Darstellung von Differentialgleichungen mit einem zellulären Automaten

Detlev Ziereisen, Florian Lüthi, i10b

2. Juni 2012

Inhaltsverzeichnis

1 Einleitung

Diese Projektarbeit (bestehend aus diesem Dokument sowie der zugehörigen Software `Tusk`) hat zum Ziel, partielle Differentialgleichungen aus dem Umfeld der physikalischen Simulationen mit einem zellulären Automaten zu berechnen sowie darzustellen. Es soll gezeigt werden, dass dies unter den konzeptionellen Restriktionen der Theorie der zellulären Automaten möglich ist, insbesondere soll die Annahme Konrad Zuses, dass die Naturgesetze diskreten Regeln folgten und das gesamte Geschehen im Universum das Ergebnis der Arbeit eines gigantischen zellulären Automaten sei [11, 13], zumindest für den Scope dieser Arbeit bestätigt werden. Ausserdem soll gezeigt werden, dass die intuitiv erwarteten Resultate in sinnvoller Rechenzeit dargestellt werden können.

Diese Projektarbeit wurde im Rahmen des Kurses Softwareprojekt 2 an der Hochschule für Technik Zürich durchgeführt. Der Quellcode sowie auch dieses Dokument sind Online auf `http://www.github.com/foyan/Tusk` zu finden.

Der Autoren Dank geht (in alphabetischer Reihenfolge der Vornamen) an Albert Heuberger für geballte mathematische Kompetenz gepaart mit unendlicher Geduld, an Lukas Eppler für die ständige Begleitung, sowie an Philippe Nahlik für die tolle Kursführung.

2 Zelluläre Automaten

> Ein zellulärer Automat ist eine regelmäßige Annordnung von Zellen. Jede Zelle kann eine endliche Zahl von Werten / Zuständen annehmen und hat eine begrenzte Zahl von Nachbarzellen, die sie beeinflussen können. Das Muster des gesamten zellulären Automaten ändert sich in einzelnen Schritten, die durch eine Reihe von Übergangsregeln bestimmt werden, die für alle Zellen gelten.[2]

Also:

Definition 1 (Zellulärer Automat)**.** Ein zellulärer Automat ist durch folgende Eigenschaften festgelegt:

- einen Zellularraum R,
- eine endliche Nachbarschaft N, wobei $\forall r \in R\,(N_r \subset R)$,
- eine Zustandsmenge Q,
- eine Überführungsfunktion $\delta : Q^{|N|+1} \mapsto Q$.

Die Zustandsübergänge erfolgen für alle Zellen nach derselben Überführungsfunktion und gleichzeitig. Die Zellzustände können wie die Zeitschritte diskret sein. [11]

Bemerkung. Aus dieser Definition folgt unmittelbar, dass der neue Zustand einer Zelle nur vom momentanen Zustand dieser Zelle sowie den Zuständen der Nachbarzellen abhängig sein kann.

Für zweidimensional organisierte R sind zwei Arten von Nachbarschaft üblich:

Von-Neumann-Nachbarschaft Die Nachbarschaft besteht jeweils aus den vier geographisch nächsten Nachbarzellen: $N_{i,j} = \{R_{i,j-1}, R_{i,j+1}, R_{i-1,j}, R_{i+1,j}\}$

Moore-Nachbarschaft Die Nachbarschaft besteht jeweils aus allen Zellen der Von-Neumann-Nachbarschaft sowie zusätzlich der diagonalen Nachbarzellen:

$$N_{i,j} = \{R_{i,j-1}, R_{i,j+1}, R_{i-1,j}, R_{i+1,j}, R_{i-1,j-1}, R_{i-1,j+1}, R_{i-1,j+1}, R_{i+1,j+1}\}$$ [7]

Beispiel (Wolfram's eindimensionales Universum)**.** Stephen Wolfram[1] definiert in [12] und in etlichen Arbeiten aus der Mitte der 1980er-Jahre einen parametrierbaren zellulären Automaten, der nur aus einer einzigen Raumdimension besteht[10]. In seiner einfachsten Ausprägung ist die Nachbarschaft N_i definiert als $\{R_{i-1}, R_{i+1}\}$, und jede Zelle kann genau zwei Zustände annehmen (tot und lebendig bzw. 0 und 1). Damit ist

$$\delta : \{0,1\}^{|\{R_{i-1},R_{i+1}\}|+1} \mapsto \{0,1\} = \{0,1\}^3 \mapsto \{0,1\}\,,$$

ergo existieren 8 mögliche Zustandsänderungen. Ein Beispiel:

alte $R_{i-1}R_iR_{i+1}$	111	110	101	100	011	010	001	000
neues R_i	0	1	1	0	1	1	1	0

Tabelle 1: Wolfram-Konfiguration 110

Man stellt nun fest, dass bei 8 möglichen Zustandsänderungen für die Anzahl der möglichen Konfigurationen gilt:

$$|K| = |Q|^{|\operatorname{dom}(\delta)|} = 2^8 = 256.$$

Jeder dieser Konfigurationen kann eine natürliche Zahl $\{0, 1, 2, \ldots, 255\}$ zugeordnet werden, indem die neuen R_i wie oben tabelliert und pro Zeile als binäre Zahl aufgefasst werden [3]:

Dadurch ist es möglich, sämtliche eindimensionalen Wolfram-Universen durch eine einzige Zahl zu identifizieren.

Als besonders spannend hat sich die in Tabelle 1 dargestellte Konfiguration 110 erwiesen, weil sie die Eigenschaft hat, ein Turing-vollständiges System zu sein [3, 12]. Dadurch ist sie die Konfiguration einer universellen Turingmaschine, die mit nur 2 Zuständen und 5 Symbolen umgesetzt werden kann – somit hat die Wolfram-Konfiguration 110 als Turingmaschine einen Umfang von $2 \cdot 5 = 10$ und zählt damit zu den kleinsten bis dato bekannten Turingmaschinen [3].

[1] Stephen Wolfram (* 29. August 1959), britischer Physiker und Mathematiker, Schöpfer der Software Mathematica sowie der Suchmaschine Wolfram Alpha [10]

111	110	101	100	011	010	001	000	Zahl
0	0	0	0	0	0	0	0	$00000000_b = 0_d$
0	0	0	0	0	0	0	1	$00000001_b = 1_d$
0	0	0	0	0	0	1	0	$00000010_b = 2_d$
					$\vdots$			
0	1	1	0	1	1	1	0	$01101110_b = 110_d$
					$\vdots$			
1	1	1	1	1	1	1	1	$11111111_b = 255_d$

Tabelle 2: Alle Wolfram-Konfigurationen

Beispiel (Game of Life)**.** Das von Conway[2] 1970 entworfene Game of Life ist eine bis heute populäre Umsetzung der Automatentheorie und insbesondere der Idee der zellulären Automaten [8].

Der ursprüngliche Entwurf befindet sich in einem zweidimensionalen R unter Verwendung der Moore-Nachbarschaft. Die Zellen können zwei mögliche Zustände $\{q_{\text{lebend}}, q_{\text{tot}}\}$ annehmen, und die Übergangsfunktion ist definiert als:

$$\delta(r, N) = \begin{cases} q_{\text{lebend}} & (r = q_{\text{tot}} \wedge \varsigma(q_{\text{lebend}}, N) = 3) \\ q_{\text{tot}} & (r = q_{\text{lebend}} \wedge \varsigma(q_{\text{lebend}}, N) < 2) \\ q_{\text{lebend}} & (r = q_{\text{lebend}} \wedge 2 \leq \varsigma(q_{\text{lebend}}, N) \leq 3) \\ q_{\text{tot}} & (r = q_{\text{lebend}} \wedge \varsigma(q_{\text{lebend}}, N) > 3) \\ q_{\text{tot}} & (\text{sonst}) \end{cases}$$

unter Zuhilfenahme der Statuszählfunktion

$$\varsigma(q, N) = \sum_{i=1}^{8} \begin{cases} 1 & (N_i = q) \\ 0 & (N_i \neq q) \end{cases}$$

3 Vom zellulären Automaten zur Differentialgleichung

Die Berechnung und Darstellung physikalischer Begebenheiten (allgemein ausgedrückt durch folgende partielle Differentialgleichung, mit $u : u(\vec{x}, t)$)

$$k_n \frac{\partial^n u}{\partial t^n} + k_{n-1} \frac{\partial^{n-1} u}{\partial t^{n-1}} + \cdots + k_1 \frac{\partial u}{\partial t} + k_0 = \frac{\partial^n u}{\partial \vec{x}^n} + \frac{\partial^{n-1}}{\partial \vec{x}^{n-1}} + \cdots + \frac{\partial u}{\partial \vec{x}}$$

mit zellulären Automaten kann durchgeführt werden, indem folgendes getan wird:

- R entspricht einer sinnvollen (groben) Diskretisierung der örtlichen Variablen $\vec{x}$ in einer, zwei oder drei Dimensionen

[2] John Horton Conway (* 26. Dezember 1937), englischer Mathematiker[9]

- Jede Zelle in R ist ein Tupel (Q, D) mit Q als einer Menge von berechnungsfernen Zustandsinformationen und den Differentialen nach der Zeit $D = \left(u, \frac{\partial u}{t}, \frac{\partial^2 u}{\partial t^2}, \cdots, \frac{\partial^n u}{t^n}\right) \in \mathbb{R}^n$
- Eine neue Generation entspricht jeweils der fortgelaufenen Zeit ∂t, welche sehr fein diskretisiert werden muss
- In der Übergangsfunktion δ steckt die eigentliche Differentialgleichung. In der Regel verändert sie nur die Elemente von D.

4 Differenzierung nach dem Ort

Wir starten mit den gegebenen Werten $u_{\vec{x}}$ (den Zuständen der Zellen aus R). Daraus erhalten wir die diskrete Funktion

$$u : R \mapsto \mathbb{R}, u(\vec{x}) = u_{\vec{x}}$$

welche durch komponentenweises Einsetzen des Differenzenquotienten $\frac{\Delta u}{\Delta \vec{x}}$ folgendermassen differenzierbar ist (unter Zuhilfenahme des Einheitsvektors $\vec{e}_i$ für jede Komponente von $\vec{x}$ sowie der Erkenntnis, dass durch die diskrete Ausgangsfunktion $\Delta\vec{x}$ komponentenweise 1 ist):

$$\left(\frac{\Delta}{\Delta \vec{x}} u\right)_{\vec{x}} = u_{\vec{x}} - \sum_{i=1}^{\dim(\vec{x})} u_{\vec{x}-\vec{e}_i} \tag{1}$$

Diese Differenzen können natürlich wiederum differenziert werden:

$$\left(\frac{\Delta}{\Delta \vec{x}} \left(\frac{\Delta}{\Delta \vec{x}} u\right)_{\vec{x}}\right)_{\vec{x}} = \left(\frac{\Delta}{\Delta \vec{x}} u\right)_{\vec{x}} - \sum_{i=1}^{\dim(\vec{x})} \left(\frac{\Delta}{\Delta \vec{x}} u\right)_{\vec{x}-\vec{e}_i}$$

Durch Einsetzen von 1 in die obige Gleichung ergibt sich dann für die Differenz zweiter Ordnung:

$$\begin{aligned}
\left(\frac{\Delta^2}{\Delta \vec{x}^2} u\right)_{\vec{x}} &= u_{\vec{x}} - \sum_{i=1}^{\dim(\vec{x})} u_{\vec{x}-\vec{e}_i} - \sum_{i=1}^{\dim(\vec{x})} \left(u_{\vec{x^*}} - \sum_{i=1}^{\dim(\vec{x})} u_{\vec{x^*}-\vec{e}_i}\right)_{\vec{x^*}:=\vec{x}-\vec{e}_i} \\
&= u_{\vec{x}} - \sum_{i=1}^{\dim(\vec{x})} u_{\vec{x}-\vec{e}_i} - \sum_{i=1}^{\dim(\vec{x})} u_{\vec{x}-\vec{e}_i} + \sum_{i=1}^{\dim(\vec{x})} u_{\vec{x}-2\vec{e}_i} \\
&= u_{\vec{x}} - 2 \cdot \sum_{i=1}^{\dim(\vec{x})} u_{\vec{x}-\vec{e}_i} + \sum_{i=1}^{\dim(\vec{x})} u_{\vec{x}-2\vec{e}_i}
\end{aligned}$$

Bemerkung (Indizes)**.** Da der Definitionsbereich der besprochenen Funktion diskret ist, verringert sich die Kardinalität des Definitionsbereichs der Differenzen mit fortschreitender Ordnung jeweils um 1 pro Dimension. Geographisch gesprochen, liegt die Differenz

zweier Zellen ja eigentlich auf der gemeinsamen Kante dieser zwei Zellen. Daher ist es in der Praxis sinnvoll, die Differenzen gerader Ordnungen (bei deren Berechnung wie eben gesehen 3 Zellen pro Dimension involviert sind) jeweils in die Zelle in der Mitte zu schieben. Im Falle der zweiten Ordnung ergibt sich damit:

$$\left(\frac{\Delta^2}{\Delta\vec{x}^2}u\right)_{\vec{x}} = \sum_{i=1}^{\dim(\vec{x})} u_{\vec{x}-\vec{e}_i} - 2\cdot u_{\vec{x}} + \sum_{i=1}^{\dim(\vec{x})} u_{\vec{x}+\vec{e}_i} \tag{2}$$

5 Integration über der Zeit

und folgenden Zusammenhängen:

$$\begin{array}{rl} u_i & u_i(t+\Delta t) = u_i(t) + \dfrac{\partial u_i(t+\Delta t)}{\partial t}\Delta t \\ \downarrow & \uparrow \\ \dfrac{\partial u_i}{\partial \vec{x}} = u_i - \sum u_{i-1} & \dfrac{\partial u_i(t+\Delta t)}{\partial t} = \dfrac{\partial u_i(t)}{\partial t} + \dfrac{\partial^2 u}{\partial t^2}\Delta t \\ \downarrow & \uparrow \\ \dfrac{\partial^2 u_i}{\partial \vec{x}^2} = -2u_i + \sum u_{i-1} + \sum u_{i+1} \underbrace{\longrightarrow}_{\frac{\partial^2 u}{\partial t^2}=k\cdot\frac{\partial^2 u}{\partial \vec{x}^2}} & \dfrac{\partial^2 u}{\partial t^2} \end{array}$$

sowie unter der Annahme, dass die Richtung der Nachbarzellen bekannt sind (d.h. jeweils der Vorgänger beziehungsweise der Nachfolger innerhalb einer Dimension bekannt sind und deswegen N auch ausgedrückt werden kann als:)

$$N^* = \begin{pmatrix} \overbrace{n_{1,1} \in N}^{\text{1. Dimension}} & \overbrace{n_{2,1} \in N}^{\text{2. Dimension}} & \cdots & \overbrace{n_{\frac{1}{2}|N|,1} \in N}^{\frac{1}{2}|N|\text{te Dimension}} & \text{Vorgänger} \\ n_{1,2} \in N & n_{2,2} \in N & \cdots & n_{\frac{1}{2}|N|,2} \in N & \text{Nachfolger} \end{pmatrix}$$

kann die Übergangsfunktion δ für die Wellengleichung folgendermassen formuliert werden:

$$\delta(r, N^*) =$$

6 δ der Ente

Zu jeder ernstzunehmenden Simulation von Wasser gehören sich darin (aktiv oder passiv) bewegende Objekte – im Idealfall sind das Enten. Die Differentialgleichung der Badeente im Wasser lautet:

$$\frac{\partial \vec{x}}{\partial t} = k \cdot \frac{\partial u}{\partial \vec{x}}$$

Sprich: der Geschwindigkeitsvektor im $\vec{x}$-Vektorfeld (also die Ableitung nach der Zeit t) ist abhängig von den Steigungen (also der Differenzen der Höhe u) entlang aller Dimensionen.

Speziell dabei ist noch, dass so eine Ente natürlich nur in endlich vielen Zellen vorkommen kann. Da die $\vec{x}$-Komponente

7 Numerische Verfahren

8 Bemerkungen zur Implementation

8.1 Wahl des Umfelds

Die Wahl des Umfelds fiel auf eine Kombination aus JavaScript und dem HTML5-`canvas`-Element für die grafische Ausgabe. Die hauptsächlichen Gründe dafür sind die einfache Umsetzung sowie die sehr gute Portabilität, vorallem auch auf portable Devices mit Touch-Bedienung. Dadurch kommt der primäre Use Case der Software (Bewegung einer Flüssigkeit mittels dem Pointing Input) besonders gut zur Geltung, weil der Input eben mit dem Finger vorgenommen werden kann.

Bedenken wurden vorgängig und während der ersten Implementierungsphase vor allem in Bezug auf die Perfomance geäussert, konnten aber im Laufe des Projekts zerstreut werden, auch weil gezeigt werden konnte, dass Optimierungen möglich sind (paralleles Berechnen, Umstellung von Vektorgrafik-Operationen auf Bitmap-Manipulationen).

Als Alternativen waren das Game-Framework XNA sowie die 3D-LED-Installation in der Haupthalle des Zürcher Hauptbahnhofs in Betracht gezogen worden.

8.2 Architektur

8.3 `CellularAutomata`

Die Klasse `CellularAutomata` hält die einzelnen Zellen, kontrolliert den Ablauf der Time-Slices und verarbeitet die Rückgabewerte der Gleichungsimplementation.

Sie besitzt eine sehr nützliche Hilfsmethode, die sowohl von ihr selber wie auch beispielsweise von der `View`-Klasse benutzt wird:

```
this.forEachCell = function(fn) {
  for (var x = 0; x < this.cols; x++) {
    for (var y = 0; y < this.rows; y++) {
      var cell = this.model[x][y];
      fn(cell);
    }
  }
};
```

8.3.1 Initialisierung der Zellen

Zu Anfang werden alle Zellen des Automaten hergestellt und initialisiert:

```
this.initCells = function() {
  this.createCells();
  this.wireNeighbourCells();
};
```

Die Daten jeder Zelle sind abhängig von der gewählten `Tusk`-Implementation und werden entsprechend von ihr selber initialisiert:

```
this.createCells = function() {
  // [...]
  if (tusk != null) {
    cell.currentData = tusk.createCellData();
  }
  // [...]
}
```

Danach werden noch alle Zellen mit ihren Nachbarn verknüpft. Da die Art der Nachbarschaft (Von-Neumann, Moore...) wiederum von der gewählten `Tusk`-Implementation abhängig ist, wird die entsprechende Implementation herangezogen[3]:

```
this.wireNeighbourCells = function() {
  if (tusk != null) {
    this.forEachCell(
      (function(automata) {
        return function(cell) {
          cell.neighbours = this.tusk.getNeighbours(cell, automata.model)
              ;
        };
      })(this)
    );
  }
}
```

Es macht sehr viel Sinn, die Zell-Nachbarschaften während der Initialisierung des Automaten vorzunehmen; zum einen, weil sie sich während der Iterierung nicht mehr ändern werden, zum anderen, weil die Zellen im Automaten in einem n-dimensionalen Array abgelegt sind, auf das dann bei jedem Iterationsschritt mindestens

$$z_{\min} = |R| \cdot |N|$$

zugegriffen werden müsste (wobei R der ganze Zellraum und N die Menge der Nachbarzellen einer Zelle sei).

8.3.2 Ein Iterationsschritt

8.4 Grafische Ausgabe

Die grafische Ausgabe des zellulären Automaten findet in einem `canvas`-Element statt, in das mittels JavaScript-Code gezeichnet wird. Die dafür zuständigen Klassen heis-

[3] Ein Wort zu der auf den ersten Blick unnötig komplizierten Definition der von `this.forEachCell()` auszuführenden anonymen Funktion: In JavaScript wird der Scope von Variablen durch die Funktion und nicht den Block begrenzt, darum gilt als Best Practice zur expliziten Scope-Gebung im Allgemeinen das Wrappen in einer anonymen Funktion und unmittelbarer Ausführung derselben [4].

sen `...CanvasPainter`, wovon verschiedene Implementationen verfügbar sind. Es wird davon ausgegangen, dass jeder `...CanvasPainter` folgende Methoden anbietet:

`begin()` Beginnt den Zeichnungszyklus.

`updateCellView(cell)` Zeichnet eine einzelne Zelle (`cell`) neu.

`drawDuck(duck, cell)` Zeichnet eine Ente (`duck`) in eine einzelne Zelle (`cell`).

`end()` Beendet den Zeichnungszyklus.

Die Methoden `begin()` und `end()` sind für Implementationen gedacht, die sich nur sinnvoll verhalten wenn sie den gesamten `canvas` neu bezeichnen können und deswegen nach einem atomaren Aufruf von `updateCellView()` einen inkonsistenten Zustand aufweisen.

In den folgenden Abschnitten sind die verfügbaren Implementationen grob erklärt. Es existiert ausserdem eine `Noop`-Implementation, die aber trivial ist und verwendet werden kann, um die grafische Ausgabe komplett abzuschalten.

8.4.1 `VectorCanvasPainter`

Diese Klasse bezeichnet den `canvas` durch Vektorgrafik-Operationen:

```
this.updateCellView = function(cell) {
  var du = cell.currentGradients; // get values
  var x = cell.x * scaleWidth; // calculate x position in pixels
  var y = cell.y * scaleHeight; //  calculate y position in pixels
  var baseColor = this.getBaseColor(); // get base painter color
  // let Tusk calculate actual cell color, and format it as 'rgb(r,g,b)'
  var color = getFormattedColor(du[this.getUIndex()], baseColor.r,
      baseColor.g, baseColor.b);
  this.context.fillStyle = color;
  // draw and fill a rectangle.
  this.context.fillRect(x, y, scaleWidth, scaleHeight);
};
```

8.4.2 `PixelCanvasPainter`

Diese Klasse bezeichnet den `canvas` durch direkte Pixel-Manipulationen. Grundsätzlich funktionert das folgendermassen [6]:

```
var myImageData = context.createImageData(cssWidth, cssHeight);
// ... do manipulation here ...
context.putImageData(myImageData, 0, 0);
```

Es muss allerdings gesagt werden, dass die Methode `putImageData()` in allen verfügbaren Implementationen (Gecko, WebKit usw.) einigermassen langsam ist. Dies wird vor allem damit erklärt, dass das zu manipulierende Pixel-Array (`myImageData.data` in unserem Beispiel) ein Array von Integern ist, das von `putImageData()` jeweils noch durchlaufen und auf Gültigkeit (Werte $\in [0:255]$) geprüft werden muss.

Andrew J. Baker zeigt nun in [1] einen Weg, durch den die Pixelmanipulationen drastisch beschleunigt werden können (solange die verwendete Implementation den Typ `Uint8ClampedArray` unterstützt, wie vom HTML5-Standard [5] eigentlich vorgesehen).

Dazu ist es erst einmal nötig, in `begin()` die Pixel-Daten vorzubereiten:

```
this.begin = function() {
  this.imageData = this.context.createImageData(WIDTH, HEIGHT);
  this.buf = new ArrayBuffer(this.imageData.data.length);
  this.buf8 = new Uint8ClampedArray(this.buf);
  this.data = new Uint32Array(this.buf);
};
```

Dadurch ist es möglich, die in `this.data` enthaltenen Daten als ganze Pixel zu bearbeiten (und nicht den Alpha-, Rot-, Grün- und Blau-Kanal einzeln):

```
this.updateCellView = function(cell) {
  /* [...] */
  for (var ix = x; ix < x + scaleWidth; ix++) {
    for (var iy = y; iy < y + scaleHeight; iy++) {
      var p = (iy * WIDTH + ix);
      this.data[p] =
        (255   << 24) | // alpha
        (color.b << 16) |
        (color.g <<  8) |
        color.r;
    }
  }
  /* [...] */
}
```

Abschliessend werden dann die manipulierten Pixel wieder zurückgeschrieben:

```
this.end = function() {
  this.imageData.data.set(this.buf8);
  this.context.putImageData(this.imageData, 0, 0);
};
```

9 Schlussfolgerung

Literaturverzeichnis

[1] Andrew J. Baker. Faster Canvas Pixel Manipulation with Typed Arrays. `http://hacks.mozilla.org/2011/12/faster-canvas-pixel-manipulation-with-typed-arrays/`, 2011. [Online; Stand 17. Mai 2012].

[2] Hans-Georg Beckmann. Zelluläre Automaten. `http://www.vlin.de/material/ZAutomaten.pdf`, 2003. [Online; Stand 28. April 2012].

[3] André Betz. *Das eindimensionale Universum. Einführung in ein informationstheoretisches Weltbild*. 1 edition, 2003.

[4] Rob Gravelle. Understanding JavaScript Closures. http://www.webreference.com/programming/javascript/rg36/index.html, 2012. [Online; Stand 20. Mai 2012].

[5] Ian Hickson. HTML Canvas 2D Context – Editor's Draft 7 March 2012. http://dev.w3.org/html5/2dcontext/, 2012. [Online; Stand 17. Mai 2012].

[6] Mozilla Developer Network. Pixel manipulation with canvas. https://developer.mozilla.org/En/HTML/Canvas/Pixel_manipulation_with_canvas, 2012. [Online; Stand 17. Mai 2012].

[7] Stefan Baur und Markus Hanselmann. Zelluläre Automaten. http://www5.in.tum.de/FA/_2005/K5/Schurr/11_CA.PDF, 2005. [Online; Stand 28. April 2012].

[8] Wikipedia. Conways Spiel des Lebens — Wikipedia, Die freie Enzyklopädie. http://de.wikipedia.org/w/index.php?title=Conways_Spiel_des_Lebens&oldid=100624377, 2012. [Online; Stand 28. April 2012].

[9] Wikipedia. John Horton Conway — Wikipedia, Die freie Enzyklopädie. http://de.wikipedia.org/w/index.php?title=John_Horton_Conway&oldid=101731222, 2012. [Online; Stand 28. April 2012].

[10] Wikipedia. Stephen Wolfram — Wikipedia, Die freie Enzyklopädie. http://de.wikipedia.org/w/index.php?title=Stephen_Wolfram&oldid=99357821, 2012. [Online; Stand 28. April 2012].

[11] Wikipedia. Zellulärer Automat — Wikipedia, Die freie Enzyklopädie. http://de.wikipedia.org/w/index.php?title=Zellul%C3%A4rer_Automat&oldid=100860810, 2012. [Online; Stand 28. April 2012].

[12] Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 1 edition, 5 2002.

[13] Konrad Zuse. *Rechnender Raum*. 1969.