Zellomat3D

Technischer Bericht

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:	TechnischerBericht - 05052005.doc	

Inhaltsverzeichnis

1. Einführung

Zweck

Dieser Bericht umfasst einen Abriss der technischen Seite des Projekts „Zellomat3D“. Dazu gehören eine Übersicht über die erzielten Ergebnisse sowie Probleme, die während des Projektes auftraten.
Dieses Dokument basiert auf den erarbeiteten Dokumenten sowie den Erfahrungen bzw. Erkenntnissen, welche während des Projektverlaufs gemacht wurden.

Gültigkeitsbereich

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

2. Projektübersicht

Motivation für die Entwicklung der Software

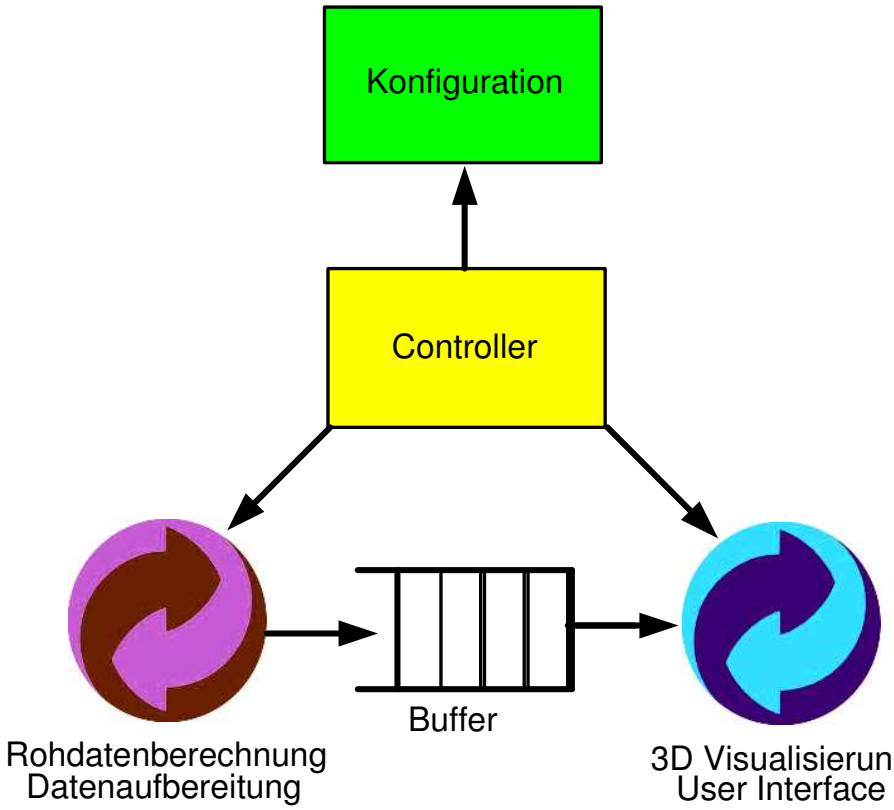
Die Theorie der Cellular Automata befasst sich seit längerem mit selbstwachsenden Organismen auf einer computertechnischen bzw. mathematischen Basis. Praktische Anwendungen sind bei adaptiven und selbstoptimierenden Systemen zu finden, die im Autonomic Computing eine grosse Rolle spielen. Jedoch nur schon die Visualisierung selbstwachsender Systeme ist von Interesse.

Ziel und Zweck des Zellomat3D

Der Zellomat3D beinhaltet verschiedenste Funktionen, um die Berechnung und die visuelle Darstellung von verschiedenen zellulären Automaten zu unterstützen. Er ermöglicht es, auf einfache Art und Weise selbst erstellte Automaten auf ihre Funktionsweise und Auswirkungen hin zu untersuchen. Durch eine hohe Verarbeitungsgeschwindigkeit und eine ansprechende 3D-Visualisierung der einzelnen Entwicklungszyklen des Automaten können schnell und einfach die Wechselwirkungen der zugrunde liegenden Regeln beobachtet werden.

3. Systemaufbau

Systemübersicht



Beschreibung

Die Applikation kann in zwei getrennte Kreisläufe unterteilt werden. Die Berechnung der Rohdaten und deren grafische Aufbereitung sowie die Visualisierung der errechneten Daten. Dazwischen fungiert ein Buffer für die Daten als Trennung zwischen den zwei Kreisläufen.
Der Controller ist für die Steuerung der ganzen Applikation verantwortlich. Das Konfigurationsmodul ist für das Parsen resp. Initialisieren der Daten, die aus den Beschreibungsdateien der ZA gelesen werden, zuständig.

Hardware

Für den Betrieb des Zellomat3D werden folgende minimale Hardwareanforderungen benötigt:

PC-System mit:

- einem Pentium 4 2.2GHz oder einem AMD Athlon XP 2000+ Prozessor
- einer NVIDIA GeForce4 oder ATI Radeon 9500 Grafikkarte mit 64MB RAM (128MB empfohlen)
- 512MB Arbeitsspeicher

Diese hohen Anforderungen sind nötig, denn auch mit der bestmöglichen Optimierung sind die Berechnung der Zyklen und deren Darstellung extrem rechenintensiv. Ausserdem sind die erzeugten Bilder sehr umfangreich, sie enthalten durch ihre Beschaffenheit eine Vielzahl an Punkten und Flächen, welche bei jeder Bewegung der Kamera geometrisch im Raum transformiert werden müssen. Daher ist eine Grafikkarte neuerer Generation unabdingbar.

3D-Technologie

Wegen der einfacheren Ansteuerung der Grafikkarte und der viel schnelleren Entwicklungszeiten wird die Darstellung in Managed DirectX 9.0 realisiert.

4. Ergebnisse

Allgemein

Es wurden fast alle Anforderungen an den Zellomat3D erfüllt. Die Ausnahme bildet die Berechnungszeit. Ein Zellraum von 128x128x128 Zellen braucht im schlechtesten Fall mehr als 10 Sekunden.

Testcase	Funktion	Priorität	Status
TC1	Framebuffer	«	**OK**
TC2	Minimale Hardware Anforderungen	« «	**OK**
TC3	Stabilität der Applikation	« «	**OK**
TC4	Hohe Framerate	« «	**OK**
TC5	Kamerasteuerung	« «	**OK**
TC6	Berechnungszeit	«	**NOK**
TC7	Umsetzung der ZA Regeln	« «	**OK**
TC8	Orientierung	«	**OK**
TC9	Parametrisierung	« «	**OK**
TC10	UC1: Öffnen eines ZA	« «	**OK**
TC11	UC2: Reset ZA	« «	**OK**
TC12	UC3: Play ZA	« «	**OK**
TC13	UC4: Step ZA	« «	**OK**
TC14	UC5: Pause ZA	« «	**OK**
TC15	UC6: Anzeigeintervall einstellen	«	**OK**
TC16	UC7: Schnappschuss eines Zyklus	«	**OK**
TC17	UC8: Informationen über das Programm anzeigen	«	**OK**
TC18	UC9: Hilfe anzeigen / ausblenden	«	**OK**
TC19	UC10: Umgebung anzeigen / ausblenden	«	**OK**
TC20	UC11: Koordinatenachsen anzeigen / ausblenden	«	**OK**
TC21	UC12: Lebensraumkubus anzeigen / ausblenden	«	**OK**
TC22	UC13: Anzeigeeigenschaften verändern	«	**OK**
TC23	UC14: Wechsel in Fenster / Vollbildmodus	«	**OK**
TC24	UC15: Änderung der Grösse des Fensters	«	**OK**
TC25	UC16: Kamerasteuerung	« «	**OK**
TC26	UC17: Beenden der Applikation	« «	**OK**

Begründung

Um aber überhaupt eine Darstellung eines solch grossen Zellhaufens zu ermöglichen, musste ein komplizierter Algorithmus in der Datenaufbereitung implementiert werden. Dieser braucht eine gewisse Zeit, um die Daten für die grafische Ausgabe aufzubereiten und zu komprimieren, da sonst die Grafikkarte mit zuviel Daten überflutet würde und gar nichts mehr zu sehen wäre.

5. Probleme und deren Lösungen

Allgemein

In den folgenden Kapiteln werden die aufgetauchten Probleme, deren Lösungen sowie die daraus folgenden Erkenntnisse zusammengefasst. Für die technischen Umsetzungen und die genaue Beschreibung der Lösungen sei auf das Dokument „StudieAlgorithmen – 05052005.doc“ verwiesen.

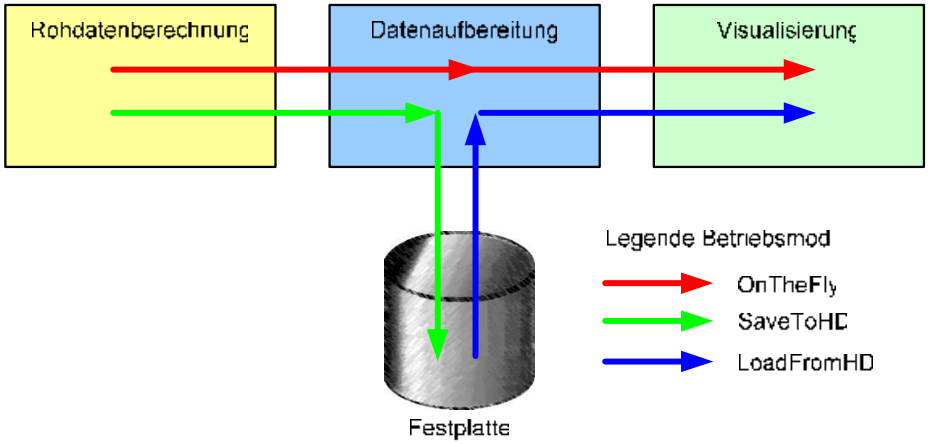
6. Problem Berechnungszeit der ZA-Zyklen

Problematik

Die Berechnung von Zyklus zu Zyklus eines zellulären Automaten ist enorm rechenintensiv. Zur Bestimmung des neuen Zustandes einer Zelle müssen deren Nachbarschaftszustände und ihr eigener Zustand mit allen Regeln des Automaten verglichen werden, was einen riesigen Aufwand für den Berechnungsalgorithmus in sich birgt.

Lösungsanzatz 1: Vorberechnung der Zyklen

Ein möglicher Lösungsansatz für dieses Problem wäre, die Berechnung der Zyklen schon im Vorfeld durchzuführen, diese Resultate auf der Festplatte abzuspeichern und sie zu einem späteren Zeitpunkt grafisch anzuzeigen.



Fazit

Einen vorberechneten Zyklus wieder in die Grafikkarte zu laden benötigt mehr Zeit als den Zyklus zu berechnen und ihn anschliessend direkt auf dem Bildschirm zu visualisieren. Somit scheidet dieser Lösungsansatz aus.

Lösungsansatz 2: Einstellbare Updateraten der Zellen

Ein Performancegewinn kann erzielt werden, wenn es Zellen gibt, die stabiler sind als andere. Eine solche stabile Zelle braucht nicht jedes Mal 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. Ein 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.
Diese Option könnte z.B. bei einem ZA, der eine Explosion simuliert, eingesetzt werden. Die Zellen nahe dem Zünder müssten mit einer hohen „Updaterate“ versehen werden, die Zellen im äusseren Bereich hingegen mit einer sehr niedrigen, da sich die Explosion exponentiell nach aussen verlangsamt.

Fazit

Die Implementation der Updaterate wurde nicht eingehend gestestet, da die diese Verbesserung zu diesem Zeitpunkt als nicht prioritär angesehen wurde. Es hat sich jedoch gezeigt, dass der zusätzliche Berechnungsaufwand minime Performanceeinbussen zur Folge hat.

Lösungsansatz 3: Selektive Berechnung der Zellen

Um den Algorithmus für die Zyklusberechnung effizienter zu gestalten, müssen alle Zellen aus der Berechnung ausgeschlossen werden, die ihren Zustand in nachfolgenden 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. Möglich ist jedoch, etliche Zellen zu bestimmen, bei denen der Zustand sicher nicht ändert. Bei allen Zellen, deren Zustand und jener aller Nachbarzellen sich im letzten Zyklus nicht geändert haben, wird sich der Zustand auch in diesem Zyklus nicht ändern. Solche Zellen brauchen in diesem Zyklus nicht berechnet zu werden.

Fazit

Aufgrund der Erkenntnisse aus den entwickelten Prototypen konnte eine erhebliche Performancesteigerung gegenüber dem vollständigen Durchrechnen aller Zellen pro Zyklus erzielt werden.
Der zusätzliche Verwaltungsaufwand übersteigt den durchschnittlichen Gewinn an Performance durch die weggelassenen Zellberechnungen nicht. Im schlechtesten Fall, wenn alle Zellen berechnet werden müssen, führt dieser Algorithmus zu Performanceeinbussen von 15%.

7. Problem Berechnungslimit der Grafikkarte

Problematik

Wo sind die Grenzen der Grafikkarte in Bezug auf die Geschwindigkeit der Geometrietransformationen und der Datentransferrate zwischen dem Arbeitsspeicher und dem Grafikbuffer? Ziel ist es, mehr als 24 Bilder pro Sekunde zu rendern, um dem Auge einen flüssigen Bildablauf des Kamerafluges vorzugaukeln.

Lösungsansatz

Es wird eine grosse Anzahl von Punkten auf dem Bildschirm angezeigt und eine rudimentäre Kamerasteuerung implementiert. Dadurch werden die Grenzen der Grafikkarte eruiert.

Fazit

Bei mehr als einer Million Punkten kommt die Grafikkarte an ihre Leistungsgrenzen. Es wird die Limite von 24 Bildern pro Sekunde unterschritten. Bei einer grossen Anzahl von Daten, die transferiert werden, stoppt der Renderprozess der Grafikkarte für eine wahrnehmbare Weile.
Diese Probleme können durch eine massive Reduktion der Datenmenge nur vermindert, aber nicht behoben werden.

8. Problem Reduktion der grafischen Datenmenge

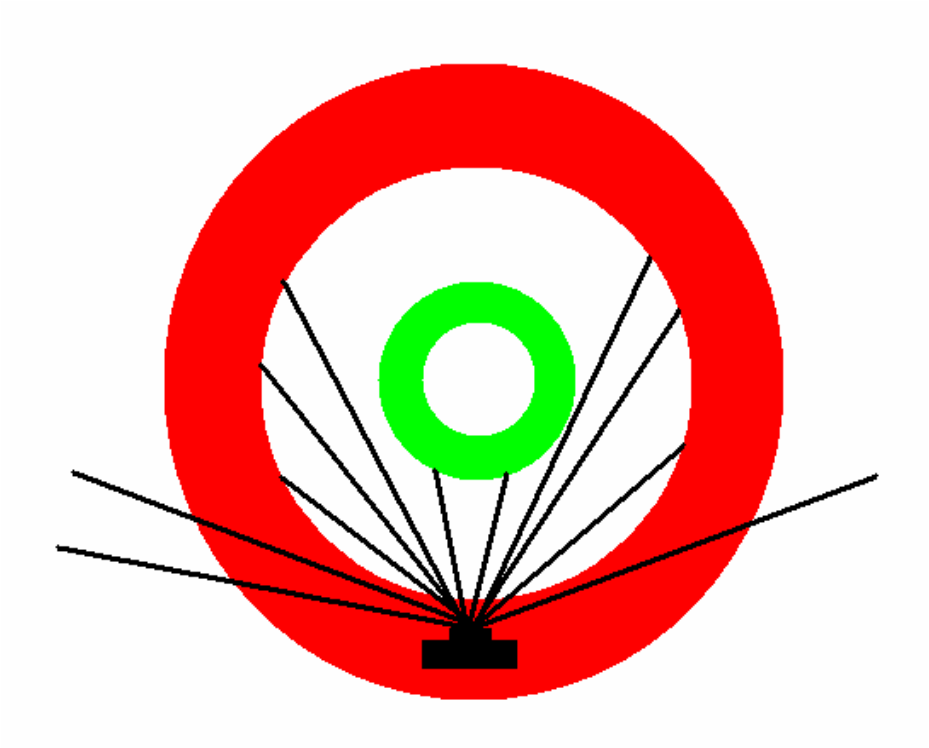
Problematik

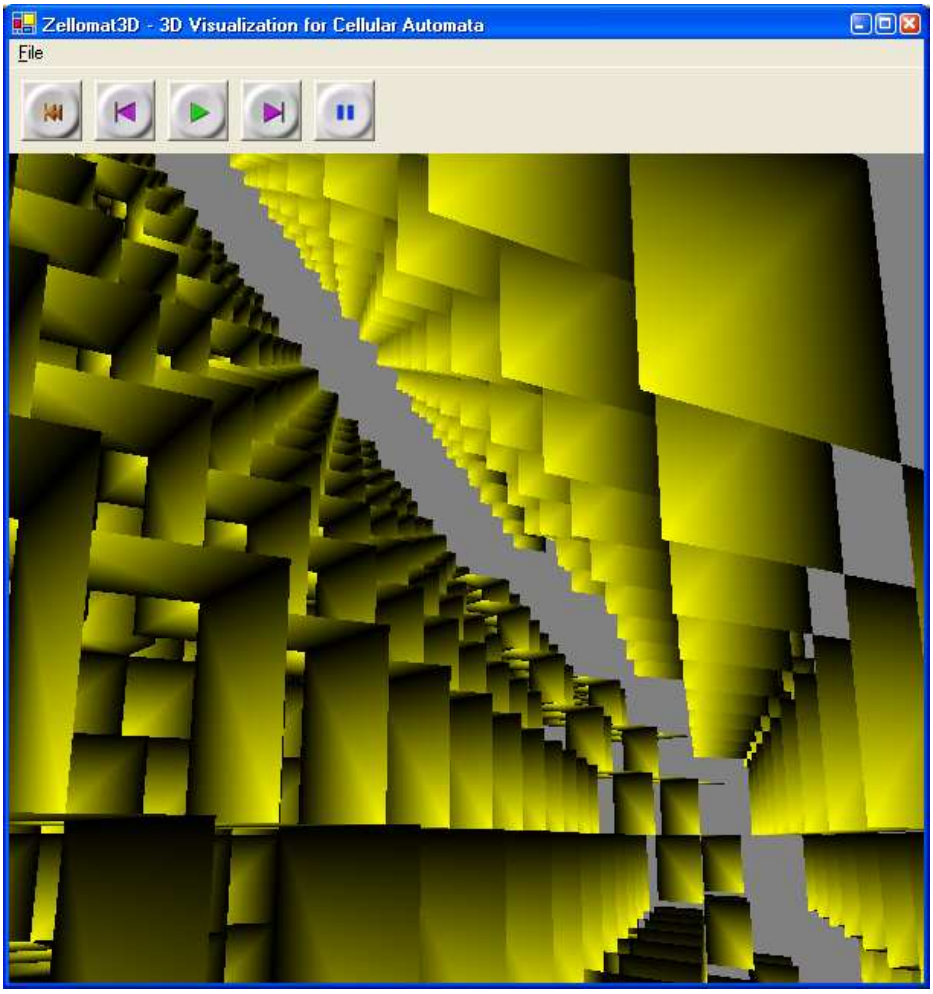
Die grafische Datenmenge muss aufgrund der eruierten Hardwarelimiten massiv reduziert werden, um eine flüssige Ausgabe der Daten auf dem Bildschirm zu ermöglichen.

Lösungsansatz 1: Reduktion durch Oberfläche (nur Aussenflächen)

Um das Datenvolumen der einzelnen Frames auf ein Minimum zu bringen, ist die Generierung einer Oberfläche 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 Rohdaten, die sich „innerhalb“ des Gebildes befinden, einfach ausser Betracht zu lassen, was zu einer enormen Datenreduktion führt.

Abstraktes Kamerasichtfeld





Reeles Kamerasichtfeld

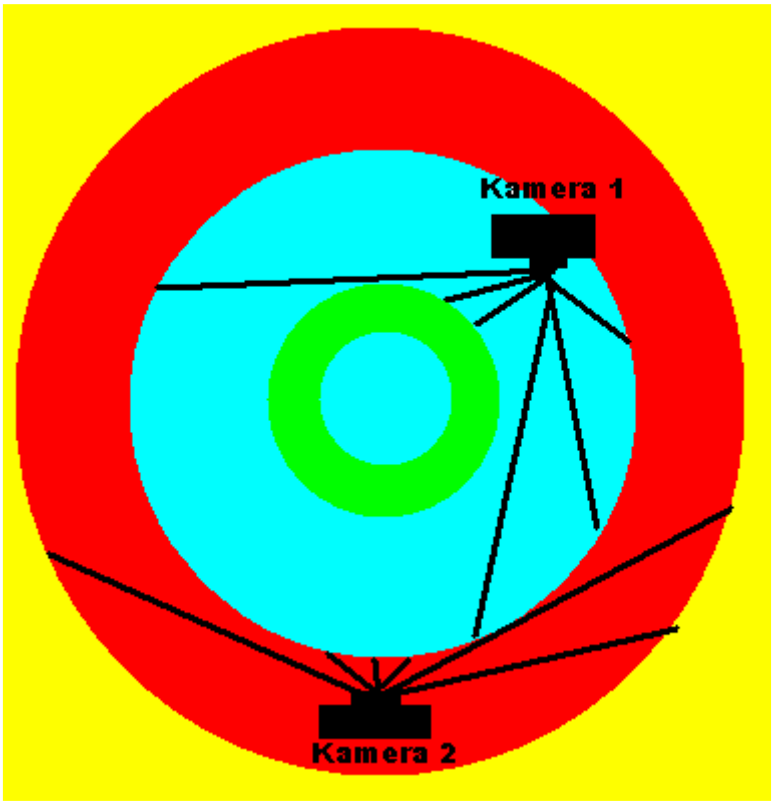
Fazit

Eine sehr hohe Reduktion der Datenmenge konnte erreicht werden. Der Algorithmus zur Reduktion entpuppte sich jedoch als nicht geeignet, da die grafische Ausgabe der ZA sehr verwirrend war, sobald man sich im Innern eines Zellhaufens befand.

Lösungsansatz 2: Reduktion durch Oberfläche mit Aussen- und Innenflächen

Um visuelle Fehlinterpretationen zu vermeiden müssen auch die Innenflächen der Zellen mit den entsprechenden Farben der Nachbarzellen dargestellt werden.

Abstraktes Kamerasichtfeld



Erklärung

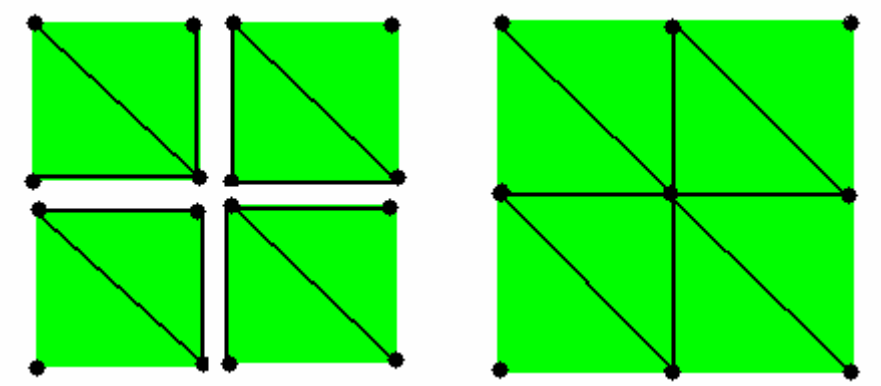
Vorbemerkung: Der hellblaue Raum stellt die Zellen mit dem Zustand 0 dar (leerer Raum).
Kamera 1 sieht die grünen Aussenflächen des grünen Rings sowie die roten Aussenflächen des roten Rings.
Kamera 2 sieht nur die Innenflächen des roten Ringes, wobei diese nicht rot erscheinen, sondern die Farbe der angrenzenden Flächen (Zellen) übernehmen, in diesem Fall also Gelb und Hellblau.

Fazit

Diese Lösung erweist sich optisch als sehr wirklichkeitsnah, denn der Benutzer kann somit nachvollziehen, in welchem Zellzustandsvolumen er sich gerade befindet und welche Zustände (Farben) die angrenzenden Nachbarn haben.
Leider braucht diese Darstellungsart doppelt so viele Geometrie- und Farbdaten wie die vorhergehende Lösung, was aber immer noch eine erhebliche Datenreduktion mit sich bringt.

Lösungsansatz 3: Reduktion der Oberflächendaten

Nun, da eine geeignete Darstellungsart gefunden wurde, muss das verbleibende Datenvolumen noch weiter dezimiert werden.



Erklärung

Die Erklärung erfolgt anhand eines einfacheren 2D-Modells. Auf dem linken Bild sind vier Flächen zu sehen, die ihre eigenen Eckpunkte definieren. Anstatt für jede Fläche eigene Eckpunkte zu generieren, kann man die gleichfarbenen Flächen zusammenfassen und die gleichen Punkte für mehrere Polygone verwenden. Es war technisch leider nicht umsetzbar, die ganze Fläche mit nur vier Punkten zu generieren.

Fazit

Bei 3D-Kuben ist eine Reduktion der Datenmenge um den Faktor sechzehn bei grossen zusammenhängenden Volumen möglich.

9. Problem des Grafikkartenbuffers

Problematik

Es muss ein Speicherplatz gefunden werden, in dem die vorberechneten und grafisch aufbereiteten Zyklen eines ZA bis zu deren Anzeige auf dem Bildschirm zwischengelagert werden können.

Lösungsansatz 1: Speichern der Daten im Arbeitsspeicher

Der Arbeitsspeicher erweist sich als genügend gross und bietet sich daher dafür an, eine Vielzahl von vorberechneten Zyklen zwischenzuspeichern.

Fazit

Wenn das Bild von der Grafikkarte zum Rendern benötigt wird, muss dieses über den Bus vom Arbeitsspeicher in den Grafikkartenspeicher transferiert werden. Die Transferzeiten sind zu vernachlässigen, da keine Verzögerungen in der 3D-Visualisierung bemerkbar sind.
Leider brauchen die Daten eines Bildes eines Zyklus enorm viel Speicher, da das RAM nicht extra für die Speicherung von grafischen Datenformaten ausgelegt ist.

Lösungsansatz 2: Speichern der Daten im Grafikkarten-speicher

Um die Datentransferzeiten zu eliminieren, kann man die vorberechneten Daten auf dem Grafikkartenspeicher ablegen. Der Vorteil dabei ist, dass die Speichereffizienz um ein Vielfaches grösser als die des RAMs ist, wodurch eine hohe Anzahl von vorberechneten Daten gebuffert werden kann.

Fazit

Eine Auflösungsänderung oder sonstige Änderungen an der Grafikkarte während des Programmablaufes haben eine Reinitialisierung der Grafikkarte und somit einen Verlust aller Daten, die sich in deren Speicher befinden, zur Folge. Konkret bedeutet das, dass man alle vorberechneten Daten verliert und wieder bei Null anfangen muss.
Der Vorteil der effizienten Speicherung der Daten überwiegt das Risiko des Totalverlusts aller vorgenerierten Daten, da Veränderungen an der Grafikkarte nur in seltenen Fällen auftreten, nämlich dann, wenn der Benutzer eine Änderung an den 3D-Parametern vornimmt.