Multiple Agents Orchestration

A Swarm Intelligence instrument for the Computational Artist

Dimitrios Kyriakoudis[1]

Abstract. Swarm Intelligence systems exhibit seemingly intelligent behavior. This property emerges as a result of the communication and interactions between many small, simple agents. However, in order for such systems to produce useful results, the problem at hand has to be clear, well-defined, and numerically expressible in a way that allows all potential solutions to be assigned a "goodness" score. This approach is ill-equipped to deal with the subjective and arbitrary nature of creative practices. This paper proposes an alternative approach to the usefulness of such systems in the form of *Multiple Agents Orchestration*, an algorithm geared towards the needs of digital creative practices, presented through two example applications in computer music and graphics.

1 INTRODUCTION

Digital technology has introduced artists to a tool of immeasurable potential and flexibility. Through the digital representation of information, one can break free from the limitations imposed by analog tools, such as acoustic musical instruments or drawing utensils, and work at a level of abstraction before unattainable: that of the pure data that give rise to the perceptual phenomena such as sound or image.

Instead of composing with the sound of different acoustic or electric instruments, all of which have well-defined sonic characteristics and responses to different playing styles, a musician can now compose with sound itself, synthesizing it from the ground up and manipulating it over time. Modern processing power is capable of supporting thousands of synthesis processes running live, in parallel, allowing for the design of rich, evolving, and customizable sonic instruments. Similarly, the visual artist can now work at the particle level, exercising control over millions of visual elements from which a perceptual entity emerges, exhibiting characteristics that are greater than the sum of its individual parts.

However, this freedom also introduces a challenge for artists, namely that of finding meaningful and useful higher-level abstractions that help one make sense of, and reason about, the arrangement of the lower-level data. Within the digital realm of pure information, one can no longer creatively depend on the built-in sonic richness and expressive capabilities of acoustic instruments. Instead, the potential for such expressivity and the interface for its control has to be accounted for in the instrument's design and implementation.

This becomes especially problematic when one is looking to control multiple, similar audiovisual synthesis processes that run in parallel. For example, in a traditional orchestra's violin section, the composer has control over many musicians playing the same instrument. Each performer's playing style and sound is unique, following the instructions on the score but ever-so-slightly differing in timbre, tempo, dynamics etc. The composer can then work with the amount of such differentiation, asking each performer to individualise their playing style to a greater or lesser extend, such as in the case of free-form vibrato or tremolo, two techniques that are almost impossible to precisely coordinate between multiple performers. This controllable convergence of individual sonic processes gives rise to the perceptually unified sound of the whole section and allows for a level of expressive control and flexibility that is unattainable through using single instruments. When digitally synthesizing sound, however, it becomes much harder to produce such subtle but effective differentiation. Due to the precise nature of digital computation, many simultaneous iterations of the same synthesis process are going to yield the exact same sound as a single one, only louder. If one desires to differentiate the parameters of such processes, then one has to resort to using either scaled randomness, or simple geometric functions such as sine waves and phasors. However, neither of these offer variation in a manner that is controllable, coherent, and organically evolving. Thus, the digital artist is confined to working with the unpredictability and limited control of randomness, or the repetitiveness and equally limited control of simple geometric functions.

Swarm Intelligence algorithms are a computational paradigm based on such interactions between simple, like-minded agents. They draw their inspiration from the way various species of animals and insects organize themselves in nature, giving rise to a total that exhibits properties that are greater than the sum of its individual parts. More importantly, they offer a small set of parameters that affects the behaviour of each individual agent and, as a consequence, the emergent behaviour of the whole. Therefore, with a few tweaks and a slightly different approach to their application, they can be used to simply and effectively control a large number of individual processes, with a few parameters at hand that can drastically change the swarm's behaviour and, thus, the resulting artistic outcome of this hybrid human-machine creative system.

2 DISPERSIVE FLIES OPTIMISATION

Dispersive Flies Optimisation (DFO), introduced in 2014 by M. Majid al-Rifaie [1], is a Swarm Intelligence algorithm in which a number of simple, independent agents traverse an n-dimensional space in search for an optimum solution to a search question. Each agent holds an n-dimensional vector, called the agent's *solution*, where n is the number of elements that have to be specified for a candidate solution to be considered complete. By passing this solution vector into the fitness function, each agent gets assigned a *fitness*, a numerical representation of how suitable its solution is for the problem at hand.

The algorithm draws its inspiration from the social swarming of various species of flies. The behaviour of such swarms can be broken

[1] Department of Computing, Goldsmiths, University of London, London SE14 6NW, UK., email: dkyri001@gold.ac.uk

down to two core components: the convergence of individual agents towards points of interest in space, such as food, and the scattering away from incoming threats.

When an agent's position reveals something promising in regards to the swarm's goal, the rest of the swarm converges on and around that position, looking to exploit that agent's findings in hopes of improving them further. If any of those agents come upon something better during this phase of exploitation, then the swarm's focus shifts and a new point of convergence gets established. As such, the swarm eventually reaches the local optimum. However, this exploitative behaviour means that the swarm can easily get distracted by local optima, thus missing the opportunity to discover the space's global best. This is where dispersion becomes useful: by introducing threats to the swarm's formation one can encourage the agents to break their social formation and randomly explore other, unexplored parts of the space that might be leading to the global, instead of local, optimum position.

The DFO uses these concepts of convergence and dispersion to tackle a fundamental issue in heuristic sciences, namely that of exploration versus exploitation. The former is necessary to gain a broad overview of the search space at hand and determine which areas are the most promising, whereas the latter is essential for narrowing down on a solution and fine tuning it to achieve maximum results. Balancing both is key for a fast, efficient, and effective algorithm, but can be hard to achieve in practice. The DFO attempts to balance these by use of the disturbance threshold, a probability that each individual's position in the search space will be disturbed during the update process.

The algorithm's implementation is straightforward and can be split in two main parts: an initialisation phase and a looping update phase.

2.1 Initialisation

During the initialisation phase, each agent gets assigned a random position for each dimension, following uniform distribution within the bounds of the search space. This serves in spreading the agents as broadly as possible, maximising the amount of space exploration in the early stages of the algorithm's running.

2.2 Update

Once the agents have been assigned a random position, the update method starts looping until either a predetermined end condition has been met, or the algorithm has run for the maximum number of iterations. Each update phase has three stages, repeated for each agent: assigning a fitness score, calculating the probability for disturbance, and, if that probability falls outside the threshold, updating the agent's position.

2.2.1 Fitness Function

The fitness function's purpose is to assign a numerical score to each agent's solution vector. This can be as simple as calculating the Euclidean distance between the solution vector and the target (or goal) vector, or otherwise incrementing or decrementing the score for each desirable or undesirable feature present in the solution vector respectively.

2.2.2 Disturbance

For each agent, a random probability for disturbance gets calculated each time through the update loop. If that probability falls within the predetermined disturbance threshold, then that agent's solution vector gets replaced with one randomly positioned within the search space. This serves as an occasional encouragement for further exploration of the search space, counteracting the swarm's overall convergent tendency.

2.2.3 Update Equation

If the agent was not disturbed in the previous stage, it will now get updated according to the positions of its best-rated peers, following the update equation:

$$x_d^t = n_d^{t-1} + U(0,1) \times (b_d^{t-1} - x_d^{t-1}) \tag{1}$$

where x is the agent being updated, t is the current iteration, d is the dimension being updated, and n and b the agent's best-scoring neighbour and the swarm's best-scoring individual respectively.

3 THE PROBLEM OF CREATING

In the context of Computational Art practices, creation can be thought of as the arbitrary arrangement of low-level data in such a way that the higher-level artwork emerges. Since all digital computation is based on the binary encoding and representation of information, every piece of music, image, or animation has to essentially be put together as a string of ones and zeroes.

In audio, this string stores a representation of the waveform in the form of levels of air pressure over time, sampled at discrete time intervals. These samples are floating point number ranging from -1 to 1, representing normalized bipolar coordinates for the speaker's cone. During playback, this stream of samples get translated to voltages that move the cone back and for. A standard, CD quality audio file contains 44,100 such samples for every second of audio in a single channel. For a 3 minute song recorded in stereo, that amounts to almost 16 million samples, while for an hour-long symphony the number is close to 320 million.

For graphics and images, the binary string holds information about the colour of each pixel that makes up the image. For a colour image, three values representing the amounts of red, green, and blue, as well as a value for the transparency, have to be specified for each pixel. For a high-definition image with a resolution of 1920$\times$1080, that amounts to almost 8.3 million values, while for a 1 minute long video of the same resolution, assuming 60 frames per second, the total comes close to 30 billion.

This poses a fundamental challenge for the computational artist: finding higher-level abstractions that allow one to control and arrange the lower-level data in a way that is meaningful and useful to one's practice.

3.1 Tension and Release

As seen earlier, the DFO's core mechanism is the interplay between convergence towards points of interest in space, and dispersion away from threats. Looking beyond the problem-loving capacities of this interaction, a direct link between the algorithm's functionality and the concept of tension and release in the context of art can be drawn. Therefore, with the right modifications, the DFO can pose a powerful higher-level controller that can heavily affect the low-level data using a few, well-defined and effective parameters.

4 MULTIPLE AGENTS ORCHESTRATION

Inspired by the DFO's ability to provide a convenient control for tension and release for a multitude of similar, multi-dimensional agents, Multiple Agents Orchestration attempts to use Swarm Intelligence to provide useful higher-level abstractions that allow one to control vast amounts of parallel processes with great flexibility and variation. It is envisioned as an instrument of digital creativity, rather than a problem-solving system, and as such takes a lot of liberties in its implementation that may be computationally inefficient but result in greater expressibility and customization.

4.1 Implementation

The prototype implementation of this algorithm, called multipleAgents, is a digital instrument for the audiovisual computational artist and performer. It has the form of a C++ library that provides an API for real-time manipulation of the different parameters that control the agents' behaviour, and provide normalised outputs that can be mapped to any number of parameters and ranges.

As the name suggests, it is designed to guide and control the behavior of a multitude of simple, similar agents, that can control any number of parameters for any sort of process, whether audio or visual. For example, each agent can represent an audio oscillator, with the different dimensions being mapped to parameters such as amplitude, frequency, panning in a stereo field etc. Similarly, each agent could represent a particle in a greater system, with the dimensions controlling parameters such as position, velocity, colour, or shape.

The resulting swarm and its emergent features can be then controlled in real time, either by changing the target vector, the parameters such as the disturbance thresholds and amounts, or any combination of both.

1. Simultaneous control of many, multi-dimensional processes.
2. Real-time adjustable number of agents and dimensions.
3. Separate disturbance thresholds, amounts, and distribution exponents for each dimension.
4. Separate update amounts and distribution exponents for each dimension.
5. Ability to instantly reset all agents to a given spot.
6. Linear interpolation between consecutive positions for every agent.
7. Choice of every how many frames the positions will be updated, interpolating in between.
8. Choice of many times the update method will be run every time before updating the positions.

4.1.1 Setup

During the setup phase, the number of agents and dimensions is specified by the user and the Swarm's parameters are initialised with default values. Since we are not necessarily looking to explore as much of the search space as possible, and since volume may be one of the parameters being controlled, each agent's solution vector is initialised with zeroes, rather than with random values within the search space. However, there is the ability to instantly reset the whole Swarm to a certain value for all dimensions, or a certain configuration of values.

4.1.2 Disturbance

During each update loop, there is a choice between using a single disturbance threshold to determine whether all of an agent's dimensions will be disturbed, or separate thresholds that allow for more independence between the different dimensions. In addition, the amount of disturbance for each dimension can be changed individually, expressed as a percentage of the normalised search space. Finally, the distribution of this disturbance can be curved up or down by means of an exponent.

During the looping update phase, a random number following uniform distribution is generated for each dimension of each agent. If that number falls outside the disturbance threshold for that dimension, then the agent's position for that dimension will remain unchanged. Otherwise, an offset position will be calculated according to the below equation:

$$x_d^t = x_d^{t-1} + U(0,1)^{exp_d} \times amt_d \times C(-1,1) \qquad (2)$$

where amt_d is the disturbance amount, between 0 and 1, for dimension d, exp_d is the disturbance exponent, $U(0,1)$ is a random number between 0 and 1 following uniform distribution, and $C(-1,1)$ is a coin toss between -1 and 1, effectively picking randomly between adding a positive or negative offset to the agent's position.

Since the agents operate in normalised space, ranging from 0 to 1, exponent values larger than 1 will curve the distribution of disturbances downwards, while exponent values less than 1 will have the opposite effect. This provides a simple and effective control for shaping the agents' dispersive behaviour in each dimension separately, allowing for rough control of each synthesis parameter's variance over space or time.

4.1.3 Fitness Function

Since the focus of Multiple Agents Orchestration is the control and shaping of the swarm's behaviour over consecutive iterations, rather than the search for an optimum solution to a well-defined numerical problem, a single, simple fitness function can be generalised for all potential applications. In order to tell how well each agent is following the specified target at any given moment, it is sufficient to calculate the Euclidean distance between one's n-dimensional position and the n-dimensional target vector. Therefore, the fitness function can be expressed as:

$$f_i^t = \sum_{d=1}^{n} \sqrt{(g_d^t - v_{id}^t)^2} \qquad (3)$$

where f_i^t is the fitness of agent i at a moment t, g_d^t is the user-specified goal value for dimension d at that moment, v_{id}^t is that agent's current value in that dimension, and n is the number of dimensions, or synthesis parameters, that each agent is controlling.

The larger the total distance between an agent's position vector and the goal vector, the larger that agent's fitness score will be. In order to find the agent closest to the desired configuration of parameters, then, it becomes a simple minimisation problem, where an n-dimensional Euclidean distance of 0 means that the agent's position is exactly what the user has specified as a target.

4.1.4 Update

The update equation is similar to that of the DFO, expanding on the control and customizability of the swarm's behaviour using the same concepts as before. An update amount parameter acts as a scalar for the radius of the circle formed around the best neighbour's position, with smaller values updating the agents in smaller intervals. The update exponent shapes the probability distribution for the agent landing within that circle, with exponents larger than 1 curving the distribution downwards, causing most agents to land closer to their best neighbour's position for that dimension. Exponents with a value of less than 1 have the opposite effect.

Another important parameter that can be controlled by the user, in real time, is that of social hierarchy. Traditionally, agents are aware of the position of the highest-rated individual in the swarm, and take it into account when using their best neighbour's position as a platform for their update. This can lead to the whole swarm converging towards the specified target value quicker. However, if one is interested more in the swarm's journey rather than its arrival to the destination, then the elitist approach, whereby the position of the best individual influences the movements of the whole swarm, can be replaced with a more local-based hierarchy. Therefore, depending on whether or not the swarm is following the elitist approach, one of the following update equations are used:

$$x_{id}^{t} = \begin{cases} n_d^{t-1} + U(0,1)^{exp_d} \times amt_d \times (b_d^{t-1} - x_{id}^{t-1}), & \text{if elitist.} \\ n_d^{t-1} + U(0,1)^{exp_d} \times amt_d \times (n_d^{t-1} - x_{id}^{t-1}), & \text{otherwise.} \end{cases} \tag{4}$$

where n_d^{t-1} is the position of the agent's best-scoring neighbour (using ring topology) for the dimension d, exp_d and amt_d are the update exponents and amounts for that dimension respectively, and b_d^{t-1} is the position in dimension d of the swarm's highest scoring individual.

An important feature of the update method is that of being able to choose how many frames the algorithm will have to go through before updating the agents' positions again, as well as how many times the update equation will be applied in a row before releasing the new positions. Between the updates, the current positions of the agents for that frame are linearly interpolated between the previous and current positions that came out of the update equation. This allows for greater control over the behaviour of the swarm, choosing between smooth, linear motions versus pointillistic jumps and how tight or loosely the swarm will be following a changing target vector.

4.2 Visualisation

The following images in figure 2 attempt to show the behavioural variety obtainable from a single swarm through different parameter configurations. Here, the same sine wave is traced four times, each time using a slightly different set of parameters to achieve significantly different aesthetics. All visualizations unfolded from left to right over 500 frames, using 80 agents.

Example 1 acts as a starting point, with the disturbance thresholds and amounts set to a value of 0.1 for each dimension. The update function was run 20 times in a row every 5 frames, linearly interpolating in between the previous and next positions for each agent.

In Example 2, the same values were used for disturbance and update thresholds, amounts, and exponents. However, this time the update function was run 10 times in a row every 20 frames. This allows the algorithm to interpolate between each consecutive set of positions in a smoother fashion, producing a set of lines instead of the first example's more pointillistic approach.

Examples 3 and 4 make use of the single disturbance threshold option, whereby each agent's dimensions are disturbed simultaneously. Different amounts and exponents are used for each dimension, which allows for separate control over the positions in the x and y axes. Example 3 uses a disturbance threshold of 0.1 and disturbance amounts of 0.5 and 0.3 respectively. However, the x dimension's exponent is set to 8, while that of the y dimension is set to 0.3. This clearly shows in the distribution of the disturbances in the different dimensions, where the x dimension has a few points disturbed further away while the y dimension has most points disturbed at the maximum amount specified.

Finally, example 4 modulates the single disturbance threshold over time, using a sine wave that loops two times over the visualization's unfolding. An update exponent value of 5 is used, curving the distribution of where each agent's updated position falls within the circle around the best neighbour. New positions are calculated every 10 frames, each time running the update method for 40 times in a row.

5 EXAMPLE APPLICATIONS

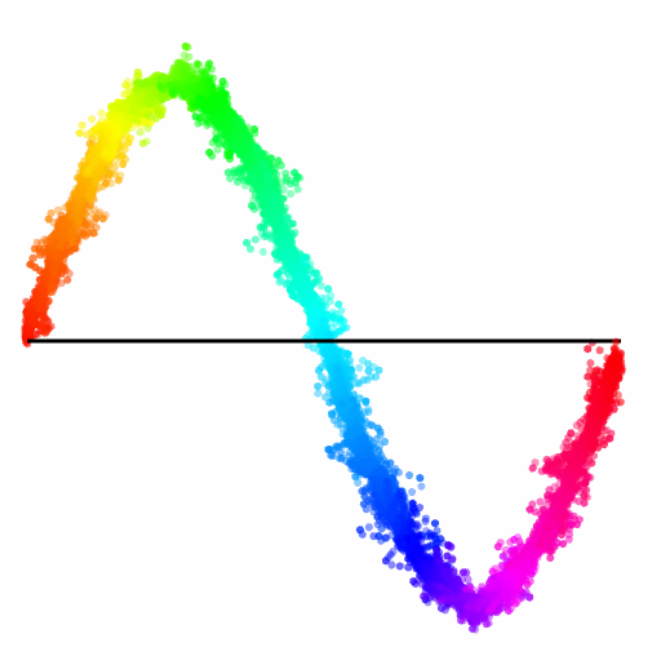
The development of the *multipleAgents* API came about as a result of an iterative feedback process, heavily informed by creative experiments that were run alongside it. Here, two such experiments are presented, one in the field of computer music and another in that of abstract graphics.

5.1 Music

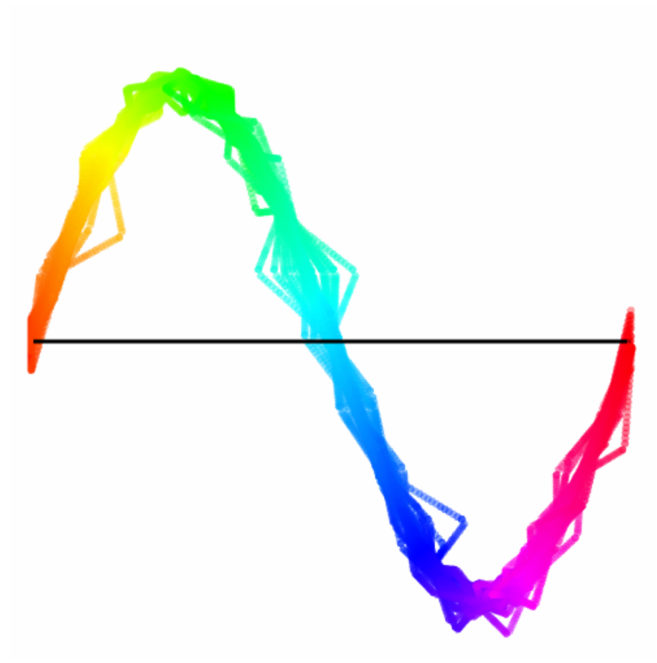
The first application of Multiple Agents Orchestration was *multipleAgents*, a multi-channel composition based on granulating Frequency Modulation synthesizers. Granulation is a form of synthesis whereby a cloud of sound emerges as the sum of many short-lived sound particles, with durations anywhere from a few to a few hundred milliseconds. These particles can either be played back from a pre-recorded file, in which case the playhead's position and the playback rate are parameters that heavily impact the resulting sound, or can be applied on real-time synthesis processes. For this piece, 12 granular synthesizers were applied over 12 Frequency Modulation synthesizers, a different synthesis method where the frequency of sine waves is modulated at extreme amounts and rates, well past the perceptual threshold of continuity, resulting in side-band frequencies emerging from that one harmonically pure sine wave. By changing the parameters of the granulation processes, as well as the underlying FM synthesis, an extremely rich and varied palette of sounds becomes available.

The piece was implemented in the Java-based creative programming environment Processing, which was handling the swarm's implementation, and the real-time sound synthesis environment SuperCollider, which was responsible for communicating the target vector changes to the swarm through the OSC protocol, receiving the normalised values of each agent for each dimension, mapping them to the desired ranges and updating the running granular and FM synthesizers.

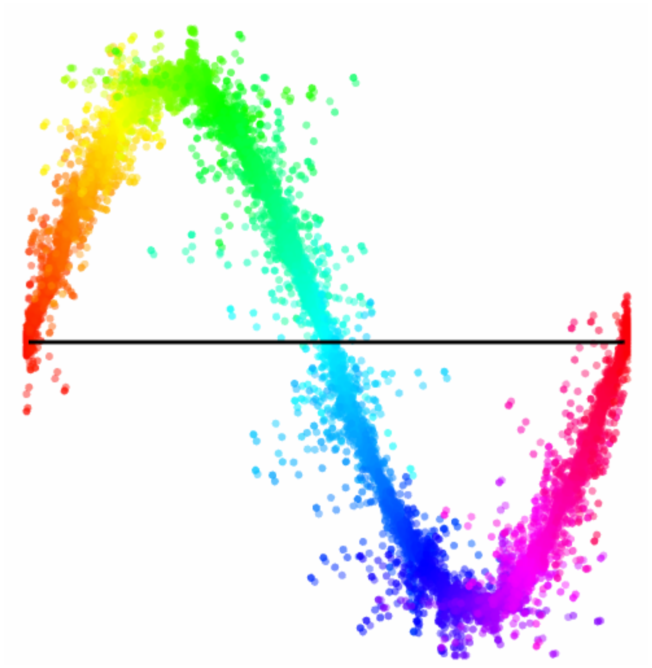
The emergent nature of granular synthesis, as well as its demand for controlling many similar, simultaneous processes, seemed like a perfect fit for the higher-level control that can be obtained using Swarm Intelligence. Furthermore, seeing as the original algorithm was inspired by the position of flies in physical space, it seemed like a great opportunity to sonnify that behaviour by mapping one of the dimensions to the panning parameter, or the distribution of sound across a set of speakers. In particular, a multi-channel sound system was used, with 6 speakers placed in a circular array around the audience. By carefully manipulating the swarm's different parameters,



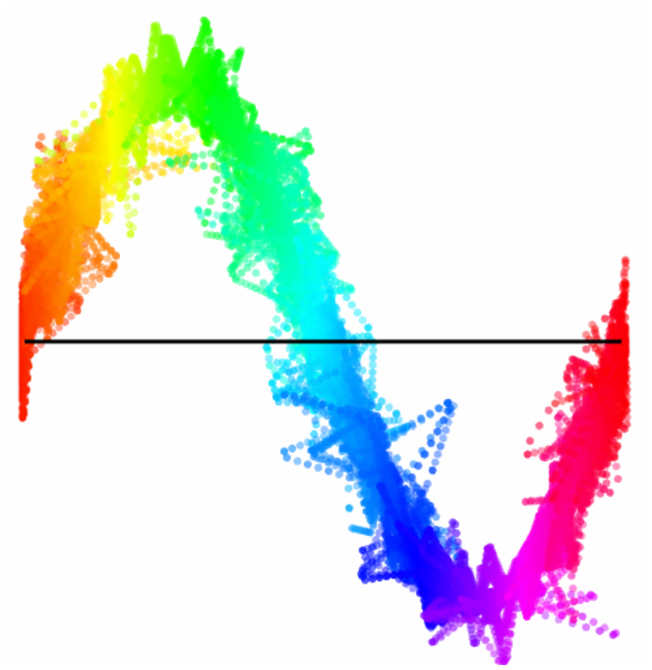
Example 1



Example 2



Example 3



Example 4

Figure 2: A visualisation of the different behaviours obtainable through different configurations of the swarm's parameters. Here, a sine wave is being specified by changing the target vector's x and y coordinates and traced by the swarm, with a hue sweep representing the algorithm's iteration.

I was able to guide it around through physical and frequency space, using no other form of communication but the target vector and the disturbance threshold and amounts.

Some of the parameters that were controlled by the swarm are:

1. Grain Amplitude
2. Grain Duration
3. Grain Panning
4. FM Carrier Frequency
5. FM Modulation Ratio
6. FM Modulation Amount

multipleAgents was performed on November 24th, 2017, during the *Resolution* event at St. James Hatcham, Goldsmiths University of London.

5.2 Graphics

This simple generative spiral in Figure 3 (appended) was generated using the next (and current) iteration of multipleAgents. The main addition of this version was that of separate disturbance thresholds and amounts for each of the dimensions.

Here, 30 agents are guided around a spiral, starting at the centre of the screen, by changing the target vector over a certain number of frames, and therefore iterations. Only two dimensions are used, representing a set of polar coordinates at which to draw a circle. A hue sweep is used to represent the passing iterations, starting from red and looping around the color wheel once.

Figure 4 shows the same swarm with the same parameters, except for one crucial difference: this time the agents follow a local neighbour approach, versus the global elitist approach. As a consequence, the swarm is less focused on following the leading agent and more focused on following the best neighbour, which results in a much more chaotic and dispersed aesthetic.

Even though each agent is merely drawing a circle during every iteration, there emerges a perceptual element of a unified spiral unfolding over space. Through some of the Gestalt principles [3] such as proximity and common fate, the circles of the individual agents get perceptually grouped into a single, complex, and organically evolving shape that gets perceived as having its own identity. This visual emergence of complex higher entities from simple components following simple rules was heavily influenced by the work of Andy Lomas [2], more specifically his video series *Hybrid Forms*.

REFERENCES

[1] Mohammad Majid Al-Rifaie, ‘Dispersive flies optimisation’, *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, (2014).

[2] Andy Lomas. Hybrid forms. http://www.andylomas.com/hybridForms.html.

[3] Max Wertheimer. Laws of organization in perceptual forms. http://psychclassics.yorku.ca/Wertheimer/Forms/forms.htm.

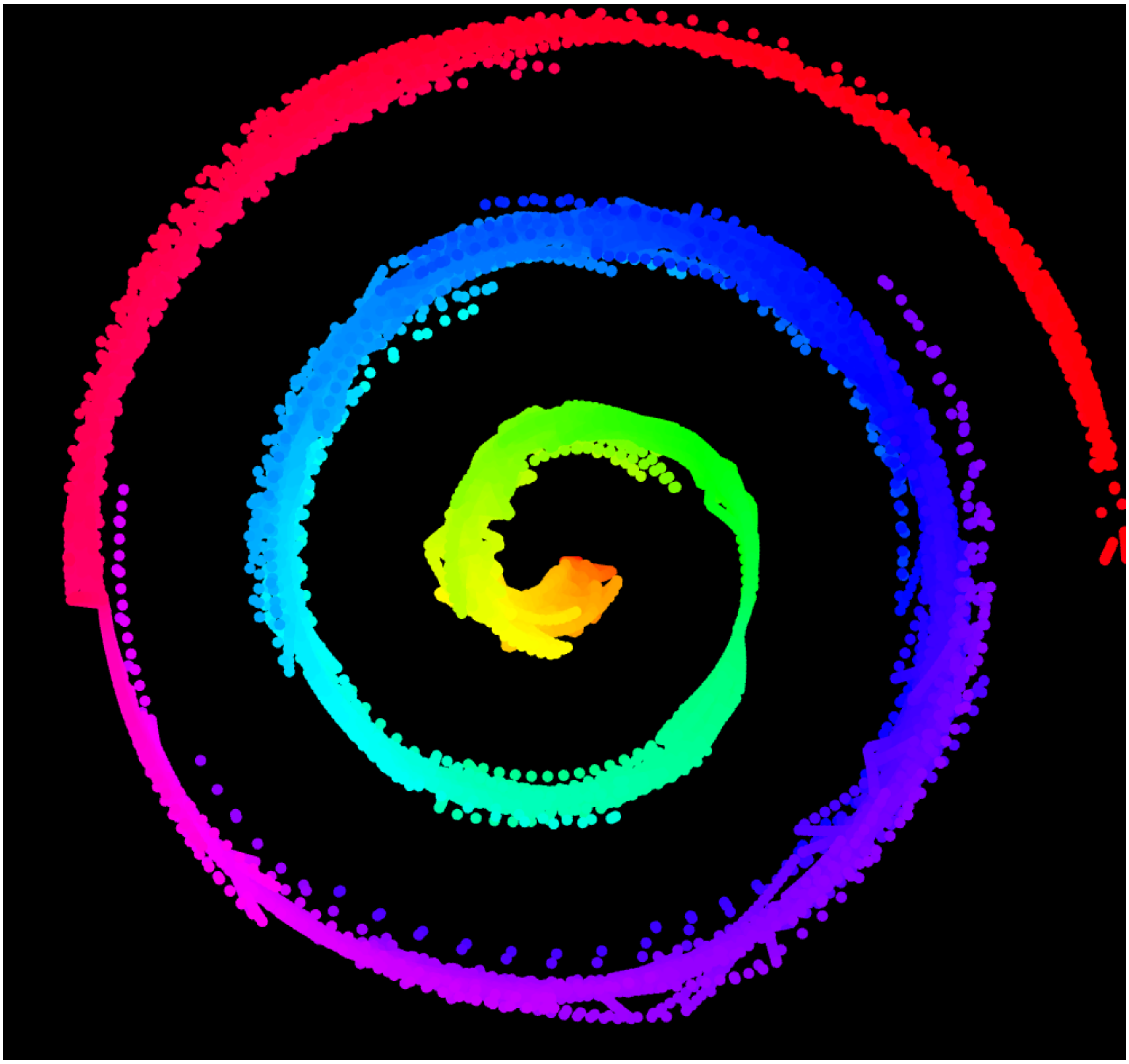


Figure 3: Chasing the spiral - Global Elitist Approach

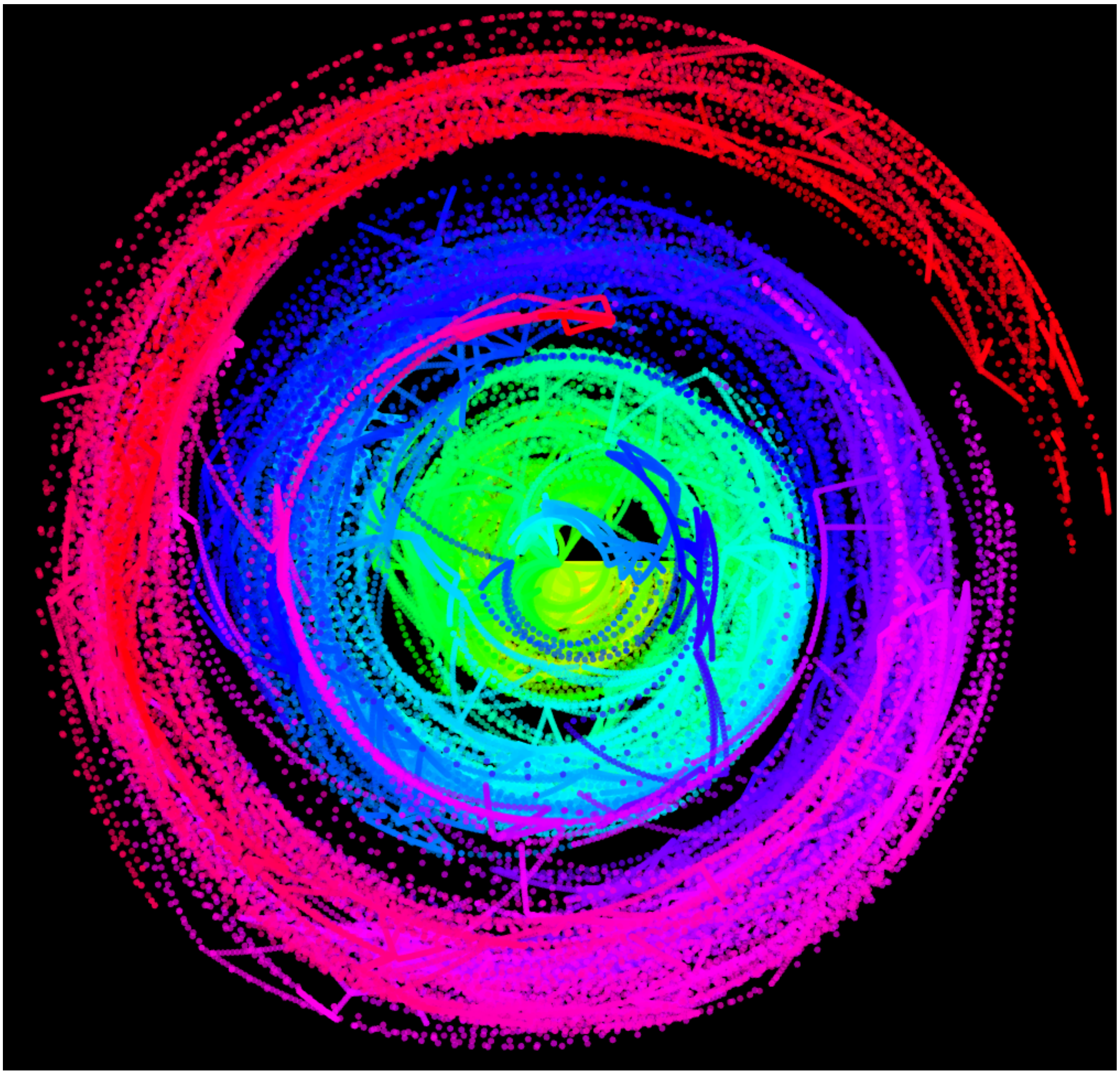


Figure 4: Chasing the spiral - Local Approach