Real time boid-simulations with shaders

Maximilian Legnar and Prof. Dr. Rasenat

***Abstract*— Basierend auf Reynolds Boid-System [1] wird ein Modell für die Simulation von Tierschwärmen entwickelt.**

Statt den simulierten Schwarm wie ein Partikelsystem zu behandeln, wird Reynolds Boid-Modell in ein Zellensystem, das mit den zellulären Automaten verwandt ist, transferiert. Diese Anpassung wird vorgenommen, damit die Simulation möglichst effizient von shader-Programmen ausgeführt werden kann.

Das Simulationsmodell unterteilt den Schwarm nicht in einzelne Individuen. Stattdessen ist der Schwarm eine im Raum ungleichmäßig verteilte Menge. Der Schwarm wird daher als leuchtende Punkte-Wolke dargestellt.

Die konzipierte Schwarmsimulation wird mittels Content-Driven Multipass Rendering prototypisch implementiert. Der Prototyp wurde erfolgreich auf Echtzeitfähigkeit geprüft.

I. Introduction and related work

Um die Bewegungsabläufe von natürlichen Tierschwärmen zu simulieren, hat sich das sogenannte Boid-Modell von Reynolds etabliert.

In dieser Arbeit werden die grundlegenden Werke von Reynolds Boid-modell [1] [2] aufgegriffen, um ein neuartiges System zu modellieren, das besonders gut von Shadern und Texturen beschrieben werden kann.

Für gewöhnlich wird Reynolds Boid-System wie ein Partikelsystem behandelt [1] [3]. In dieser Arbeit wird der Schwarm hingegen von einem Zellensystems beschrieben, in dem jede Zelle örtlich gebunden ist. Jede Zelle kann dabei von mehreren Individuen besucht werden. Diese Uminterpretation erfolgt, damit die Simulation auf effiziente Weise von shader-programmen ausgeführt werden kann.

II. Method

A. Das Boid-System als Zellensystem

Das Boid-System ist mit dem Partikelsystem verwand [1] und kann daher für gewöhnlich auch als solches behandelt werden.

Wesentlicher Bestandteil eines Partikelsystems ist die Bewegung, also die Positionsänderung, von Partikeln. Ein Shader kann ein Pixel allerdings nicht ohne weiteres bewegen. Das hängt damit zusammen, dass ein Pixel-Thread nicht in der Lage ist, Werte in andere Pixel zu schreiben. Jedes Pixel kann sein Rechenergebnis nur in sich selbst schreiben.

Aufgrund dieser Einschränkung verabschieden wir uns an dieser Stelle vom Gedanken des Partikelsystems. Stattdessen wird das Boidsystem von einem Zellen-System beschrieben.

Weil shader-programme auf die Verarbeitung von Texturen optimiert sind, wird dieses Zellensystem mithilfe von Texturen beschrieben. Ein Pixel aus einer Textur repräsentiert eine Zelle. Die Individuen des simulierten Schwarms (Boids) bewegen sich dabei durch nebeneinanderliegende Zellen hindurch. Jede Zelle kann eine reelle Menge von Boids, $D_i >= 0$, besitzen.

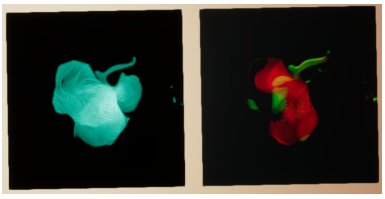

Fig. 1. Left: A texture, that represents the boid-density, D. Each pixel is a real skalar value that represents the amount of boids, contained in this cell. Right: A texture, called V, that holds the control-vectors, v, for each cell. Each boid will follow its own control-vector

In den nächsten Kapiteln werden Reynolds drei Grundverhaltensregeln, cohasion, seperation und alignment, umformuliert. Aus jeder Verhaltensregel werden sogenannte Steuervektoren, $\vec{v}$, generiert, wie in Abbildung 1 gezeigt. Die Summe aller Steuervektoren ergibt den Gesamtsteuervektor, $\vec{V}$, in dessen Richtung sich Boids bewegen.

B. Cohasion

Der Steuervektor, der aus der Verhaltensregel der Kohäsion entsteht, wird $\vec{v}_c$ genannt. $\vec{v}_c$ ist im wesentlichem ein Vektor, der zum Schwerpunkt aller gefundenen Nachbarn zeigt. Jede Zelle ist ein eigener Thread und berechnet ihr eigenes Kohäsions-Steuersignal, $\vec{v}_c$. Um $\vec{v}_c$ zu berechnen, wird die Boid-Dichte-Textur, D, benötigt. D gibt an, wie viele Boids in einer Zelle vorhanden sind.

Gegeben sei eine Zelle, X, die sich am Ort $x = (x_u \quad x_v)$ befindet. Diese Zelle berechnet ein Kohäsions-Steuersignal für diesen Ort, $\vec{v}_c(x)$, mithilfe von N gegebenen Wahrnehmungsvektoren. Mit einem Wahrnehmungsvektor, $\vec{s}_i$, kann eine Stichprobe, D_i, aus D genommen werden.

$$D_i(x) = D(x + \vec{s}_i \cdot \frac{1}{R}) \tag{1}$$

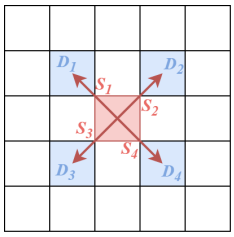

Fig. 2. Die rot markierte Zelle heißt X und befindet sich an Position x. Sie besitzt 4 Wahrnehmungsvektoren, $\vec{s}_i$. Mit ihnen ist die Zelle in der Lage Stichproben, D_i, aus der Boid-Textur, D, zu entnehmen.

Mithilfe der Stichproben, D_i, wird dann das Kohäsions-Steuersignal, $\vec{v}_c$, wie folgt berechnet:

$$\vec{v}_c(x) = \frac{\sum_{i=1}^{N}(\vec{s}_i \cdot D_i(x))}{\sum_{i=1}^{N} D_i(x)} \tag{2}$$

Im Zähler aus Gleichung 2, wird im Grunde nichts geringeres getan, als Vektoren von der Zelle X zu allen gefundenen Nachbarn zu spannen.

Zum Schluss wird der Mittelwert aus allen gespannten Vektoren gebildet, indem durch die Anzahl aller gefundenen Nachbarn, $\sum_{i=1}^{N} D_i$, geteilt wird.

C. Separation

Das Separations-Steuersignal, $\vec{v}_s$, arbeitet im Gegensatz zur Kohäsion mit Vektoren, die von den entdeckten Nachbarn weg und zum gesteuerten Boid hin zeigen. Die Größe des Steuervektors, $\vec{v}_s$, steigt umso mehr an, je näher ein Nachbar ist. Reynolds empfiehlt hierfür mit umgekehrter Proportionalität zu arbeiten [2]. So entsteht schlussendlich Formel 3:

$$\vec{v}_s(x) = \sum_{i=1}^{N}(-D_i(x) \cdot e(\vec{s}_i) \cdot \frac{1}{|\vec{s}_i|}) = -\sum_{i=1}^{N}(D_i(x) \cdot \frac{\vec{s}_i}{|\vec{s}_i|^2}) \tag{3}$$

D. Alignment

Das Steuersignal, $\vec{v}_a$, soll dafür sorgen, dass sich ein Boid der Bewegungsrichtung und Geschwindigkeit seiner Nachbarschaft anpasst. Diesmal ist also nicht nur die Menge, D, der Nachbarschaft von Interesse, sondern auch deren Bewegungsrichtungen, $\vec{V}$.

In einer Zelle können sich mehrere Boids aufhalten. Das Bedeutet, dass $\vec{V}$ den Steuervektor von mehreren Boids repräsentieren könnte. Dies wird bei der Berechnung von $\vec{v}_a$, in Formel 4, berücksichtigt.

$$\vec{v}_a(x) = \frac{\sum_{i=1}^{N}(\vec{V}_i(x) \cdot D_i(x))}{\sum_{i=1}^{N}(D_i(x))} - \vec{V}(x) \tag{4}$$

mit

$$\vec{V}_i(x) = \vec{V}(x + \vec{s}_i \cdot \frac{1}{R}) \tag{5}$$

Zuerst wird im Zähler (Gleichung 4) des ersten Terms die Summe aller Steuervektoren aus der Nachbarschaft gebildet. Danach wird diese Summe durch die Anzahl an gefundenen Steuersignalen geteilt. Die Anzahl der gefundenen Steuersignale entspricht der Anzahl der gefundenen Boids, $\sum_{i=1}^{N}(D_i(x))$.

Nachdem die durchschnittliche Bewegungsrichtung der Nachbarschaft ermittelt wurde, wird davon die eigene Bewegungsrichtung, $\vec{V}(x)$, abgezogen, so wie es auch in [2] getan wird.

E. rules of boid movement

In order to find a formula, that discripe how the boids moves through the cell-system, we define following rules:

1) Die Anzahl der Boids ist konstant:

$$\sum_{i=1}^{R}\sum_{j=1}^{R} D(i,j) = konstant \tag{6}$$

2) Ein Steuervektor entleert und befüllt dieselbe Menge Boids: Ein Steuervektor, $\vec{V}$, verursacht den Transport einer Menge Boids, D_V.

Die Menge D_V wird darauf hin auf N andere Zellen verteilt.

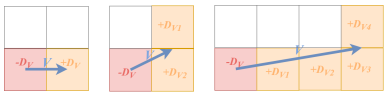

Fig. 3. Es lassen sich beliebig viele Verteilungsmethoden modellieren. Die rot markierte Zelle verteilt Boids in Richtung von V. Die Verteilende Zelle verliert dabei D_V Boids. Genau diese Menge wird dann in die orange markierten Zellen verteilt. Hierfür können unterschiedliche Verteilungsarten konzipiert werden.

Die Summe aller N Teilmengen, D_{Vi}, muss genau so groß sein, wie die verloren gegangene Menge D_V.

$$\sum_{i=1}^{N}(D_{V_i}) = D_V \tag{7}$$

Für die Verteilung von D_V können, so wie in Abbildung **??** vorgeschlagen, unterschiedliche Verteilungsmethoden eingesetzt werden. D_{Vi} kann einer einzigen Zelle überreicht oder auf mehrere Zellen verteilt werden.

3) Maximale Bewegungsreichweite eines Boids: Bei der Advektion von Texturen sollte ein Pixel pro Zeitschritt nicht weiter als eine Pixellänge bewegt werden. Ansonsten können Instabilitäten entstehen, wodurch sich die bewegte Textur, laut [4], explosionsartig aufblähen kann.

III. Evaluation

Der im Prototyp simulierte Schwarm kann durch Einsatz einfacher Zusatzverhaltensregeln von einem menschlichen Spieler gejagt oder verfolgt werden. Die Bewegungen des Schwarms können bei entsprechender Parametrisierung realistisch und lebendig gestaltet werden. Ohne mit zufällig generierten Steuervektoren arbeiten zu müssen, bewegen sich manche Schwärme scheinbar in zufällige Richtungen, als ob die virtuellen Lebewesen ein eigenes Ziel verfolgen würden.

Die Performance des implementierten Prototyps erwies sich, entgegen aller Erwartungen, als überraschend gut. Die Messungen aus Kapitel **??** lassen vermuten, dass sich das konzipierte Simulationsmodell durchaus in der Praxis anwenden lässt. Künftige Videospiele könnten von diesen interaktiven Schwarm-Wesen profitieren, zumal sich die entwickelte Simulation von herkömmlichen Shadern ausgeführt werden kann. Shader-Programme, die mithilfe der Unreal Engine 4 implementiert wurden, lassen sich ohne große Umstände auf einige Plattformen portieren.

Trotz allem ist das bestehende System durchaus verbesserungswürdig. Besonders negativ aufgefallen ist der in Kapitel **??** angesprochene Verlust von Boids. Diese Problematik konnte aus Zeitgründen nicht weiter untersucht

werden, weswegen noch keine konkrete Erklärung für das Verschwinden von Individuen existiert.

REFERENCES

[1] C. W. Reynolds, Flocks, herds, and schools: a distributed behavioral model, Computer Graphics.
[2] C. Reynolds, Steering behaviors for autonomous characters (1999).
[3] O. Tschesche, Parallele simulation und visualisierung von großen tierschwärmen mit opencl, Bachelorthesis.
[4] R. Fernando, GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics - Chapter 38, Pearson Higher Education, 2004.