Exploring the emergent transport networks in algorithmic approximations of *Physarum polycephalum*

An exploration into emergent behaviour in simulations of cellular automata and particle physics simulations

Henry Grantham-Smith

2022

Abstract

Emergence is a little understood topic of study, but one that holds a massive bearing on the world, and *Physarum polycephalum* is a heavily studied organism that has been shown to display complex emergent behaviours, and rudimentary intelligence. Within my EPQ I sought to explore emergence by means of a 'hands-on' approach, allowing me to explore the complexities of emergent behaviour by simulating *Physarum polycephalum*, and its potential for problem solving and network generation.

I used Unity Game Engine to handle graphics rendering, and C# with HLSL as my languages of choice. This allowed me to focus on implementation of my algorithm, without overemphasis on boilerplate code. Overall I am very happy with my result, and how closely the behaviours of my simulation match the behaviours of *Physarum polycephalum*. The patterning of multiple of my simulations match the transport networks of *Physarum polycephalum* almost exactly, and the potential scope for efficiently solving network based problems is massive due to the real-time nature of my simulation.

Contents

Glossary

Physarum polycephalum

a plasmodial slime mould of significant study and note due to its observed behaviours. Yellow in appearance when in the plasmodial stage of its life cycle. 1, 5–7, 13

amoeboid myxamoebae

a form of cell that typically inhabits dryer conditions, often displaying amoeboid characteristics despite not belonging to the classification amoeba. 6

biflagellated swarm cell

a form of cell that has two flagella, primarily existing within wet environments. 6

bilinear

a filtering mode for images, that typically results in smoother images that look less 'pixelated'. 9

boilerplate code

The code that sits behind the scenes of a program, dealing with much of the functionality that is crucial for a program to operate. Boilerplate code is often handled by libraries or engines. 1, 5

C#

an Object oriented programming language created by Microsoft, and is the primary language used by Unity game engine. 1, 5

chemoattractant

a chemical that is used by organisms as an attractant, often between organisms that operate on swarm mechanics, chemoattractants attract the associated organism and often allow for coordination between individuals within a swarm. 9, 12

class

the blueprint of an object within the object oriented programming paradigm. A class will outline all the attributes and behaviours of and object. 7

compute shader

a shader script that handles data and computation by utilising the GPU, allowing for faster processing as the computational load is ported to a separate component. 4, 8

cytokinesis

the process following nuclear mitosis, where the entire cell splits in two, each half containing half the nuclei present. 6

diploid

a state of an organism's genetic information whereby the organism's nuclei contain all of its genetic code. 6

emergent behaviour

a behaviour that stems from emergent interactions of an object's constituent parts. These are behaviours that are not coordinated by a control center but rather stem interactions of objects or their components. 1, 5

gamete

a sex cell for an organism, typically two gametes will fuse to form a new fertile cell known as a zygote. 6

haploid

a state of an organism's genetic information whereby the organism's nuclei contain half of its genetic code. 6

HLSL

High Level shader language. Used to program compute shaders and other shader scripts. 1, 5

kernel

the main component of a shader script, it defines the behaviour and functionality of the shader. 9

meiosis

a form of cell division that halves the present genetic information. It will result in 4 genetically different cells each containing half the original volume of genetic information. 6

miotic division

a process by which a cell or nuclei, splits in two, while duplicating the genetic information as to maintain the quantity of genetic code in each nuclei. 6

multi-nucleated

an organism that exists with multiple nuclei within its individual cell(s). Nuclei are membrane-bound organelle that contain the genetic information for the entire organism. 4, 6

object oriented programming

a programming paradigm that features a concept of self contained objects that exist with their own functions and attributes. 3

plasmodial slime mold

a type of single celled, multi-nucleated organism that undergoes multiple individual phases, including a characteristic plasmodial phase wherein the organism exists as a single cell, up to a foot in diameter. 5, 6

RGBA

a colour space defined by the fact that it contains 4 channels. each channel is a float value that controls the quantity of red, green, blue and transparency, known as alpha. 9

shader

a script that can write to and change 2D images, by directly writing to change each pixel within that image. 3, 4

sporangia

storage sites for spores, often found in fungi. 6

true slime mold

another name for plasmodial slime mold. 5

zygote

the primordial cell that contains all the genetic information for the organism, and will kick start the required process to form and develop the entire organism. 6

My EPQ

1.1 Introduction

As an exploration into the complexities of emergence and emergent behaviour I decided to try to model the behaviour of *Physarum polycephalum*. My intention was to simulate the process by which *Physarum polycephalum* explores its immediate surroundings in an attempt to find food, leading to complex network generation. In the process exploring how approximations of emergence and emergent behaviour can be algorithmically structured and observed for the sake of better understanding. While also developing a system that would yield visually appealing results.

When researching around this subject it very quickly became apparent how incredible emergence, and by extension how interesting *Physarum polycephalum*, is. I quickly decided that the best way to explore the behaviour of *Physarum polycephalum* was to attempt to model it algorithmically, building a program that would simulate the behaviours that evolve from the emergent properties of *Physarum polycephalum*, and other plasmodial slime molds.

I built the entire program in C# and HLSL, using the Unity game engine in order to manage rendering, and boilerplate code for my simulation. In the end I ended up with a program that I am very happy with, and one that I feel certainly manages to simulate not only the behaviours of *Physarum polycephalum* but also the nature of emergent behaviours at play.

1.2 Order from Chaos

Emergence is best described as a rule of our universe, intrinsic to its fundamental laws. Many small objects come together to form a more complex structure that has properties not present in any one of its original constituent parts. It describes how complexity can grow from simple rules.

Consider water; the molecule H_2O has no properties that make it "wet", the concept and feeling of "wetness" is a property that evolves from the many interactions of smaller molecules on a microscopic scale. Equally as such, try to define a nation. What is a nation? It is not the land, it is not the symbols, the culture, the language or even the people. All of these elements change, so is that to say that nations don't exist? No, nations certainly exist, and have a measurable impact on the planet. Thus, a nation is best described not as any one of its original constituent parts, but rather, as an emergent property of the interactions of the people who live within its borders. The actions of that nation are controlled by the individual decisions of every member of its population, yet a nation has far more power than any one person could ever have when acting on their own.

Emergence has a massive impact on all things around us. From the interactions of molecules making the many dead parts of our cells into a living organism; to nations, created by the interactions of humans; to galaxies, defined as massive clouds of gas and stars manipulated by its own gravity [28]. Emergence is iterative and evident at every level of organisation or structure within the universe. For this reason is it possible, even, to debate the very nature of consciousness? Is the mind an emergent property of the brain? A question that is far beyond the scope of this paper, but one that truly highlights how little we still know about the nature of emergence, and the impact it has on everything around us.

1.3 The Blob (Physarum polycephalum)

Physarum polycephalum is a true slime mold, also known as a plasmodial slime mold. Belonging to the Kingdom Protista and the Class Myxomycetes [19], [26]. Plasmodial slime molds follow a very complex life cycle, one that has previously led to difficulty when classifying them [3].

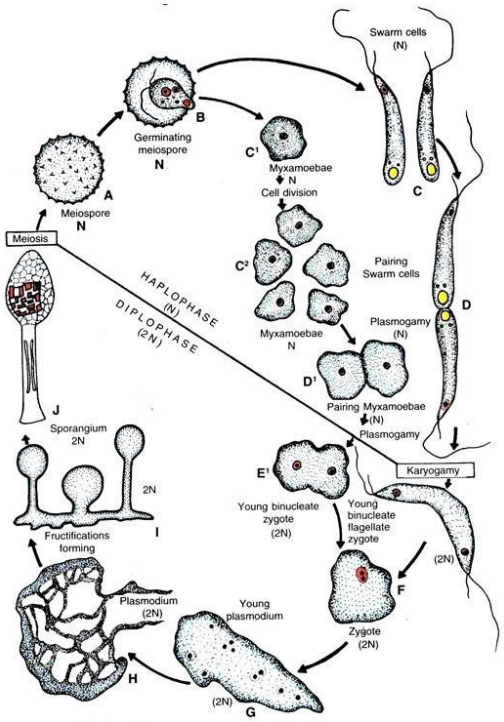


Figure 1.1: The life cycle of *Physarum polycephalum* [13]

When haploid gametes fuse to form a diploid zygote. The diploid zygote undergoes miotic division forming a multi-nucleated plasmodium [13], a single celled organism with many millions of nuclei. This occurs since the plasmodium never undergoes cytokinesis, and hence the cell never splits when in the plasmodial stage [13]. It is this stage that characterises plasmodial slime molds like *Physarum polycephalum* [11]. Slowly the slime mold will dry out, beginning the next stage in a plasmodial slime molds life cycle. The drying plasmodium will form stalks, topped with sporangia. Haploid spores are formed within the sporangia through meiosis [13]. The spores disperse, spreading out over a very large area [8]. When in optimal conditions, the spores will germinate into haploid gametes. A single spore has the ability to germinate into two different types of gamete, dependent upon the moisture levels. Biflagellated swarm cells are formed when the moisture level is very high, whereas amoeboid myxamoebae are formed if the moisture levels are any lower [8]. Finally, two gametes need meet in order to fuse into a single diploid zygote cell and start the cycle over again[8], [11], [13].

Plasmodial slime molds are most recognisable during their plasmodial stage, when they exist as a single multinucleated cell. The single cell is filled with a thick soup of molecules and fluid, known as protoplasm [2]. Plasmodial slime molds have also been known to grow to impressive sizes, measuring as large as a foot in diameter, impressive since it is a single cell [11]. When in this stage, *Physarum polycephalum* uses peristaltic movements of its protoplasm to move [2], [8]. It will use projections of itself to explore and forage in its surroundings, growing across a space and then reorganising to form a network between food sources [11], [16]. This is the most impressive behaviour displayed by *Physarum polycephalum* as it appears to display intelligence, or at the very least a form of problem solving [2].

This is significant since by doing so, *Physarum polycephalum* very apparently displays emergent behaviour. It allows us an insight into how truly fantastic the phenomenon of emergence is. Intelligence and problem solving are characteristics currently associated with developed frontal and parietal lobes [31]. Despite this, *Physarum polycephalum* has no nervous system, and with it, a complete lack of any centralised decision-making region including a brain [2].

By forming a network between food sources, *Physarum polycephalum* is in essence able to solve "the traveling salesman problem". This mathematical challenge is outlined as follows:

A salesperson has N number of cities to visit and wishes to sell their produce at each location. What is the shortest, and therefore most efficient path for the salesperson to follow?

For a small value of N the challenge is relatively simple to solve. By finding all possible connections of the points, and then choosing the shortest path the problem is quickly solved. However, the complexity of the problem increases exponentially not linearly. Thus, to solve the problem for just a few nodes it could take uncountable iterations. This quickly shows how impressive the task

completed by the slime mould is, since the task is in many cases too computationally heavy for a computer using a brute force methodology.

Using this behaviour of *Physarum polycephalum*, an experiment was devised by Atsushi Tero from Hokkaido University [11]. Oats were placed on a scale map of Tokyo and its surrounding area, with each oat representing a major station surrounding Tokyo's city centre [7]. This changed the implications of the slime molds behaviour; no longer was the slime mold connecting a network of randomly distributed points, but rather it was solving an infrastructure problem that took many engineers, many years to design [7], [32]; and remarkably, when the slime mold branched out and connected each oat in an intricate network the resulting web almost exactly matched the railway infrastructure [7], [11]. The slime mold had successfully found the most efficient route between each point.

1.4 The Fundamental Algorithm

1.4.1 Ants

As a starting point, I began by looking at ants, and their use of various chemoattractants to communicate and develop swarm intelligence. Ants have also been shown to solve multiple complex problems including the travelling salesperson problem [10], making them an excellent candidate for a first look at algorithmic approximations of emergence.

My initial algorithm was intended to simulate the movement of a single ant. Each ant would be modelled with a class, existing as an individual instance, with their own set of attributes:

x, y	acting as relative positions to the origin, in the horizontal and vertical directions respectively, stored as two float/Real numbers
SenseRange	which is simply a float number representing how far the ant can detect chemoattractants from
v	acting as the velocity component for each ant
θ	acting as the relative offset in radians with a range of $0 \leq \theta < 2\pi$

Every frame, the ant's relative x and y would be updated in the direction θ with a magnitude of v (multiplied by the time since the last frame was drawn as to make the simulation frame rate independent), and a marker would be placed to act as a chemoattractant. Once the frame was drawn θ would be updated to point towards the nearest chemoattractant within a distance of their SenseRange. If no chemoattractant existed, then θ would remain the same. Finally random variance would be added to θ allowing variation in the ant's movement, which would also give rise to a wandering mechanic within the ant's behaviour, closely modeling the logic behind the implementation of a particle physics simulation [4].

1.4.2 Slime

I based my overarching logic for my final simulation, upon this principle of independent agents acting together all influenced by their surroundings, but ultimately unaware of the greater simulation.

Very much alike my ant simulation, each agent within the simulation space had two main components that were individual to it:

x, y	acting as two dimensional coordinates relative to the origin, in the horizontal and vertical directions respectively. $\{x, y\} \subset \mathbb{R}$
θ	acting as the relative offset in radians with a range of $0 \leq \theta < 2\pi$

On top of this each separately defined species has its own set of attributes that apply to each of the agents assigned to that species of slime mold.

It should be noted that, when I reference species from this point forward, I am not referencing biological species, but rather different permutations of *Physarum polycephalum* with subtly different settings, leading to different patterns of cohesion and organisation.

n	denoting the number of agents that will populate the simulation
v	the velocity of all agents in the simulation
SpawnMode	the initial starting configuration for agents when the simulation starts
DecayRate	the rate at which chemoattractant fades from the simulation
DiffuseRate	the rate at which chemoattractractant spreads across the simulation space
SensorOffset	the distance from the agent checked by the algorithm
SensorSize	the size of the area of pixels checked by the algorithm
SensorSpacing	the angle between each sensor
Turn v	the speed at which an agent is able to change its relative direction
r, g, b, a	the red, green, blue and alpha channels used to colour each agent while the simulation is running (useful for distinguishing between different species)

1.4.3 Textures and Data Storage

The algorithm is also based on four main render textures:

Trail Texture stores the agent data and chemoattractant trails.

Diffused Texture uses the DiffuseRate to blur the chemoattractant trails before storing the resulting values in each pixel.

Colour Texture the diffused texture with a re-colour shader applied, it uses the stored RGBA values within the species data.

Final Texture this is the texture rendered to the screen each frame, derived from the Colour Texture.

and the size of the textures is the same between all four. It is the size of these textures that denotes the size of the simulation space. I used 1080 x 920 px as a standard size for all of my simulations. I used each pixel to denote discrete space, leading to a simulation much like John Conway's 'Game of Life'; A cellular automata defined by a set of simple rules [15], [17].

When the simulation starts for the first time it calls to the Set method. This method handles the loading of all the data into the simulation, before also passing that same data into each of the compute shaders that will handle the simulation and textures.

Once all the relative data is retrieved the algorithm initialises a compute buffer and an array, both set to a size of n. The array will store the relative x, y and θ for each agent, while the buffer will do the same for each frame of the simulation, and each texture on that frame.

1.4.4 Spawning

once the data is passed in, and the simulation starts the first frame, the simulation calls to a function that handles agent setup. The function is passed the array of agents, the width and height of the simulation, and a parameter denoting what spawn mode to start the simulation in. There are four main starting configurations. For each Agent their x, y and θ are set, and the array is returned with updated Agent values.

Point where all agents spawn in a single point at the centre of the simulation space.

$$x = \frac{W}{2} \tag{1.1}$$

Where W is defined as the width of the simulation space

$$y = \frac{H}{2} \tag{1.2}$$

Where H is defined as the height of the simulation space

$$\theta = 2\pi(R) \tag{1.3}$$

Where R is some random value unique to each agent, $0 \leq R \leq 1.0$, $R \in \mathbb{R}$.

Random where all agents spawn with a random position within the simulation space

$$x = R_1 \tag{1.4}$$

$$y = R_2 \tag{1.5}$$

$$\theta = 2\pi(R_1) \tag{1.6}$$

Where R_1, R_2 and R_3 are unique random values,
$0 \leq R_1 \leq$ Simulation space width, $0 \leq R_2 \leq$ simulation space height $0 \leq R_3 \leq 1.0$
$\{R_1, R_2, R_3\} \subset \mathbb{R}$.

Random Circle where all agents spawn on a random location within a circle that is $0.15\times$ the simulation space height, with a random θ.

$$x = W + 0.15W(\sqrt{R_1})\cos(2\pi R_2) \tag{1.7}$$

$$y = W + 0.15H(\sqrt{R_3})\sin(2\pi R_4) \tag{1.8}$$

$$\theta = 2\pi(R_5) \tag{1.9}$$

Where $A = \{R_1, R_2, R_3, R_4, R_5\}$, $0 \leq A \leq 1.0$, $A \subset \mathbb{R}$,
$W =$ the simulation space width, and $H =$ the simulation space height

Inward Circle where all agents spawn on a random location within a circle that is $0.15\times$ the simulation space height, and all agents are oriented to face towards the centre of the screen.

$$x = W + 0.15W(\sqrt{R_1})\cos(2\pi R_2) \tag{1.10}$$

$$y = W + 0.15H(\sqrt{R_3})\sin(2\pi R_4) \tag{1.11}$$

$$\theta = \arctan\left(\frac{\frac{H \div 2 - y}{\sqrt{(H \div 2 - y)^2 + (W \div 2 - x)^2}}}{\frac{W \div 2 - x}{\sqrt{(H \div 2 - y)^2 + (W \div 2 - x)^2}}}\right) \tag{1.12}$$

Where $A = \{R_1, R_2, R_3, R_4, R_5\}$, $0 \leq A \leq 1.0$, $A \subset \mathbb{R}$,
$W =$ the simulation space width, and $H =$ the simulation space height

Following the assignment of the Agent data, the array data is passed into the agent compute buffer. Where the main simulation kernel is given the starting values of all the Agents. On the first frame the starting configuration acts as the data that is rendered to the screen without the simulation running. This involves simply passing the Agent data into into the main kernel where the texture is directly copied onto the final texture, and each pixel is coloured white for each agent's respective x and y, while the rest of the texture remains black.

1.4.5 Running the Simulation

for all following frames the data is passed to the Run Simulation Method:

This method handles all data for the simulation, and manages the movement of data between kernels and the main program. foremost it checks that the Trail Texture does not equal null. Should it be found that the Trail Texture does not exist, the trail texture is initialised to a new RenderTexture object. The Render texture is set to use RGBA colour space, and random read and write is enabled. Finally the filter mode is set to bilinear, and the texture is created.

This same process is repeated for each texture.

Following the creation of these textures, all the textures are assigned to their corresponding kernels. and each of the compute shaders are dispatched. Including:

pragma kernel Update Which handles the behaviour for each Agent, based on their θ and local chemoattractant intensity.

pragma kernel Diffuse Blurs the entire texture with an averaging algorithm.

pragma kernel ReColour takes the relative value from 0 to 1 for each pixel and finds the dot product with r, g, b, and a to obtain a re-coloured version of the diffused texture.

Finally the method finishes by copying the diffused texture to the trail texture, and the Colour texture to the final texture. This is accomplished with a function that simply transfers data from the source texture to the target texture.

1.4.6 Pragma Kernels

Update

The Update kernel foremost checks that the current id $\not\geq n$.

Following this it generates a pseudo random value from a predetermined hash function, the hash function receives a single seed:

$$RandomValue = \frac{ffg(seed)}{4294967295} \tag{1.13}$$

where $RandomValue$ has a range of $R : [0, 1]$, and the functions g and f are defined as:

$$g(x) = 2654435769 \left(x^{2747636419}\right) \tag{1.14}$$

$$f(x) = 2654435769 \left(x^{(x \gg 16)}\right) \tag{1.15}$$

To calculate a value for each individual agent the hash function is iterated on twice, using the first hashed value to generate a seed for the final random value that will be used by that agent for that frame of the simulation.

$$RandomValue = \frac{ffg\left(y \times width + x + \frac{ffg(x+(time \times 100000))}{4294967295}\right)}{4294967295} \tag{1.16}$$

The next key part of the algorithm for agent behaviour is the sense function. This function acts as the main vision for the Agent. The method is passed both the Agent in reference, and a sensor angle offset.

Sensor Angle	defined as θ + sensor angle offset. This will act as the relative offset based on the Agent's relative facing.
$SensorCenterX$	defined as $\cos(SensorAngle)$
$SensorCenterY$	defined as $\sin(SensorAngle)$
sum	used as a running total for the overall weight of that relative area.

The sense method checks a square of size SensorSize, centered on $(SensorCenterX, SensorCenterY)$. The algorithm loops through each pixel individually, within the allocated size. In order to account for index values that could be greater than the size of the simulation space the x and y coordinates for each pixel to be used are first checked, and corrected. The x and y coordinates are defined as:

$$x = \min(width - 1, \max(0, x)) \tag{1.17}$$

$$y = \min(height - 1, \max(0, y)) \tag{1.18}$$

Following the calculation of the x and y coordinates for each pixel, the sum is incremented by the value of the chemoattractant stored in the pixel with the respective coordinates. It is this sum value that is returned by the sense method. Due to this edge detection algorithm the chemoattractant in each of the boarder pixles does have the potential to have as much as the weighting multiplied by the SensorSize^2, dependent on the location of the sensor.

Back within the main Update kernel, following the generation of the pseudo random number, the main rules for movement are outlined. Foremost three areas are sampled; using the sense method, the kernel stores three values to represent each 'chunk' of vision available to the Agent. The first area is directly ahead of the Agent, where the sense method is passes a sense angle offset of zero. Following this each of the remaining two sections are passed $\pm$ SensorSpacing to rotate

the region of focus both clockwise and anti-clockwise. This results in the centers of each region having a radial distance of SensorSpacing between each of them.

The next phase of the kernel handles which direction the agent will face based on the results from the Sense regions. In the case that there is the strongest quantity of chemoattractant directly ahead, θ is not changed. In the case that either the clockwise, or anti-clockwise regions is more heavily weighted than the other, then θ will be updated. $\theta = \theta \pm (RandomValue \times TurnSpeed)$. It is the random value that adds enough variation as to allow the simulation to evolve organically over time. In the occasion that both the clockwise and anti-clockwise regions have equal values then a random turn value is calculated to prioritise a single direction over another for that frame, this constitutes the random searching behaviour of each agent.

$$\theta = \theta + (2(TurnSpeed) \times (RandomValue - 0.5)) \tag{1.19}$$

Following this the new x and y are calculated based on the facing direction of the agent.The next pixel the Agent occupies is then calculated as:

$$x = x + (\cos(x) \times v \times \delta) \tag{1.20}$$

$$y = y + (\sin(x) \times v \times \delta) \tag{1.21}$$

Where δ is defined as the time since the last frame. This multiplication scales the movement by the relative frame rate. Allowing the movement of the Agent to be independent of the frame rate.

The next crucial step is to confine the agents to the simulation space. If the target pixel is located outside the bounds of the texture then a new position is calculated. First the random value is recalculated.

$$RandomValue = \frac{ffg(RandomValue \times 4294967295)}{4294967295} \times 2\pi \tag{1.22}$$

This prevents the same random value from being used a second time over and scales it to a range of $R : [0, 2\pi]$.

I used much of the same logic as for if the pixel was outside of the search range within the sense function, whereby the x and y are reset to pixels within the simulation space using the same algorithms (1.17)(1.18), and θ is reset to face a random direction. This can result in the Agent leaving the simulation space again. However, probability dictates that eventually the agent's θ will be reset to a value that allows it to re-enter the simulation.

$$\theta = RandomValue \tag{1.23}$$

This set of equations will only run if the Agent is found to occupy a pixel outside the simulation, saving the kernel from excess computation.

The final and most important step taken by this kernel is to update the Trail Texture. This is done by finding the Trail Texture pixel with location x and y. The value of this cell is set to 1 in all RGBA channels, this essentially colours the associated pixel white. This step is significant as it acts as the primary record of the location of each agent, including where they have been over time. Since this Texture is updated each frame, the path of each agent can be followed by tracing the path of white pixels.

Diffuse

The next step for the simulation is to pass the data from the Trail Texture into the Diffuse kernel. This kernel uses two main real numbers, a value for the DiffuseRate and a value for the DecayRate. A significant feature of this kernel, is its dependence on the Trail Texture. However, as a rule, the Diffuse kernel is unable to write to the Trail Texture, rather it creates its own texture map that inherits all the data from the Trail Texture. This avoids conflicts of data on the GPU allowing for more sequential assignment of data between frames, it is this texture that is defined as the Diffused Texture.

To apply the necessary blur to the texture the following algorithm is applied:

Original Weight	defined as the weight of the cell from the Trail Texture that is currently being iterated over.
Blurred Colour	defined as the sum divided by the scale of the blur.
Diffuse Weight	used as the weight of diffusion for that frame in reference real time.
sum	used as a running total for the overall weight of that relative pixel.

We start by initialising the value of sum to 0, and Original Weight to the value of our pixel from the Trail Texture. Following this, we iterate over a three by three pixel grid centred on our target pixel. We clamp our pixels to sit within the simulation space using the same algorithm as in our sense function (1.17)(1.18). Following this we set our sum to the sum of each cell within the three by three grid, assuming each pixel is not outside the bounds of the simulation space. This algorithm does again more heavily weight pixels on the edge of the simulation. Blurred Colour is then set to store the average of each of these cells accomplished as,

$$BlurredColour_1 = \frac{sum}{9} \tag{1.24}$$

Diffuse Weight is then calculated as

$$DiffuseWeight = \min\left(1, \max\left(0, DiffuseRate \times \delta\right)\right) \tag{1.25}$$

Where δ is the time since the last frame was drawn, this addition keeps the diffusion of chemoattractants independent of frame rate, again allowing for real time rendering.

Finally Blurred Colour is recalculated as

$$BlurredColour_2 = OriginalColour \times (1 - DiffuseWeight) + BlurredColour_1 \times DiffuseWeight \tag{1.26}$$

The pixel in Diffused Texture is then set to:

$$\max\left(0, BlurredColour_2 - DecayRate \times \delta\right) \tag{1.27}$$

Where δ is the time since the last frame was drawn. This final calculation is used to fade the chemoattractant from the scene while also clamping the value to a range of $D : [0, 1]$.

The most crucial step outside of the kernels to take place between each frame is the copying of the Diffuse Texture to The Trail Texture, this is the fundamental step that defines how the Agents react to and set down chemoattractants.

Recolour

The final kernel to address is the ReColour kernel. This phase is entirely optional, as it simply applies a colour to the simulation, this helps to determine between different species of slime mold, but is essentially aesthetic. The kernel takes an input of each pixel and divides each colour channel into their respective value. Since the simulation runs off of single values all the corresponding channels will be identical to begin with, this lack of complexity makes re-colouring very simple. I simply calculate each channel as

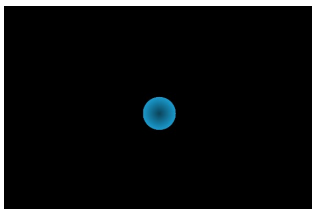
$$Channel = value \cdot colour \tag{1.28}$$

Where *Channel* represents each of the RGBA channels. *value* represents the numerical value stored in each cell of the simulation, and *colour* is the target colour when $value = 1$.

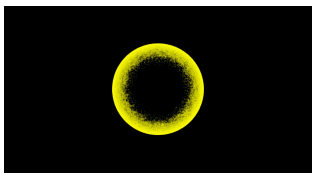
To finish the entire single iteration of a frame, the re-coloured texture is copied to the Final Texture which is rendered to the screen.

1.5 Implementation

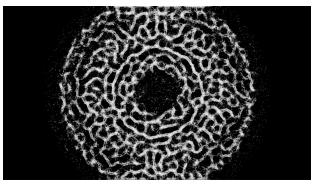
I found with my implementation, the sensitivity of the simulation very closely resembled the characteristics of a chaotic system; the tiniest change in settings, results in vastly different outcomes. The largest impact I found was associated with the cohesion of the Agents. This was impacted by a culmination of each of the settings acting in unison, as seen in figure 1.2 particularly between examples such as figure 1.2d and figure 1.2f.



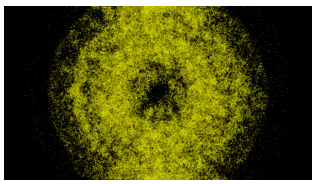
(a) The starting frames of a slime mold with high cohesion and a spawn mode of point



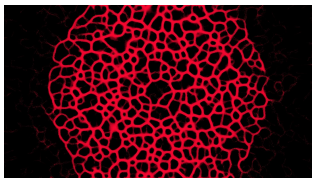
(b) a slime mold with high cohesion but low sense range and turn speed



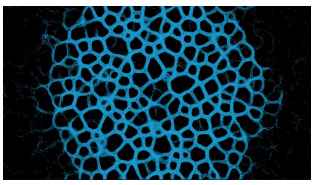
(c) this species has a very high cohesion but a quite limited sense range while also moving very slowly.



(d) this species has almost no vision, leading to a complete lack of cohesion



(e) this species exhibits the closest and most accurate portrayal of behaviours exhibited by *Physarum polycephalum*



(f) this species has fantastic sense range and size, and can move comparatively very quickly leading to harsher shapes

Figure 1.2: A series of frames from my simulation exhibiting different species with different behaviours as a result. (Each frame was taken at a different stage in each of the simulations to best showcase the behaviours)

One of the largest changes with my implementation to the final algorithm was, without a doubt, my addition of colour fading. (When switching between different species). This was implemented to account for live displays of my simulation, allowing me to change the settings of the simulation while it was still running. This change was purely aesthetic and did not change the behaviour of the simulation at all, but I felt it was important, since I denoted different species with different colours, and I felt a fade between colours was the most visual way to denote a change of settings within the simulation.

1.6 Evaluation

Overall, I am really happy with the result of my simulation. In a lot of different configurations the core behaviors of my simulation matched the interactions of *Physarum polycephalum*, and most of the patterning formed by the simulation match its organic counterpart. While it only inhabits a 2D plain, the simulation displays organisation, and cohesion, leading to complex pattern formation in the form of a network that continuously changes in response to external stimuli. I feel that in this way I have managed to simulate a close approximation to *Physarum polycephalum* and the emergence it displays.

Emergence, is a fundamental and complex part of our universe, it is present in all things, and working towards understanding this phenomenon, not only gives us insight into the complexities of organisms such as *Physarum polycephalum*, but a greater understanding of the complexities of the wider world that surrounds us.

Bibliography

[1] aldoaldoz, *Langton's Ant*, Jan. 2009. [Online]. Available: https://www.youtube.com/watch?v=1X-gtr4pEBU (visited on 03/04/2022).

[2] K. Alim, N. Andrew, and A. Pringle, "Physarum," English, *Current Biology*, vol. 23, no. 24, R1082–R1083, Dec. 2013, Publisher: Elsevier, ISSN: 0960-9822. DOI: 10.1016/j.cub.2013.09.040. [Online]. Available: https://www.cell.com/current-biology/abstract/S0960-9822(13)01187-1 (visited on 03/07/2022).

[3] W. Armstrong, *Slime Mold Photos*, Sep. 2020. [Online]. Available: https://www2.palomar.edu/users/warmstrong/slime1.htm (visited on 03/07/2022).

[4] Artificial Life Lab Graz, *How life emerges from a simple particle motion law: Introducing the Primordial Particle System*, Dec. 2016. [Online]. Available: https://www.youtube.com/watch?v=makaJpLvbow (visited on 03/04/2022).

[5] A. Avetisyan, *Opengl - Getting data back from compute shader*, Dec. 2016. [Online]. Available: https://stackoverflow.com/questions/40573213/getting-data-back-from-compute-shader (visited on 03/07/2022).

[6] C. Barcellos, *C++ - Interpolate from one color to another*, May 2017. [Online]. Available: https://stackoverflow.com/questions/13488957/interpolate-from-one-color-to-another (visited on 03/07/2022).

[7] BBC Earth Lab, *Can Slime Mould Solve Mazes? — Earth Lab*, Dec. 2018. [Online]. Available: https://www.youtube.com/watch?v=HyzT5b0tNtk (visited on 03/07/2022).

[8] B. Britannica, *Slime mold — organism — Britannica*, Mar. 2019. [Online]. Available: https://www.britannica.com/science/slime-mold (visited on 03/04/2022).

[9] A. Chen, *How to change a color from an image with HLSL*, Jan. 2012. [Online]. Available: https://social.msdn.microsoft.com/Forums/vstudio/en-US/d8b5b9c9-d359-423c-bb4d-4580a2d03a4b/how-to-change-a-color-from-an-image-with-hlsl (visited on 03/07/2022).

[10] A. c. —. http://www.ardamis.com, *The Emergent Virtues of Slime Molds — Dataisnature*, en, Sep. 2009. [Online]. Available: https://www.dataisnature.com/?p=535 (visited on 03/04/2022).

[11] F. Jabr, *How brainless slime molds redefine intelligence — Nature*, Article, Nov. 2012. [Online]. Available: https://www.nature.com/articles/nature.2012.11811 (visited on 03/04/2022).

[12] J. Jones, "Characteristics of pattern formation and evolution in approximations of physarum transport networks," en, *Artificial Life*, vol. 16, no. 2, Mar. 2010, Publisher: Massachusetts Institute of Technology Press (MIT Press), ISSN: 1064-5462, 1530-9185. DOI: 10.1162/artl.2010.16.2.16202. [Online]. Available: https://uwe-repository.worktribe.com/output/980579 (visited on 03/07/2022).

[13] S. Kritartha, *Life Cycle of Slime Mold (With Diagram) — Physarum*, en-US, Nov. 2016. [Online]. Available: https://www.biologydiscussion.com/fungi/life-cycle-of-slime-mold-with-diagram-physarum/63069 (visited on 03/07/2022).

[14] N. Mind, *Changing some color of a texture using C# in Unity*, Jul. 2016. [Online]. Available: https://gamedev.stackexchange.com/questions/124463/changing-some-color-of-a-texture-using-c-in-unity (visited on 03/07/2022).

[15] Numberphile, *Inventing Game of Life (John Conway) - Numberphile*, Mar. 2014. [Online]. Available: https://www.youtube.com/watch?v=R9Plq-D1gEk (visited on 03/04/2022).

[16] F. Patino-Ramirez, A. Boussard, C. Arson, and A. Dussutour, "Substrate composition directs slime molds behavior," en, *Scientific Reports*, vol. 9, no. 1, p. 15 444, Oct. 2019, Number: 1 Publisher: Nature Publishing Group, ISSN: 2045-2322. DOI: 10.1038/s41598-019-50872-z. [Online]. Available: https://www.nature.com/articles/s41598-019-50872-z (visited on 03/07/2022).

[17] Rational Animations, *Epic conway's game of life*, Oct. 2011. [Online]. Available: https://www.youtube.com/watch?v=C2vgICfQawE (visited on 03/04/2022).

[18] J. Seigel, *Color Interpolation Between 3 Colors in .NET*, Apr. 2020. [Online]. Available: https://stackoverflow.com/questions/1236683/color-interpolation-between-3-colors-in-net (visited on 03/07/2022).

[19] *Slime Moulds: General Characteristics, Cellular and Acellular Slime Moulds and Their Life Cycle*, en-US. [Online]. Available: https://byjus.com/neet/slime-moulds/ (visited on 03/07/2022).

[20] U. Technologies, *Unity - Manual: Compute shaders*, en, 2020. [Online]. Available: https://docs.unity3d.com/Manual/class-ComputeShader.html (visited on 03/07/2022).

[21] ——, *Unity - Manual: Custom Render Textures*, en, 2020. [Online]. Available: https://docs.unity3d.com/2019.3/Documentation/Manual/class-CustomRenderTexture.html (visited on 03/07/2022).

[22] ——, *Unity - Manual: GPU instancing*, en, 2020. [Online]. Available: https://docs.unity3d.com/Manual/GPUInstancing.html (visited on 03/07/2022).

[23] ——, *Unity - Manual: Shaders*, 2020. [Online]. Available: https://docs.unity3d.com/Manual/Shaders.html (visited on 03/07/2022).

[24] ——, *Unity - Manual: Texture arrays*, en, 2020. [Online]. Available: https://docs.unity3d.com/Manual/class-Texture2DArray.html (visited on 03/07/2022).

[25] ——, *Unity - Scripting API: Texture2D.GetPixel*, en, 2020. [Online]. Available: https://docs.unity3d.com/ScriptReference/Texture2D.GetPixel.html (visited on 03/07/2022).

[26] *UWL Website*. [Online]. Available: http://bioweb.uwlax.edu/bio203/2010/renner_brad/classification.htm (visited on 03/07/2022).

[27] Verge Science, *What self-driving cars can learn from brainless slime mold*, May 2018. [Online]. Available: https://www.youtube.com/watch?v=40f7_93NIgA (visited on 03/04/2022).

[28] *What is a galaxy? — nasa space place – nasa science for kids*. [Online]. Available: https://spaceplace.nasa.gov/galaxy/en/ (visited on 03/09/2022).

[29] *What is plasmodial slime mold? — Technology Trends*. [Online]. Available: https://www.primidi.com/what_is_plasmodial_slime_mold (visited on 03/07/2022).

[30] *What is true slime mold? — Technology Trends*. [Online]. Available: https://www.primidi.com/what_is_true_slime_mold (visited on 03/07/2022).

[31] *Where does intelligence reside in the brain?* en-US. [Online]. Available: https://bigthink.com/neuropsych/brain-intelligence/ (visited on 03/07/2022).

[32] World Science Festival, *What is Slime Mold and is it Intelligent?* Nov. 2019. [Online]. Available: https://www.youtube.com/watch?v=sEX3F2h8cjA (visited on 03/04/2022).