Swarm Intelligence

Himanshu Masand
University of Pennsylvania

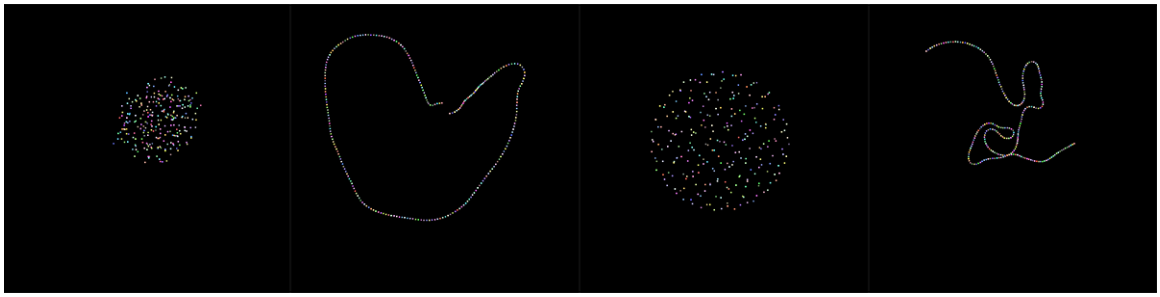

Figure 1: *Swarm intelligent system as a flock (simulating a school of fish or birds), as a line (simulating ants or ducklings) or having a spherical formation*

Abstract

Swarm intelligence is the collective behavior of decentralized, self-organized systems. Such systems are composed of homogenous individuals. The behavior of each individual in a swarm depends on its local perception of its neighborhood. Simulating such a multi-agent system is computationally expensive because each agent must consider all of the others, if only to identify its neighbors. For large systems, simple implementations are too slow since computation grows as the square of agent population. GPUs can perform all the computation in parallel and can therefore give a tremendous performance boost. This paper describes an implementation using the compute shaders of OpenGL 4.3 to utilize the parallelism of the GPU to simulate simple behaviors in swarm intelligent systems.

CR Categories:
I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—Intelligent agents
1.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: flocking, self-organization, behavioral animation, particle system, gpu

Links: DL PDF

1 Introduction

Swarm intelligence is a widely observed phenomenon in animal societies. Flocks of birds, school of fish or even ants walking in a single file are examples of self-organized swarm systems. Such systems are composed of simple homogenous individuals interacting locally with one another and with their environment. Swarms in natural systems have long been studied in biology. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local interactions between such agents lead to the emergence of intelligent global behavior, unknown to the individual agents.

Craig Reynolds[Reynolds 1987] was the first person to simulate flocking in an artificial swarm system. Since then, there have been numerous implementations of his work on the CPU. However, CPU implementations are slow and therefore limit the number of agents that can be simulated. GPUs on the other hand provide highly parallel methods which result in greater speeds and simulations with very large number of agents.

With the release of OpenGL 4.3, many new features were released. One of them is the compute shaders. OpenGL was lacking compute shaders and even for very light computation, developers had to switch to OpenCL. With the introduction of compute shaders in OpenGL, it is now possible to do computation in OpenGL itself. This paper describes how to use compute shaders to implement a particle system exhibiting Reynolds behaviors.

2 Related Work

Craig Reynolds developed a very simple and elegant model for steering autonomous agents[Reynolds 1987] which was independent of the characters means of locomotion. His paper described behaviors on three levels classified according to complexity. Using the simple individual behaviors like seek and flee, and combining them with group behaviors like cohesion, separation and alignment resulted in complex behaviors like flocking, leader following etc. These algorithms were used to implement the swarm intelligent system.

3 Implementation

The particle system consists of the current state and the next state of each particle. A state consists of the position and velocity of the particle. Therefore, for each particle, the new position and velocity is calculated in every frame. All the computation is done in the compute shader.

The new velocities are computed using the algorithms described by Craig Reynolds[Reynolds 1987]. The cohesion behavior is implemented by first calculating the center of mass of the particles by averaging their current positions using equation (1).

$$X_{cm} = \frac{\sum x_i}{\sum i} \tag{1}$$

The cohesion velocity is then calculated as a vector from the particles current position (x) to the center of mass of all particles Xcm

$$V_{cohesion} = X_{cm} - x \tag{2}$$

The alignment velocity is calculated as the average velocity of all particles:

$$V_{align} = \frac{\sum v_i}{\sum i} \tag{3}$$

The separation velocity is calculated as the sum of the inverse of the separation distances di

$$V_{separate} = \sum \frac{d_i}{||d_i||^2} \tag{4}$$

The flocking behavior is created as a weighted sum of separation, cohesion and alignment velocities. In the line following behavior, each particle seeks the previous particle. The velocities are computed using the equation (5)

$$V_i = P_{i-1} - P_i \tag{5}$$

The first particle simply follows the mouse position. This also results in a leader following behavior where the mouse pointer is the leader. The swarm can be constrained to a spherical formation by balancing the cohesion and separation velocities to reach equilibrium. The radius of the formation can be adjusted by changing the weights of separation and cohesion components.

The velocities calculated using the above equations are then used to determine the next position of the particle. Instead of being calculated in the shader, the group velocities like cohesion, alignment and separation are being calculated on the CPU and then saved in a shader storage buffer object (introduced in OpenGL 4.3). The advantage of using the shader storage buffer object is that it can read and write arbitrary data in the same C-like way as they were created and can also treat parts of the buffer as an array of structures[Bailey 2012]. The final velocities for flocking, line following and spherical formation are then computed in the compute shader. The final position of each particle in the current frame is saved in the position buffer. The OpenGL rendering draws the particles by reading this buffer.

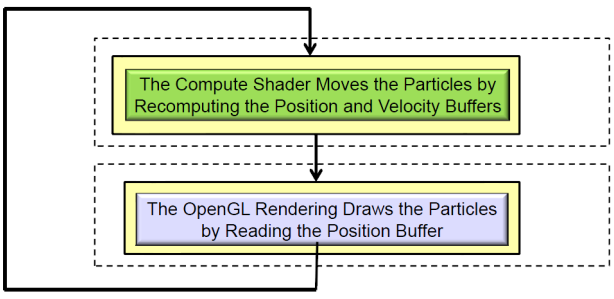

Figure 2: *Interaction between the compute shader and rendering* [Bailey 2012]

4 Results

The system provides three kinds of swarm behaviors. These can be seen in Figure 1. The first behavior simulates a school of fish or birds where they move in a flock. The second behavior simulates ants or ducklings where each individual follows another individual with one being the leader. The third behavior simulates a spherical formation while following a leader (mouse pointer).

The next important result is the performance of the system. As mentioned above, even simple implementations for large systems are too slow since computation grows as the square of agent population. Therefore, the GPU implementation should provide a huge performance improvement. As expected, the implementation resulted in some very good performance numbers.

All the simulations were run on an NVIDIA GeForce GTX660M with a resolution of 1000x1000. The simulations ran at interactive speed even with a few million particles. Increasing the number of particles to more than 12 million resulted in visible lag. The system was able to handle a maximum of 32 million particles with a workgroup size of 1024. The speed at which this simulation ran was approximately 400 million particles per second. The maximum speed attained was 650 million particles per second when the simulation consisted of 4 million particles with a workgroup size of 128.

5 Future Work

Although the system gives a tremendous performance, a lot of work can still be done on the rendering side. Currently, the system renders the particles as points. This can be enhanced by supporting objects or at least spheres with velocity vectors (as lines). Another extension could be adding collision avoidance to the system. The avoidance velocity can be computed using the Reynold's model and added to the already existing velocity to avoid obstacles.

The system can also be further optimized to result in improved performance.

6 Conclusion

This paper has described a method to create different types of swarm behaviors using the compute shaders of OpenGL 4.3. The system is capable of simulating swarms of fish, birds, ants and ducks. In addition to it, the system also enables geometric formations like spheres. It has also produced expected performance improvement over the CPU.

References

BAILEY, M., 2012. Opengl compute shaders. ACM SIGGRAPH 2012, August.

REYNOLDS, C. 1987. Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH 87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM, 25–34.