# Real time boid-simulations with compute shading

Maximilian Legnar and Prof. Dr. Rasenat

***Abstract*— Based on Reynolds boid system [1], a model for the simulation of animal swarms is being developed.**

**Instead of treating the simulated swarm as a particle system, Reynold's Boid model is transferred to a cellular system akin to cellular automata. This adjustment is made so that the simulation can be executed as efficiently as possible by shader programs.**

**The simulation model does not divide the swarm into individuals. Instead, the swarm is an unevenly distributed amount in space. The swarm is therefore depicted as a glowing point cloud.**

**The designed swarm simulation is prototypically implemented using content-driven multipass rendering. The prototype was successfully tested for real-time capability.**

## I. Introduction and related work

In order to simulate the movements of natural animal swarms, the so-called boid model was established by Reynolds.

In this work, the fundamental works of Reynolds Boid model [1] [2] are taken up to model a novel system that is particularly well described by shaders and textures.

Usually Reynolds Boid system is treated like a particle system [1] [3]. In this work, however, the swarm is described by a cellular system in which each cell is locally bound. Each cell can be visited by several individuals. This reinterpretation is done so that the simulation can be performed efficiently by shader programs.

## II. Method

### A. The boid system as a cell system

The boid system is related to the particle system [1] and therefore can usually be treated as such.

An essential component of a particle system is the movement, ie the positional change, of particles. However, a shader cannot easily move a pixel. This is because a pixel thread is unable to write values to other pixels. Each pixel can write its calculation result only in itself.

Because of this restriction, we say goodbye to the idea of the particle system at this point. Instead, the boid system is described by a cell system. Because shader programs are optimized for processing textures, this cell system is described using textures. A pixel from a texture represents a cell. The individuals of the simulated swarm (Boids) move through adjacent cells. Each cell can have a real set of boids, $D_i >= 0$.

The following chapters reformulate Reynold's three basic behavioral rules, cohasion, separation, and alignment. From each behavior rule, so-called control vectors, $\vec{v}$, are generated, as shown in Figure 1. The sum of all control vectors gives the total control vector, $\vec{V}$, in the direction of moving boids.

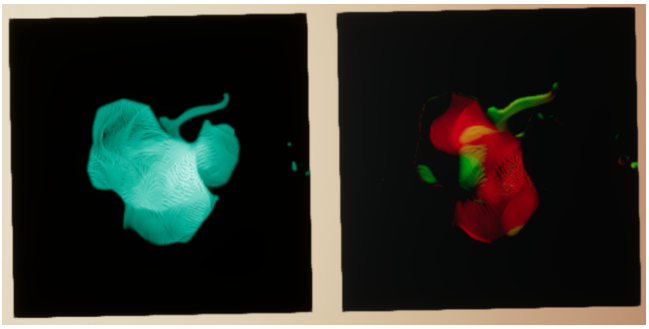

Fig. 1. Left: A texture, that represents the boid-density, $D$. Each pixel is a real skalar value that represents the amount of boids, contained in this cell. Right: A texture, called $V$, that holds the control-vectors, $v$, for each cell. Each boid will follow its own control-vector

### B. Cohasion

The control vector, which results from the behavioural rule of cohesion, is called $\vec{v}_c$. $\vec{v}_c$ is essentially a vector that points to the center of gravity of all found neighbours. Each cell is a separate thread and calculates its own cohesion control vector, $\vec{v}_c$. To calculate $\vec{v}_c$, the boid density texture, $D$, is needed. $D$ indicates how many boids are present in a cell.

Given a cell, $X$, which is located at $x = (x_u \quad x_v)$. This cell calculates a cohesion control signal for this location,$\vec{v}_c(x)$, using $N$ given perceptual vectors. With a perceptual vector, $\vec{s}_i$, a sample, $D_i$, can be taken from $D$.

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

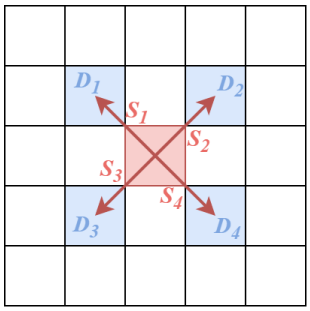


Fig. 2. This is how boids, which are inside a cell, sees the virtual world: The red marked cell is called $X$ and is located at position $x$. It has 4 perceptual vectors, $\vec{s}_i$. With them, the cell is able to take samples, $D_i$, from the boid texture, $D$.

Using the samples, $D_i$, the cohesion control signal, $\vec{v}_c$, is then calculated as follows:

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

In the numerator of Equation 2, basically nothing less is done than to clamp vectors from cell $X$ to all found neighbours.

Finally, the mean of all the stretched vectors is formed by dividing by the number of all neighbours found, $\sum_{i=1}^{N} D_i$.

### C. Separation

The separation control vectors, $\vec{v}_s$, works in contrast to cohesion with vectors pointing away from the discovered neighbours and towards the controlled boid. The size of the control vector, $\vec{v}_s$, increases the closer a neighbour is. Reynolds recommends working with reverse proportionality [2]. This is how Formula 3 finally emerges:

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

### D. Alignment

The control signal, $\vec{v}_a$, is intended to ensure that a boid adapts to the direction of movement and speed of its neighbourhood. This time not only the amount, $D$, of the neighbourhood is of interest, but also their directions of movement, $\vec{V}$.

A cell can contain several boids. This means that $\vec{V}$ could represent the control vector of several boids. This is considered in the calculation of $\vec{v}_a$, in formula 4.

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

with

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

First, in the numerator (equation 4) of the first term, the sum of all the control vectors from the neighbourhood is formed.

Thereafter, this sum is divided by the number of control signals found. The number of detected control signals corresponds to the number of found boids, $\sum_{i=1}^{N}(D_i(x))$.

After the average direction of motion of the neighbourhood has been determined, the own direction of movement, $\vec{V}(x)$, is subtracted from this, as it is done in [2].

### E. rules of boid movement

After calculating $V$, the swarm must be moved in the direction of the given control vectors. How a group of boids, $D_i$, will distribute through the cell system, is modeled with following rules:

*1) The number of boids is constant:*

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

Formel 6 specifies that the amount of all existing boids must remain constant.

*2) A control vector empties and fills the same amount of boids:* A control vector, $\vec{V}$, causes the transport of a lot of boids, $D_V$. The set $D_V$ is then distributed to $N$ other cells.

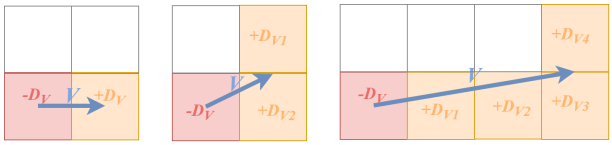


Fig. 3. Any number of distribution methods can be modeled. The red marked cell distributes boids in the direction of $\vec{V}$. The distributing cell loses $D_V$ Boids. Exactly this amount is then distributed into the orange labeled cells. For this, different types of distribution can be designed.

The sum of all $N$ subsets, $D_{Vi}$, must be exactly the same as the lost quantity $D_V$.

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

For the distribution of $D_V$ different distribution methods can be used, as suggested in Figure 3. $D_{V_i}$ can be passed to a single cell or distributed to multiple cells.

*3) Maximum range of movement of a boid:* In the advection of textures, one pixel per time step should not be moved more than one pixel. Otherwise, instabilities may occur, causing the moving texture to expand, according to [4].

## III. Evaluation

The swarm simulated in the prototype can be hunted or tracked by a human player using simple additional rules. The movements of the swarm can be made realistic and vivid with appropriate parameterization. Without having to work with randomly generated control vectors, some swarms seem to be moving in random directions, as if the virtual beings were following their own target.

The performance of the implemented prototype proved to be surprisingly good. Performance measurements (on a GTX 1050-TI) suggest that the designed simulation model can be applied in practice. Future video games could benefit from these interactive swarm creatures, especially as the developed simulation of traditional shaders can be performed. Shader programs implemented using the Unreal Engine 4 can be easily ported to some platforms.

Particularly notable was the loss of boids. For reasons of time, this problem could not be further investigated, which is why there is no concrete explanation for the disappearance of individuals.

## 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.