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 separate Individuen. Stattdessen ist der Schwarm eine im Raum ungleichmäßig verteilte Menge. Der Schwarm wird daher als leuchtende Punkte-Wolke, von einer Textur dargestellt.

Die konzipierte Schwarmsimulation wird mittels Content-Driven Multipass Rendering prototypisch implementiert und in Echtzeit ausgeführt.

I. Einleitung2

Im Rahmen dieser Arbeit wird Reynolds Boid-Simulationsmodell so angepasst, dass alle für die Simulation nötigen Arbeitsschritte möglichst effizient von Shadern ausgeführt werden können.

Wegen der besonderen Funktionsweise von Shadern können manche Arbeitsschritte aus Reynolds Simulationsmodell nicht auf direktem Wege übernommen und implementiert werden. Daher werden manche Gesetzmäßigkeiten aus dem gegebenen Boid-Modell uminterpretiert und umformuliert. Bei Umformulierungen wird stets darauf geachtet, die grundlegenden Prinzipien des Boid-Systems möglichst unverändert zu lassen

II. Related work

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

Für gewöhnlich wird ein Boid-System wie ein Partikelsystem behandelt [1], so wie in [3]. In dieser Arbeit wird der Schwarm hingegen mithilfe eines Zellensystems beschrieben, in dem jede Zelle örtlich gebunden ist. Jede Zelle kann dabei von mehreren Individuen besucht werden.

Reynolds Boid-System wird, wie in [3], meist mithilfe von Grafikkartenschnittstellen wie *OpenCV* oder *CUDA* berechnet. In dieser Arbeit wird statt dessen auf die Verwendung von herkömmlichen Shadern gesetzt.

III. 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 hinein schreiben.

Aufgrund dieser Einschränkung verabschieden wir uns an dieser Stelle vom Gedanken des Partikelsystems. Stattdessen wird das Boidsystem von einem Zellen-System beschrieben, wie bei shaderbasierten Flüssigkeitsströmungssimulationen [4] auch.

Das bedeutet, dass verwendete Texturen kein Partikelsystem, sondern ein Zellensystem repräsentieren. Ein Pixel aus einer Textur beschreibt kein umherfliegendes Boid, sondern eine Zelle, die eine reelle Menge von Boids, D_i, enthalten kann. Die Individuen des Schwarms bewegen sich dabei durch nebeneinanderliegende Zellen hindurch.

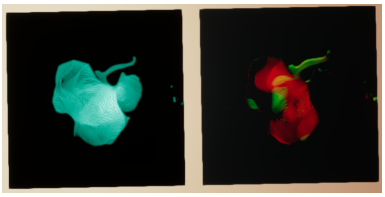

Fig. 1. Left: A texture, that represents the boid-density, D. Each pixel is a real skalar value that represents the amount of boids, contained in this cell. Right: A texture, called V, that holds the control-vectors, v, for each cell. Each boid will follow its own control-vector

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

Das Steuersignal, das aus der Verhaltensregel der Kohäsion resultiert, wird $\vec{v}_c$ genannt. $\vec{v}_c$ ist vereinfacht gesagt 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.

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

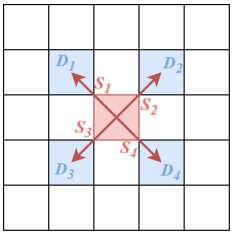

Fig. 2. Die rot markierte Zelle heißt X und befindet sich an Position x. Sie besitzt 4 Wahrnehmungsvektoren, $\vec{s}_i$. Mit ihnen ist die Zelle in der Lage Stichproben, D_i, aus der Boid-Textur, D, zu entnehmen.

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:

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

Auch hier werden zuerst Vektoren zu allen Nachbarn gespannt, indem das Produkt aus D_i und $\vec{s}_i$ gebildet wird. Weil in diesem Fall die Größe der gespannten Vektoren aber noch gesondert behandelt werden muss, werden sie vorerst normiert. So ergibt sich $D_i \cdot e(\vec{s}_i)$. Natürlich werden alle Einheitsvektoren noch umgedreht, damit abstoßende Kräfte entstehen. All diese abstoßenden Einheitsvektoren werden dann mit $1/|\vec{s}_i|$ gewichtet, so wie von Reynolds in [2] empfohlen.

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: Beim Transport der Boids sollte stets auf die Einhaltung des folgenden Gesetzes geachtet werden:

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

Formel 6 legt fest, dass die Anzahl aller vorhandenen Boids konstant bleiben muss. Die Anzahl soll im Verlaufe der Simulation weder sinken noch fallen. Beim Boid-Transport muss darauf stets geachtet werden.

2) Ein Steuervektor entleert und befüllt dieselbe Menge Boids: Gegeben sei eine Zelle X bei x. Sie besitzt einen Steuervektor $\vec{V}$ und D Boids.

Der Steuervektor, $\vec{V}$, verursacht den Transport einer Menge Boids, D_V, wobei
$D_V <= D$ (damit Gesetz 6 eingehalten wird).

Durch den Transport verliert Zelle X D_V Boids, während genau dieselbe Menge in ihrer Umgebung verteilt wird. Die Menge D_V wird auf N andere Zellen verteilt, diese Zellen werden als *besuchte* Zellen bezeichnet. Jede besuchte Zelle bekommt eine Teilmenge von D_V, namens D_{Vi} gutgeschrieben.

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.

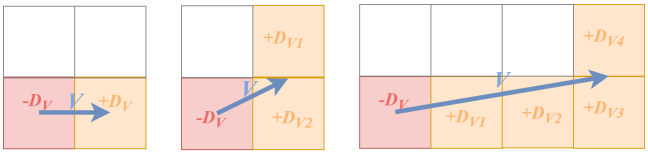

Fig. 3. Es werden drei unterschiedliche Verteilungsarten gezeigt. Die rot markierte Zelle verteilt Boids in Richtung von V. Die Verteilende Zelle verliert dabei D_V Boids. Genau diese Menge wird dann in die orange markierten Zellen verteilt. Hierfür können unterschiedliche Verteilungsarten konzipiert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

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.

Daher sollte bei der Simulation darauf geachtet werden, dass ein Boid pro Simulationsschritt keine Zelle überspringt. Diese Regel sollte bei der Konzipierung von Verteilungsmethoden berücksichtigt werden. Verteilungsmethoden wie die Methode, die in Abbildung **??** ganz rechts zu sehen ist, sollten daher vermieden werden.

F. Boid-movement formula

Nachdem in Kapitel **??** festgelegt wurde, nach welchen Gesetzen und Formeln sich der Schwarm im Zellensystem verteilen soll, kann eine allgemein gültige Formel für den Transport von D aufgestellt werden.

Grundsätzlich geben Zellen pro Simulationsschritt Boids an Nachbarzellen ab und nehmen neue Boids von ihren Nachbarn auf. Dieser Prozess kann über Formel 8 ausgedrückt werden:

$$D(x, t + dt) = D(x, t) - D_{leave}(x, t) + D_{enter}(x, t) \quad (8)$$

In Gleichung 8 aktualisiert eine Zelle, die sich bei x befindet, ihre Boids. Hierfür wird $D_{leave}(x, t)$ von $D(x, t)$ abgezogen und $D_{enter}(x, t)$ zu $D(x, t)$ addiert.

Zunächst wird es darum gehen, den bis jetzt unbekannten Termen D_{leave} und D_{enter} näher zu kommen.

Gegeben sei eine Zelle X am Ort x. Sie besitzt $D(x)$ Boids. Außerdem besitzt sie einen Steuervektor, $\vec{V}(x)$, für ihre Boids. $\vec{V}(x)$ repräsentiert den derzeitigen Bewegungswunsch der in X enthaltenen Boids. Im Steuervektor ist auch die Information enthalten, wie viele Boids aus der Zelle fließen werden, $D_{leave}(x)$. Die Länge des Steuervektors stellt dabei die Dringlichkeit des Bewegungswunsches dar. Ist $\vec{V}(x)$ groß, haben es die Boids im übertragendem Sinne eilig.

$$D_{leave}(x) \sim |\vec{V(x)}| \quad (9)$$

Außerdem ist $D_{leave}(x)$ neben $|\vec{V(x)}|$ auch abhängig von $D(x)$. $D_{leave}(x)$ ist schließlich eine Teilmenge von $D(x)$, wobei $D_{leave}(x) <= D(x)$.

Daraus ergibt sich schlussendlich die finale Formel für $D_{leave}(x)$:

$$D_{leave}(x) = |\vec{V(x)}| \cdot D(x) \quad (10)$$

Außerdem muss dabei stets Gleichung 11 erfüllt sein, damit $D_{leave}(x) <= D(x)$ gilt.

$$|\vec{V(x)}| <= 1 \quad (11)$$

Bei der Generierung der Steuersignale sollte also stets darauf geachtet werden, das ein Steuervektor niemals größer als 1 wird.

An dieser Stelle sollte die folgende Erkenntnis nicht fehlen: Die transportierte Menge, D_V, ist genauso groß wie die verlorene Menge, D_{leave}.

$$D_{leave}(x) = D_V(x) \quad (12)$$

Wie D_{Enter} berechnet wird, hängt davon ab, nach welchem Verteilungsgesetz die transportierte Menge, D_V, verteilt wird.

Ein besonders einfaches Verteilungsgesetz ist die Verteilung der Menge D_V in eine Nachbarzelle. Diese Methode bringt dabei den Vorteil eines geringen Rechenaufwands mit sich.

Eine Zelle, X, muss all Ihre direkten Nachbarn durchsuchen, um in Erfahrung zu bringen, ob Steuervektoren aus ihrer Nachbarschaft auf sie selbst zeigen. Die Nachbarsuche kann mithilfe von unterschiedlich orientierten Testvektoren, $\vec{T_i}$, erfolgen, so wie in Abbildung **??** dargestellt.

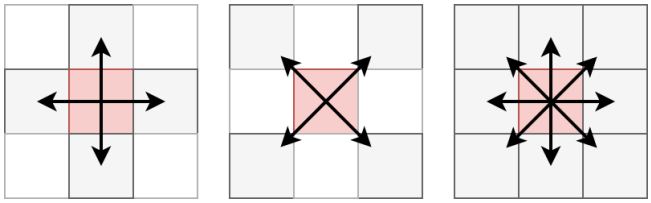

Fig. 4. Die rot eingefärbte Zelle untersucht ihre Nachbarschaft mit N_c Testvektoren, T_i. Die Testvektoren repräsentieren auch die möglichen Bewegungsrichtungen die ein Boid einschlagen kann. Wenn nur 4 Testvektoren verwendet werden, heißt das auch, dass sich die Boids in nur 4 unterschiedliche Richtungen bewegen können.

Stellt sich heraus, dass ein benachbarter Steuervektor auf Zelle X zeigt, nimmt Sie D_{Enter_i} Zellen vom Nachbar auf.

$$D_{Enter_i} = D_{Leave}(x + \frac{\vec{T_i}}{R}) = D(x + \frac{\vec{T_i}}{R}) \cdot |\vec{V}(x + \frac{\vec{T_i}}{R})| \quad (13)$$

Nachdem Zelle X all ihre N_c Nachbarn untersucht hat, sind n_c Nachbarzellen bekannt, die ihre Boids an X abgeben. Es gilt $0 <= n_c <= N_c$. Aus jedem, der n_c Nachbarn, entsteht ein D_{Enter_i}. Zelle X nimmt dann insgesamt $D_{Enter}(x)$ Boids aus ihrer Nachbarschaft auf:

$$D_{Enter}(x) = \sum_{i=1}^{n_c} (D_{Enter_i}) \quad (14)$$

IV. 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.