Computational Design Seminar

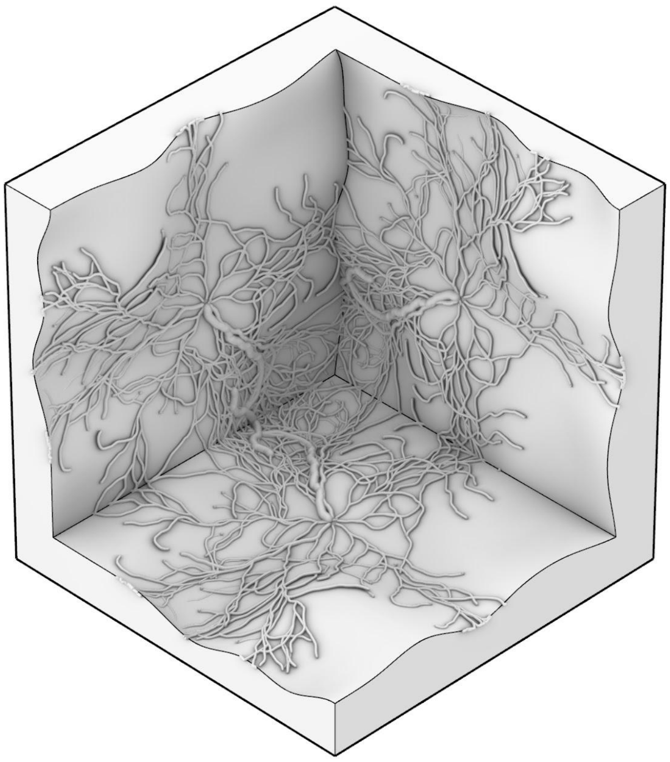
2020/21 Wintersemester

16.02.2021

Slime Mold Growth

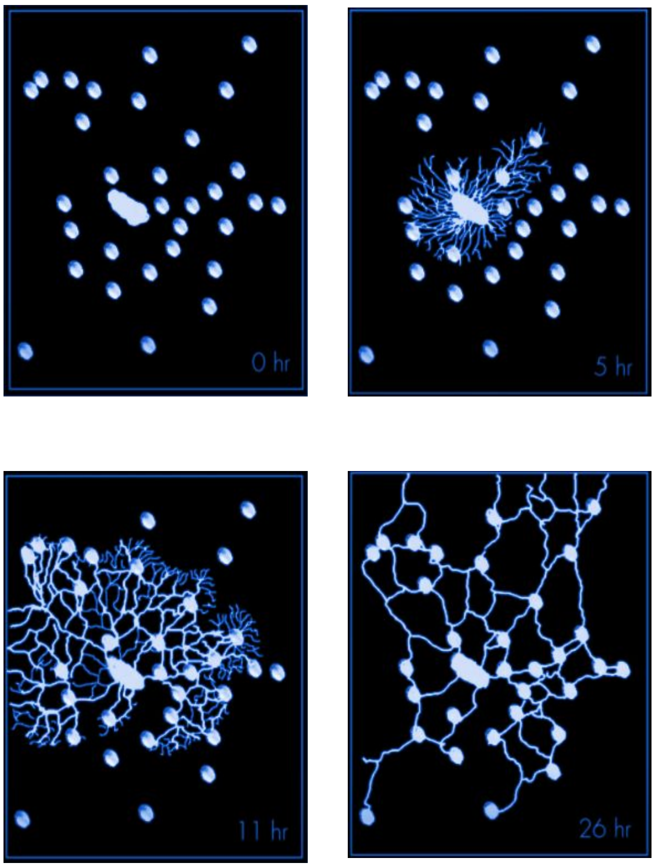
Computational Emulation

Harrison Hildebrandt



Biological Role Model

The biological role model we chose to emulate was myxomycete, more commonly known as slime mold. Slime mold spreads through spores to locate food with the help of airborne chemicals. Slime mold exists as single-celled organisms until they find an abundance of food, after which they join together and 'move' by shifting themselves into a new shape. But in this search for food, slime molds show properties that can best be summarized as 'efficient'. They move between different nodes of food surrounding them in an attempt to create a network between them where each part can be properly nourished, and connect to all other parts in the most efficient known manner possible.



Principle Extraction

Slime Mold	**Python**
Spreading through Spores	Updating Agent Position
Avoid Self-Intersecting Growth	Process "Self Avoidance"
Locate Food Source	Append Food Source Position to List
Continue Search for Other Food from Found Food	Place New Agents at Found Food Location
Converge Paths Between Food Sources	Trace Shortest Agent Paths between Food Sources
Thicken Connecting veins, Fade Unnecessary veins	Increase Mesh Pipe Radius and Reduce ARGB Alpha

Classes and Methods

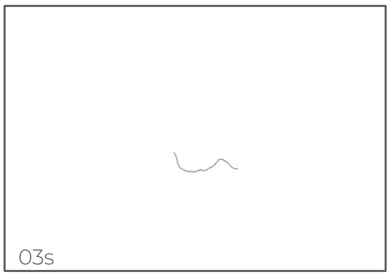
```
class ParticleSystemNetwork:
....def __Init__
....def CreateParticleSystem()
....def Update()

class ParticleSystem:
....def __Init__
....def CreateVeins()
....def SplitVeins()
....def Update()
```

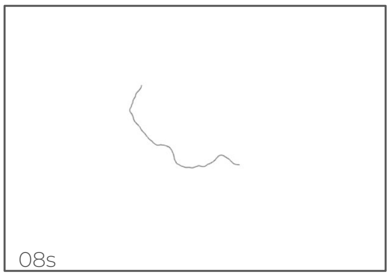
```
class Particle:
....def __Init__
....def Update()
....def ProcessVeinAvoidance()
....def ProcessAttractor()
....def ProcessInitialFoodBoundary()
....def StopVeinGrowth()
....def CreateNewSystem()
....def ProcessFoodSource()
        ....def RemoveFoodParticle()
        ....CreateParticleSystem()

FindPathBack():
```

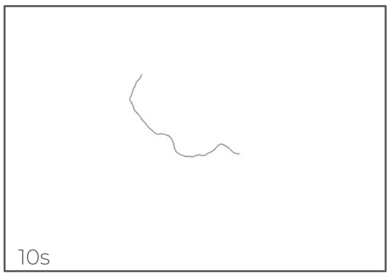
Particle.Update()

The first step in emulating slime mold growth was creating a class that copied the action of a single cell. The class 'Particle' stores the current position of the instance and a list of its previous positions as Point.3ds. When 'else' is executed, the instance runs '.Update()' which updates the particle position and appends it's previous position to the list. Originally, the .Update() only rotated the Particle.Velocity by a random double between -0.2 to 0.2 radians about the Z Axis. The agent is visualized by drawing a Polyline through its history.

```
if iReset:
....particle = Particle()

else:
....Update particle
```



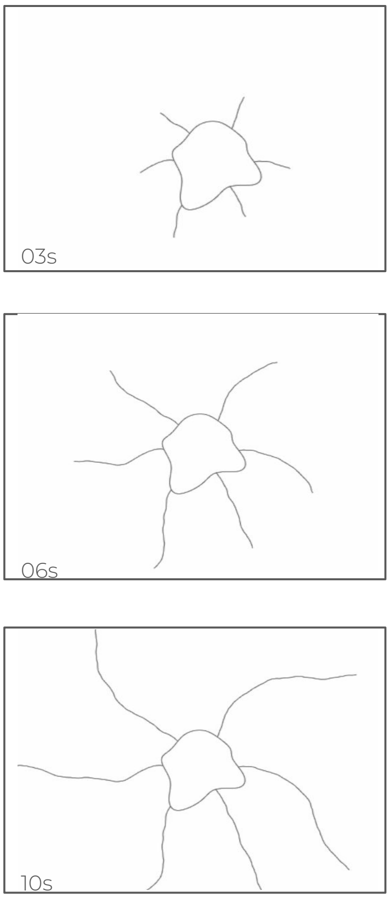




ParticleSystem.Update()

Next step was creating multiple instances of Particle which happens in the class ‘ParticleSystem’. ParticleSystem is initialized by providing the boundary for placing the particles and defining how many particles to generate in the ParticleSystem. Similar to Particle, ParticleSystem runs a .Update() method. This method cycles through all Particles in Particle System and executes their .Update() method.

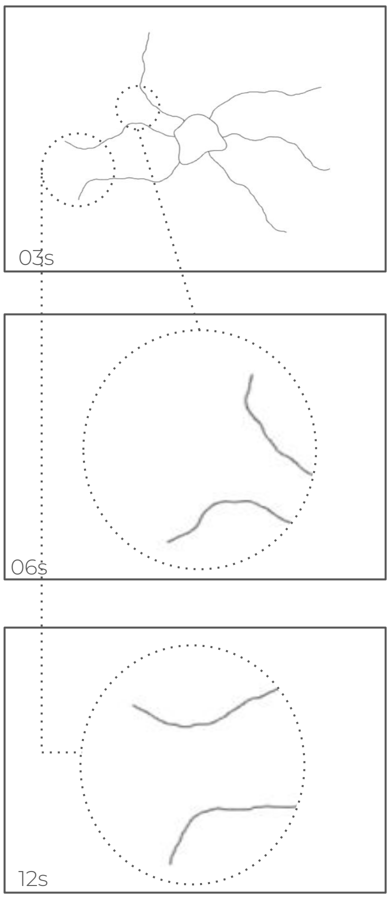
```
def __init__(self):
   for i in range(0, Input Number of Particles):
   ....Start Particle() on Food Source Boundary
   ....Append Particle() to ParticleSystem.Particles

def Update(self):
   for i in range(0, Length(ParticleSystem.Particles)):
    ....Update Particle()
```



Particle.ProcessSelfAvoidance()

The veins in the simulation were converging or self-intersecting which is not seen in slime mold. To combat the convergence, the code has the method Particle.ProcessSelfAvoidance() which is executed in Particle.Update(). The method checks if the current particle position is close to another particle or it’s history. If the particle is close to another vein, Particle.Velocity moves proportionally (to its distance away) from the vein.

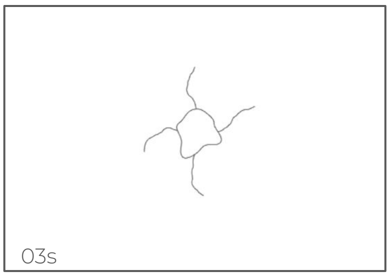
```
for Particle in Particle System:
....Find Closest Point on Nearby Veins
....Calculate Distance Vector to Closest Point

....if Particle Approaches Vein:
.......Update Particle.Velocity to MoveAway from Vein

....elif Particle Hits Vein:
.......Stop Particle.Velocity
```

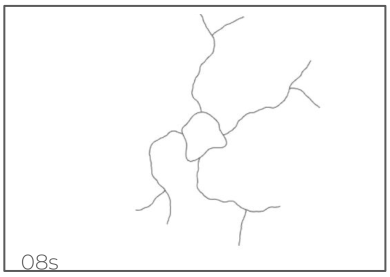


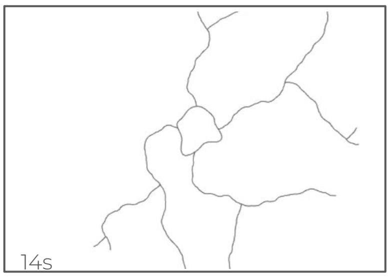
Particle.CreateNewParticle()

The next method focuses on the ‘branching’ aspect of the slime mold. Once a Particle’s ‘.Vein’ reaches a certain length, a new Particle (originating at the splitting point) is added to the Particle System. The new Particles get updated and behave like the rest.

```
if Vein.Length == SplitLength
....Create NewParticle at SplitLength
....Add NewParticle to ParticleSystem.Particles()
```


03s

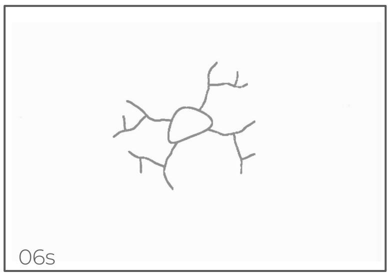

08s

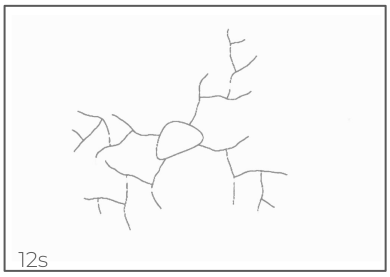

14s

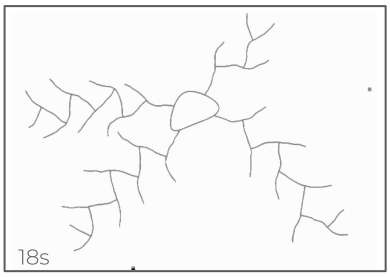
Particle.StopGrowth()

Once a single cell of slime mold goes long enough without finding food, the cell stops growing. The same logic is translated to Python. Once a Particle.Vein’s length reaches a maximum distance, the Particle.Update() sets the particle’s velocity to 0.

```
if Vein.Length > Maximum Distance without Finding Food:
....Particle.Velocity = 0
```


06s

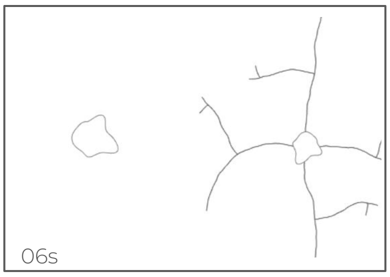

12s

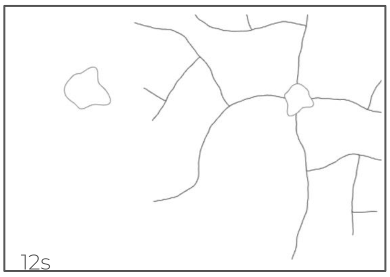

18s

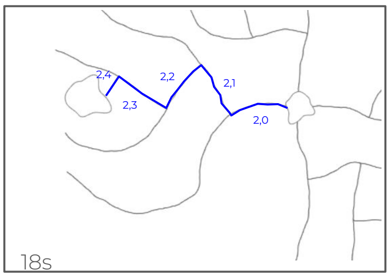
Particle.FindPathBack()

When slime mold finds a food source, it traces a path back to a food source it had previously found. The code traces back the agent's path between the sources. Each particle carries a '.TreeBranch' and a '.BranchLevel' to index it's order in the system. If a Particle with .TreeBranch: 3 and .Branch Level: 4 finds the food source, the system will know to trace back the path from that index.

```
if Particle Finds Food:
....Stop Particle.Velocity
....Tell the System it Found Food
....Find Path between Connected Food Sources
```



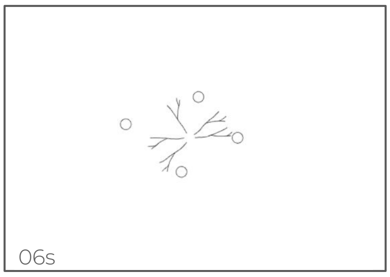


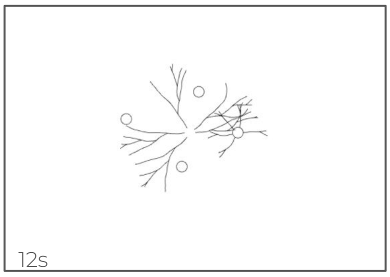


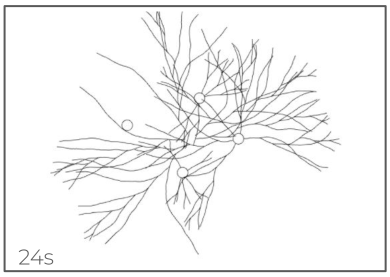
ParticleSystem.__Init__()

Once the slime mold finds one food source, it continues searching for the next. The class ParticlesystemNetwork hosts multiple ParticleSystems. Once a Particle finds a food source, a new ParticleSystem is initiated in the ParticleSystemNetwork at the food source. The result is all the food sources being connected together.

```
if Particle.Position == Food.Position:
....Create NewParticleSystem at Food.Position
....Append NewParticleSystem to ParticleSystemNetwork
```



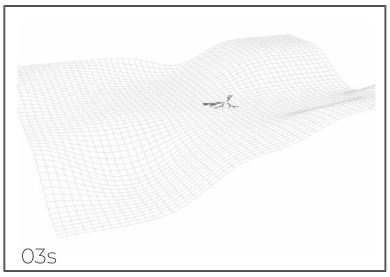


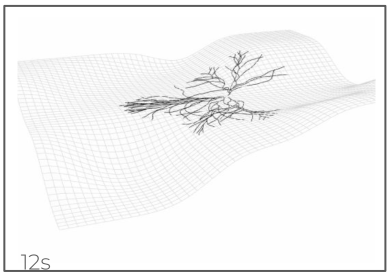


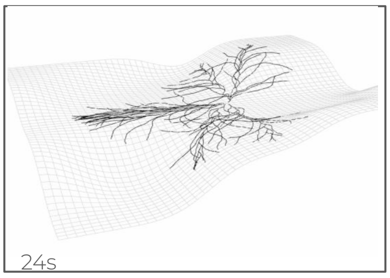
Particle.Project()

Many different formal options were explored. One option was to have the particle movement unconstrained in every axis and let the particles explore freely within a 'BoundingBox'. The other option was to project every particle and food source to a surface to emulate the slime mold growing on an uneven surface such as on a log.

```
Project Particle.Position to Surface:
Project Particle.Velocity.Tip to Surface:
Particle.Velocity = Particle.Position-Particle.Velocity.Tip

Particle.Update()
```



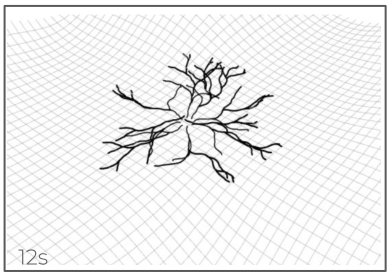


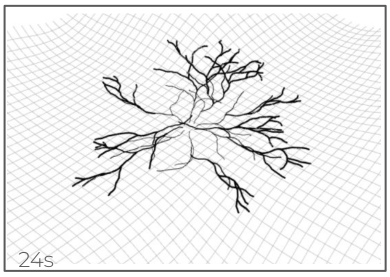


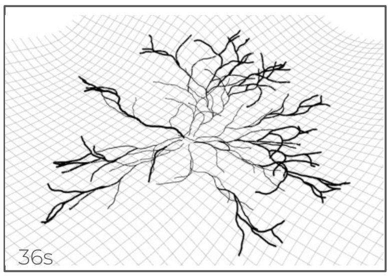
Particle.Radius()

The final adjustment to the code was to assign each particle a '.Radius' and a '.Transparency'. Growing particles would all have the same radius but particles that had reached the maximum length without finding food would start to get smaller and fade while the radius of particles that had found food would grow.

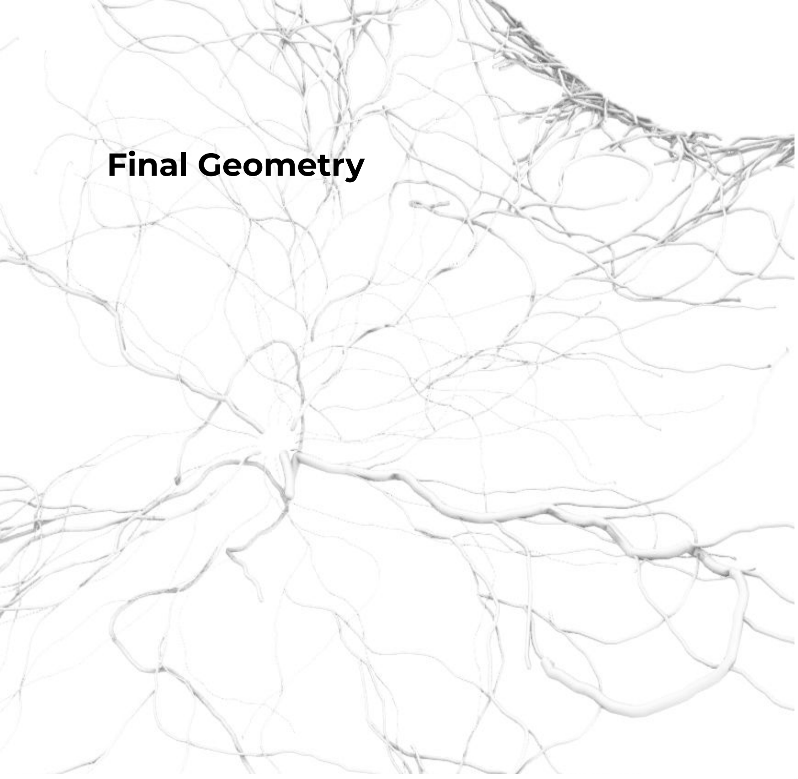
```
particle.Radius = 5

if particle == MaxLength and has not Found Food:
....particle.Radius -= 1
elif particle == FoodConnecting Vein:
....particle.Radius -= 1
```

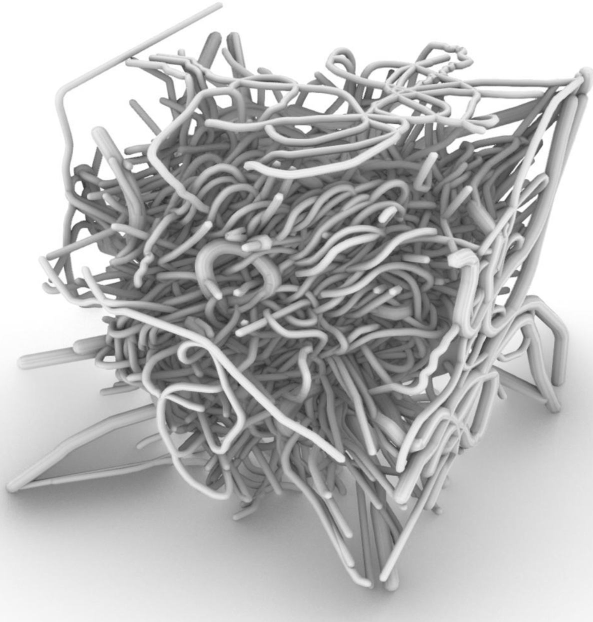


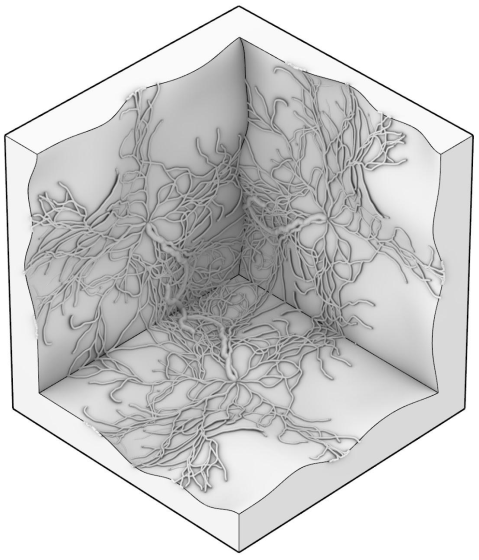


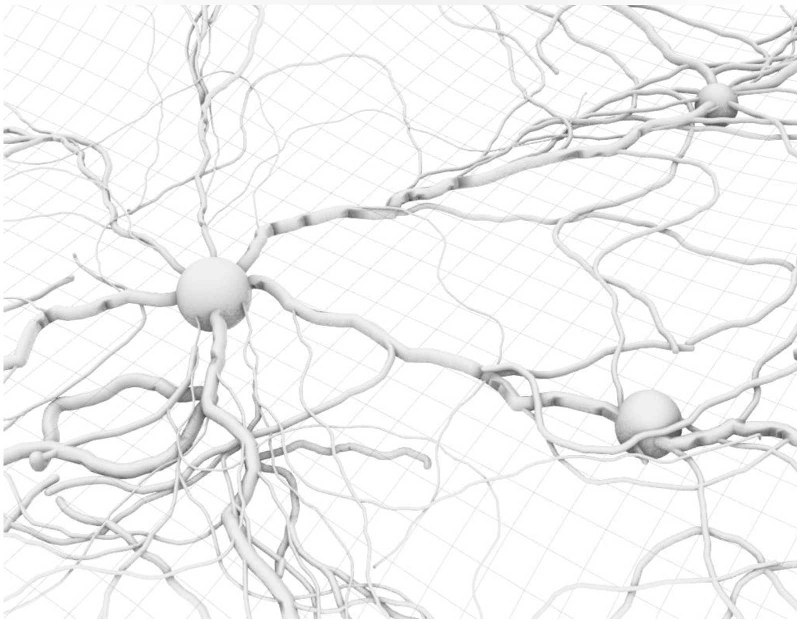


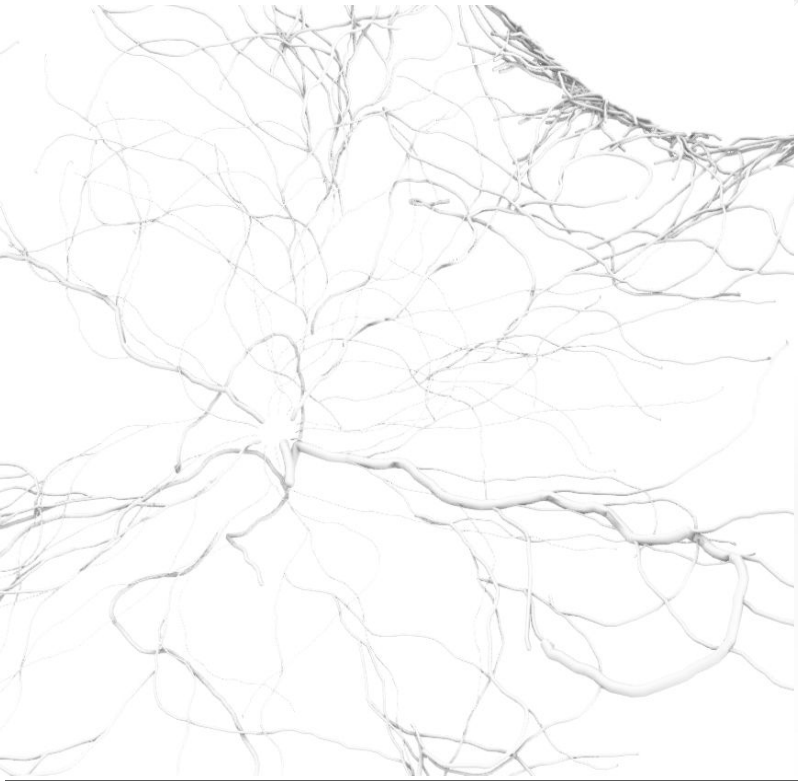
Final Geometry

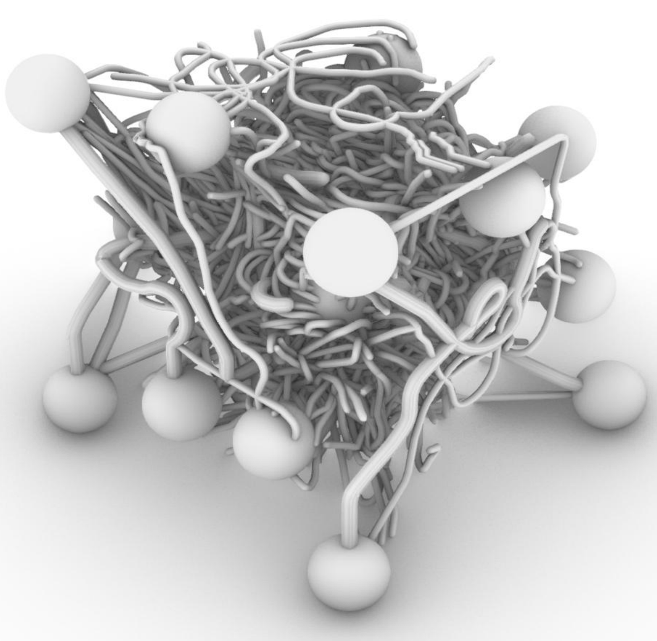




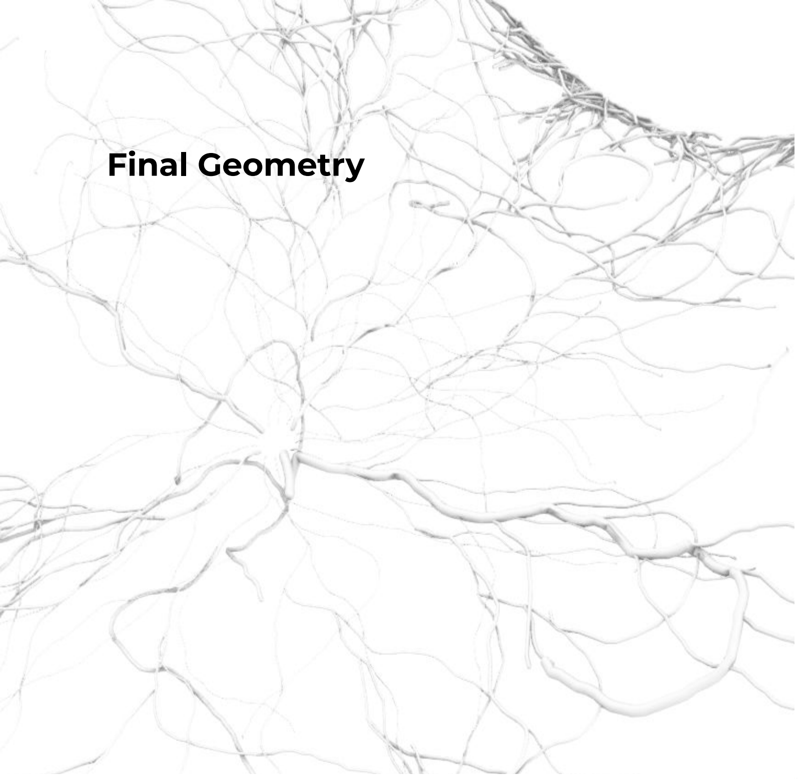


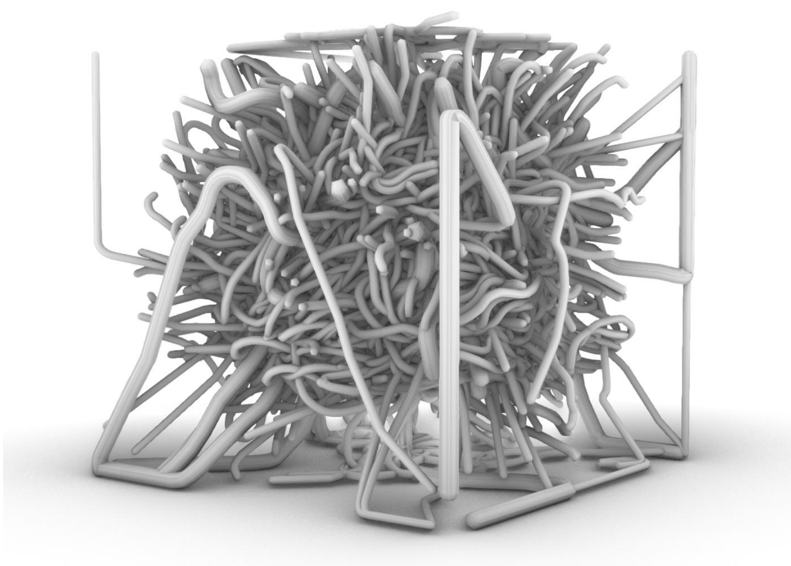


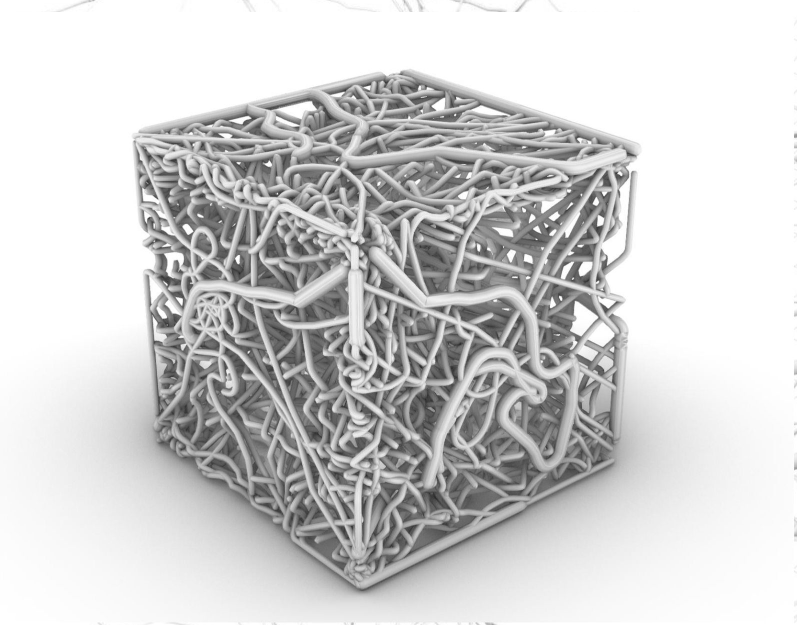


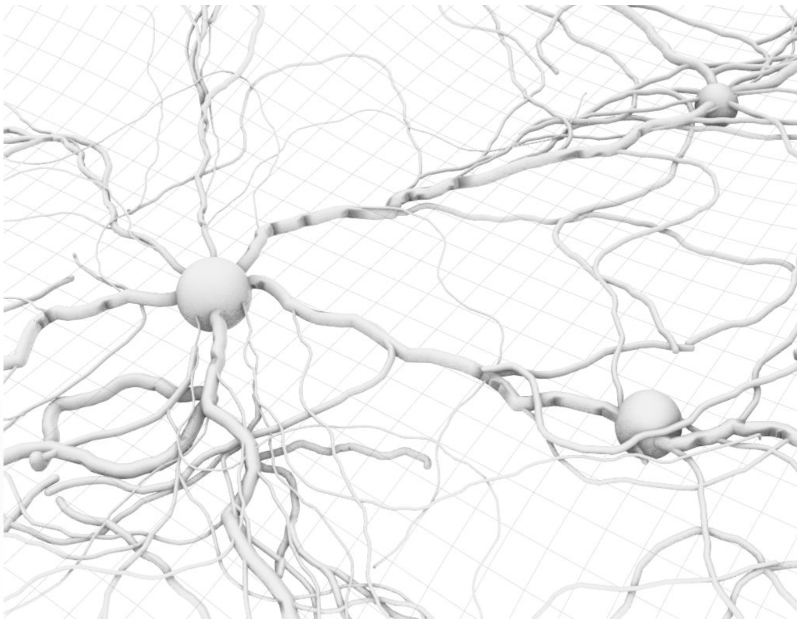


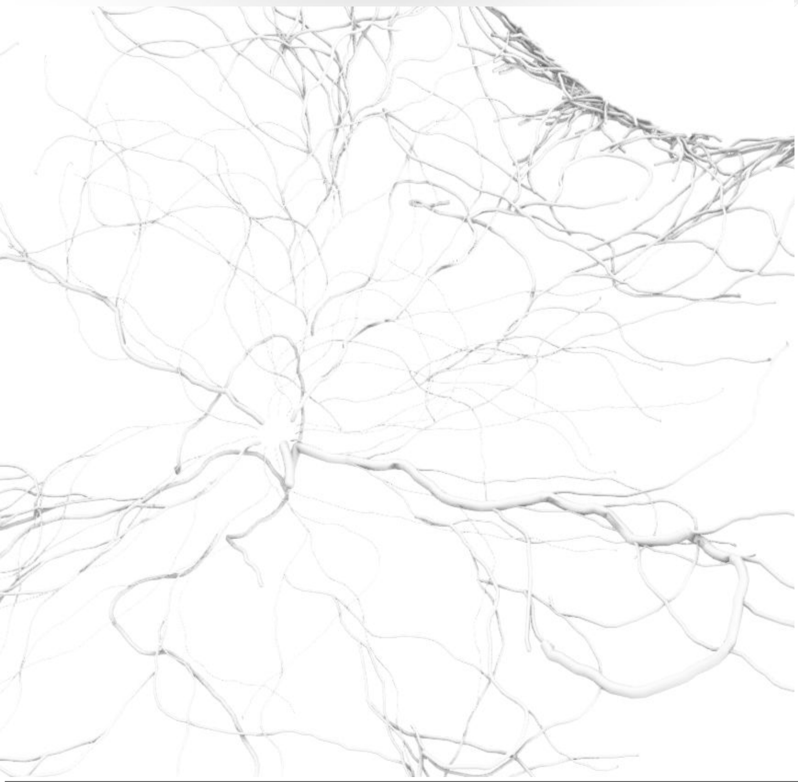
Final Geometry











References

1. https://www.sciencefriday.com/articles/dussutour-slime-mind/
2. https://maxfin13.medium.com/simulating-stimulating-slime-mold-with-graphs-d2140381f285
3. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
4. https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/
5. Page 4: Nakagaki, et al: Rules for biologically-inspired adaptive network design
6. Li, Shimin, et al. “Slime Mould Algorithm: A New Method for Stochastic Optimization.” *Future Generation Computer Systems*, Elsevier Science Div, 2020, pp. 300–323.
7. Suzuki, Y., and Toshiyuki Nakagaki. *Natural Computing and Beyond Winter School Hakodate 2011, Hakodate, Japan, March 2011 and 6th International Workshop on Natural Computing, Tokyo, Japan, March 2012, Proceedings*. Springer Japan, 2013. Adaptive Path-Finding and Transport Network Formation by the Amoeba-Like Organism Physarum pages 14-29