# Zellomat3D

## Analyse

| | | |
|---|---|---|
| **Projekt:** | 3D Cellular Automata Simulator – Diplomarbeit – SS/2005 | |
| **Auftraggeber:** | Hochschule Rapperswil HSR | |
| **Betreuer:** | Eduard Glatz – Prof. Dipl. Ing. ETH | eglatz@hsr.ch |
| **Mitarbeiter:** | Michael Florin | loop@loop.li |
| | Andreas Weinmann | a.weinmann@gmx.ch |
| **Ablage**: | Analyse - 06042005.doc | |

# Inhaltsverzeichnis

# 1. Einführung

**Zweck**

Dieses Dokument dient dazu, die Analyse der entstehenden Applikation aufzuzeigen. Darunter wird eine Übersicht über den Aufbau und die Architektur gegeben sowie erklärt, warum einzelne Implementierungen von Teilanforderungen so gelöst wurden.

**Gültigkeitsbereich**

Dieses Dokument gilt für die Diplomarbeit "Zellomat3D", welche im SS/2005 an der Hochschule Rapperswil HSR durchgeführt wurde.

# 2. Zelluläre Automaten

**Beweggründe**

Um überhaupt eine sinnvolle Analyse der zu erstellenden Applikation zu machen, müssen im Vorfeld einige Abklärungen hinsichtlich der Variationsmöglichkeiten von zellulären Automaten gemacht werden.
In diesem ersten Teil der Analyse werden verschiedene Arten von Automaten auf ihr Verhalten, ihre Komplexität und ihre Anforderung an den Zellomat3D hin untersucht.

**Definition eines ZA**

Eine Zelle eines ZA hat endliche viele Zustände. Die Zellen sind in einer geometrischen Struktur angeordnet, welchen die Nachbarschaftsbeziehung definiert. Die neuen Zustände aller Zellen werden simultan anhand des Zustandes der Zelle und allen ihren Nachbarn bestimmt.

**Klassifizierung**

Stephen Wolfram teilt in seinen Theorien[1] die verschieden Automatentypen in vier Grundklassifizierungen auf:

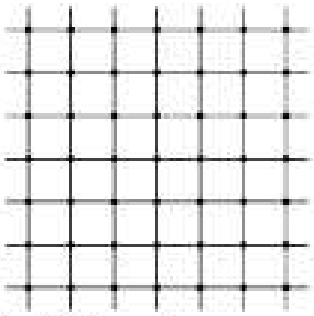
- Automaten der Klasse 1 entwickeln sich aus beliebigen Anfangszuständen zu einem unveränderlichen Endzustand.
- Automaten der Klasse 2 bilden im Laufe ihrer Entwicklung Muster aus, die sich periodisch für alle Zeiten wiederholen.
- Automaten der Klasse 3 zeigen ein chaotisches Verhalten, ihre Muster lassen keine Periodizität erkennen. Gelegentlich tritt Selbstähnlichkeit auf.
- Automaten der Gruppe 4 entwickeln komplizierte, räumlich voneinander getrennte Strukturen; sie sind instabil und aperiodisch.

**Dimensionen**

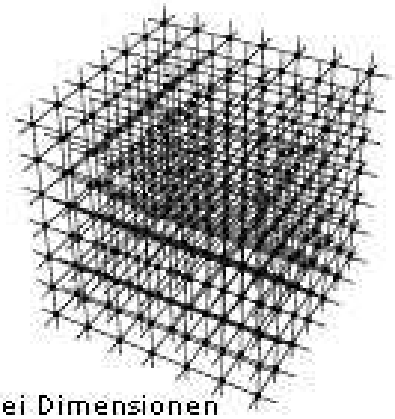
Diese Klassifizierungen sind auf Automaten in jeder Dimension anwendbar. Aber wie wirkt sich der Schritt in eine nächst höhere Dimension auf die Anzahl der Berechnungen pro Zyklus eines Automaten aus? Hier eine Hochrechnung:

[1] http://web.archive.org/web/20010617131722/www.light-edition.midroth.com/back/analy/2hec19.htm

Eine Dimension



Zwei Dimensionen



Drei Dimensionen

**Hochrechnung**

Vorgaben:

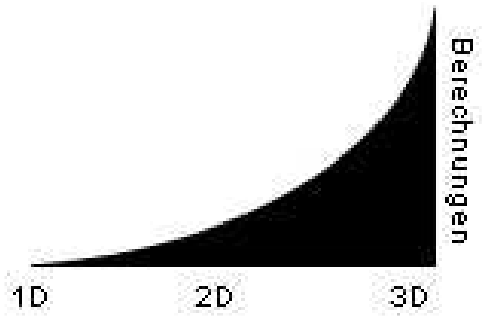
- 50 Berechnungszyklen
- Seitenlänge des Zellraums = 40 Zellen.

Ergebnisse:

- Eine Dimension:
  $50 \times 40^1$ = 2'000 Berechnungen
- Zwei Dimensionen:
  $50 \times 40^2$ = 80'000 Berechnungen
- Drei Dimensionen:
  $50 \times 40^3$ = 3'200'000 Berechnungen

**Erkenntnisse**

Wie man leicht erkennen kann, steigt mit dem Schritt in eine nächst höhere Dimension die benötigte Rechenleistung stark an. Da die zu entwickelnde Applikation mit der dritten Dimension arbeitet, muss unbedingt auf eine hohe Berechnungsgeschwindigkeit der einzelnen Zellen sowie auf einen niedrigen Speicherbedarf des Arbeitsspeichers geachtet werden.

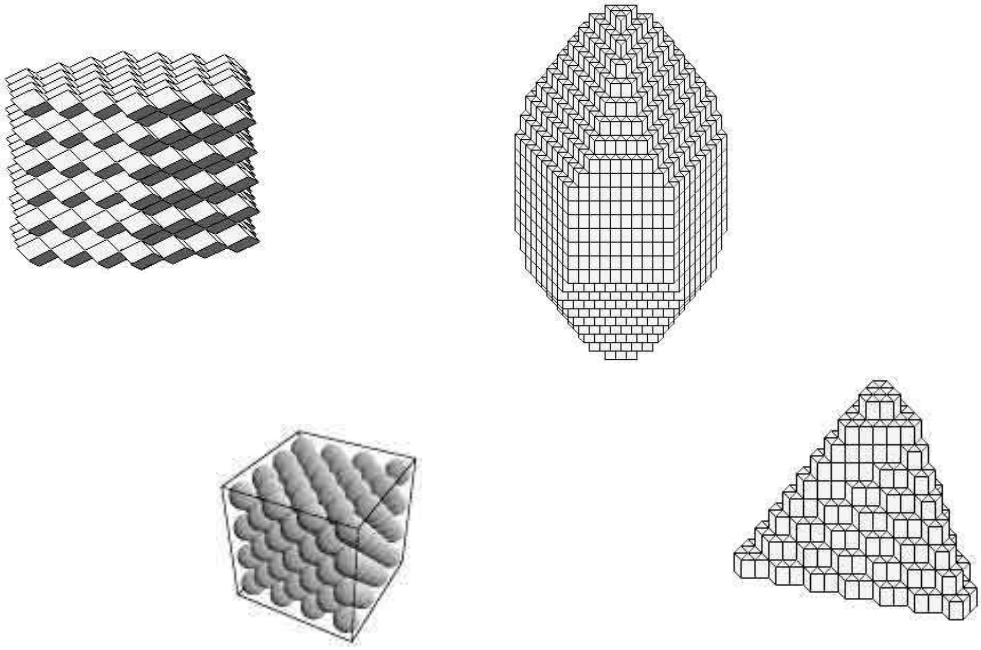



**Entscheidung**

Es muss ein Algorithmus gefunden werden, der all diese Eigenschaften realisieren kann, denn sonst wird die Hardware viel zu schnell an ihre Leistungsgrenzen stossen.

**Formen**

Die Anordnung der Zellen untereinander kann auf unterschiedlichste Art und Weise realisiert werden. Von Tetraeder, Hexaeder, Oktaeder über Dodekaeder bis zu Ikosaeder sind alle geometrischen Formen denkbar, solange sie ohne Schwierigkeiten in einem Gitter aneinander gereiht werden können.

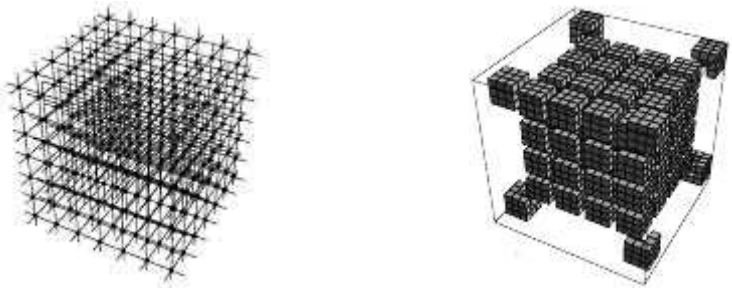


**Erkenntnis**

Es ist also möglich, die unterschiedlichsten Zellenanordnungen zu realisieren.

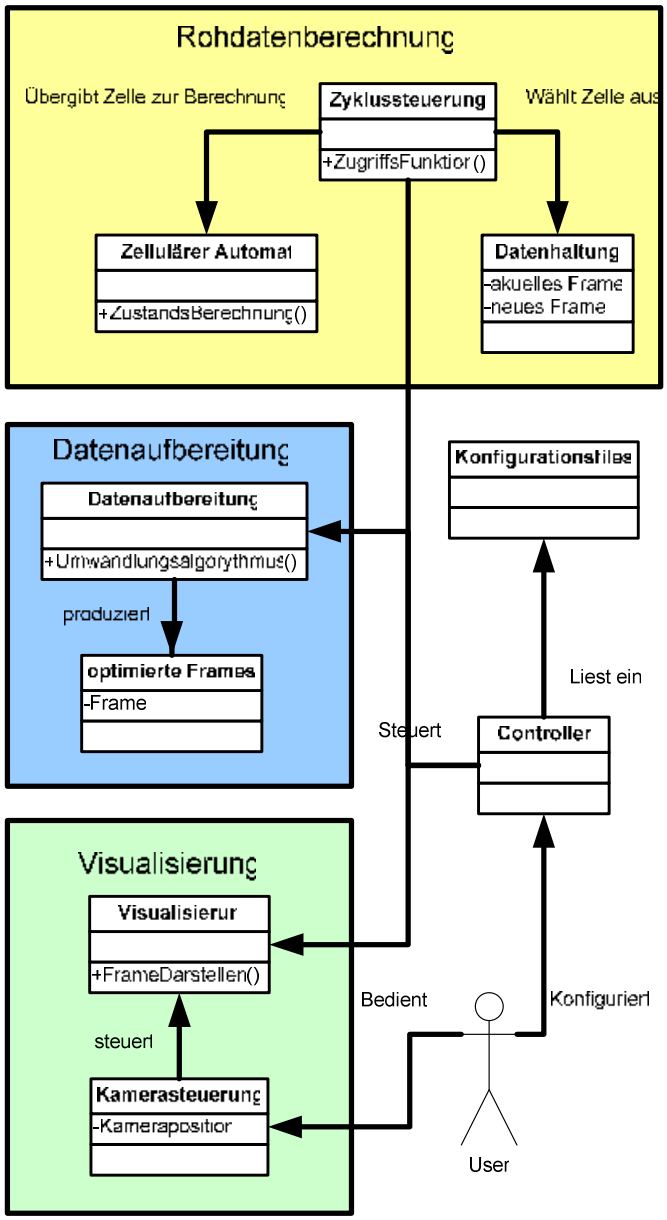
**Entscheidung**

Um das logisch-räumliche Denken der Teammitglieder nicht zu überfordern, werden in der zu entwickelnden Applikation ausschliesslich hexagonale Gitter verwendet und darin gleichseitige Kuben als Zellen definiert.



# 3. Gesamtüberblick

**Domainmodell**

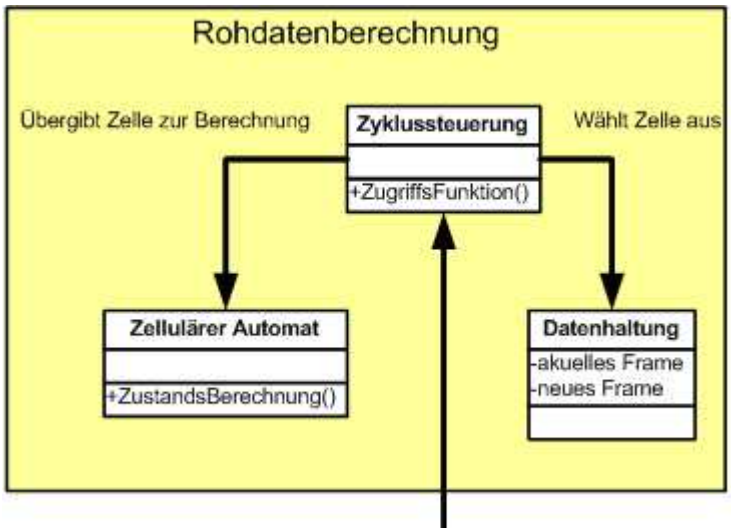



**Beschreibung**

Die Funktionen dieses Projekts können grob in vier Teile aufgeteilt werden. Diese vier Teile sind die Rohdatenberechnung, die Datenaufbereitung, die Visualisierung und der Controller. Der Controller ist für die Steuerung des gesamten Programmablaufes zuständig. Die Parameter für die Initialisierung der Applikation befinden sich in den Konfigurationsdateien. Die Rohdatenberechnung erstellt die Rohdaten. Die Datenaufbereitung transformiert die Rohdaten in visualisierbare Frames. Die Visualisierung stellt die Frames auf dem Monitor dar. Sie erlaubt dem Benutzer auch, die Kameraposition zu steuern.

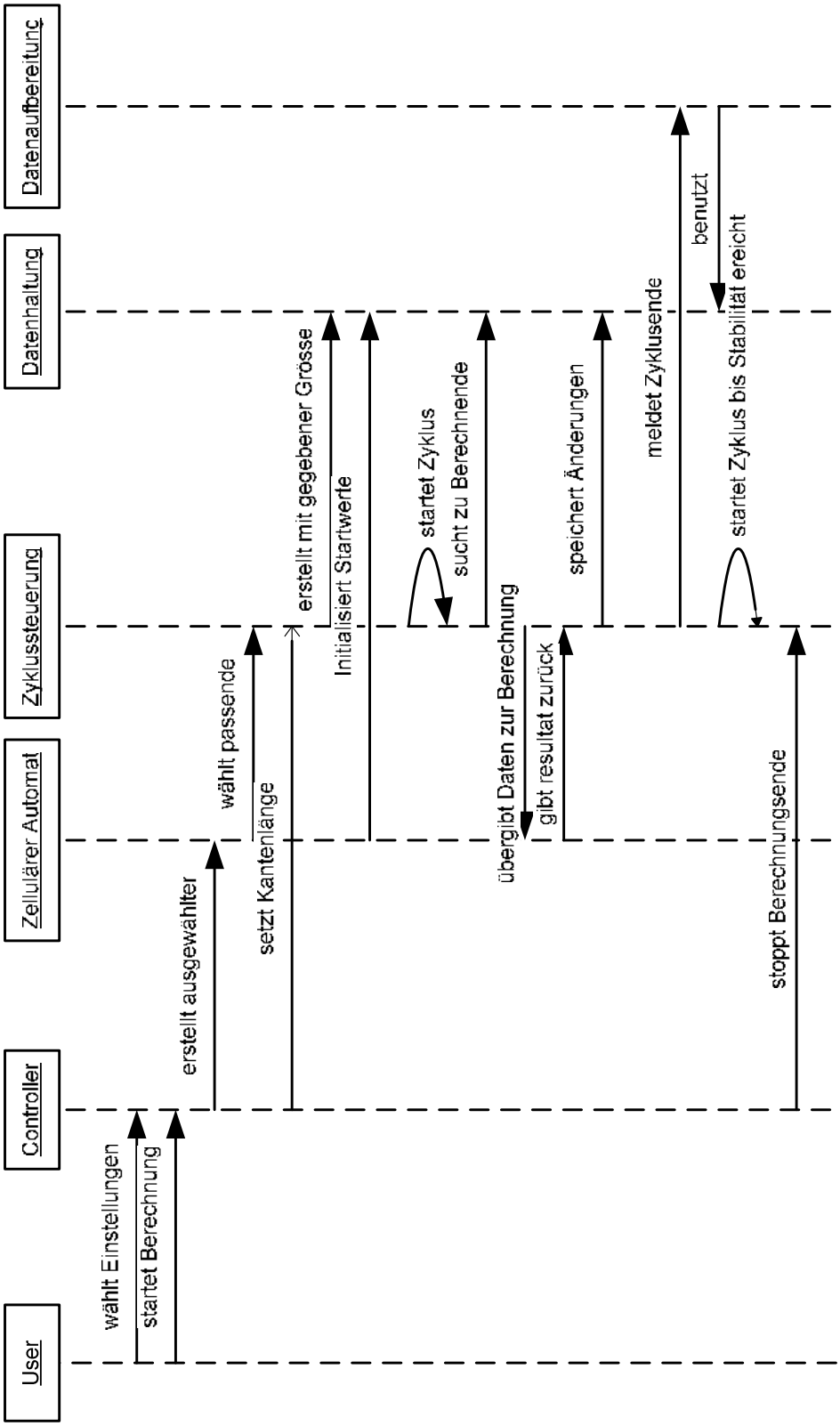
# 4. Modul Rohdatenberechnung

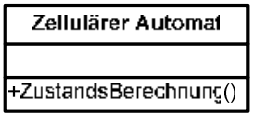
**Domainmodell**




Das Berechnungsmodul beinhaltet die Funktionsbereiche „zellulärer Automat“, „Zyklussteuerung“ und „Datenhaltung“ der Zellen.
Der Controller dient zur Parametrisierung sowie zur Steuerung der Rohdatenberechnung. Die „zellulären Automaten“ beinhalten die Regeln, die das Verhalten der Zellen bestimmen. Die „Zyklussteuerung“ bestimmt, welche Zellen von dem ZA berechnet werden müssen. Alle Zellen werden in der „Datenhaltung“ zusammengefasst und in einer passenden, möglichst wenig „speicherhungrigen“ Form im Arbeitsspeicher abgelegt.

**Sequenzdiagramm**






**Zellulärer Automat**

In einem ZA werden die Regeln definiert, die das Verhalten des Zellomat3D bestimmen. Anhand dieser Regeln wird er in jedem Zyklus die Zustände der Zellen neu bestimmen.

**Zustands-berechnung der Zellen**

Der ZA liest die benötigten Nachbarzustände einer Zelle ein. Aus diesen Zuständen berechnet er den neuen Zustand der Zelle. Die Änderungen der Zellzustände dürfen erst übernommen werden, wenn alle Zellen ihren „neuen“ Zustand berechnet haben, denn sonst würden gewisse Zellen schon einen Zustand in die Berechnung mit einbeziehen, der im gleichen Zyklus entstanden ist. Dies würde dem Prinzip eines ZA widersprechen.

**Erkenntnis**

Es braucht für jede Zelle einen „aktuellen“ Zustand, der abgelesen wird, und einen „neuen“ Zustand, auf den geschrieben wird. Wenn alle Zellen eines Zyklus’ berechnet worden sind, werden die “neuen“ Zustände zu den “aktuellen“ Zuständen. Im nächsten Zyklus werden wieder „neue“ Zustände berechnet usw.

**Zyklussteuerung**


Die Zyklussteuerung ist für die Auswahl der zu berechnenden Zellen verantwortlich. Der trivialste Algorithmus wählt der Reihe nach alle Zellen im ganzen Lebensraum aus.

**1. Optimierungs-möglichkeit für die Zyklussteuerung**

Durch diesen trivialen Algorithmus würden aber auch die Zellen berechnet, die ihren Zustand im anstehenden Zyklus nicht ändern. Der ideale Algorithmus würde nur die Zellen bestimmen, deren Zustände nach der Berechnung anders sein werden als vor der Berechnung. Genau diese Zellen zu bestimmen ist aber nicht möglich. Dagegen können etliche Zellen bestimmt werden, bei denen der Zustand sich sicher nicht ändert. Bei allen Zellen, deren eigener Zustand und jener aller Nachbarzellen sich im letzten Zyklus nicht geändert haben, wird sich auch der Zustand in diesem Zyklus nicht ändern. Eine solche Zelle braucht nicht berechnet zu werden.

**2. Optimierungs-möglichkeit für die Zyklussteuerung**

Ein zusätzlicher Performancegewinn kann erzielt werden, wenn es Zustände von Zellen gibt, die stabiler sind als andere. Eine solche stabile Zelle bräuchte nicht jedes Mal neu berechnet zu werden, wenn eine unstabile Zelle berechnet werden muss. Um dies zu realisieren, wird mit dem Zustand zusätzlich auch jedes Mal ein Faktor gesetzt, der bestimmt, nach wie vielen Zyklen die Zelle erst wieder neu berechnet werden muss. Der Nachteil dieser Optimierung ist, dass sich durch diesen Faktor das Datenvolumen pro Zelle in der Datenhaltung vergrössert. Dieser Faktor muss nämlich in jeder Zelle gespeichert werden.

**Erkenntnis**

Es ist nun ein Algorithmus zu finden, der diese Gesetzmässigkeiten ausnutzt. Der zusätzliche Verwaltungsaufwand darf aber den Gewinn an Performance durch die weggelassenen Zellberechnungen nicht wieder zunichte machen. Da im schlechtesten Fall alle Zellen berechnet werden müssen, kann ein solcher Algorithmus auch zu Performanceeinbussen führen. Man muss abschätzen, ob die durchschnittliche Performance besser oder schlechter ist als der einfachste und trivialste Algorithmus.

**Datenhaltung**

| Datenhaltung |
| --- |
| -akuelles Frame<br>-neues Frame |
| |

**Anforderungen an die Datenhaltung**

Die Konzeption der Datenhaltung ist sehr wichtig. Sie muss einerseits eine einfache und schnelle Zyklussteuerung ermöglichen und anderseits auch eine sehr schnelle Berechnung der Zellen durch den ZA unterstützen. Ausserdem muss darauf geachtet werden, dass man nicht an die Grenze des Arbeitsspeichers stösst. Es werden folgende Ansprüche an die Datenhaltung gestellt:

- Geringe Grösse
- Schneller Zugriff auf die Nachbarzellen
- Gute Unterstützung der Zyklussteuerung

**Grösse der Datenhaltung**

Das Datenvolumen pro Zelle muss wegen der grossen Menge von Zellen sehr klein gehalten werden, denn wenn die maximale Grösse des Arbeitspeichers überschritten wird, frisst das Pagingverfahren des Betriebssystems den grössten Teil der Systemperformance, wodurch das Arbeiten verunmöglicht wird.
Die geforderte geringe Grösse kann mittels folgenden Punkten erreicht werden:

- Die Position der Zelle muss schnell aus der Datenstruktur berechnet werden können, damit sie nicht in der Zelle selber gespeichert werden muss.
- Es muss auf den Gebrauch von passenden Datentypen geachtet werden.
- Die Struktur der Datenhaltung darf nicht viel Overhead enthalten.

**Schneller Nachbarschafts-zugriff**

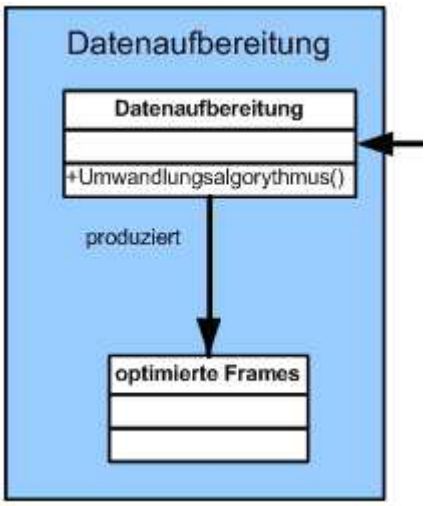
Es muss einfach sein, die Nachbarzellen einer Zelle zu bestimmen. Denn jede Zelle braucht für die Errechnung ihres neuen Zustands die Zustände Nachbarzellen.

**Gute Unterstützung der Zyklussteuerung**

Es muss effizient bestimmt werden können, welche Zellen sich im letzten Zyklus geändert haben und daher neu berechnet werden müssen. Das Beste wäre eine Lösung, die nicht alle Zellen auf Änderungen testen muss.

# 5. Modul Datenaufbereitung

**Domainmodell Modul Datenaufbereitung**




**Erklärung**

Die Datenaufbereitung funktioniert nach dem Prinzip der Umwandlung und Optimierung der Rohdaten in ein grafisches Format, um eine ansprechende visuelle Darstellung der einzelnen Frames zu erreichen.

**Ablauf**

Die Rohdaten werden zuerst aus dem Modul Rohdatenberechnung ausgelesen, anschliessend so umgewandelt, dass sie dem erforderlichen grafischen Format für die Anzeige entsprechen und dann dem Modul Visualisierung zugeführt.

**Darstellungsart**

Um den Vorgaben aus Stephen Wolframs Buch möglichst nahe zu kommen, könnte man für jede Zelle einen Kubus generieren.

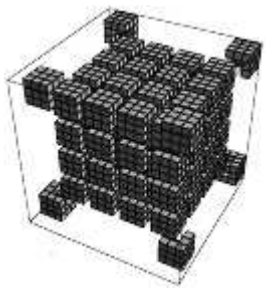


**Abbildung 1:** Zellulärer Automat aus „Stephen Wolfram, a new kind of science“

Jeder Zustand einer Zelle wird fest einer Farbe zugeordnet. Die Zuordnung kann aus der Farbentabelle des Benutzerhandbuches heraus gelesen werden.
Um das Verhalten des gesamten Systems erfassen zu können, muss man sich frei im Raum bewegen können.

**Optimierungen**

Warum sollte man in jedem Frame alle Zellen darzustellen versuchen? Was passiert mit den Zellen, die zwar existieren, aber vom Betrachter nicht gesehen werden, weil sie von anderen Zellen verdeckt werden?

**1. Optimierungs-möglichkeit**

Man könnte versuchen, die Zellen, die von anderen Zellen verdeckt werden, grafisch nicht darzustellen. Dadurch würde sich das Datenvolumen der einzelnen Frames um ein Vielfaches reduzieren.
Um diese Optimierung durchzuführen zu können, müsste aber die Blickrichtung des Beobachters schon im Vornherein bekannt sein, was leider zu diesem Zeitpunkt nicht der Fall ist, da sich der Betrachter während der nachfolgenden Visualisierung frei im Lebensraum des ZA bewegen können soll.

**2. Optimierungs-möglichkeit**

Um das Datenvolumen der einzelnen Frames trotzdem auf ein Minimum zu bringen, wäre die Generierung eines „Meshes“[2] sehr von Vorteil.
Konkret bedeutet das, dass man so etwas wie ein „Tuch“ über alle zusammenhängenden Bereiche von Zellen des gleichen Zustandes legt und nur die generierte Oberfläche visualisiert. Dieses Verfahren ermöglicht es, die Rohdatenmenge beträchtlich zu reduzieren.

**Einschränkungen**

Bei der 2. Optimierungsmöglichkeit ist auch der „Worstcase“ nicht ausser Acht zu lassen: In unserem Fall wäre dies ein zellulärer Automat, bei welchem benachbarte Zellen immer unterschiedliche Zustände besitzen. Dies würde zur Folge haben, dass sich die generierte Oberfläche um jede einzelne Zelle legen würde. Ausser viel Rechenarbeit entstünde dann keine Reduktion der Datenmenge in einem Frame.

**Entscheidung**

Die 2. Optimierungsmöglichkeit wird in der Applikation Zellomat3D aufgrund ihrer überwiegenden Vorteile realisiert.
Sie eignet sich ideal für die Generierung von Kuben und hat, ausser bei einem „Worstcase-Szenario“, immer eine Reduktion der Datenmenge der Frames zur Folge.

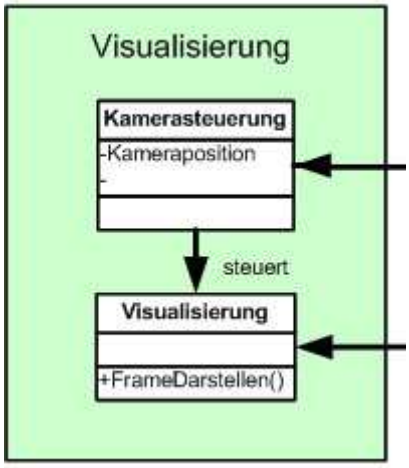
**Begründung**

Auf die Datenreduktion der einzelnen Frames wird deshalb besonderen Wert gelegt, weil die Datentransferraten des AGP-Busses und die Speichergrössen der Grafikkarten nicht unendlich gross sind. Durch diese Reduktion ist es also möglich, Automaten mit grösseren Lebensräumen zu simulieren.

[2] Mesh: Datenformat für grafische Daten, z.B. Spielfiguren und Gegenstände in 3D-Egoshootern

# 6. Modul Visualisierung

**Domainmodell Modul Visualisierung**




**Erklärung**

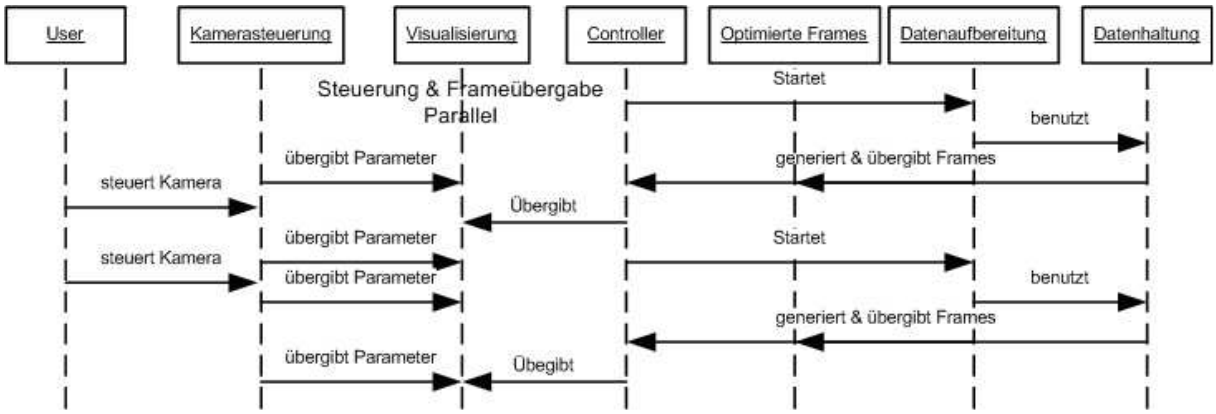
Das Modul Visualisierung ist zuständig für die grafische 3D-Ausgabe auf dem Bildschirm. Es bereitet die Grafikkarte auf die bevorstehenden Aufgaben vor, regelt die Interaktionen des Benutzers mit der Kamerasteuerung und ist verantwortlich für den reibungslosen Ablauf der Visualisierung der einzelnen Frames.

**Ablauf**

Das Modul Datenaufbereitung liefert die optimierten Frames, die dann vom Modul Visualisierung in die Grafikkarte geladen und dort gerendert am Monitor dargestellt werden.
Während dieses Prozesses werden zeitgleich die Steuerung der Kamera und zusätzliche Einstellungen, die der Benutzer wünscht, an die Grafikkarte weitergegeben. Diese generiert dann anhand der gesetzten Parameter und der übermittelten Daten das Bild.

**Sequenzdiagramm**

Die Visualisierung sollte in der Lage sein, parallel Daten verarbeiten zu können. Während das neue Frame geladen wird, kann der Benutzer ungestört seinen Flug durch die Welt der zellulären Automaten fortsetzen.



**Erweiterungen**

Mögliche Erweiterungen, um die visuelle Darstellung der Zyklusschritte des zellulären Automaten noch weiter aufzuwerten, wären zum Beispiel besondere Effekte, darunter Reflexionen oder Schattierungen. Möglich wäre auch die Platzierung der Gebilde in einen Raum mit ansprechendem Design. Der Visualisierung sind eigentlich keine Grenzen gesetzt. Wenn man mitverfolgt, was heutige Grafikkarten an Rechenpower mit sich bringen, kann man unglaubliche Szenarien kreieren und diese in Echtzeit berechnen lassen.

**Beispiel**

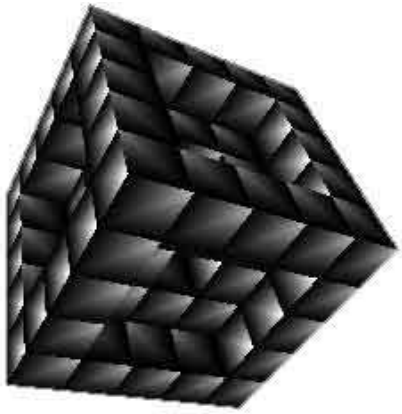


**Abbildung 2:** Kubus aus dem Internet, gezeichnet in 3D Studio Max

**Einschränkungen**

Eine Einschränkung in dieser Diplomarbeit wird die limitierte Datentransferrate der Grafikkarte sein. Diese Hardwarekomponenten wurden nämlich nicht dazu entwickelt, eine sequentielle Abarbeitung von Frames zu bewältigen, sondern nur dazu, die Animation von statischen Elementen zu gewährleisten.
In unserer Applikation müssen für jede Zelle sechs Seitenflächen dargestellt werden, welche je durch vier Punkte bestimmt sind. Diese grosse Anzahl von Punkten übersteigt sehr schnell die Leistungsfähigkeit des Grafikprozessors, da er bei Veränderung der Betrachtungsposition jeden einzelnen Punkt geometrisch transformieren muss.

**Entscheidung**

Es reicht nicht, eine Oberfläche zu erzeugen, darüber hinaus muss darauf geachtet werden, zu ihrer Darstellung möglichst wenige Punkte zu verwenden. Das heisst konkret, dass die Eckpunkte der Flächen mehrfach genutzt werden sollten.
Als erste Priorität muss zuerst etwas am Bildschirm erscheinen, bevor weitere Effekte und Features hinzukommen. Alles Folgende wird erst bei vollständigem Erreichen der Visualisierung der Zyklen eines zellulären Automaten entwickelt, da diese Erweiterungen sehr zeitaufwändig sind.