wird die Viskosität der Teilchen geregelt: Die Stoßregeln der grünen Teilchen werden zu einem gewissen Prozentsatz außer Kraft gesetzt.

Man hat also folgende Parameter zur freien Verfügung:

- Oberflächenspannung ($0 \ldots 100\%$).
- Stoßwahrscheinlichkeit der grünen Teilchen ($0 \ldots 100\%$).
- Anfängliche Teilchendichte ($0 \ldots 100\%$).
- Wandsprenkeldichte ($0 \ldots 100\%$[2]).
- Größe des Einsetzgebiets meist 24×24 Pixel.
- Größe des Gebiets in dem Teilchen entnommen werden.
- Größe des Automaten insgesamt. Fast alle Simulationen wurden mit Automaten der Größe 1024×1024 gemacht. Bei kleineren Automaten machen sich Randeffekte zu schnell bemerkbar, größere sind von der Rechenzeit und Speicherbedarf nicht machbar.
- Wahl der Randbedingungen. Den Rand des Automaten kann man durch Wände begrenzen, ansonsten gelten automatisch periodische Randbedingungen.

Natürlich ist dies ein sehr abtraktes Modell für einen Tumor, aber es gibt ja auch nicht *den* Tumor sondern eine Vielzahl verschiedener Arten.

Hätte man erst mal das "viscous fingering" im Griff, könnte man ohne größere Probleme auch eine Kombination mit dem Richardson Modell anstreben und auch Einflüsse wie die Nahrungsmittelversorgung durch Einführung eines Nahrungsmittelfeldes im Automaten berücksichtigen. Geändert würde dadurch nur der bisher sehr einfache Einsetzschritt.

Das Modell zeichnet sich also durch hohe Flexibilität aus.

[2] Für die Oberflächenspannung, die Stoßwahrscheinlichkeit der grünen Teilchen und die anfängliche Teilchendichte ist eine Variation nur in ganzen Prozentschritten möglich. Für die Wandsprenkeldichte wählt man eine Zahl zwischen 0 und 1000, kann also in zehntel Prozentschritten variieren.

Kapitel 6

Ergebnisse

6.1 Entmischung zweier Lattice Gase

Abb. 6.1 zeigt die Entmischung eines 256×256 Automaten. Sowohl die grünen (hellgrauen) als auch die roten (dunklen) Teilchen führen stets alle möglichen Stöße des um ein Ruheteilchen erweiterten Modells aus. Die Randbedingungen sind periodisch, die anfängliche Dichte beträgt 50% d. h. 3.5 Teilchen pro Zelle.

Eine Zelle ist hellgrau, falls mehr grüne als rote Teilchen sich auf ihr befinden. Überwiegen die roten, wird sie dunkelgrau. Die schwarzen Sprenkel bezeichnen Zellen auf denen sich gleich viele rote wie grüne Teilchen, oder aber gar keine Teilchen befinden.

Auf einer SGI 4D35/S unter IRIX4.0.5 konnte eine Rechengeschwindigkeit von etwa 7 Sekunden für einen kompletten Zeitschritt erreicht werden. Dies ist allerdings die real verbrauchte Zeit und nicht die CPU Zeit. Jeder Zeitschritt in der Zählung der Bildunterschrift besteht aus den drei Teilschritten Propagation, Stoßschritt und Einsetzen. Da keine Teilchen eingesetzt werden, bleibt der Einsetzschritt stets ohne Folgen. Trotzdem wurde das selbe Programm verwendet das auch Teilchen einsetzen kann.

Abb. 6.2 zeigt einen Automaten mit deutlich einfacheren Regeln. So werden bei den grünen Teilchen nur head-on-head Zweiteilchenstöße und bei den roten alle Stöße mit verschwindendem Gesamtimpuls ausgeführt. Aufgrund der grünen Ruheteilchen, die nur durch gemischtfarbige Stöße in Bewegung gebracht werden, haften die grünen Blasen quasi am Untergrund fest.

6.1.1 Fazit

Mit den einfacheren Regeln konnte selbst nach 90000 Zeitschritten keine Entmischung erreicht werden, während mit den besseren Regeln bereits nach etwa 18000 Zeitschritten vollständige Entmischung auftrat. Damit kann man zufrieden sein.

In [25] wird eine ähnliche Simulation mit einem 128×128 Automaten vorgestellt. Die Regeln sind etwa ein Mittelding aus meinen einfachen Regeln und den vollständigen Regeln mit allen Stößen. Die Entmischung tritt nach 15000 Schritten auf, die Rechengeschwindigkeit ist etwa vergleichbar.

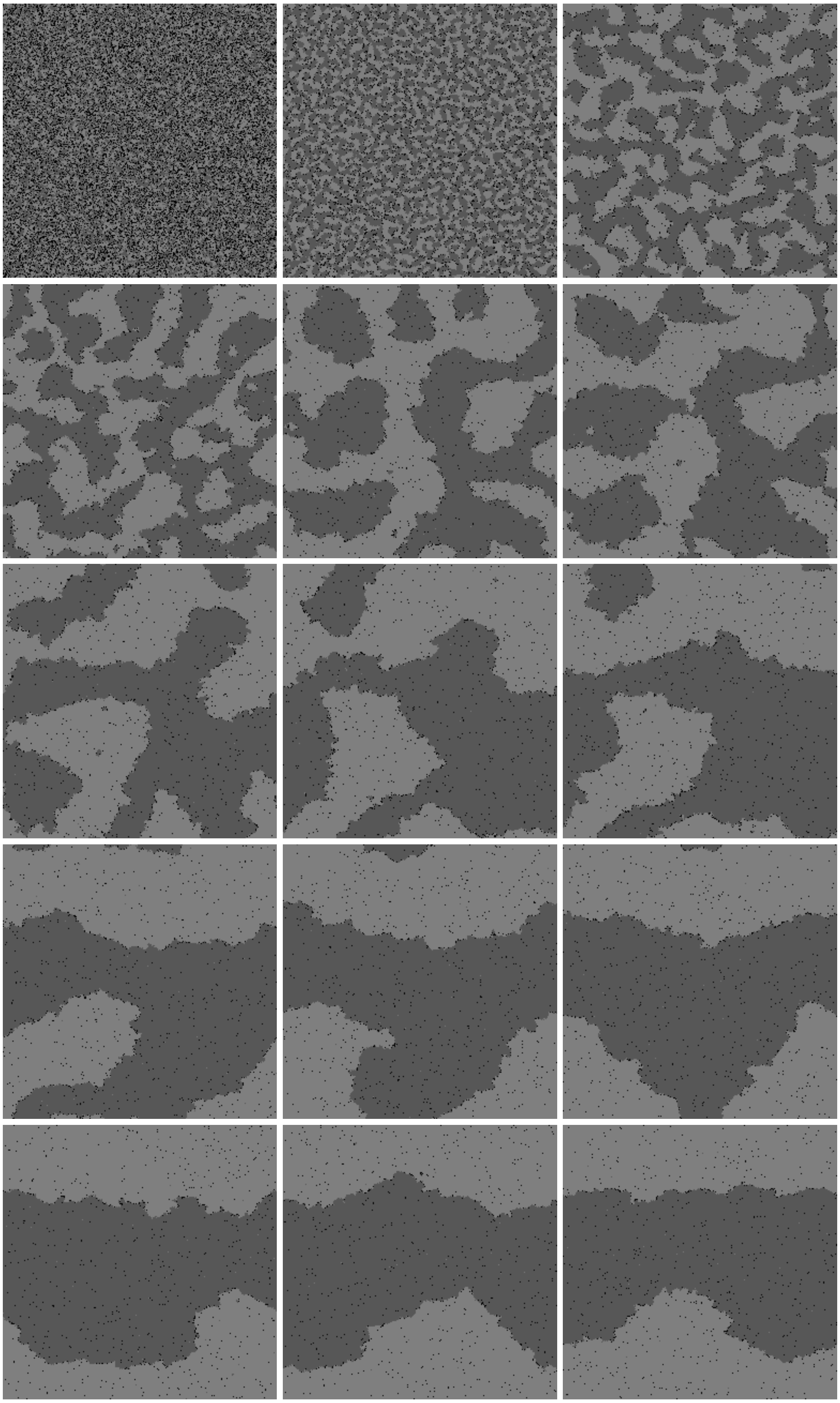

Abbildung 6.1: Entmischung eines 256×256 Automaten nach 1, 67, 1000; 2000, 4000, 5000; 7000, 9000, 10000; 13333, 16666, 18333; 20000, 23333 und 26666 Zeitschritten. Die etwas seltsame Numerierung kommt daher, daß jeder Zeitschritt dreigeteilt ist und daher mit drei multipliziert eine glatte Zahl ergibt.

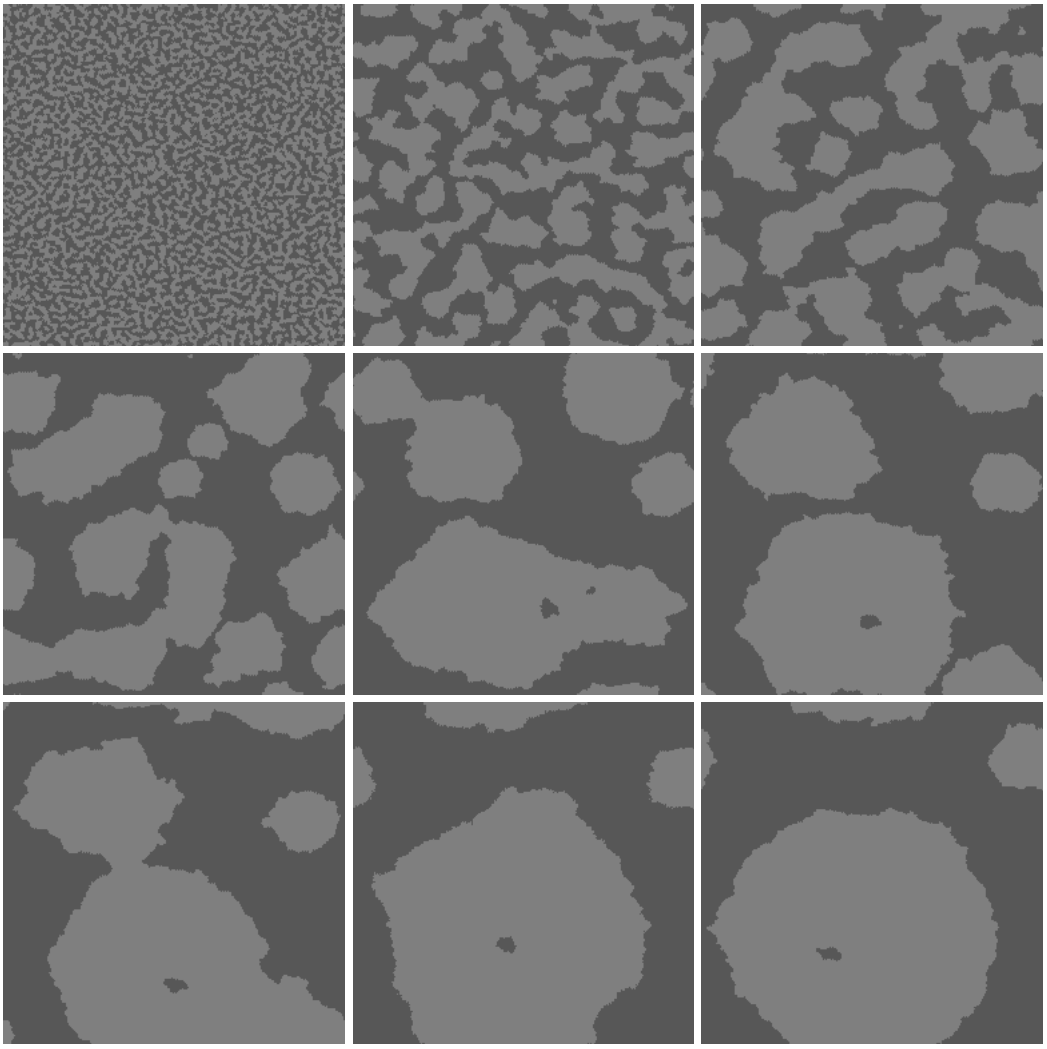

Abbildung 6.2: Entmischung eines 256×256 Automaten mit einfacheren Regeln nach 25, 2500, 5000; 10000, 25000, 45000; 50000, 60000 und 90000 Zeitschritten.

6.2 Wachstumsprozesse

Weniger erfolgreich als die Versuche mit der Entmischung waren die Simulationen der Wachstumsprozesse. Die Abb. 6.3, 6.4,6.5 und 6.6 zeigen Versuche bei denen denen "grüne"[1]Teilchen eingesetzt werden. Die Stoßregeln sind noch verhältnismäßig einfach. Die grünen Teilchen machen nur symmetrische (head-on-head) Zweierstöße, die, falls ein Ruheteilchen als drittes Teilchen beteiligt ist, dieses in Bewegung setzen. Die roten Teilchen machen alle Stöße mit Gesamtimpuls Null.

Durch diese Stoßregeln werden unphysikalische Erhaltungssätze erzeugt, was sich dahingehend auswirkt, daß sich in den Bildern 6.4 und 6.5 hauptsächlich die hexagonale Symmetrie des zugrundeliegenden Gitters wiederspiegelt.

Der Unterschied der beiden Bilder liegt hauptsächlich darin, wie groß das Gebiet ist in welchem Teilchen entnommen werden. Da dieses Gebiet gleichmäßig auf dem Rand verteilt liegt, haben die Teilchen ungehinderteren Zugang als dies in der Nähe des kompakteren Einsetzgebietes der Fall ist. Daher muß das Gebiet in dem Teilchen entnommen werden deutlich kleiner gewählt werden als das Einsetzgebiet. In Abb. 6.5 ist dieses Gebiet halb so groß wie das Einsetzgebiet, aber trotzdem noch deutlich zu groß: Die Teilchenzahl nimmt mit der Zeit stark ab.

Daß Abb. 6.6 nicht die hexagonale Symmetrie wiederspiegelt, liegt nicht an viel

[1]Die Zellen mit mehr grünen Teilchen sind wieder hellgrau markiert. Die Unterschiede in der Helligkeit und Kontrast zwischen den einzelnen Bildern kommt daher, daß die ursprünglich farbig erzeugten Bilder "von Hand" in Graustufenbilder verwandelt wurden um den Kontrast zu erhöhen.

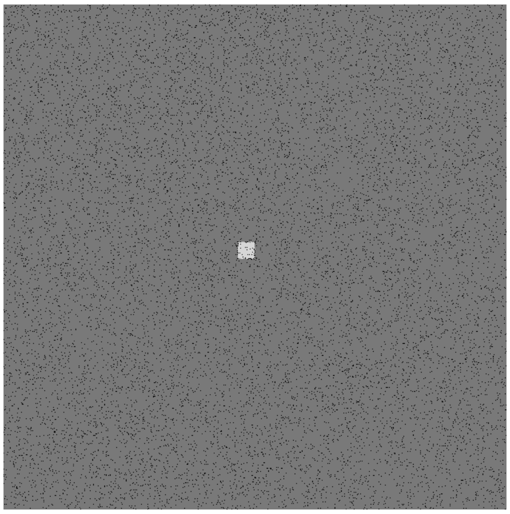

Abbildung 6.3: Typische Anfangsbedingung für eine Wachstumssimulation. Das Quadrat in der Mitte ist 24×24 Pixel groß.

besseren Stoßregeln sondern an einem anderen Fehler in meinem Programm. Dieser bewirkt, daß sich ab und zu (ein paar hundert mal pro Zeitschritt – das ist nicht viel bei über 3 Millionen Teilchen) ein grünes in ein rotes Teilchen umwandelt. Um diesen Fehler zu finden mußten ausführliche Teilchenzählungen durchgeführt werden.

Realistischere Ergebnisse erreicht man durch Implementierung weiterer Stoßregeln. In den Abb. 6.7, 6.8 und 6.9 sind bei beiden Teilchensorten grundsätzlich alle Stöße aus Abb. 4.3 möglich. Bei den grünen Teilchen wird jedoch die Stoßwahrscheinlichkeit für alle Stöße gleichmäßig auf 15% heruntergesetzt. Außerdem werden die roten Teilchen eingesetzt, da diese die geringere Viskosität besitzen.

Der Unterschied zwischen diesen drei Versuchen besteht in der unterschiedlichen Dichte der Wandsprenkel. Nur durch relativ hohe Wandsprenkeldichten konnte ein Effekt erreicht werden der "viscous fingering" zumindest ähnelt. Man muß schon aufpassen, daß man die Perkolationsschwelle von ca. 0.65 [44] für die Wandsprenkel nicht überschreitet.

Abb. 6.10 und 6.11 zeigen noch etwas Teilchenstatistik zum Versuch mit der mittleren Wandsprenkeldichte (Abb. 6.8). Für die anderen Automaten sieht diese Statistik ähnlich aus, nur daß bei höherer Wandsprenkeldichte die Einsetzrate fällt.

Für den Automaten in Abb. 6.9 wurde noch eine Analyse der fraktalen Dimension der Randlinie durchgeführt. Das Ergebnis kann man in Abb. 6.12 bewundern. Die erhaltene Dimension von $D = 1.37 \pm 0.1$ stimmt optimistisch. Die fraktale Dimension von viscous fingering beträgt laut [43] $1.4 \ldots 1.7$.

Versuche mit verschiedenen anderen Kombinationen von Parameterwerten brachten nichts wesentlich anderes.

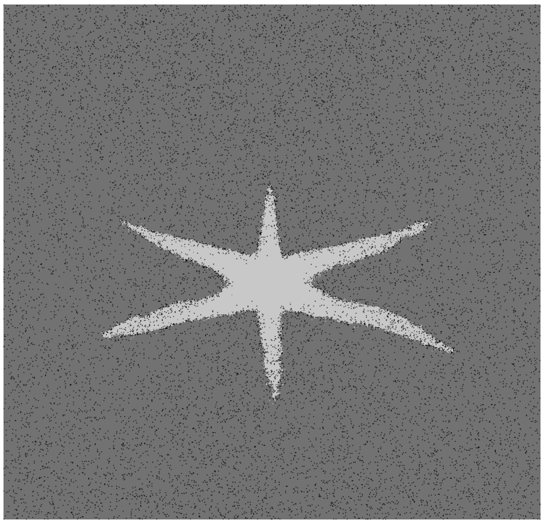

Abbildung 6.4: Hier wurden folgende Parameterwerte verwendet: Oberflächenspannung 85%, Anfängliche Teilchendichte 40%, keine Wandsprenkel, Größe des Einsetzgebiets 32×32. Die Randbedingungen sind periodisch.

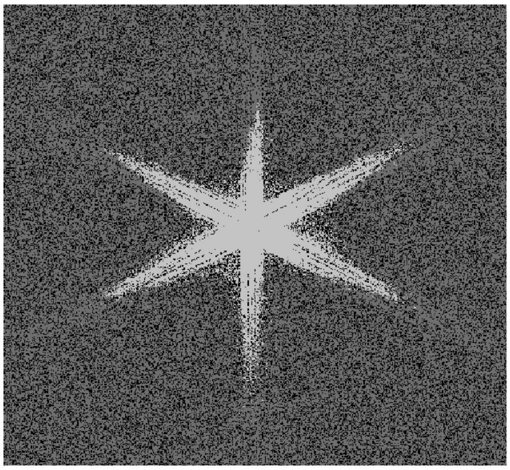

Abbildung 6.5: Hier wurden folgende Parameterwerte verwendet: Oberflächenspannung 85%, Anfängliche Teilchendichte 40%, Wandsprenkeldichte 0.3%, Größe des Einsetzgebiets 24×24. Die Randbedingungen sind durch Wände gegeben.

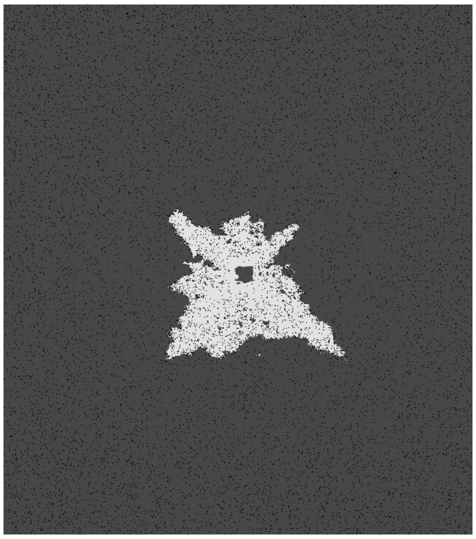

Abbildung 6.6: Hier wurden folgende Parameterwerte verwendet: Oberflächenspannung 85%, Anfängliche Teilchendichte 45%, Wandsprenkeldichte 0.5%, Größe des Einsetzgebiets 32×32. Das Bild zeigt einen Ausschnitt des fehlerhaften Automaten nach 6666 Zeitschritten (bzw 20000 Teilschritten).

6.2.1 Fazit

Die Simulation von "viscous fingering" ohne Wandsprenkel kann als gescheitert betrachtet werden. Bevor man weitere Versuche in dieser Richtung unternimmt sollte man sich vielleicht noch eingehender mit der Theorie dieser Art von Instabilität befassen.

Möglicherweise ist die Abmessung der "Finger" einfach zu groß – größer als die Abmessung meiner Automaten. Auch scheint mir die Verbindung zur Strömung durch poröse Medien etwas unklar. Während z. B. [42] diese Analogie ausdrücklich betont, teilt [43] und auch [45] viscous fingering in eine Universalitätsklasse mit "Diffusion limited aggregation" ein, während Strömung durch poröse Medien zur Universalitätsklasse "Perkolation" gehört.

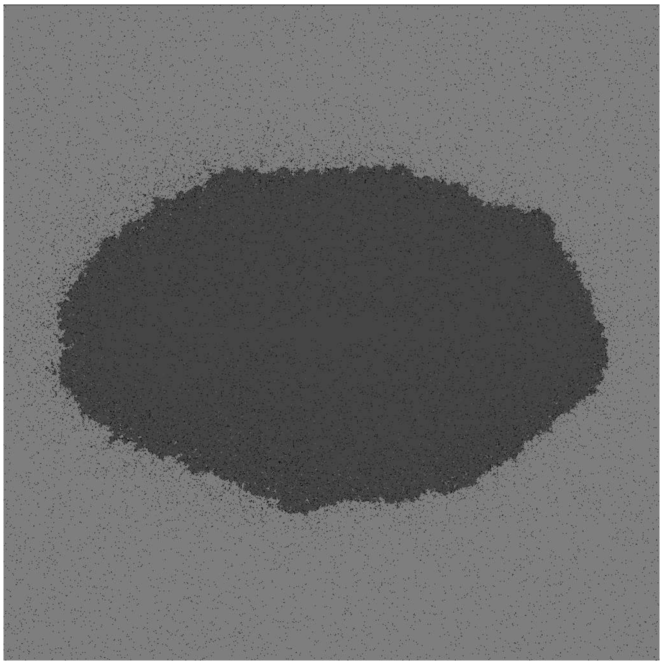

Abbildung 6.7: Hier wurden folgende Parameterwerte verwendet: Oberflächenspannung 70%, Anfängliche Teilchendichte 40%, Wandsprenkeldichte 0%, Größe des Einsetzgebiets 24×24. Auf 40 Zellen am Rand verteilt werden Teilchen entnommen. Die Randbedingungen werden durch Wände realisiert. Das Bild zeigt den Automaten nach 6000 Zeitschritten.

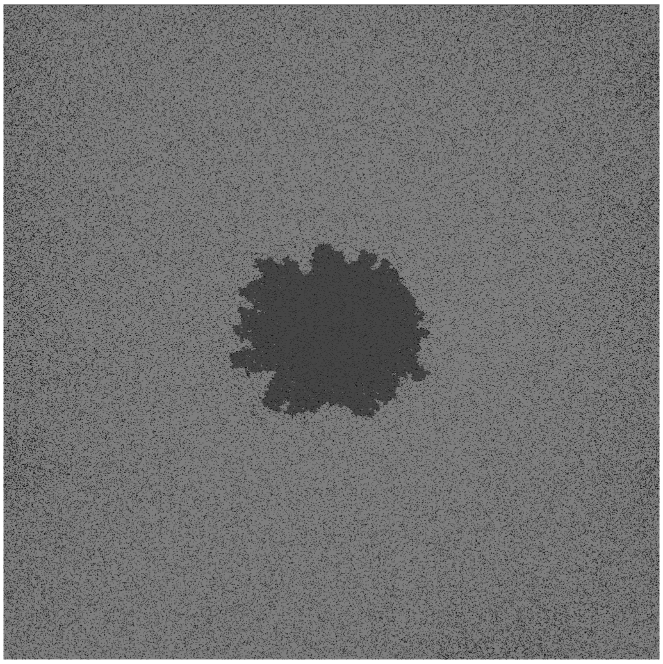

Abbildung 6.8: Hier wurden folgende Parameterwerte verwendet: Oberflächenspannung 70%, Anfängliche Teilchendichte 40%, Wandsprenkeldichte 9%, Größe des Einsetzgebiets 24×24. Auf 40 Zellen am Rand verteilt werden Teilchen entnommen. Die Randbedingungen werden durch Wände realisiert. Das Bild zeigt den Automaten nach 10000 Zeitschritten.

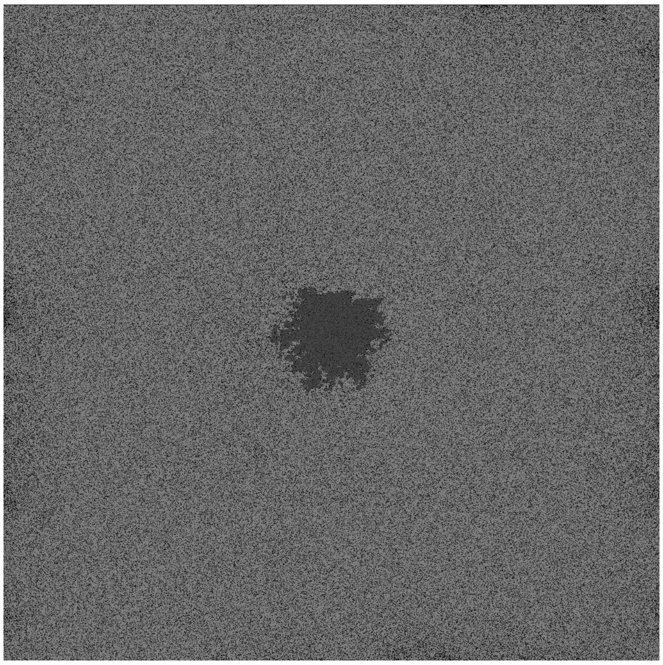

Abbildung 6.9: Hier wurden folgende Parameterwerte verwendet: Oberflächenspannung 70%, Anfängliche Teilchendichte 40%, Wandsprenkeldichte 25%, Größe des Einsetzgebiets 24×24. Auf 40 Zellen am Rand verteilt werden Teilchen entnommen. Die Randbedingungen werden durch Wände realisiert. Das Bild zeigt den Automaten nach 18000 Zeitschritten.

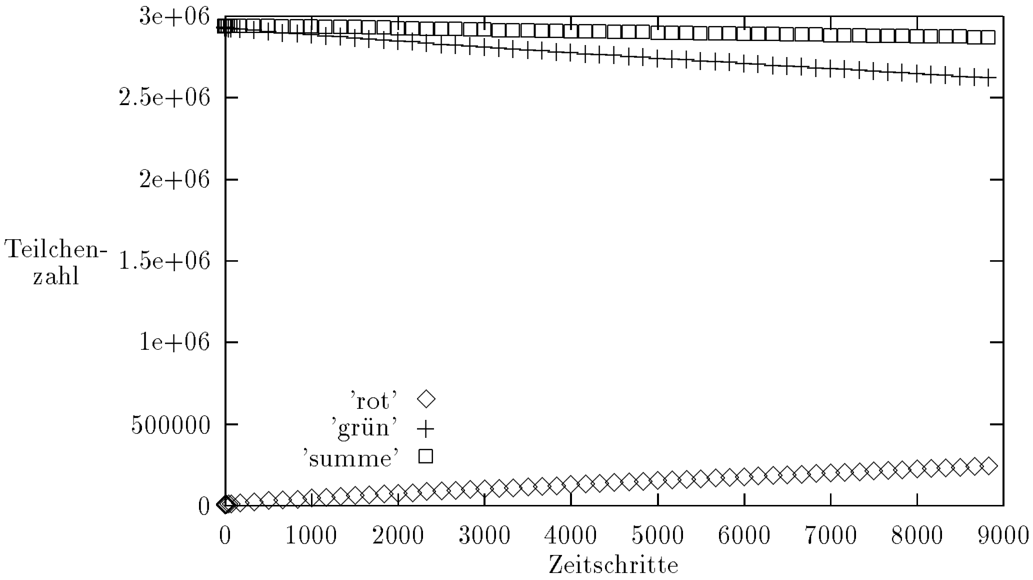

Abbildung 6.10: Teilchenstatistik für beide Teilchensorten und die Gesamtzahl der Teilchen zum Automaten aus Abb. 6.8.

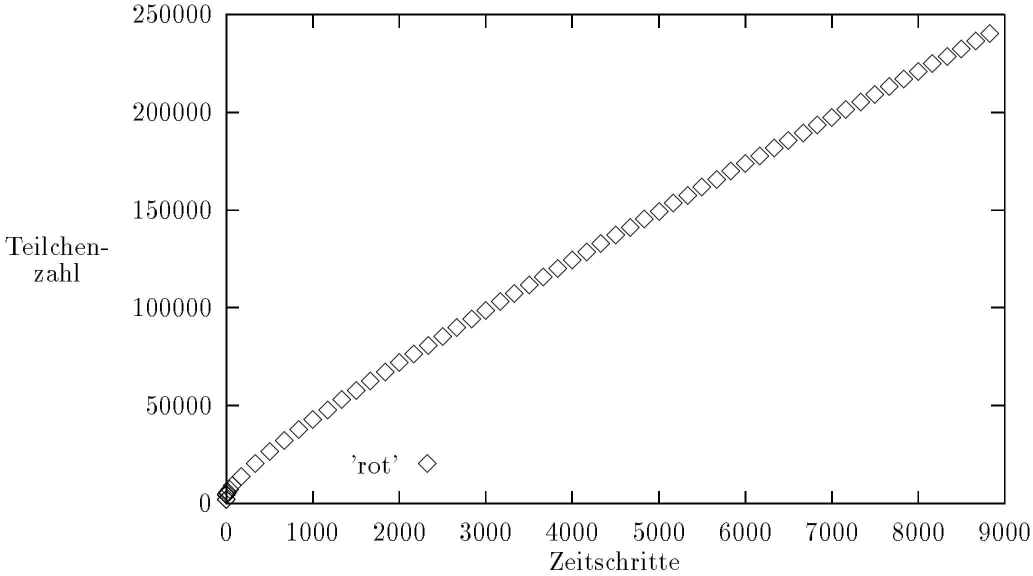

Abbildung 6.11: Teilchenstatistik nur für die roten Teilchen zum Automaten aus Abb. 6.8.

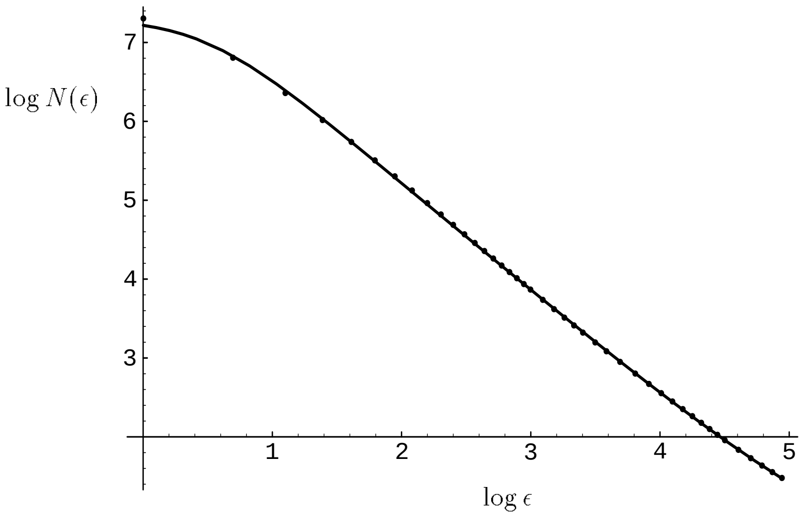

Abbildung 6.12: Der Automat aus Abb. 6.9 wurde mit einem Boxcountingalgorithmus untersucht. Die fraktale Dimension beträgt 1.37 ± 0.1

6.3 the program

Mit dem folgenden *Cellang*-Programm wurden die meisten Simulationen gemacht. Es implementiert ein zweiphasiges Lattice Gas mit Ruheteilchen, allen möglichen Stößen und Einsetzen von Teilchen. Mit den Erläuterungen aus den vorigen Kapiteln und den im Programm enthaltenen Kommentaren sollte es in groben Zügen verstehbar sein.

```
# Copyright by Juergen Dollinger 1995
# Simulation of two immiscible fluids using particles of zero velocity.
# Using all possible collisions for particles of both types.
# Using only collisions which can be done by rotating
# for mixed case, do recoloring instead.
# immi_ruhe_all_miti.cla : 3 different timesteps
# 1.) particles travel
# 2.) collisions and recoloring
# 3.) put in / put out particles

# Global m4 Makros
# Surface Tension 0..100
define(OBSP,67)

# collision probability of green particles 0..100
define(COLP,15)

# input particle of type
define(INPPT,1)

# The hexagonal neighborhood is made of alternating parts
# of the Moore Neighborhood
#                        _ _
#                      _|_|_|
# which =1 or 3, 5:   |_|Z|_|
#                       |_|_|
#                       _ _
#                      |_|_|_
# which =0 or 2, 4:    |_|Z|_|
#                      |_|_|
#
#                               / \
#                            nw/   \ ne
#                             /     \
#                          w |       | e
#                            |       |
#                             \     /
#                           sw \   / se
#                               \ /

2 dimensions of
  # color of display
  color of 0..3

  # make a neighborhood and define places where
  # particles are put in or taken out.
```

```
  # which = 4 or 5 : take out
  # which = 2 or 3 : put in
  #
  which of 0..5

  # If the cell is a wall, it reflects particles!
  #
  wall of 0..1

  # The presence (non-zero) of a particle "traveling" in the given
  # direction. The non-zero value indicates the kind of particle.
  # 1:red,  2:green
  #
  particle[] for e, ne, nw, w, sw, se of 0..2

  # particle with velocity 0
  restparticle of 0..2

  random of 0..99
end

# Hexagonal neighborhood.
#
neighbor[] for e, ne, nw, w, sw, se
     := [1, 0], [1, 1], [0, 1], [-1, 0], [0, -1], [1, -1] when cell.which%2
     := [1, 0], [0, 1], [-1, 1], [-1, 0], [-1, -1], [0, -1] otherwise
if time%3=0 then
  # let particles pass through, compute color, no collisions

  # Find out how many particles of each kind are incident on this cell.
  one := cell.restparticle = 1
  two := cell.restparticle = 2
  forall i
    one := one + 1 when neighbor[i].particle[i+%3] = 1
    two := two + 1 when neighbor[i].particle[i+%3] = 2
    # Particles travel through undisturbed.
    cell.particle[i] := neighbor[i+%3].particle[i]
  end
  cell.color := 3 when cell.wall
             := 1 when one > two
             := 2 when two > one
             := 0 otherwise
elsif time%3=1 then

  # Find out how many particles of each kind are in this cell.
  #
  one := cell.restparticle = 1
  two := cell.restparticle = 2
  forall i
    one := one + 1 when cell.particle[i] = 1
    two := two + 1 when cell.particle[i] = 2
  end

  if cell.wall then
```

```
  # reflect back to incoming direction.
  # particle has same direction as the neighbor where the
  # particle comes from.
  #
  forall i
    cell.particle[i] := cell.particle[i+%3]
  end

elsif one = 0 then
  #
  # Only particles of type two (less collisions)
  #
  if cell.random<COLP then
    if two = 2 then #collision?
      if cell.restparticle = 2 then
        forall i
          if cell.particle[i] then
            cell.particle[i] := 0
            cell.particle[i+%1] :=2
            cell.particle[i-%1] :=2
            cell.restparticle := 0
          end
        end
      else
        stoss := 0 # initialize: no collision
        momencoll := 0 # initialize: no collision producing restparticle
        forall i
          momencoll := momencoll |
                (cell.particle[i+%2] & cell.particle[i])
          # collision producing restparticle
          stoss := stoss |
                (cell.particle[i+%3] & cell.particle[i])
          # head on head collisions
        end
        forall i
          cell.particle[i] := cell.particle[i-%1] # counterclockwise
                        when stoss & cell.random>=COLP/2
                         := cell.particle[i+%1] # clockwise
                        when stoss & cell.random<COLP/2
          if momencoll & cell.particle[i+%2] & cell.particle[i] then
            cell.particle[i] := 0
            cell.particle[i+%2] := 0
            cell.particle[i+%1] := 2
            cell.restparticle := 2
          end
        end
      end
    elsif two =3 & cell.restparticle = 2 then
      stoss := 0 # initialize: no collision
      momencoll := 0 # initialize: no collision with nonzero momentum
      forall i
        stoss := stoss |
              (cell.particle[i+%3] & cell.particle[i])
        momencoll := momencoll |
```

```
          (cell.particle[i+%2] & cell.particle[i])
    # collision producing restparticle
  end
  if stoss then
    if cell.random<COLP/4 then
      cell.particle[] := 0, 2, 0, 2, 0, 2 # __/
      cell.restparticle := 0              #   \
    elsif cell.random<COLP/2 then
      cell.particle[] := 2, 0, 2, 0, 2, 0 #  \__
      cell.restparticle := 0              #  /
    else
      forall i
        cell.particle[i] := cell.particle[i+%1]
                            when cell.random<3*COLP/4 #clockwise
                         := cell.particle[i-%1]
                            otherwise #counterclockwise
      end
    end
  elsif momencoll then
    forall i
      if cell.particle[i+%2] & cell.particle[i] then
        if cell.random<COLP/2 then
          cell.particle[i+%2] := 0  #  /     \
          cell.particle[i+%1] := 2  # *   ->  --- i+%1
          cell.particle[i+%3] := 2  #  \       \
          cell.restparticle := 0    #   i       i
        elsif cell.random>=COLP/2 then
          cell.particle[i] := 0      #  /       /
          cell.particle[i+%1] := 2   # *   ->  ---
          cell.particle[i-%1] := 2   #  \     /
          cell.restparticle := 0
        end
      end
    end
  end
elsif two =3 & cell.restparticle = 0 then
  stoss := 0 # initialize: no collision
  momencoll := 0 # initialize: no collision with nonzero momentum
  forall i
    momencoll := momencoll |                       #   __
          (cell.particle[i+%3] & cell.particle[i]) #    \
    stoss := stoss |
        ( cell.particle[i]     #
        & cell.particle[i+%2]  #   \ /
        & cell.particle[i+%4]) #    |
  end
  if stoss then
    if cell.random<COLP/4 then
      forall i
        cell.particle[i] := cell.particle[i+%1] #rotate 3 particles
      end
    elsif cell.random<COLP/2 then
      cell.particle[] := 2, 0, 0, 2, 0, 0 #  _._
      cell.restparticle :=2               #
```

```
    elsif cell.random<3*COLP/4 then
      cell.particle[] := 0, 2, 0, 0, 2, 0 #  /
      cell.restparticle :=2              # /
    else
      cell.particle[] := 0, 0, 2, 0, 0, 2 # \
      cell.restparticle :=2              #  \
    end # if cell.random<
  elsif momencoll then
    forall i
      if cell.particle[i] & !cell.particle[i+%3] then
        if cell.random<COLP/2 then
          cell.particle[i+%1]:=2
          cell.particle[i-%1]:=2  #  /
          cell.restparticle :=2   # *
          cell.particle[i+%2]:=0  #  \
          cell.particle[i-%2]:=0
          cell.particle[i]:=0
        elsif cell.particle[i+%1] then
          cell.particle[i+%1]:=0
          cell.particle[i+%2]:=2  # \_
          cell.particle[i-%1]:=2  #  \
          cell.particle[i-%2]:=0
        else
          cell.particle[i+%1]:=2  #  i+%1
          cell.particle[i+%2]:=0  #  /_i
          cell.particle[i-%1]:=0  # /
          cell.particle[i-%2]:=2  #i-%2
        end
      end
    end
  end
elsif two = 4 then #4-particle collision?
  if cell.restparticle then
    stoss := 0 # initialize: no collision
    momencoll := 0 # initialize: no collision with nonzero momentum
    forall i
      momencoll := momencoll |                      #   __
          (cell.particle[i+%3] & cell.particle[i]) #    \
      stoss := stoss |
        ( cell.particle[i]     #
        & cell.particle[i+%2]  #   \ /
        & cell.particle[i+%4]) #    |
    end
    if stoss then
      if cell.random<COLP/4 then
        forall i
          cell.particle[i] := cell.particle[i+%1] #rotate 3 particles
        end
      elsif cell.random<COLP/2 then
        cell.particle[] := 0, 2, 2, 0, 2, 2 #  \/
        cell.restparticle := 0              #  /\
      elsif cell.random<3*COLP/4 then
        cell.particle[] := 2, 0, 2, 2, 0, 2 #  __\__
        cell.restparticle := 0              #     \
```

```
    else
      cell.particle[] := 2, 2, 0, 2, 2, 0 #  _/_
      cell.restparticle := 0              #  /
    end
  elsif momencoll then
    forall i
      if cell.particle[i] & !cell.particle[i+%3] then
        if cell.random<COLP/2 then
          cell.particle[i+%1]:=2
          cell.particle[i-%1]:=2  #
          cell.restparticle :=0   # __/__
          cell.particle[i+%2]:=0  #    \
          cell.particle[i-%2]:=0
          cell.particle[i+%3]:=2
        elsif cell.particle[i+%1] then
          cell.particle[i+%1]:=0
          cell.particle[i+%2]:=2  # \_
          cell.particle[i-%1]:=2  #  \
          cell.particle[i-%2]:=0
        else
          cell.particle[i+%1]:=2  #  i+%1
          cell.particle[i+%2]:=0  #  /_i
          cell.particle[i-%1]:=0  # /
          cell.particle[i-%2]:=2  #i-%2
        end
      end
    end
  end
else # no restparticle
  stoss := 0 # initialize: no collision
  forall i
    stoss := stoss +
          ( cell.particle[i+%3] & cell.particle[i] )
  end
  if stoss = 4 then # \/
                    # /\
    if cell.random<COLP/4 then
      forall i
        cell.particle[i] := cell.particle[i+%1] # clockwise
      end
    elsif cell.random<COLP/2 then
      forall i
        cell.particle[i] := cell.particle[i-%1] # counterclockwise
      end
    elsif cell.random<3*COLP/4 then
      cell.particle[] := 0, 2, 0, 2, 0, 2 # __/
      cell.restparticle := 2              #   \
    else
      cell.particle[] := 2, 0, 2, 0, 2, 0 #  \__
      cell.restparticle := 2              #  /
    end
  else
    forall i
      if cell.particle[i] & cell.particle[i+%3]            # _/_
```

```
               & cell.particle[i+%1] & cell.particle[i-%1] then  #  \
              cell.particle[i+%3] :=0
              cell.restparticle := 2
              if cell.random<COLP/2 then  #\
                 cell.particle[i+%1] :=0  # *-- i
                 cell.particle[i+%2] :=2  #  \

              else                        #  /
                 cell.particle[i-%1] :=0  # *-- i
                 cell.particle[i-%2] :=2  #/
              end
            end
          end # forall i
        end # if stoss = 4
      end # if cell.restparticle
    elsif two = 5 & cell.restparticle then
      stoss := 0 # initialize: no collision
      forall i
        stoss := stoss +
              ( cell.particle[i+%3] & cell.particle[i] )
      end
      forall i
        if stoss=4 then
          cell.particle[i]:= cell.particle[i+%1] when cell.random<COLP/2
                          := cell.particle[i-%1] when cell.random>=COLP/2
        elsif cell.particle[i] & cell.particle[i+%3]           # _/_
            & cell.particle[i+%1] & cell.particle[i-%1] then  #  \
          cell.particle[i+%3] :=0 #
          cell.restparticle := 0  # \/_
          cell.particle[i+%2] :=2 # /\
          cell.particle[i-%2] :=2 #
        end
      end
    elsif two = 5 & !cell.restparticle then
      forall i
        if !cell.particle[i+%3] then
          cell.particle[i+%3]:=2 #    \  /            /
          cell.particle[i+%2]:=0 #     \/___  ->  ___*/___
          cell.particle[i-%2]:=0 #i+%3 /\            \
          cell.restparticle:=2   #    /  \            \
        end
      end
    else # Particles travel through undisturbed.
      # do nothing (already traveled)
    end
  end #if cell.random<COLP
elsif two = 0 then
  #
  # Only particles of type one (all possible collisions)
  #
  if one = 2 then #ZweierStoss?
    if cell.restparticle = 1 then
      forall i
        if cell.particle[i] then
```

```
        cell.particle[i] := 0    #
        cell.particle[i+%1] :=1  # *--  -> /
        cell.particle[i-%1] :=1  #         \
        cell.restparticle := 0   #
      end
    end
  else
    stoss := 0 # initialize: no collision
    momencoll := 0 # initialize: no collision producing restparticle
    forall i
      momencoll := momencoll |
              (cell.particle[i+%2] & cell.particle[i])
      stoss := stoss |
              (cell.particle[i+%3] & cell.particle[i])
    end
    forall i
      cell.particle[i] := cell.particle[i+%1]
                        when stoss & cell.random>=50
                       := cell.particle[i-%1]
                        when stoss & cell.random<50
      if momencoll & cell.particle[i+%2] & cell.particle[i] then
        cell.particle[i] := 0
        cell.particle[i+%2] := 0
        cell.particle[i+%1] := 1
        cell.restparticle := 1
      end
    end
  end
elsif one = 3 then
  if cell.restparticle then # 2 particle + restparticle collision?
    stoss := 0 # initialize: no collision
    momencoll := 0 # initialize: no collision with nonzero momentum
    forall i
      momencoll := momencoll |
              (cell.particle[i+%2] & cell.particle[i])
      stoss := stoss |
            ( cell.particle[i+%3] & cell.particle[i])
    end
    if stoss then
      if cell.random<25 then
        cell.particle[] := 0, 1, 0, 1, 0, 1 # __/
        cell.restparticle := 0              #   \
      elsif cell.random<50 then
        cell.particle[] := 1, 0, 1, 0, 1, 0 #  \__
        cell.restparticle := 0              #  /
      else
        forall i
          cell.particle[i] := cell.particle[i+%1] when cell.random>74 #clockwise
                        := cell.particle[i-%1] when cell.random<75 #counterclockwise
        end
      end # if cell.random<
    elsif momencoll then
      forall i
        if cell.particle[i+%2] & cell.particle[i] then
```

```
        if cell.random<50 then
          cell.particle[i+%2] := 0  #  /     \
          cell.particle[i+%1] := 1  # *   ->  --- i+%1
          cell.particle[i+%3] := 1  #  \       \
          cell.restparticle := 0    #   i       i
        elsif cell.random>=50 then
          cell.particle[i] := 0      #  /       /
          cell.particle[i+%1] := 1   # *   ->  ---
          cell.particle[i-%1] := 1   #  \     /
          cell.restparticle := 0
        end
      end
    end
  end # if stoss
else  # no restparticle, 3 particle collision?
  stoss := 0 # initialize: no collision
  momencoll := 0 # initialize: no collision with nonzero momentum
  forall i
    momencoll := momencoll |                        #   __
          (cell.particle[i+%3] & cell.particle[i]) #    \
    stoss := stoss |
        ( cell.particle[i]     #
        & cell.particle[i+%2]  #   \ /
        & cell.particle[i+%4]) #    |

  end
  if stoss then
    if cell.random<25 then
      forall i
        cell.particle[i] := cell.particle[i+%1] #rotate 3 particles
      end
    elsif cell.random<50 then
      cell.particle[] := 1, 0, 0, 1, 0, 0 #  _._
      cell.restparticle := 1              #
    elsif cell.random<75 then
      cell.particle[] := 0, 1, 0, 0, 1, 0 #  /
      cell.restparticle := 1              # /
    else
      cell.particle[] := 0, 0, 1, 0, 0, 1 # \
      cell.restparticle := 1              #  \
    end # if cell.random<
  elsif momencoll then
    forall i
      if cell.particle[i] & !cell.particle[i+%3] then
        if cell.random<50 then
          cell.particle[i+%1]:=1
          cell.particle[i-%1]:=1  #  /
          cell.restparticle :=1   # *
          cell.particle[i+%2]:=0  #  \
          cell.particle[i-%2]:=0
          cell.particle[i]:=0
        elsif cell.particle[i+%1] then
          cell.particle[i+%1]:=0
          cell.particle[i+%2]:=1  # \_
```

```
            cell.particle[i-%1]:=1  #  \
            cell.particle[i-%2]:=0
          else
            cell.particle[i+%1]:=1  #  i+%1
            cell.particle[i+%2]:=0  #  /_i
            cell.particle[i-%1]:=0  # /
            cell.particle[i-%2]:=1  #i-%2
          end
        end
      end
    end # if stoss
  end # if cell.restparticle

elsif one = 4 then #4-particle collision?
  if cell.restparticle then
    stoss := 0 # initialize: no collision
    momencoll := 0 # initialize: no collision with nonzero momentum
    forall i
      momencoll := momencoll |                        #   __
            (cell.particle[i+%3] & cell.particle[i]) #    \
      stoss := stoss |
          ( cell.particle[i]     #
          & cell.particle[i+%2]  #   \ /
          & cell.particle[i+%4]) #    |
    end
    if stoss then
      if cell.random<25 then
        forall i
          cell.particle[i] := cell.particle[i+%1] #rotate 3 particles
        end
      elsif cell.random<50 then
        cell.particle[] := 0, 1, 1, 0, 1, 1 #  \/
        cell.restparticle := 0              #  /\
      elsif cell.random<75 then
        cell.particle[] := 1, 0, 1, 1, 0, 1 #  __\__
        cell.restparticle := 0              #    \
      else
        cell.particle[] := 1, 1, 0, 1, 1, 0 #  _/_
        cell.restparticle := 0              #  /
      end
    elsif momencoll then
      forall i
        if cell.particle[i] & !cell.particle[i+%3] then
          if cell.random<50 then
            cell.particle[i+%1]:=1
            cell.particle[i-%1]:=1  #   i+%1
            cell.restparticle :=0   # __/__i
            cell.particle[i+%2]:=0  #   \
            cell.particle[i-%2]:=0  #   i-%1
            cell.particle[i+%3]:=1
          elsif cell.particle[i+%1] then
            cell.particle[i+%1]:=0
            cell.particle[i+%2]:=1  # \_
            cell.particle[i-%1]:=1  #  \
```

```
          cell.particle[i-%2]:=0
        else
          cell.particle[i+%1]:=1  #  i+%1
          cell.particle[i+%2]:=0  #  /_i
          cell.particle[i-%1]:=0  # /
          cell.particle[i-%2]:=1  #i-%2
        end
      end
    end #forall i
  end
else # no restparticle
  stoss := 0 # initialize: no collision
  forall i
    stoss := stoss +
          ( cell.particle[i+%3] & cell.particle[i] )
  end
  if stoss = 4 then #\/
              #/\
    if cell.random<25 then
      forall i
        cell.particle[i] := cell.particle[i+%1] # clockwise
      end
    elsif cell.random<50 then
      forall i
        cell.particle[i] := cell.particle[i-%1] # counterclockwise
      end
    elsif cell.random<75 then
      cell.particle[] := 0, 1, 0, 1, 0, 1 # __/
      cell.restparticle := 1              #   \
    else
      cell.particle[] := 1, 0, 1, 0, 1, 0 #  \__
      cell.restparticle := 1              #  /
    end
  else # stoss=2
    forall i
      if cell.particle[i] & cell.particle[i+%3]             # _/_ i
          & cell.particle[i+%1] & cell.particle[i-%1] then  #  \
        cell.particle[i+%3] :=0
        cell.restparticle := 1
        if cell.random<50 then     #\
          cell.particle[i+%1] :=0  # *-- i
          cell.particle[i+%2] :=1  #  \

        else                       #  /
           cell.particle[i-%1] :=0  # *-- i
           cell.particle[i-%2] :=1  #/
        end
      end
    end # forall i
  end # if stoss = 4
end # if cell.restparticle
elsif one = 5 & cell.restparticle then
  stoss := 0 # initialize: no collision
  forall i
```

```
      stoss := stoss +
            ( cell.particle[i+%3] & cell.particle[i] )
    end
    forall i
      if stoss=4 then
        cell.particle[i]:= cell.particle[i+%1] when cell.random<50
                        := cell.particle[i-%1] when cell.random>=50
      elsif cell.particle[i] & cell.particle[i+%3]            # _/_
            & cell.particle[i+%1] & cell.particle[i-%1] then  #  \
        cell.particle[i+%3] :=0 #
        cell.restparticle := 0  # \/_
        cell.particle[i+%2] :=1 # /\
        cell.particle[i-%2] :=1 #
      end
    end
  elsif one = 5 & !cell.restparticle then
    forall i
      if !cell.particle[i+%3] then
        cell.particle[i+%3]:=1 #    \  /             /
        cell.particle[i+%2]:=0 #     \/___  ->  ___*/___
        cell.particle[i-%2]:=0 #i+%3 /\             \
        cell.restparticle:=1   #    /  \             \
      end
    end
  else # (one>4 or one<2) Particles travel through undisturbed.
    #do nothing (already traveled)
  end
else
  # two different types of particles
  #
  # do only collisions which can be done by rotating
  # i.e. collisions which do not alter restparticle
  # and with net zero momentum
  # if there is no collision do ``recoloring''

  # Call
  # 2*Cos((i-k) * pi/3)
  # as cosik[i-%k]
  cosik[] for 6 := 2, 1, -1, -2, -1, 1

  # if a collision will happen set stoss :>0
  if one + two = 2 then #ZweierStoss?
    stoss := 0 # initialize: no collision
    forall i
      stoss := stoss |
        ( cell.particle[i+%3] & cell.particle[i])
    end
  elsif one + two = 3 then
    stoss := 0 # initialize: no collision
    if cell.restparticle then # 2 particle collision
      forall i
        stoss := stoss |
              ( cell.particle[i+%3] & cell.particle[i] )
      end
```

```
  else # 3 particle collision?
    forall i
      stoss := stoss |
            ( cell.particle[i]
            & cell.particle[i+%2]
            & cell.particle[i+%4])
      #      \ /          __
      # Only  |  but not   \
    end
  end # if cell.restparticle
elsif one + two = 4 then #4-particle collision?
  if cell.restparticle then # 3 particle collision?
    stoss := 0 # initialize: no collision
    forall i
      stoss := stoss |
            ( cell.particle[i]
            & cell.particle[i+%2]
            & cell.particle[i+%4])
      #      \ /          __
      # Only  |   but not  \
    end
  else # 4-particle collision?
    vstoss := 0 # initialize: no collision
    forall i
      vstoss := vstoss +
            ( cell.particle[i+%3] & cell.particle[i] )
    end
    stoss := 1 when vstoss = 4
          := 0 otherwise
  end # if cell.restparticle
elsif one + two = 5 & cell.restparticle then
  vstoss := 0 # initialize: no collision
  forall i
    vstoss := vstoss +
          ( cell.particle[i+%3] & cell.particle[i] )
  end
  stoss := 1 when vstoss = 4
        := 0 otherwise

else # one + two =6 (one + two <2 cannot happen because else
    # one=0 or two=0)
  stoss := 0 # Only recoloring
end #if one + two =

if stoss then
  #
  # compute work := color flux * colorfield
  #
  minwork := -72
  forall l:0..5 # rotate by l steps
    work := 0
    forall i:0..5 # all neighbours
      neigborcolor := 0
      forall k
```

```
        neigborcolor := neigborcolor+1 when neighbor[i].particle[k] = 1
                     := neigborcolor-1 when neighbor[i].particle[k] = 2
      end
      neigborcolor := neigborcolor+1 when neighbor[i].restparticle = 1
                   := neigborcolor-1 when neighbor[i].restparticle = 2
      forall k:0..5
        localcolor := 1 when cell.particle[l+%k] = 1
                   := -1 when cell.particle[l+%k] = 2
                   := 0 otherwise

        work := work + cosik[i-%k] * neigborcolor * localcolor

      end # forall k
    end # forall i

    # Because of the missing minus minwork is maximum
    if work > minwork | (work = minwork & cell.random<50) then
      minwork := work
      direction := l
    end
  end # forall l
  forall i
    cell.particle[i] := cell.particle[i] when direction = 0
                     := cell.particle[i+%1] when direction = 1
                     := cell.particle[i+%2] when direction = 2
                     := cell.particle[i+%3] when direction = 3
                     := cell.particle[i+%4] when direction = 4
                     := cell.particle[i+%5] when direction = 5
  end
elsif cell.random < OBSP then # ‘‘recoloring’’ but dont increase
                    # surface tension too much
                    # OBSP=100 will be maximum surface tension.
  # recoloring: all particles stay at their positions
  # but change their colour. The number of particles of each color will stay
  # constant.
  test[] for 6 := 0,0,0,0,0,0
  resttest := 0
  if cell.restparticle then
    resttest := 2
  end
  minwork := -72
  forall l:0..5 # Start at Position l with particles of type 1
    work := 0
    if cell.particle[l] then
      tone := one
      forall k:0..5 # 6 possible particles
        if cell.particle[l+%k] then
          if tone > 0 then
            test[l+%k] := 1
            tone := tone - 1
          else
            test[l+%k] := 2
          end
        end
```

```
    end # forall k
    forall i:0..5 # sum over all neighbours
      neigborcolor := 0
      forall k
        neigborcolor:=neigborcolor+1 when neighbor[i].particle[k]=1
                    :=neigborcolor-1 when neighbor[i].particle[k]=2
      end
      neigborcolor:= neigborcolor+1 when neighbor[i].restparticle=1
                  := neigborcolor-1 when neighbor[i].restparticle=2
      forall k
        localcolor := 1 when test[k] = 1
                   := -1 when test[k] = 2
                   := 0 otherwise
        work := work + cosik[i-%k] * neigborcolor * localcolor
      end #forall k
    end # forall i
    # Because of the missing minus minwork is maximum
    if work > minwork | (work = minwork & cell.random <50) then
      minwork := work
      forall j
        cell.particle[j] := test[j]
      end
      cell.restparticle := resttest
    end
  end # if cell.particle[l]
end # forall l

# begin with restparticle
if cell.restparticle then
  forall l:0..5 # Like above but restparticle := 1
           # Begin with particles of type 1 at direction l
    work := 0
    resttest := 1
    if cell.particle[l] then
      tone := one - 1
      forall k:0..5 # 6 possible particles to continue
        if cell.particle[k+%l] then
          if tone > 0 then
            test[k+%l] := 1
            tone := tone - 1
          else
            test[k+%l] := 2
          end
        end
      end # forall k
      forall i:0..5 # sum over all neighbors
        neigborcolor := 0
        forall k
          neigborcolor := neigborcolor+1 when neighbor[i].particle[k] = 1
                       := neigborcolor-1 when neighbor[i].particle[k] = 2
        end
        neigborcolor := neigborcolor+1 when neighbor[i].restparticle = 1
                     := neigborcolor-1 when neighbor[i].restparticle = 2
        forall k
```

```
                localcolor := 1 when test[k] = 1
                           := -1 when test[k] = 2
                           := 0 otherwise
                work := work + cosik[i-%k] * neigborcolor * localcolor
              end #forall k
            end # forall i
            # Because of the missing minus minwork is maximum
            if work > minwork | (work = minwork & cell.random <50) then
              minwork := work
              forall j
                cell.particle[j] := test[j]
              end
              cell.restparticle := resttest
            end
          end # if cell.particle[l]
        end # forall l
      end # if cell.restparticle
    else # do not recolor to decrease Surface Tension
    end # if stoss
  end # if cell.wall ... elsif one=0 ... elsif two=0 else ...
  # mix random values
  cell.random := (cell.random + [1,0].random
              + [0,1].random + [-1,0].random + [0,-1].random )%100
else # if time%3=
  if cell.which>1 & cell.which<4 then
    forall i
      cell.particle[i] := INPPT # input a particle
    end
    cell.restparticle := INPPT # input a particle
  end # if cell.which>1 & cell.which<4
  if cell.which>3 then
    forall i
      cell.particle[i]:=0 # if there was a particle, take it out
    end
    cell.restparticle := 0 # if there was a particle, take it out
  end
end # if time%3=
```

Literaturverzeichnis

[1] Gardner, Martin, “The Fantastic Combinations of John Conway’s New Solitaire Game of ‘Life’,”, *Scientific American*, **223**:4 (April 1970), 120–123.

[2] Howard Gutowitz (maintainer), “Frequently Asked Questions About Cellular Automata” anonymous ftp: **alife.santafe.edu**:‘`/pub/topics/cas`’
www: http://alife.santafe.edu/alife/topics/cas/ca-faq/ca-faq.html.

[3] Andreas Deutsch, (Abt. Theoretische Biologie Uni Bonn) “Towards Analyzing Complex Swarming Patterns in Biological Systems with the Help of Lattice-Gas Automata”, *Proceedings of the 2nd ECMBM* Lyon (Dez. 93).

[4] Stephen Wolfram, “Origins of Randomness in Physical Systems”, *Physical Review Letters* **55** No. 5 (1985) 449–452.

[5] Stephen Wolfram, “Random Sequence Generation by Cellular Automata”, *Advances in Applied Mathematics* **7** (1986) 123–169.

[6] Howard Gutowitz. Cryptography with dynamical systems. In N. Boccara, E. Goles, S. Martinez, and P. Picco, editors, *Cellular Automata and Cooperative Phenomena*, pages 237–274. Kluwer Academic Publishers, 1993.

[7] Stephen Wolfram, “Statistical mechanics of cellular automata”, *Reviews of Modern Physics* **55** No. 3 (1983) 601–644.

[8] Stephen Wolfram, “Universality and complexity in cellular automata”, *Physica D*, **10** (1984) 1–35.

[9] Stephan Wolfram. *Theory and applications of cellular automata.* World Scientific, Singapore, 1986. ISBN 9971-50-124-4 pbk.

[10] Stephan Wolfram. *Cellular Automata and Complexity: Collected Papers.* Addison-Wesley, 1994. revision of [9] Paperback: ISBN 0-201-62664-0, Hardcover: ISBN 0-201-62716-7.

[11] Norman Margolus, “Physics–like Models of Computation” *Physica D*, **10** (1984) 81–95.

[12] Nigel Allison, Mike Sales, “CART A cellular automata research tool”, *Microprocessors and Microsystems* **16** No. 8 (1992) 403–415.

[13] Josef J. Strout, Newsgroup: `comp.theory.cell-automata`
Message-ID:<Pine.SUN.3.90.950303074845.26908A-100000@helmholtz>
Date: Fri, 3 Mar 1995 07:59:57 -0800 (PST)

[14] J. Hardy, Y. Pomeau, *Journal of Mathematical Physics*, **13** (1972), 1024.

[15] J. Hardy, Y. Pomeau, O. de Pazzis, “Time evolution of a two-dimensional model system. I. Invariant states and time correlation functions”, *Journal of Mathematical Physics*, **14** (1973), 1746.

[16] J. Hardy, O. de Pazzis, Y. Pomeau, “Molecular dynamics of a classical lattice gas: transport properties and time correlation functions”, *Physical Review A*, **13** (1976), 1949.

[17] U. Frisch, B. Hasslacher, Y. Pomeau, “Lattice-Gas Automata for the Navier-Stokes Equation”, *Physical Review Letters*, **56** No. 14 (1986) 1505–1508.

[18] Stephen Wolfram, “Cellular Automaton Fluids 1: Basic Theory”, *Journal of Statistical Physics* **45** Nos. 3/4 (1986) 471–526.

[19] U. Frisch, D. d’Humières, B. Hasslacher, Y. Pomeau, J. Rivet, “Lattice Gas Hydrodynamics in Two and Three Dimensions”, *Complex Systems* **1** (1987) 649–707.

[20] D. d’Humières, P. Lallemand, “Numerical Simulations of Hydrodynamics with Lattice Gas Automata in Two Dimensions”, *Complex Systems* **1** (1987) 599–632.

[21] I. Karlin, private communications.

[22] Bruce Boghosian, private communications.

[23] G. Baumann, M. Grmela, T. F. Nonnenmacher, “Discrete Boltzmann-like equations: a dynamical formulation”, *Physica A*, **187** (1992) 503–518.

[24] J. A. Somers and P. C. Rem, “Analysis of Surface Tension in Two-Phase Lattice Gases”, *Physica D* , **47** (1991) 39–46.

[25] Daniel H. Rothman and Jeffrey M. Keller, “Immiscible Cellular-Automaton Fluids”, *Journal of Statistical Physics* **52** Nos. 3/4 (1988) 1119–1127.

[26] Andrew K. Gunstensen and Daniel H. Rothman, “A Lattice-Gas Model for Three Immiscible Fluids”, *Physica D* , **47** (1991) 47–52.

[27] Chen, Shiyi, Hudong Chen and Gary D. Doolen, “How the Lattice Gas Model for the Navier–Stokes Equation Improves When a New Speed is Added”, *Complex Systems*, **3** (1989), 243–251.

[28] F. J. Higuera and J. Jiménez, “Boltzmann Approach to Lattice Gas Simulationen”, *Europhysics Letters* **9** (7) (1989), 663–668.

[29] D. Hiebeler, A brief review of CA packages. *Physica D* , **45** (1990), 463–467.

[30] Eckart, J. Dana, “A Parallel Extendible Scalable Cellular Automata Machine: PE–SCAM”, *Proceedings of the ACM* 20^{th} *Annual Computer Science Conference*, March 3–5, 1992, 467–472.

[31] Eckart, J. Dana, “A *Cellular* Automata Simulation System”, *SIGPLAN Notices*, **26**:8 (August 1991), 80–85.

[32] Eckart, J. Dana, “Cellang: Language Reference Manual” Wird mit dem Paket *Cellular* ausgeliefert.

[33] Eckart, J. Dana, “A *Cellular* Automata Simulation System”, Aktuelle Version wird mit dem Paket *Cellular* ausgeliefert.

[34] Langton, Chris and Dave Hiebeler, *Cellsim*, available by anonymous ftp .

[35] Toffoli, Tommaso and Norman Margolus, *Cellular Automata Machines: A New Environment for Modeling*, The MIT Press, Cambridge, Massachusetts, 1987.

[36] Rudolf-Erich Klemke, "Zur Anomalie der FNM/FAD-Reaktionen als Ursache der Karzinogenese und des Zelltodes" *Schriftenreihe Krebsgeschehen* **10** vfm-Verlag Heidelberg.

[37] P. A. Riley, "Control of Proliferation of Normal and Neoplastic Cells in Culture" in Karger, "Regulation of Growth in Neoplasia" Basel (1981) 131–198.

[38] Richard Durrett and Thomas M. Liggett, "The shape of the limit set in Richardson's Growth Model", *The annals of Probability*, **9** No. 2 (1981) 168–193.

[39] T. Vicsek, "Fractal Growth Phenomena", World Scientific Publishing Co. Singapore 1989.

[40] Maury Bramson, David Griffeath, "On the Williams-Bjerknes Tumor Growth Model I" *The annals of Probability*, **9** No. 2 (1981) 173–185.

[41] Maury Bramson, David Griffeath, "On the Williams-Bjerknes Tumor Growth Model: II" *Math. Proc. Camb. Phil. Soc.* **88** (1980) 339–357.

[42] A. J. Degregoria, L. W. Schwartz, "A boundary-integral method for two-phase displacemant in Hele-Shaw cells", *J. Fluid Mech*, **164** (1986) 383–400.

[43] Hideki Takayasu, "Fractals in the physical sciences" Manchester University Press 1990.

[44] Dietrich Stauffer, "Introduction to Percolation Theory", Taylor & Francis 1985 ISBN 0-85066-315-6.

[45] Armin Bunde, Shlomo Havlin, "Fractals and Disordered Systems", Springer 1991.

Dank

Ich danke Herrn Prof. Dr. T. F. Nonnenmacher für die Vergabe des interessanten Themas und seine Unterstützung.

Herrn Prof. Dr. P. Reineker danke ich für sein Interesse an der Arbeit und die Erstellung des zweiten Gutachtens.

PD. Dr. G. Baumann, als Betreuer, hatte immer einen Ordner voll Wissen parat. Dr. A. Barth war als Computerspezialist gefragt und interessierte sich immer für die neusten Ergebnisse. Cornelius Krämer hat sich an einem langen Abend meine Ausführungen angehört und einen wichtigen Fehler in meinen *Cellang*-Programmen gefunden. Auch Ralf Metzler und am Anfang auch Gernot Haager haben nicht nur (aber auch) Tee gekocht und Rämsche gefressen.

Gegen Ende erklärten mir noch Dr. I. Karlin und Bruce Boghosian, ein Lattice Gas Spezialist aus den USA, wie es sich mit der Viskosität verhält.

Dana J. Eckart stellte mir seine Programmiersprache *Cellang* zur Verfügung, und war auch gleich bereit einige Erweiterungen einzufügen.

Den anderen die noch nicht aufgeführt wurden, danke ich ganz besonders.

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit unter Verwendung der angegebenen Quellen und Hilfsmittel selbständig angefertigt habe.

Ulm, 29. März 1995 Jürgen Dollinger