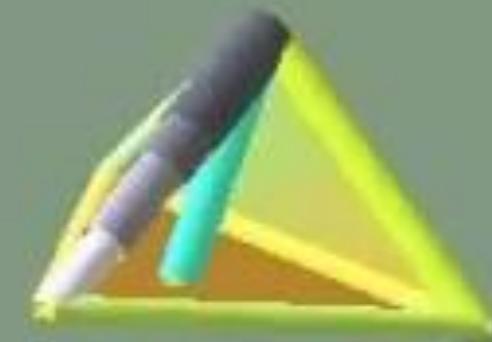
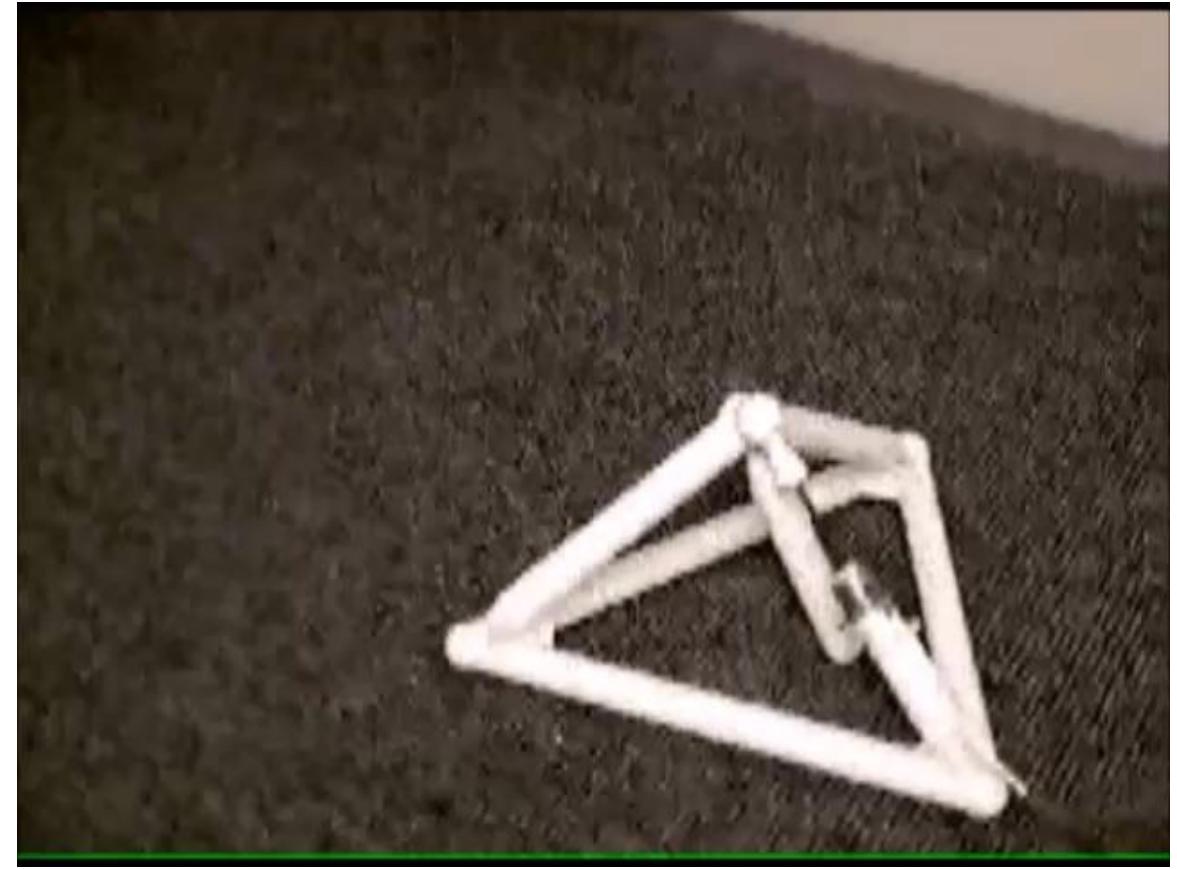
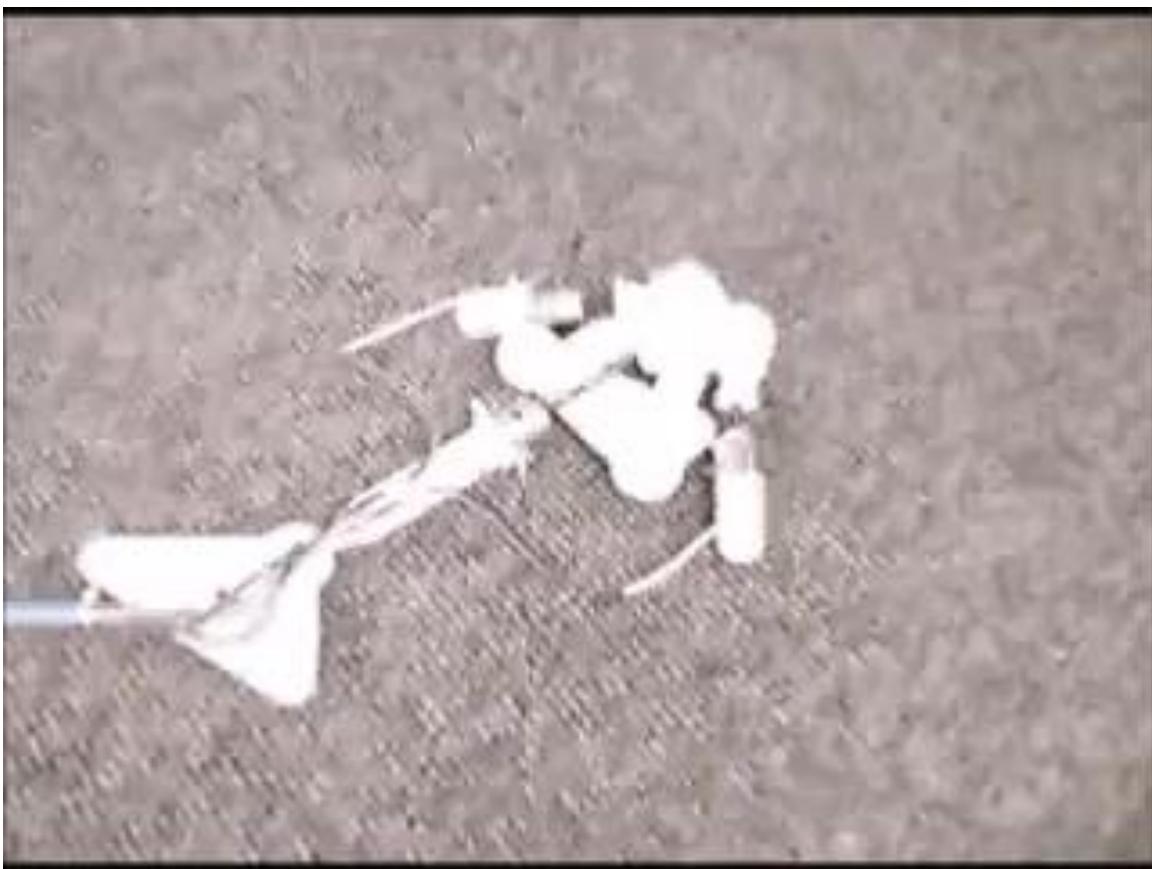


Assignment #3 Project

MECS 4510
Evolutionary Computation
Hod Lipson





MECS 4510
**Evolutionary Computation and
Design Automation**
Fall 2019

**Instructor: Hod Lipson
TA: Joni Mici**

Assignment 3

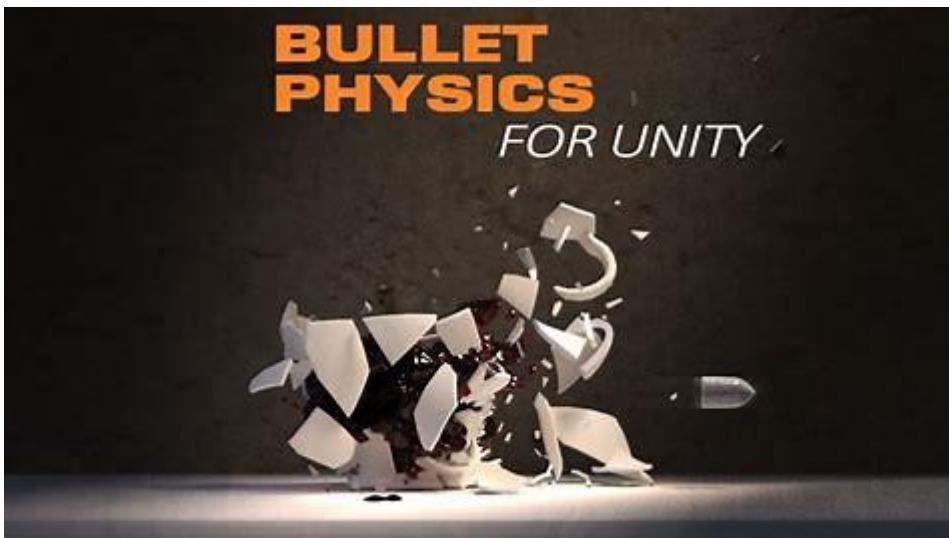
- 3a Develop a 3D physics simulator
 - Demonstrate a bouncing cube and a “breathing” cube
- 3b Evolve a fixed-shape robot
 - Manually design a robot, evolve a controller to make it move
- 3c Evolve both the shape and behavior of a robot
 - Demonstrate a moving robot of various shapes and behaviors
- Final Presentation

| | |
|--|-----------------------------|
| | Mock exam |
| | Review |
| | Project final presentations |

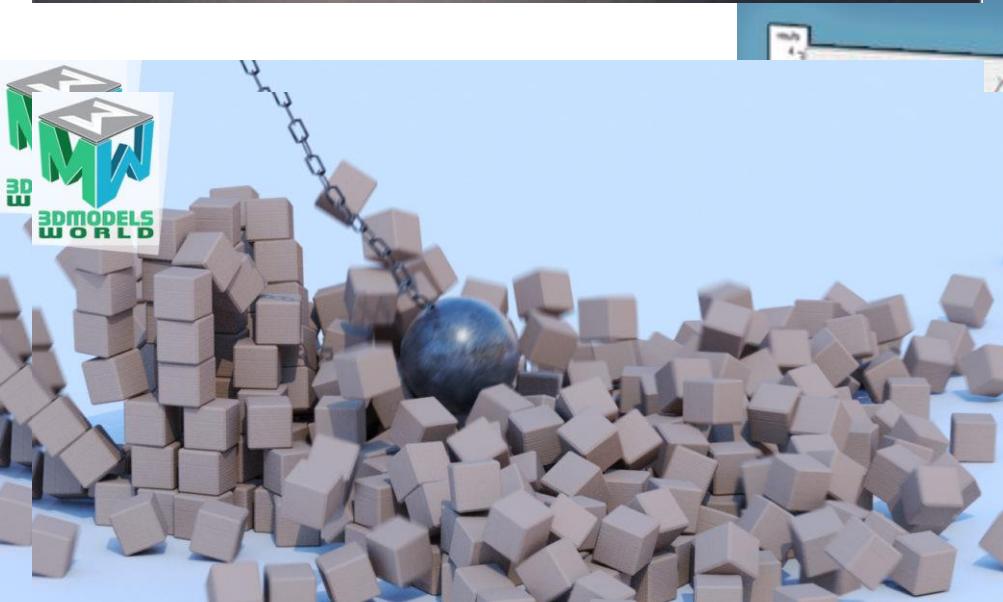
3a. Build a 3D simulator

Basic steps

Existing Physics Simulators



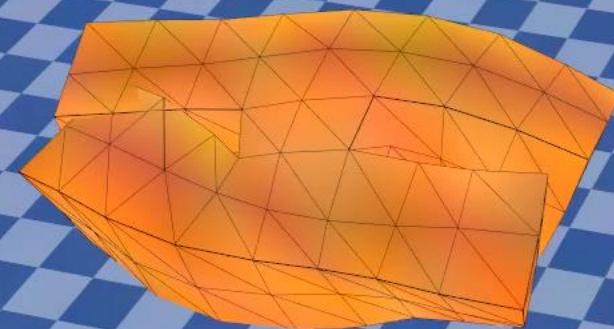
GAZEBO



A screenshot of a web-based Bullet Physics demo. On the left, there's a menu with items like "Lau Editor", "Bullet Physics", "Raycasted bath", "Quane of life", "Volcanic Simulation", "SmoothLife", and "Rotating Cube". In the center, there's a 3D view of various colored 3D objects (cubes, spheres) on a grid. To the right, there's descriptive text about Bullet Physics and a controls section with instructions for moving objects.



Cube



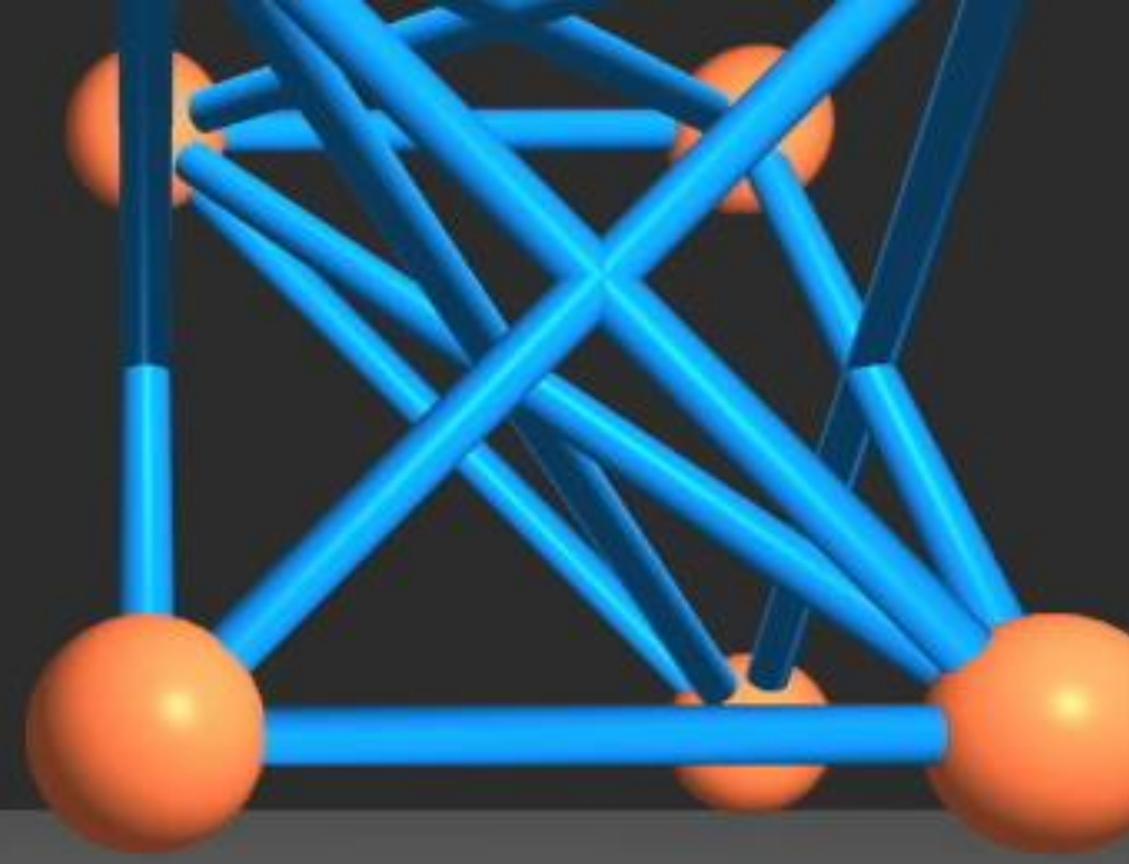
Build your own!

Why build your own physics simulator?

- Simpler than learning to use other simulator
- Simpler to integrate with your EA
- Robust (wont crash) with garbage generated by an EA
- More versatile physics
- An extra line on your resume

Basic primitives

- Masses
 - Point masses (no volume)
 - Have position, velocity and acceleration
- Springs
 - Massless
 - Connect two masses
 - Have a rest length and stiffness
 - Apply restoring forces on the masses

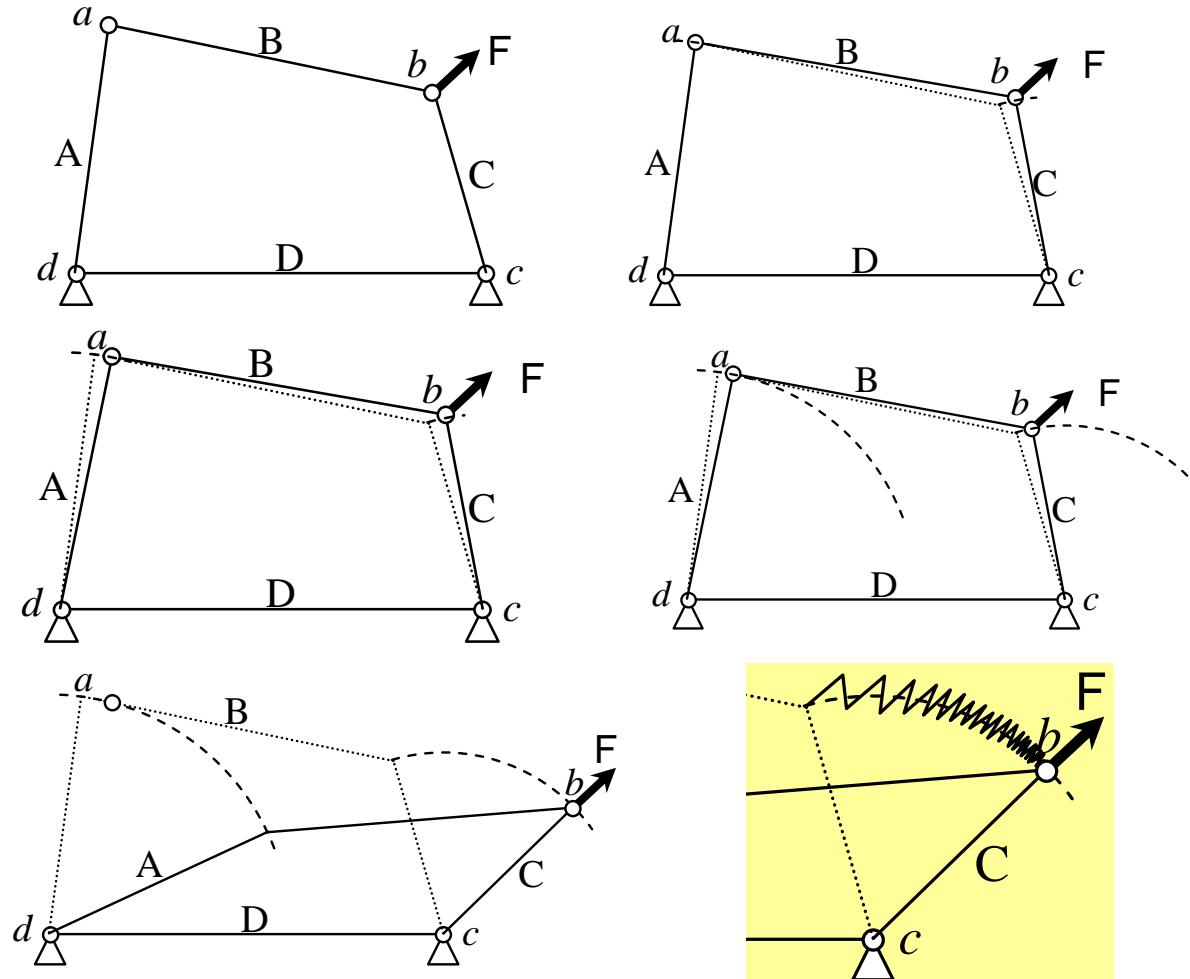


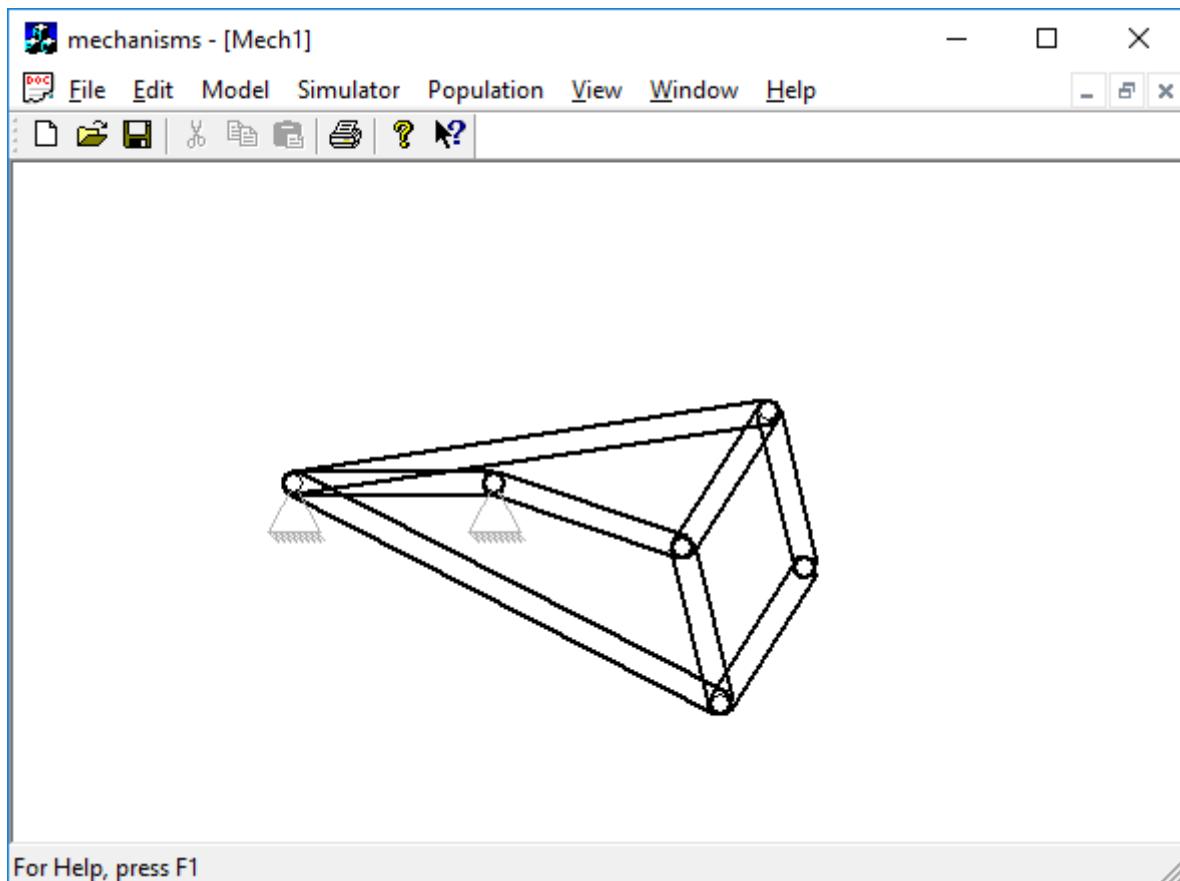
Basic simulator

- Choose a discrete time step dt
 - Not larger than a millisecond ($dt=0.001$).
 - Set global time variable $T = 0$
- At each time step:
 - $T = T + dt$
 - Interaction step:
 - Compute and tally all the forces acting on each mass from springs connected to it
 - Integration step:
 - Update position and velocity of each atom using Newton's laws of motion $F=ma$.

Stick to consistent unit system, e.g. MKS

Relaxation Simulation





First: Data structures

- Mass
 - Mass m (scalar)
 - Position \mathbf{p} (3D Vector)
 - Velocity \mathbf{v} (3D Vector)
 - Acceleration \mathbf{a} (3D Vector)
 - External forces \mathbf{F} (3D Vector)
- Spring
 - Rest length L_0 (Scalar)
 - Spring constant k (scalar) ($=EA/L$ for bars)
 - Indices of two masses m_1, m_2

Dynamic Simulation

- Time increment
 - $T = T + dt$
- Interaction step
 - For all springs:
 - Calculate compression/extension using $F=k(L-L_0)$
 - For all masses:
 - Tally forces on each mass
 - Add gravity, collision, external forces (if any)
- Integration Step
 - Calculate $a = F/m$ (from $F=ma$)
 - Update $v = v + dt*a;$
 - Update $p = p + v*dt$
- Repeat forever or until reaches equilibrium

dt = simulation timestep (0.001)

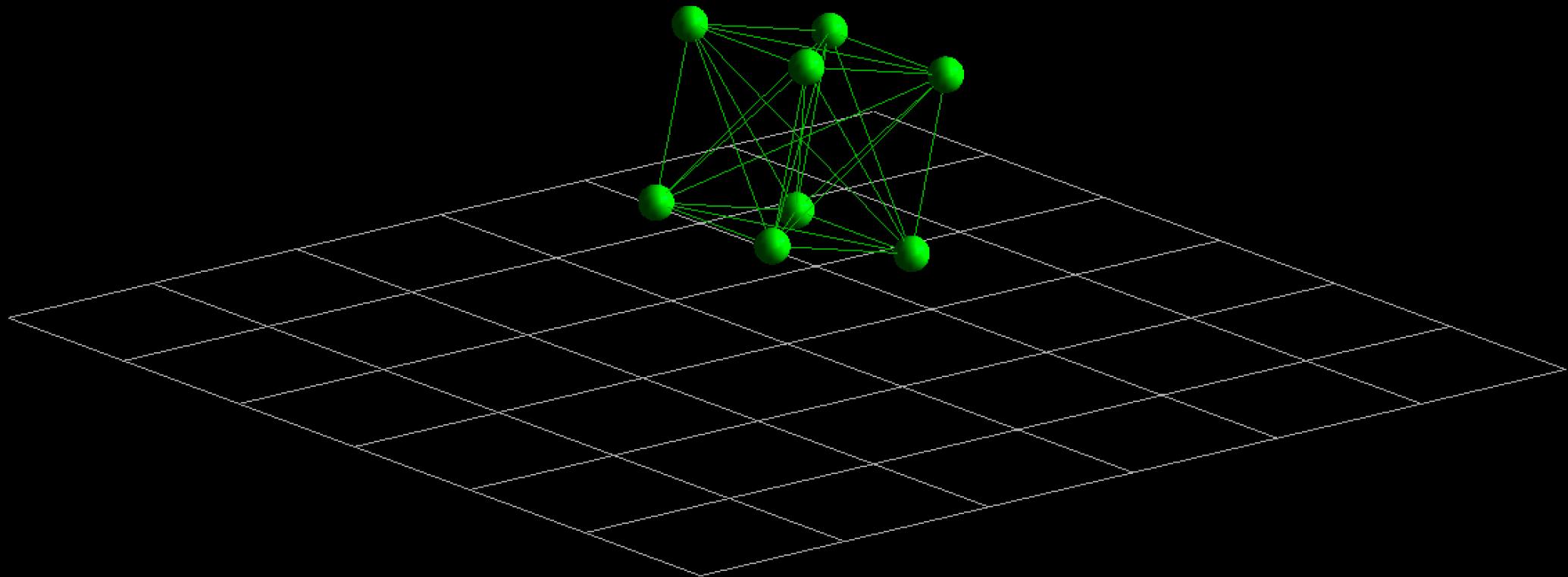
Dynamic = real time; geometry, mass, forces, velocities, accelerations

Dynamic Simulation

- Time increment
 - $T = T + dt$
- Interaction step
 - For all springs:
 - Calculate compression/extension using $F=k(L-L_0)$
 - For all masses:
 - Tally forces on each mass
 - Add gravity, collision, external forces (if any)
- Integration Step
 - Calculate $a = F/m$ (from $F=ma$)
 - Update $v = v + dt*a;$
 - If mass is fixed, set $v=0$
 - If dampening, reduce speed $v=v*0.999$
 - If colliding with ground, apply restoring force
 - Update $p = p + v*dt$
- Repeat forever or until reaches equilibrium

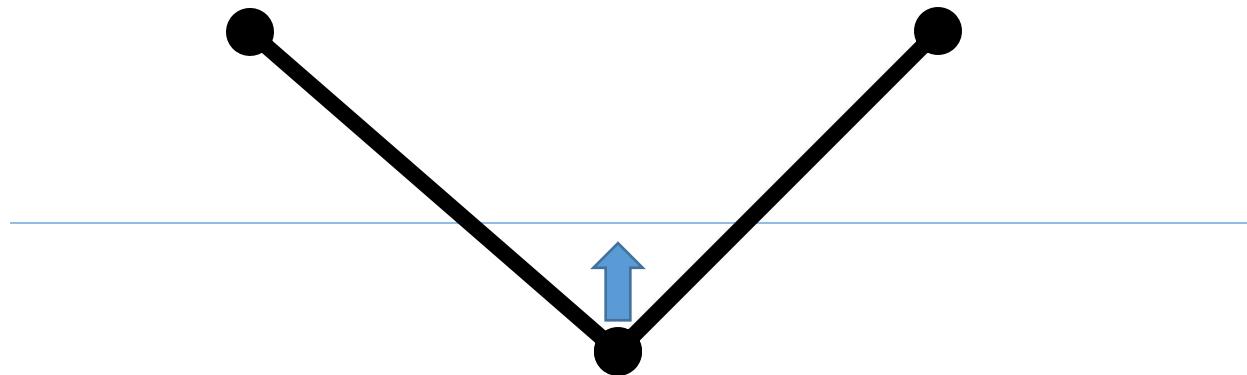
dt = simulation timestep (0.001)

Dynamic = real time; geometry, mass, forces, velocities, accelerations



How to handle collisions with the ground?

- Apply a ground reaction force F_G once a node goes below ground ($z < 0$)
 - $F_G = (0, 0, -z * K_G)$
 - K_G is the spring coefficient of the ground, e.g. 100,000



110

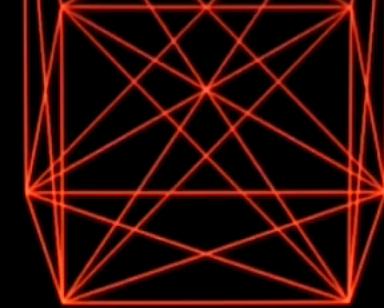
Aut

N.

Aut

Read

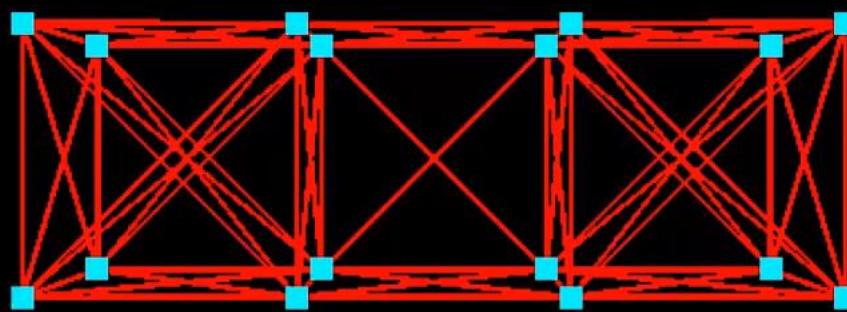
phy

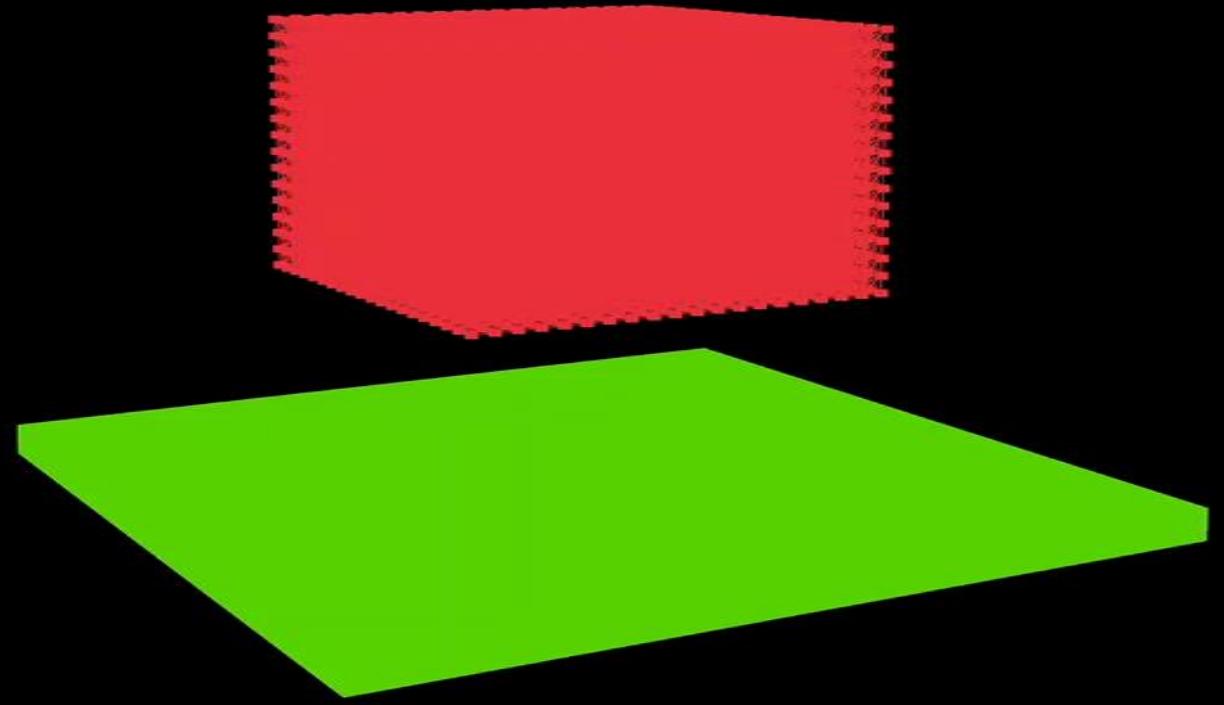


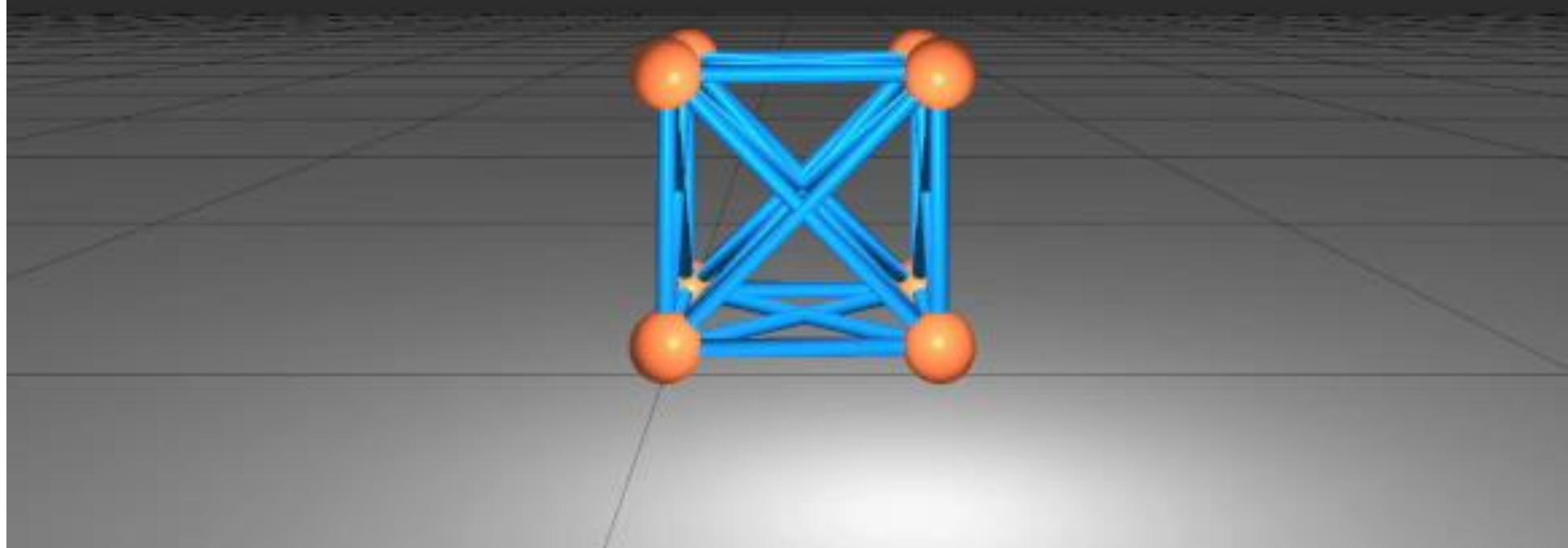
```
const double GRAVITY = 9.81;
const double damping = 0.9999;
const double DT = 0.0008; // simulation timestep
const double friction_mu_s = 1; // friction coefficient rubber-concrete
const double friction_mu_k = 0.8; // friction coefficient rubber-concrete
const double k_vertices_soft = 2000; // spring constant of the edges
const double k_ground = 200000; // constant of the ground reaction force
double omega = 10; // this is arbitrary, you can use any that you see fit
```

Create more complex structures

- Add more cubes
 - Share vertices and share springs
- Try making structures out of tetrahedra or other primitives

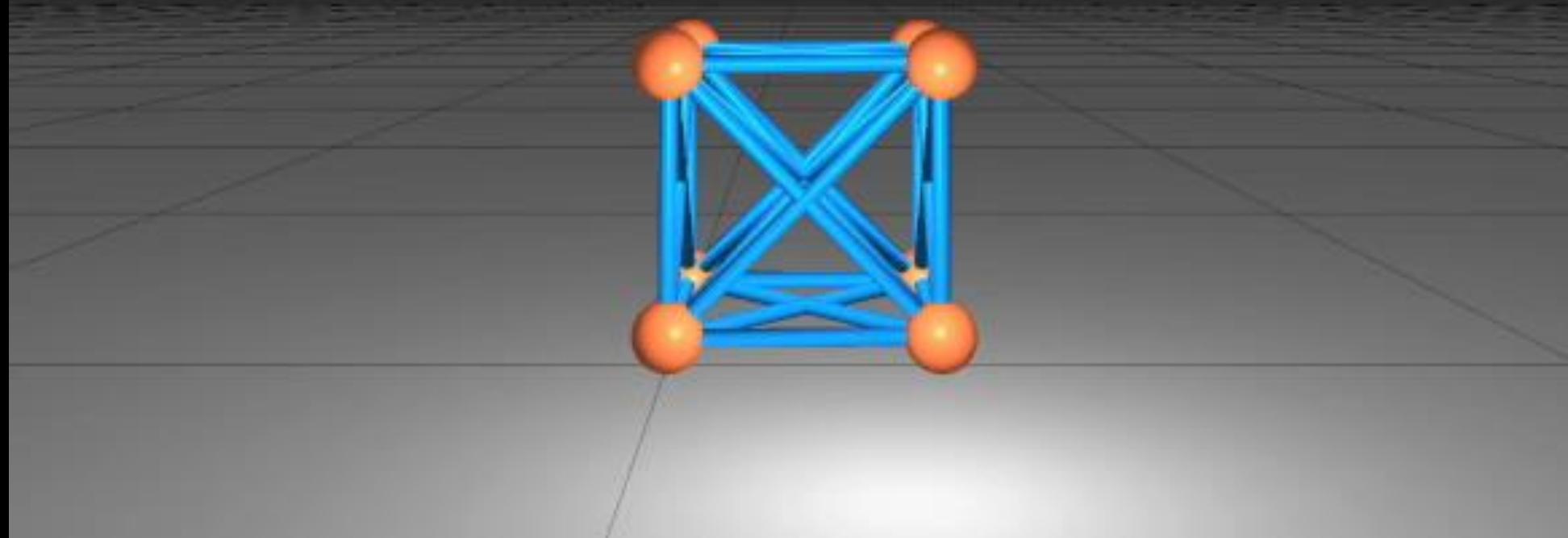






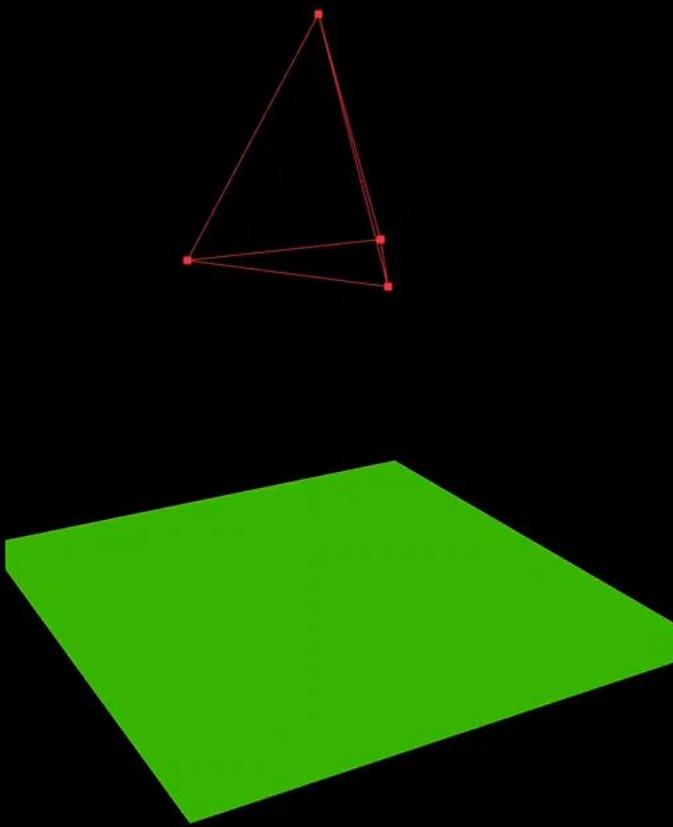
Create animated structures

- Dynamically change L_0 of various springs
 - For example, change
 - $L_0=a+b \cdot \sin(\omega T+c)$
- The coefficient ω is a global constant that determines the frequency of oscillations.
 - Setting $\omega = 1$ will result in a period of 2π sec
- Choose a, b, c randomly within a reasonable range, for each spring.
 - Keep some springs at a constant rest length by choosing $b=0$



Simulation efficiency

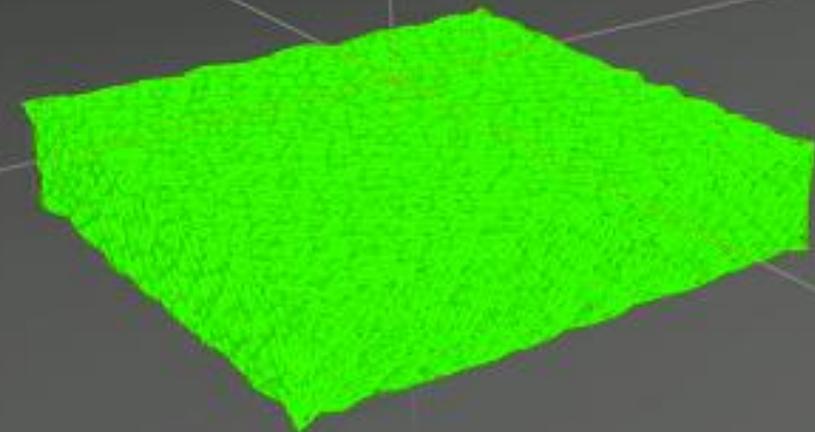
- Measured in spring evaluations per second
 - Per real second, not per simulated second
- Typical values
 - Python: 10,000 springs/sec
 - C++: 1.5 Million springs/sec per CPU core
 - CUDA/GPU: 600 Million springs/sec (TitanX 2500 cores)
- If efficiency is low, keep structures simple



WORM

Bars: 270033

Time: 0.38 s



Random Lattice

Spacing cutoff: 0.004 m

Bounds: 0.1875 x 0.0375 x 0.1875 m

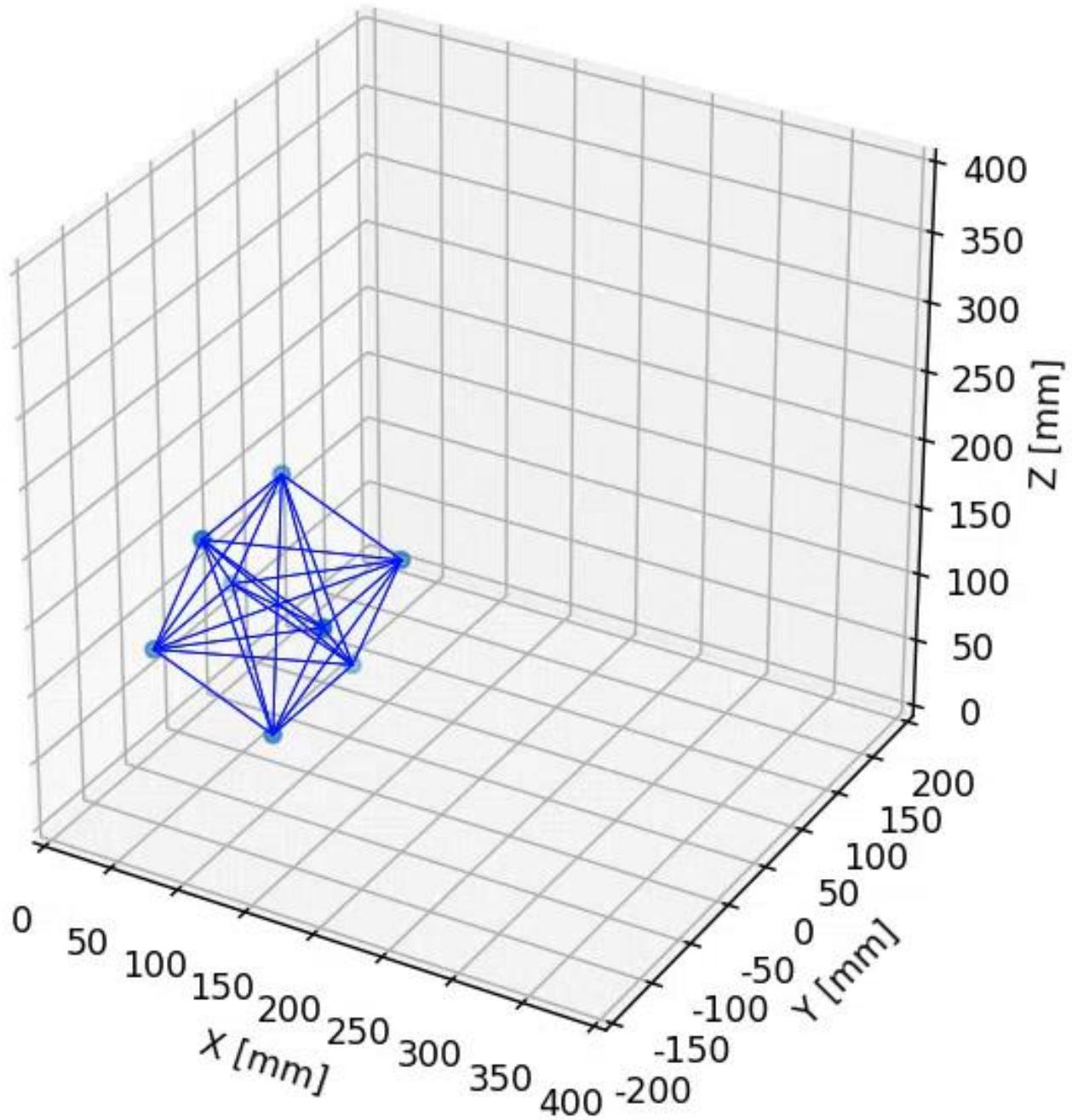
Material: NYLON

Change material property

- Set spring constant to represent hard and soft materials
 - $k = 10000$ Hard material
 - $k = 1000$ soft material
 - Change k as a function of spring length (nonlinear material)
 - Change k with time?
- Note: Higher k needs smaller simulation dt

3D Graphics

- Visualization is important
 - 3D graphics available for python, MATLAB, C++
 - Draw all springs and masses
 - Draw ground plane clearly (e.g. grid in different colors)
 - Draw external forces
- Use color for visualization, e.g.
 - Change spring color with stress (e.g. red in tension, blue in compression)
 - Change spring color depending on material type, k, etc.
- You don't need to draw the springs and masses every time step.
 - For example, you can have a time-step of $dt=0.0001$ sec but draw the cube only every 100 time-steps



VPython

3D Programming for Ordinary Mortals

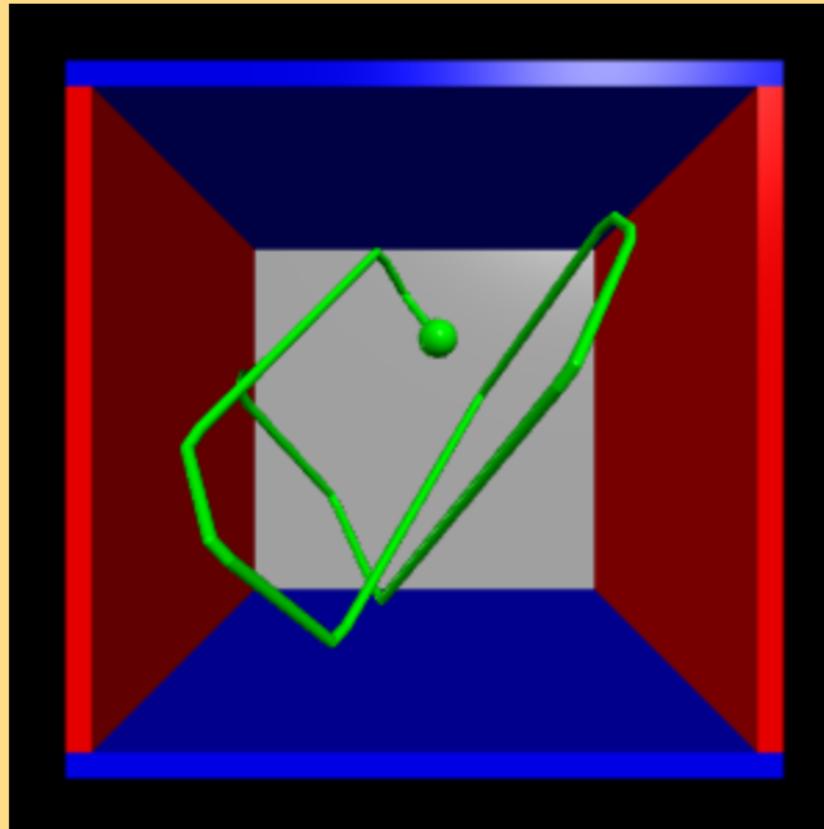
[Examples](#)

[Documentation](#)

[GlowScript VPython User Forum](#)

[VPython 7 User Forum](#)

[glowscript.org](#)



VPython makes it easy to create navigable 3D displays and animations, even for those with limited programming experience. Because it is based on Python, it also has much to offer for experienced programmers and researchers.

OpenGL resources

OpenGL is useful for rendering:

For c++ (this is probably the fastest):

GLFW: <https://www.glfw.org/>

Tutorials:

<http://www.opengl-tutorial.org/>

<https://learnopengl.com/>

For Java (slower):

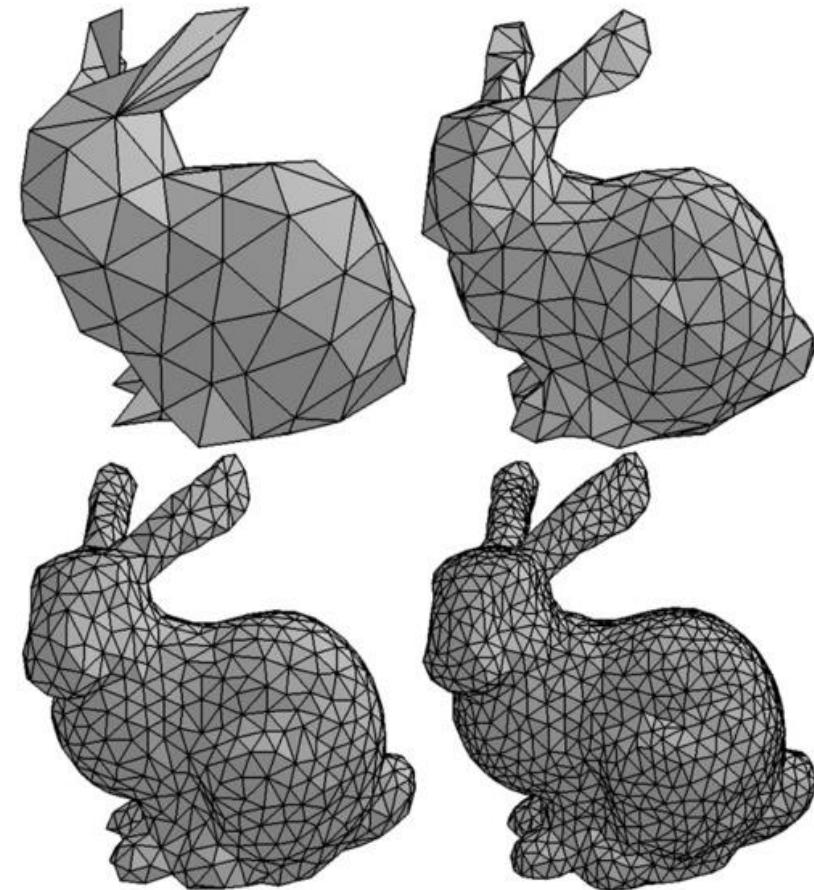
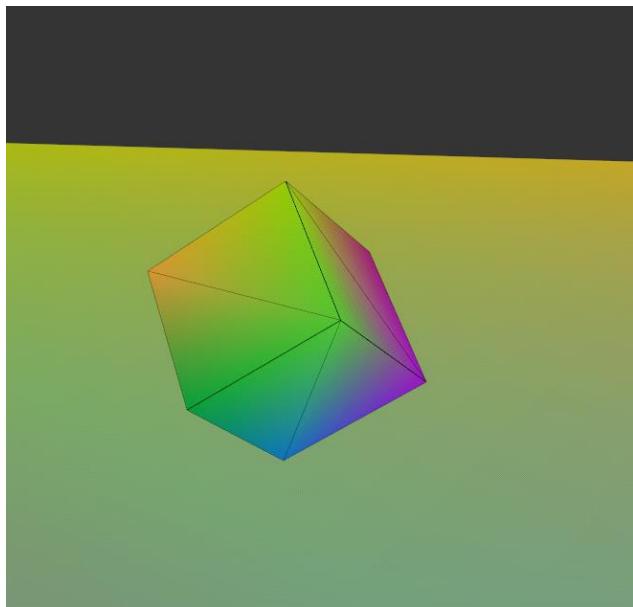
LWJGL: <https://www.lwjgl.org/>

For python (even slower):

ModernGL: <https://github.com/moderngl/moderngl>

Solid appearance

- You can make the robot look solid by shading (filling in) all the exterior triangles of the robot.



How to easily simulate a structure made of many springs and masses

Hod Lipson

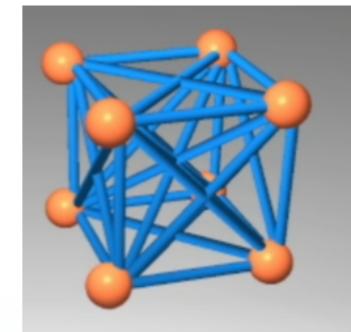
These instructions explain how to create a basic spring-mass physics simulation. You can implement this in any language. You will need to be able to plot some 3D graphics (lines and points). Assume that x, y plane is horizontal and z axis is pointing upwards.

First, let's implement a bouncing cube, like this:

<https://1drv.ms/v/s!Ap6pJEdQihYkhZAzoncgCWfIjUSQRw>

Implementation Steps

1. Create data structures for masses and springs
 - a. Create two lists (arrays): One for springs and one for masses.
 - b. Each mass needs the following attributes: m (mass, [kg]); p (3D position vector [meter]), v (3D velocity vector, meter/s), a (3D acceleration vector, meters/s²)
 - c. Each spring has the following properties: k (spring constraint, [N/m]), L_0 (original rest length, [meters]), and the indices of the two masses it connects: m_1, m_2
2. Populate data structure with an initial test structure – a 3D cube
 - a. Create an initial cube sized 0.1x0,1x01m, total weight 0.8 Kg.
 - b. Initialize corner masses with 0.1Kg masses at all 8 corners
 - c. Create 28 springs connecting all pairs of masses: 12 edges, 12 short diagonals and 4 long diagonals. Set **$k = 10,000$ (or 1000)**, for example.
 - d. Set the rest length of all springs to their initial Euclidean length and the velocities and accelerations of each mass to zero
 - e. Plot the cube in 3D to make sure it looks ok. (you will need some 3D graphics library that can draw lines, like OpenGL)
3. Set Global variables of the simulation:



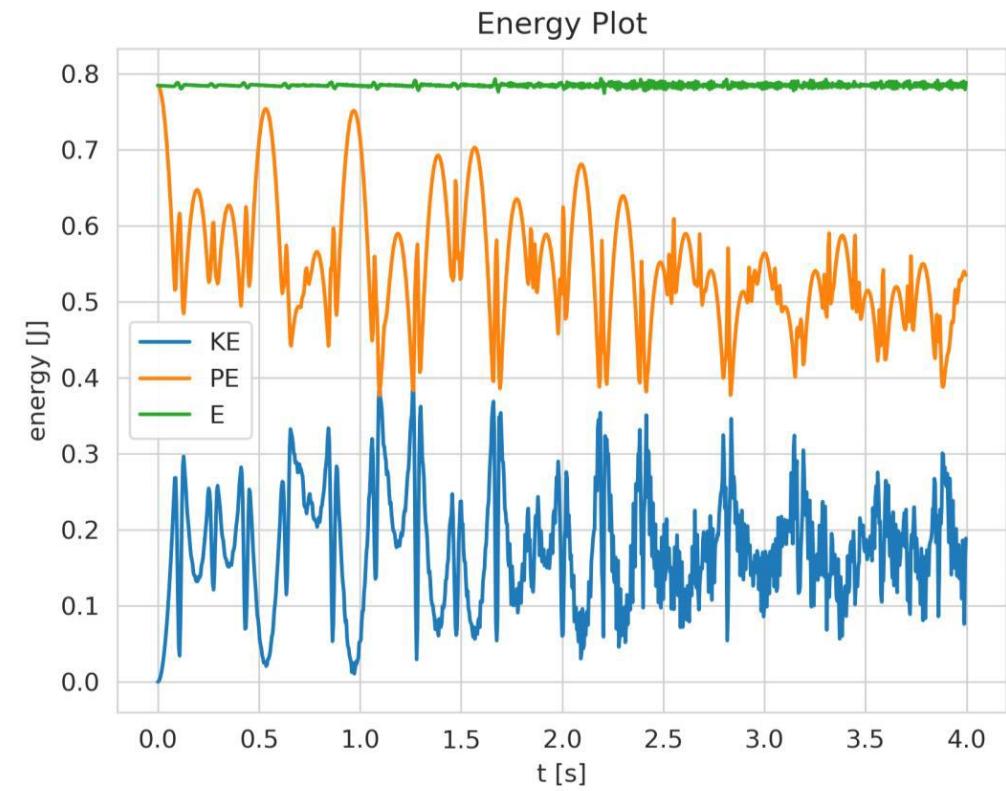
General tips

- Choosing k and dt is tricky. If the structure is too “wobbly”, k is too small. If the structure “vibrates”, k is too high. If the structure “explodes”, your timestep dt or k is too large. Generally, set dt to a small step (e.g. 1E-5) and set k to a low stiffness (e.g. 10,000), and increase them from there*.
- Once you get it to work, you can easily parallelize the loop in 4a and 4b to use all your CPU cores. For example, in C++ use OMP. If you are really ambitious, you can even use GPU/CUDA.
- Be sure to use double precision numbers for all your calculations.

*Some students reported good results with $k=100,000$ and $dt = 0.0008$

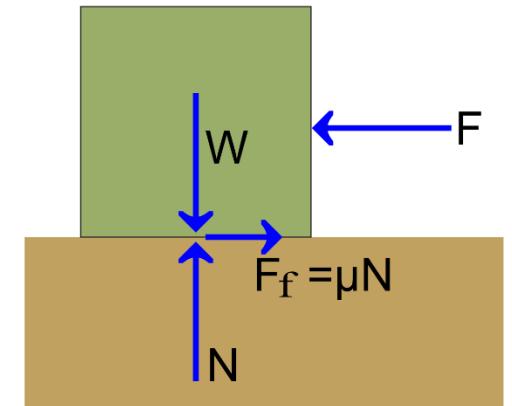
Validation

- If there is no damping or friction, energy should remain constant
 - Plot the total energy of the cube as function of time (kinetic energy, potential energy due to gravity, potential energy in the springs, as well as the energy related to the ground reaction force). The sum should be nearly constant. If it is not constant, you have some bug



Add friction

- When a mass is at or below the ground, you can simulate friction and sliding. Assume μ_s is a coefficient of friction and μ_k is the kinetic coefficient
- Calculate the horizontal force $F_H = \sqrt{F_x^2 + F_y^2}$ and the normal force F_n
- If $F_n < 0$ (i.e. force pointing downward) then:
 - If $F_H < -F_n * \mu_s$ then zero the horizontal forces of the mass
 - If $F_H \geq -F_n * \mu_s$ then reduce the horizontal force of the mass by $\mu_k * F_n$
- Update the mass velocity based on the total forces, including both friction and ground reaction forces



The screenshot shows a 3D rendering application interface. At the top, there is a menu bar with options: File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help. Below the menu is a toolbar with icons for project management, file operations, and search. The main area is divided into several panes:

- Project:** Shows the project structure with files like CMakeLists.txt, language_support.cmake, main.cpp, ColorFragmentShader.fragmentshader, TransformVertexShader.vertexshader, and README.md.
- Code Editor:** Displays the main.cpp file containing GLSL code for initializing a OpenGL window and creating a shader program.
- Event Log:** Shows build logs with timestamps and duration.
- Run:** Shows the current run configuration set to "example".
- Structure:** Shows the file structure of the project.
- File View:** Shows the contents of main.cpp.

The central part of the interface displays a 3D scene. A perspective camera is looking down at a cube. The cube has a color gradient texture, transitioning from yellow at the top to red at the bottom. It is positioned on a checkered floor plane. The background is dark, and the overall interface is a light gray color.

```
glPointSize(10);

// Create and compile our GLSL program from the shaders
GLuint programID = LoadShaders( "TransformVertexShader.vertexshader", "ColorFragmentShader.fragmentshader" );

Shader shaderCube("TransformVertexShader.vertexshader", "ColorFragmentShader.fragmentshader");
GLuint programID = shaderCube.ID;
```

```
float aaa[16]=
main
```

```
vertices p =
-0.0183013 0.05 0.168301
-0.0683013 0.05 0.0816987
-0.0683013 -0.05 0.0816987
-0.0183013 -0.05 0.168301
0.0683013 -0.05 0.118301
0.0683013 0.05 0.118301
0.0183013 0.05 0.0316987
0.0183013 -0.05 0.0316987
```

```
3:19 AM Build finished in 8 s 768 ms
3:20 AM Build finished in 8 s 617 ms
3:21 AM Build finished in 8 s 695 ms
3:21 AM Build finished in 8 s 5645 ms
3:22 AM Build finished in 8 s 659 ms
3:23 AM Build finished in 1 s 310 ms
```

```
454:28 LF+ UTF-8 Git: master Context: example [D]
```

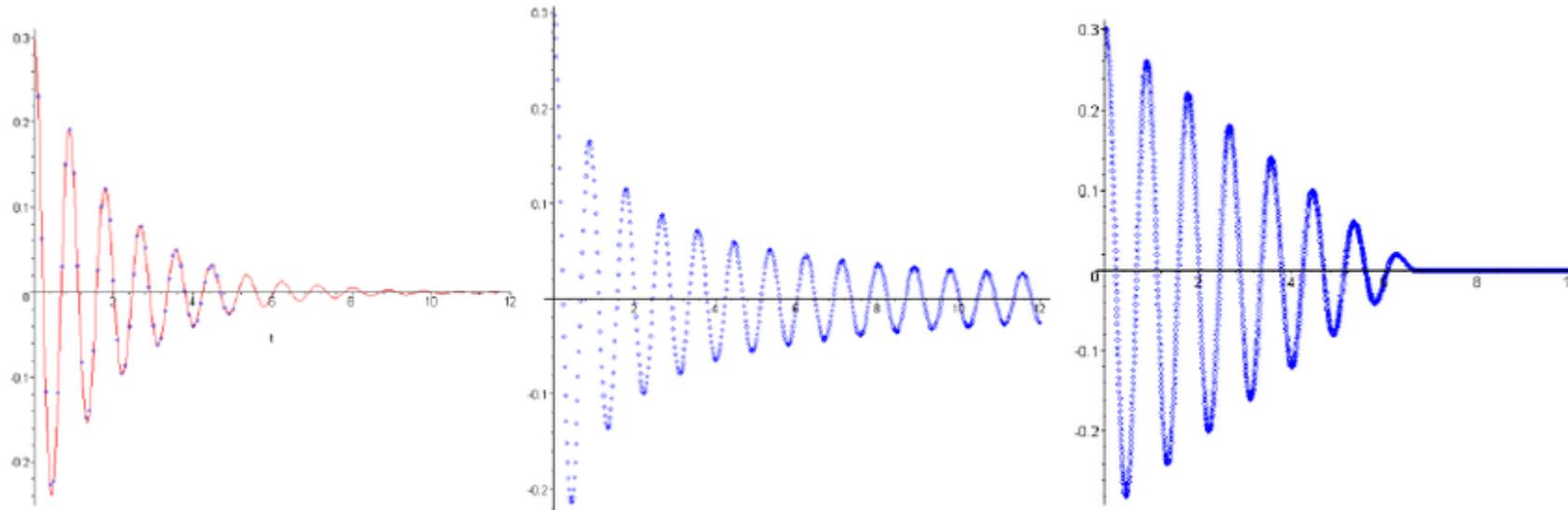


Coefficients of Friction for Common Surfaces

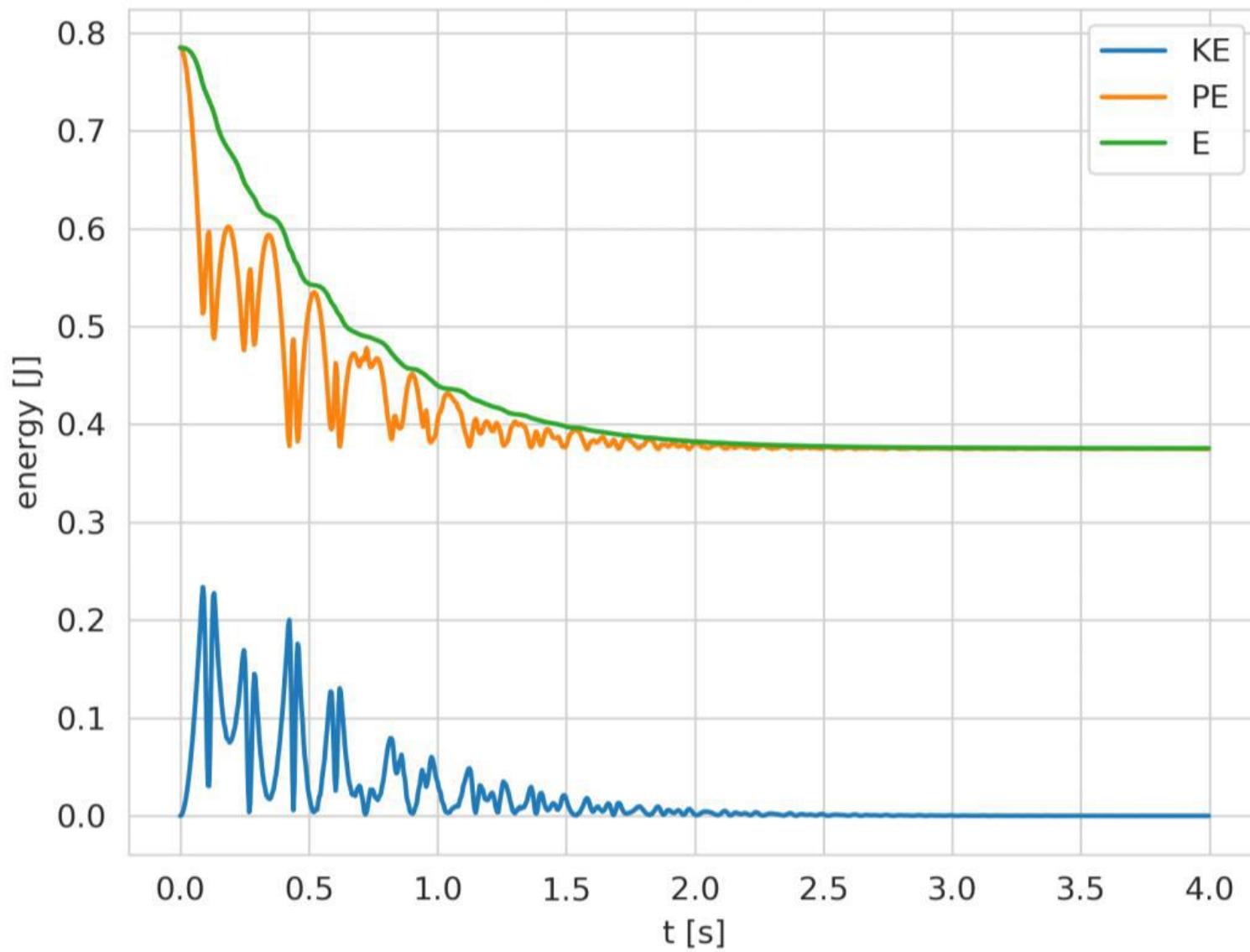
| Materials | μ_s value | μ_k value |
|-----------------------------|---------------|---------------|
| Steel on Steel | 0.74 | 0.57 |
| Aluminum on Steel | 0.61 | 0.47 |
| Copper on Steel | 0.53 | 0.36 |
| Rubber on Concrete | ~1.0 | 0.8 |
| Wood on Wood | 0.25 – 0.50 | 0.2 |
| Glass on Glass | 0.94 | 0.4 |
| Waxed wood on wet snow | 0.14 | 0.10 |
| Waxed wood on dry snow | 0.01 | 0.04 |
| Metal on Metal (lubricated) | 0.15 | 0.06 |
| Ice on Ice | 0.10 | 0.03 |
| Teflon on Teflon | 0.04 | 0.04 |
| Synovial joints in Humans | 0.01 | 0.003 |

Add dampening

- Velocity dampening
 - Multiply the velocity \mathbf{V} by 0.999 (or similar) each time step
 - Helps reduce vibration, wobbling, and snapping



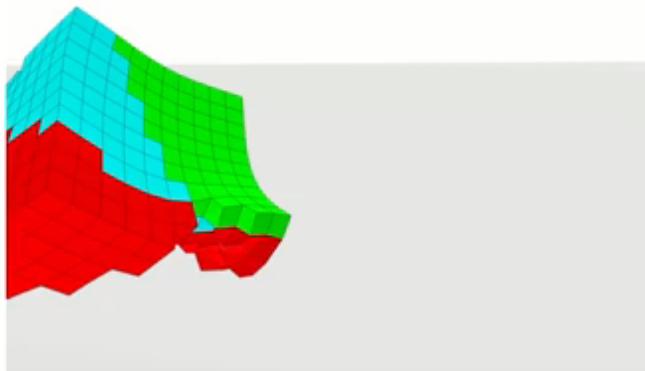
Energy Plot,damping=0.999



Add dampening

- Velocity dampening
 - Multiply the velocity V by 0.999 (or similar) each time step
 - Helps reduce vibration and wobbling
- **Add water/viscous dampening**
 - **Apply drag force $F_D=cv^2$ in the opposite direction of v (c is drag coef).**

Recently soft robots have been evolved
to run, and to grow like plants



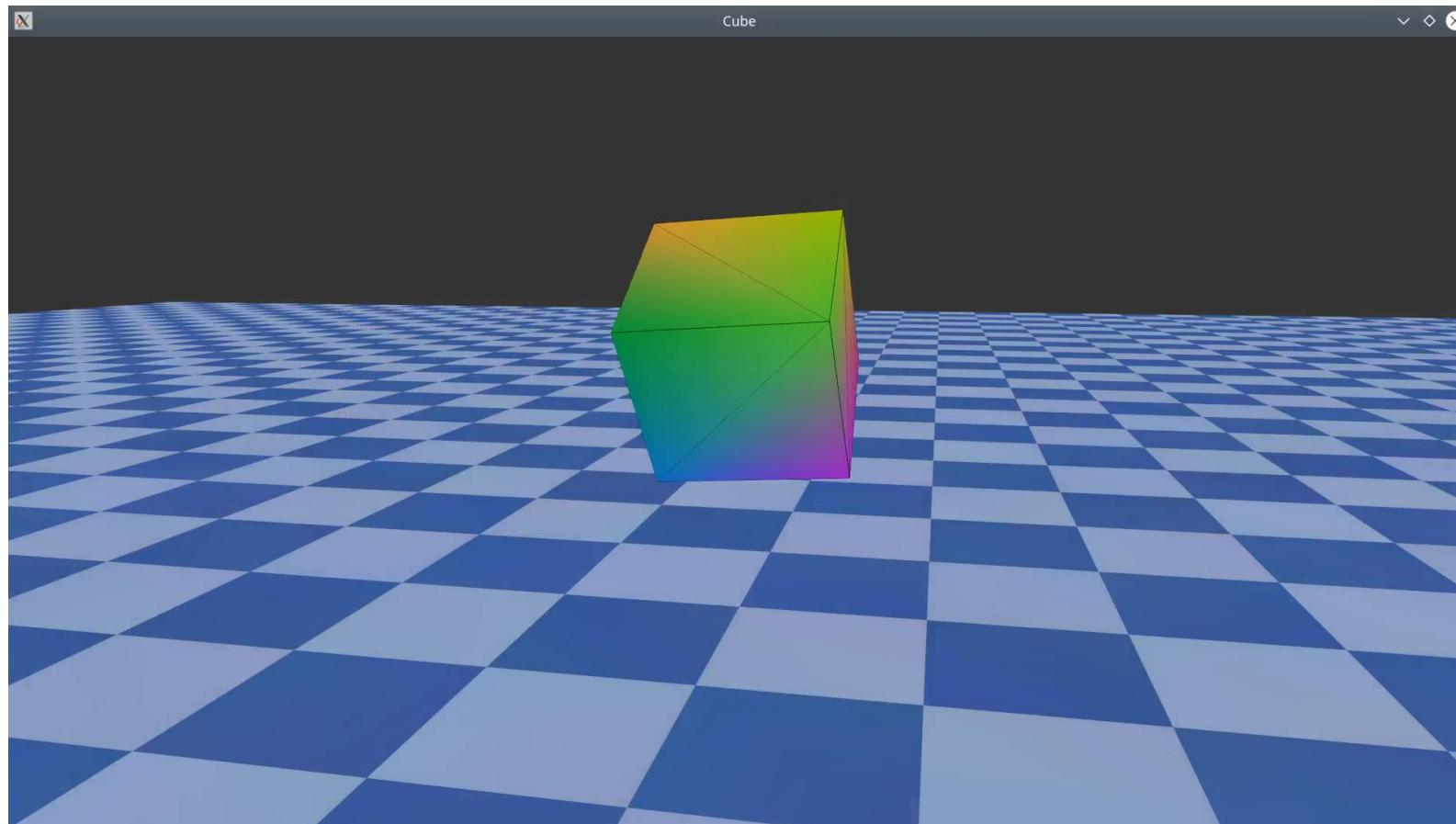
Cheney et al., 2013



Corucci et al., 2016

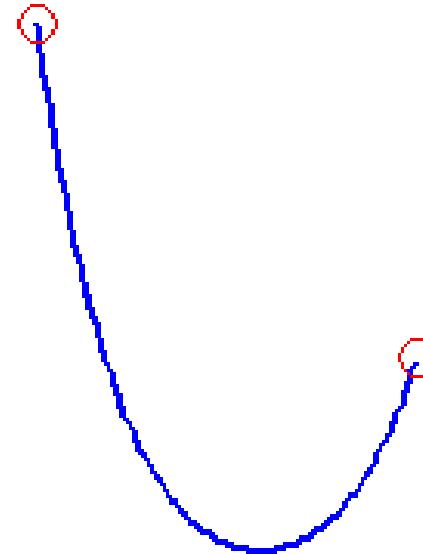
Anchors

- Anchor points by setting $v=0$
- Constrain points to a curve or surface, e.g. set $v_x=0$



Add nonlinear behavior

- Replace $F=k(L-L_0)$ with some nonlinear behavior
- Cable
 - $F=k(L-L_0)$ if $L>L_0$
 - $F=0$ if $L\leq 0$



Final submission for 3a: Breathing cube

- For each spring set k and $L_0 = a+b\sin(\omega t+c)$
 - ω is global frequency. a,b,c,k are spring parameters (k is spring coef)
 - Cycle period is $2\pi/\omega$

Summary:

3a: Create and demonstrate a physics simulator

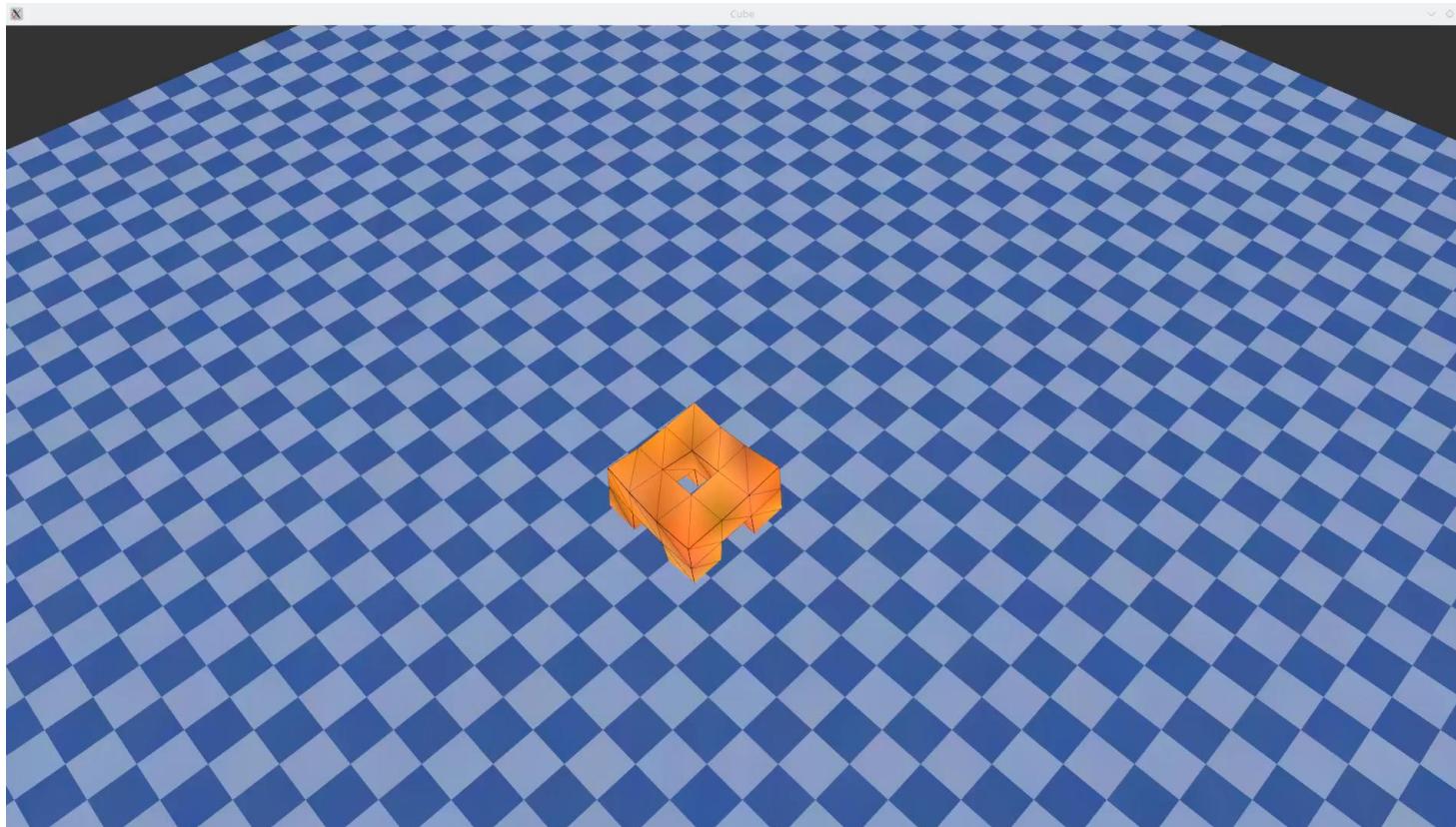
1. Cover page includes all information
2. General quality of the report (grammar, layout)
3. Video of bouncing cube with at least two bounces
4. Video of breathing cube with at least two breathing cycles provided
5. Cube bounces correctly
6. Cube launched with a slight spin (creating complex bounce)
7. Multiple cubes dropped simultaneously (up to 10) in random orientations
8. Description of all simulation parameters provided
9. Description of all breathing parameters provided
10. All Energy curves provided
11. Energy plot for bouncing cube is constant
12. Simpler problem(s) tested for debugging, e.g. tetrahedron.
13. Cube faces and floor are shaded to have solid appearance
14. Post video of cube on piazza (provide screenshot and link)

Assignment 3b

Evolve a robot with a fixed morphology

Assignment 3b

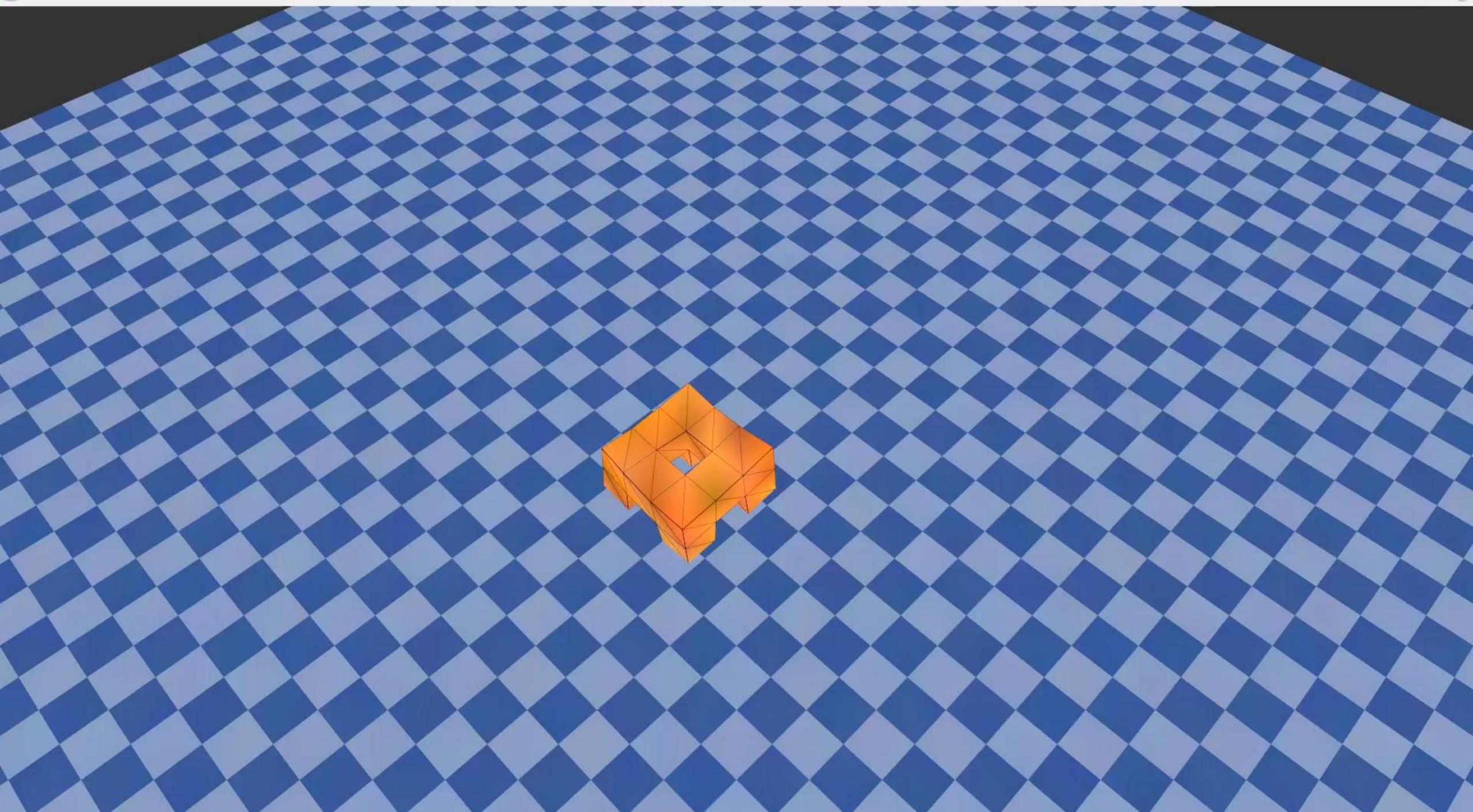
- Parametrically evolve spring rest lengths coefficients to make a manually designed robot move



X

Cube

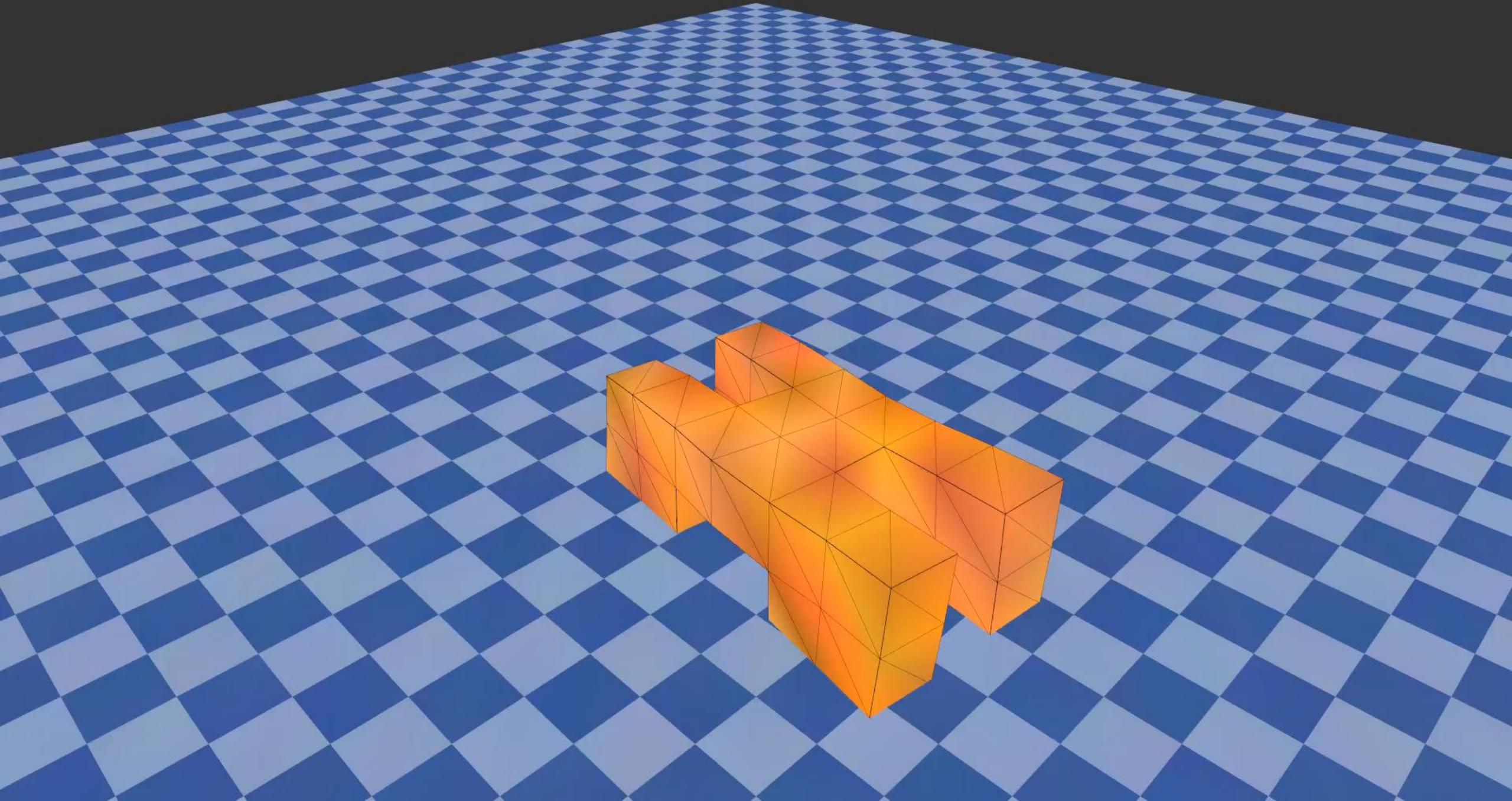
X



X

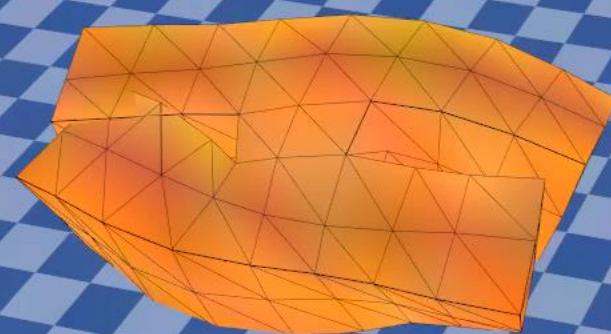
Cube <2>

X



X

Cube



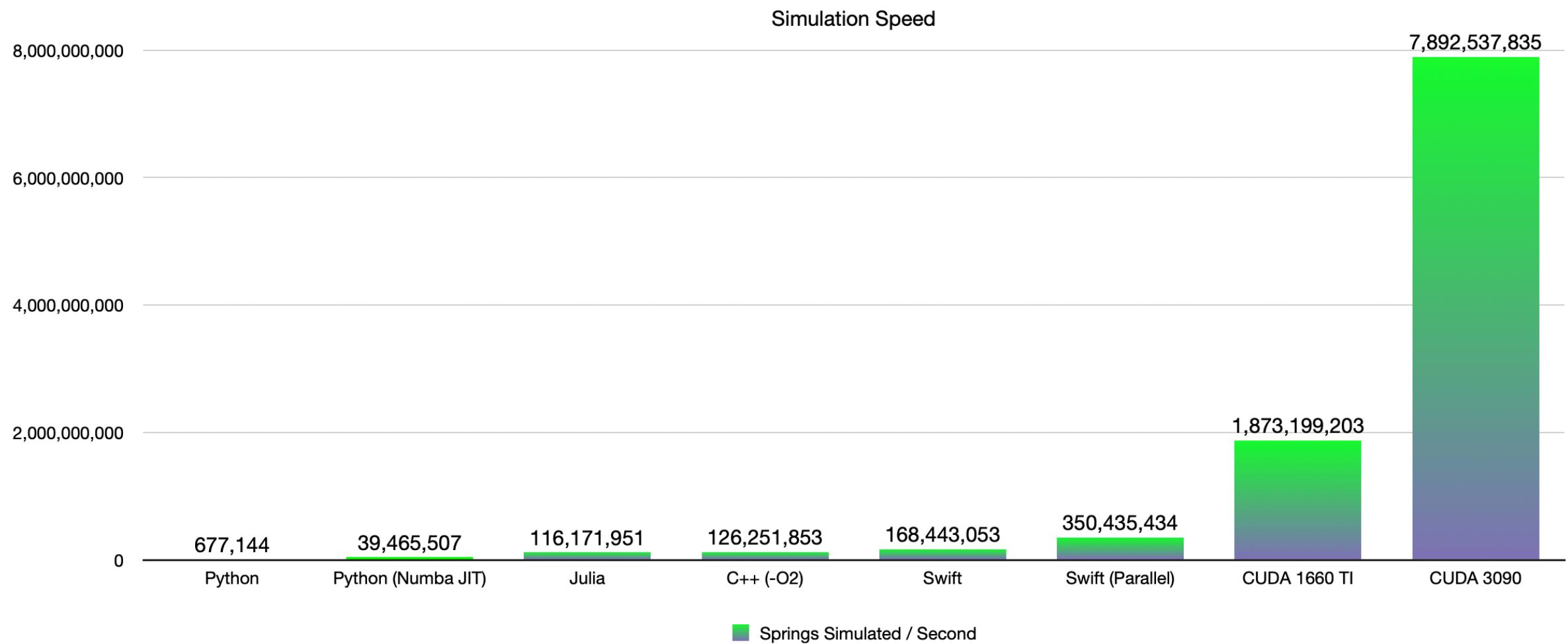


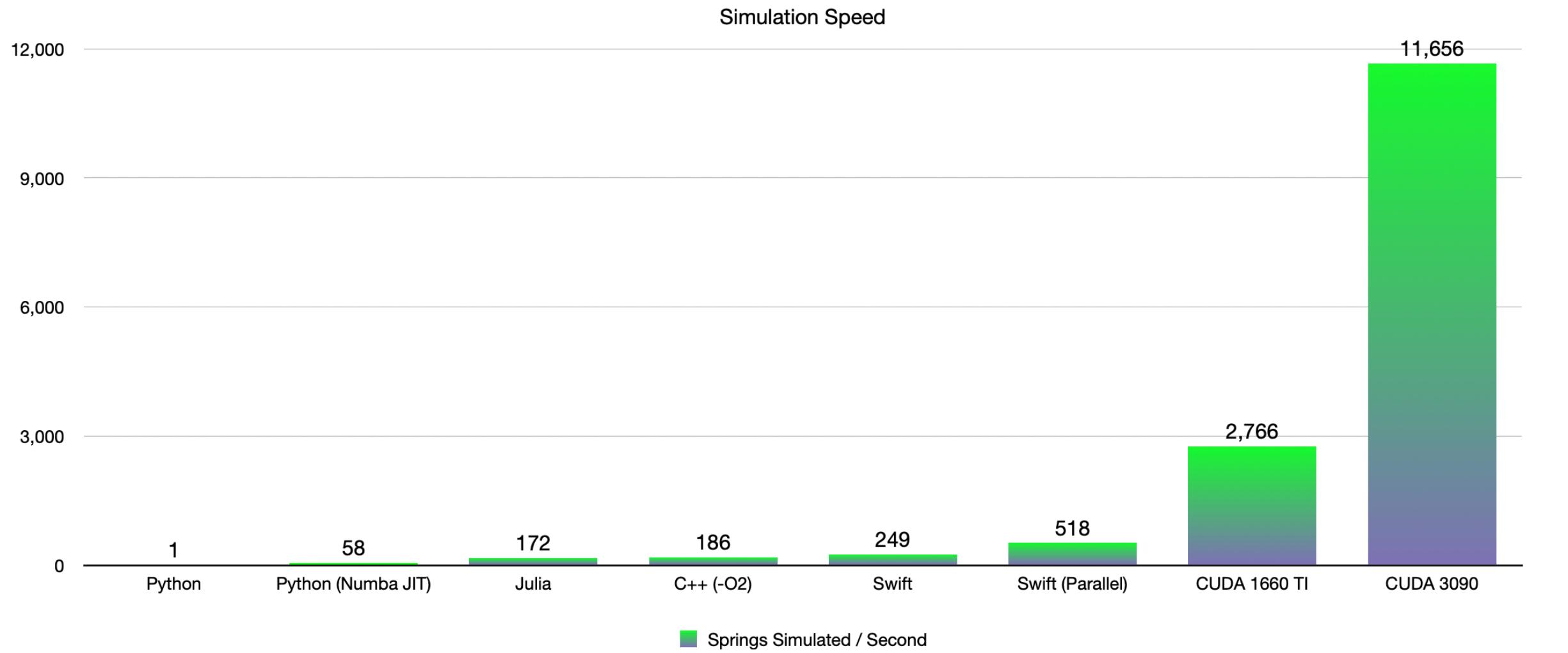
Cube



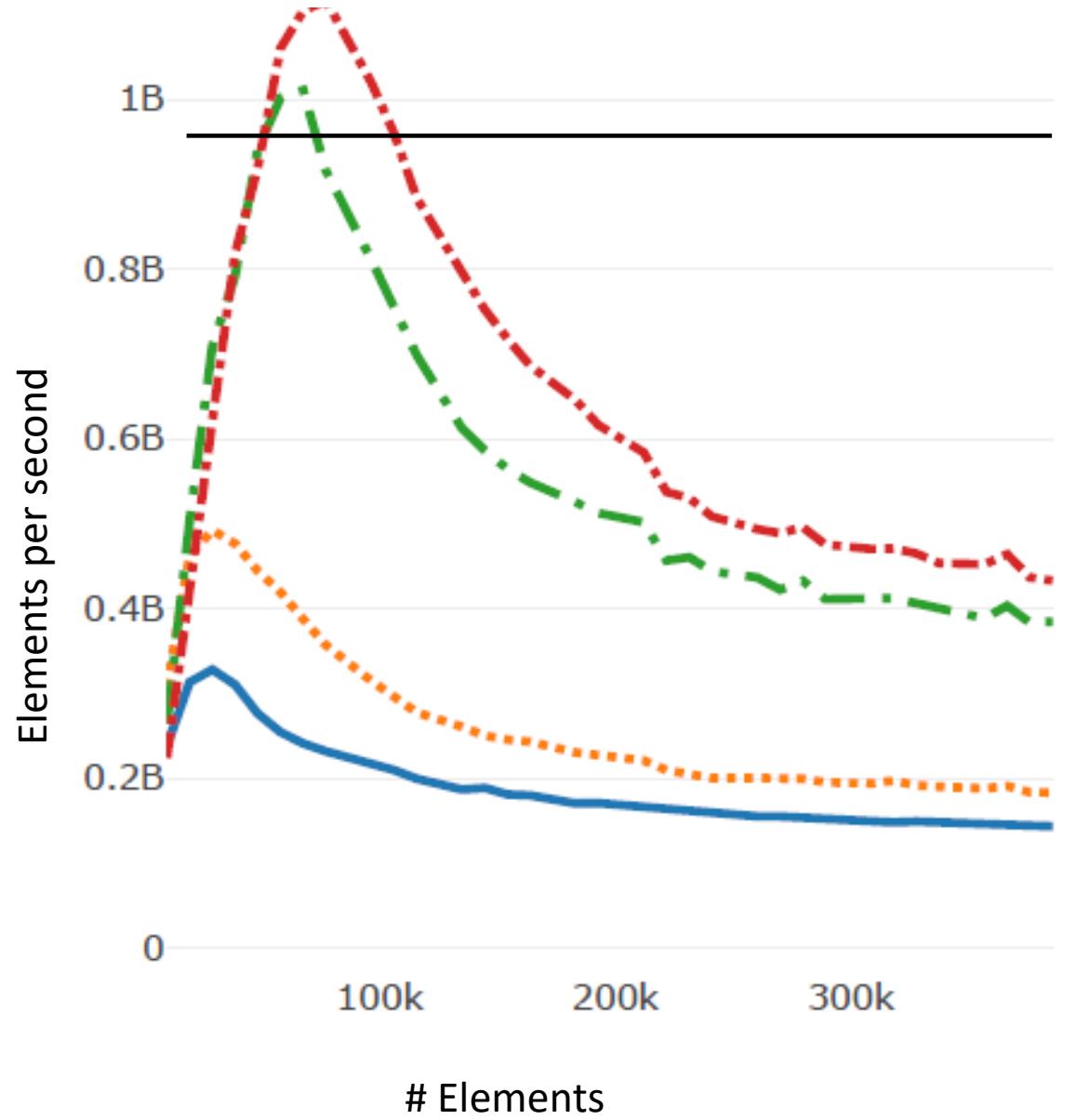
Evolve a robot with a fixed morphology

1. Cover page includes all information
2. General quality of the report (grammar, layout)
3. Video of fastest robot with at least three cycles shown
4. Video of fastest robot bouncing on the ground
5. Nice rendering of the robot (bars, spheres, ground plane, shaded faces)
6. Multiple robots showed in a single video
7. Description of all simulation parameters
8. Description of all robot parameters
9. Description of all evolutionary parameters
10. Learning curve
11. Dot chart
12. Diversity chart
13. Discussion of what works and did not work
14. Performance (speed of robot) in robot maximum diameter per cycle.
15. Innovative robot shown
16. Alternative representations tried
17. Post video of robot on piazza (provide screenshot and link)





Max



GPU- Accelerated Physics Simulations with CUDA

- GPU (CUDA) accelerated physics simulation sandbox capable of running complex spring/mass simulations in a fraction of the time of traditional CPU-based software while the CPU continues to perform optimizations.
- Capable of simulating 100,000 springs at about 4000 frames per second, or about 400,000,000 spring updates per second

```
int main() {  
  
    Mass * m1 = sim.createMass(Vec(1, 0, 0)); // create two masses at positions (1, 0, 0), (-1, 0, 0)  
    Mass * m2 = sim.createMass(Vec(-1, 0, 0));  
    Spring * s1 = sim.createSpring(m1, m2); // connect them with springs  
  
    Cube * c1 = sim.createCube(Vec(0, 0, 3), 1); // create fully-connected cube with side length 1 and center (0, 0, 3)  
  
    Plane * p = sim.createPlane(Vec(0, 0, 1), 0); // constraint plane z = 0 (constrains object above xy plane)  
  
    sim.setTimeStep(0.0001) // increment simulation in 0.0001 second intervals  
    sim.setBreakpoint(5.0); // ends simulation at 5 seconds  
    sim.run(); // run simulation  
  
    // perform other optimizations on the CPU while the simulation continues to run, like topology optimization, robotics, etc.  
  
    return 0;  
}
```

Titan

Interested in CUDA library? <https://www.creativemachineslab.com/titan-library.html>

Contact Sofia Wyetzner sofiaw@stanford.edu or Boxi Xia bx2150@columbia.edu

Evolving Motion

- For each spring set k and $L_0 = a+b*\sin(\omega t+c)$
 - ω is global frequency. a,b,c,k are spring parameters (k is spring coef)
 - Cycle period is $2\pi/\omega$
- Evolve locomotion
 - Fitness: net distance travelled by center of mass
- Representation
 - Direct encoding: Chromosome with a,b,c,k for each spring
 - Indirect encoding: Chromosome specifies how a,b,c,k are determined

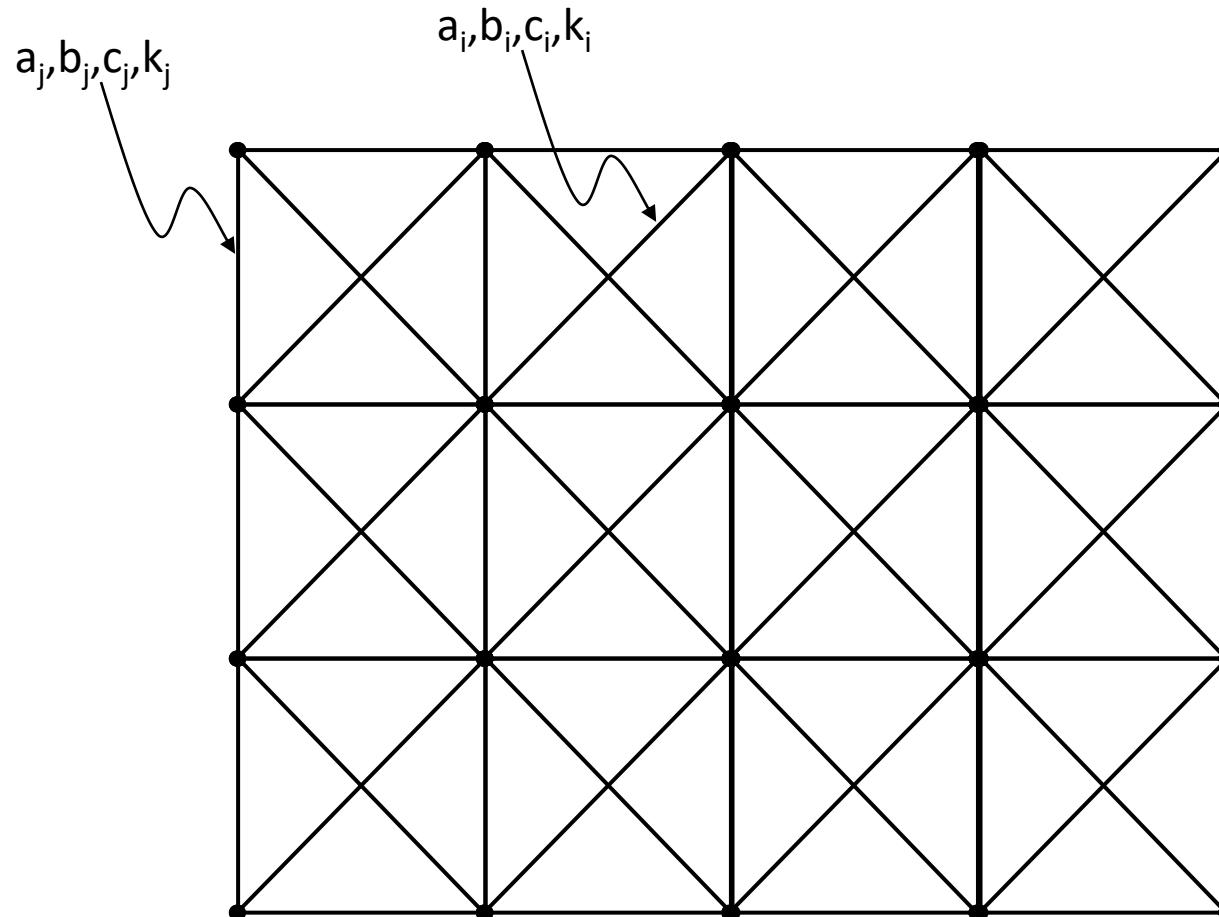
Direct encoding

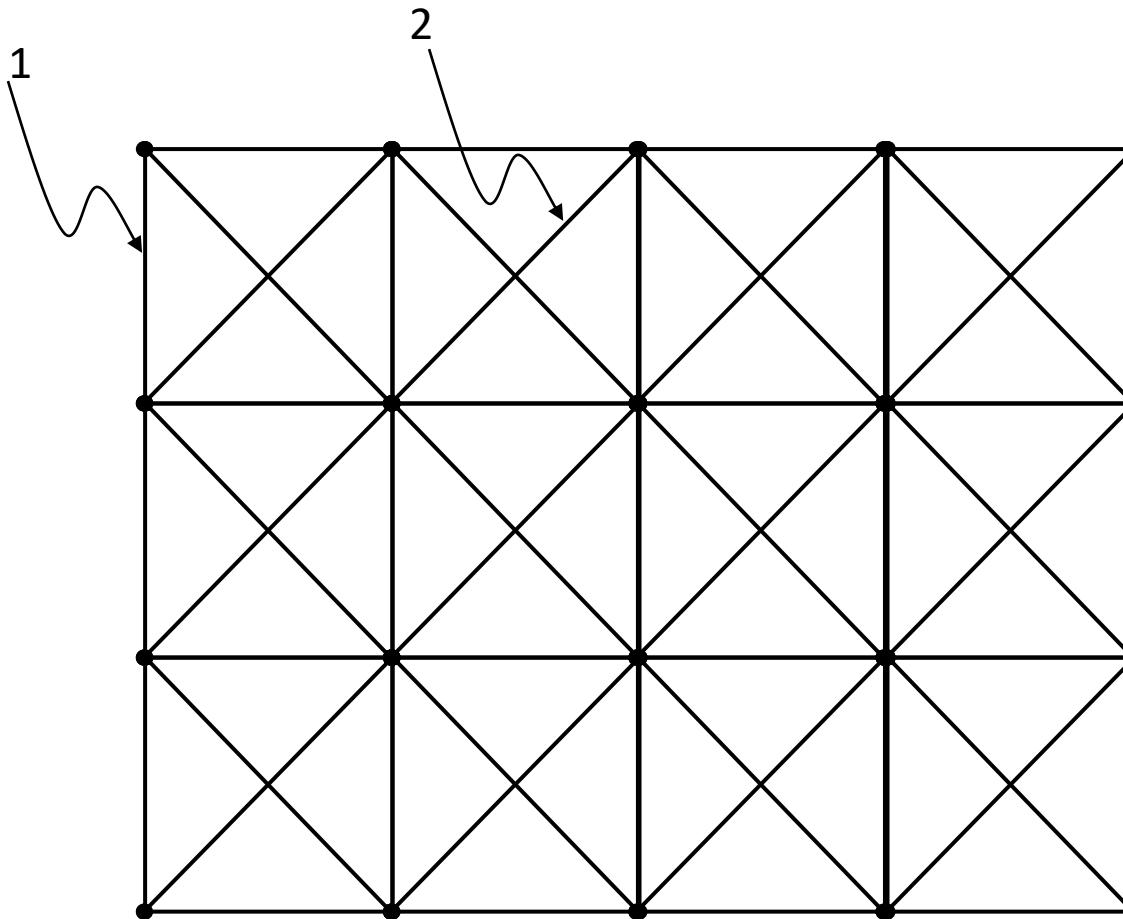
| |
|----------------------|
| a_1, b_1, c_1, k_1 |
| a_2, b_2, c_2, k_2 |
| ... |
| a_n, b_n, c_n, k_n |

$n = \text{number of springs}$

Fitness? Mutation? Crossover?

$$L_0 = a + b \cdot \sin(\omega t + c)$$

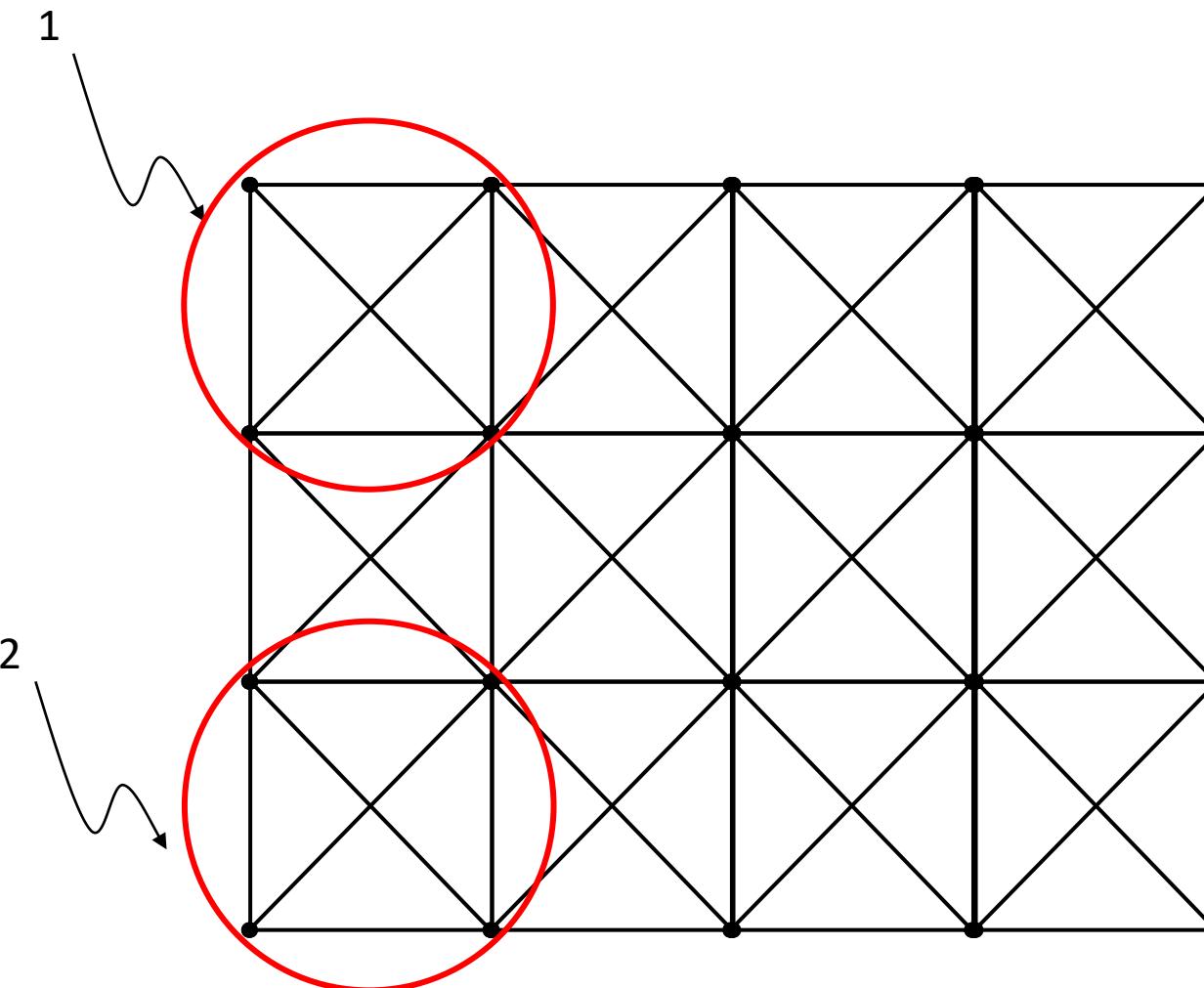




$$L_0 = a + b \sin(\omega t + c)$$

1. $k=1000$ $b=c=0$
2. $K=20,000$ $b=c=0$
3. $K=5000$ $b=0.25$ $c=0$
4. $K=5000$ $b=0.25$ $c=\pi$

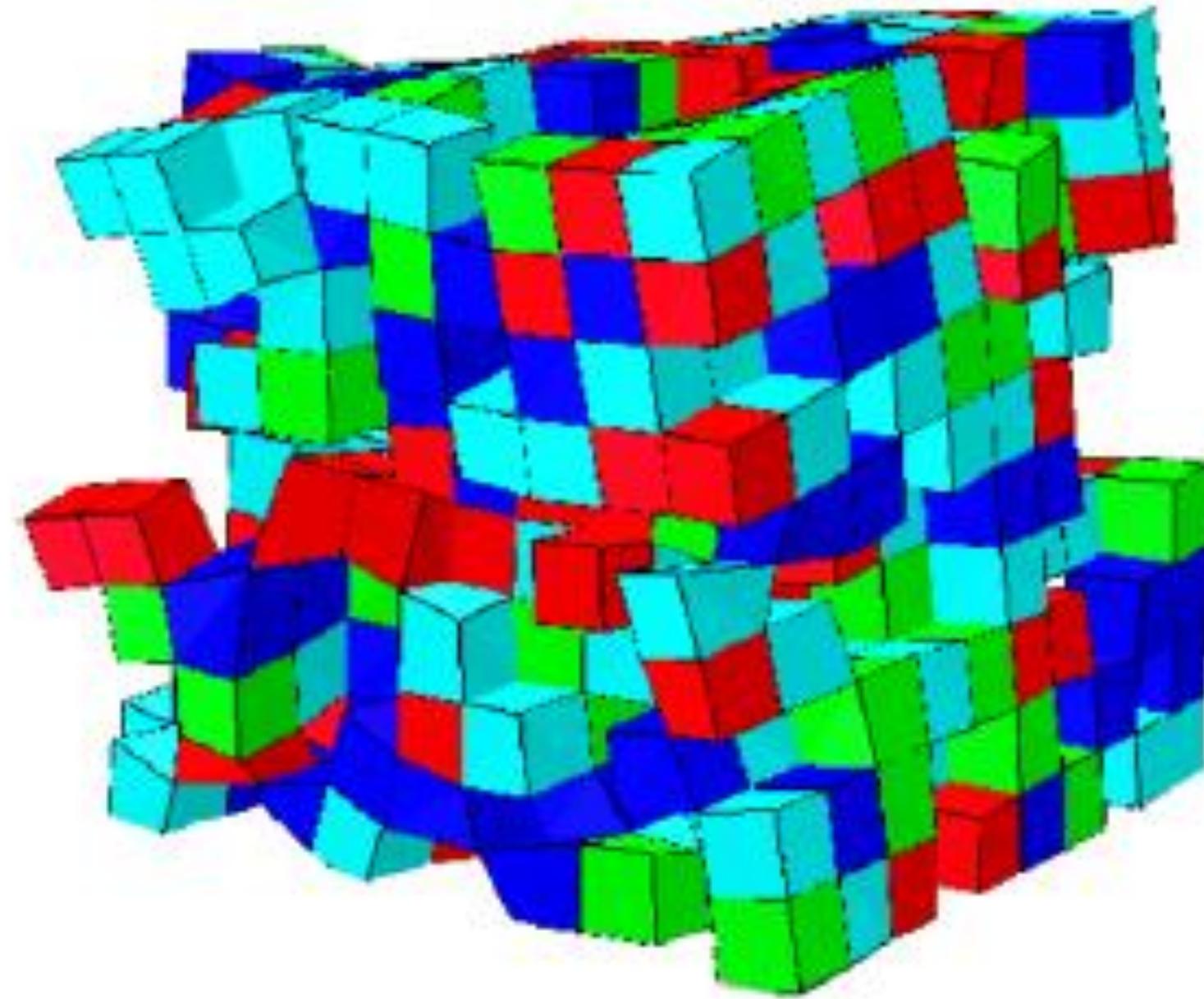
Cycle period: $T=2\pi/\omega$



$$L_0 = L_{0_int} * [1 + b * \sin(wt + c)]$$

1. $k=1000$ $b=c=0$
2. $K=20,000$ $b=c=0$
3. $K=5000$ $b=0.25$ $c=0$
4. $K=5000$ $b=0.25$ $c=\pi$

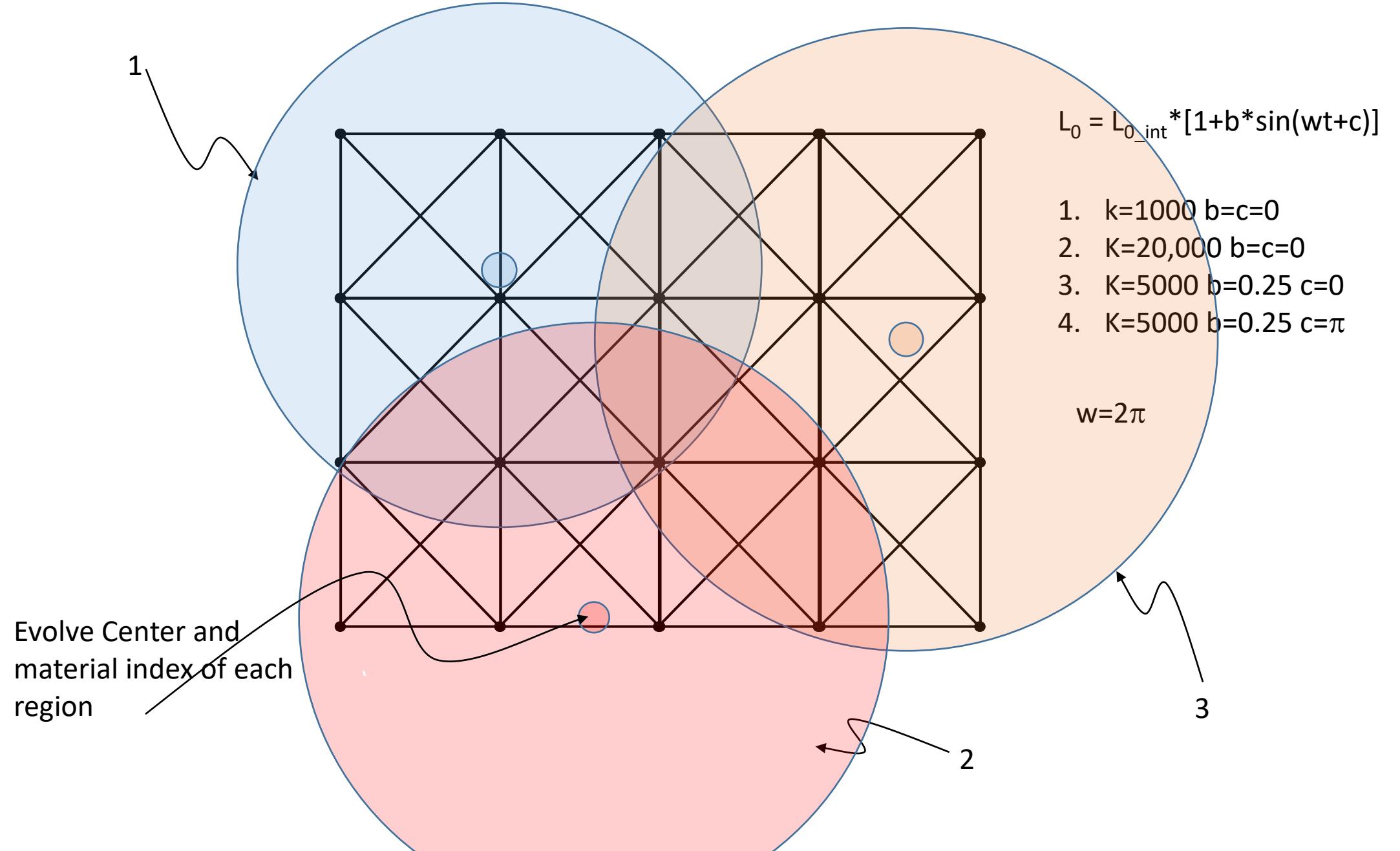
$$w=2\pi$$



Breakout

- Suggest a representation that would indirectly encode spring parameters for a large robot



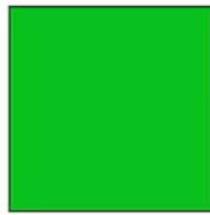




Muscle: contract then expand



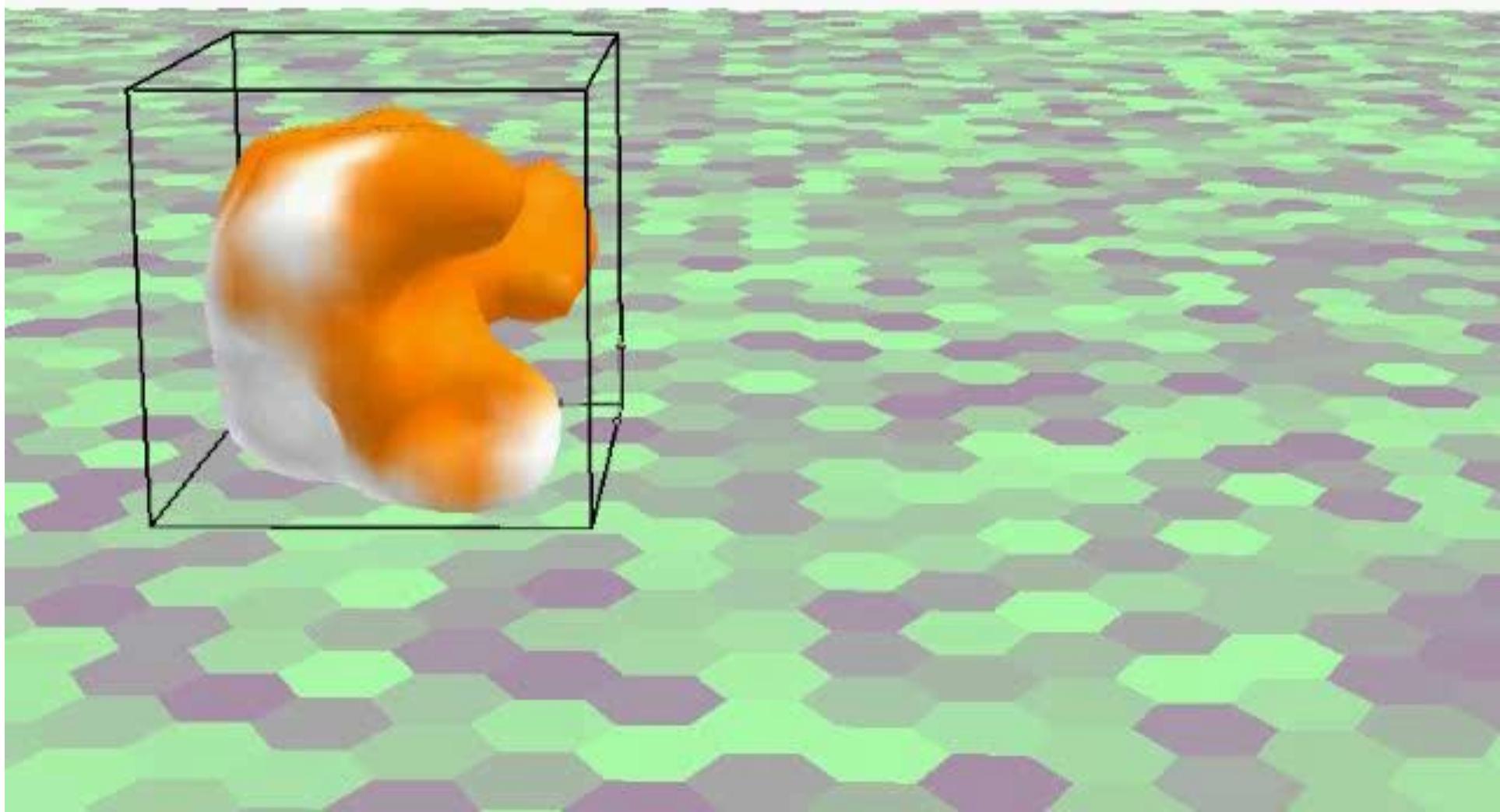
Tissue: soft support



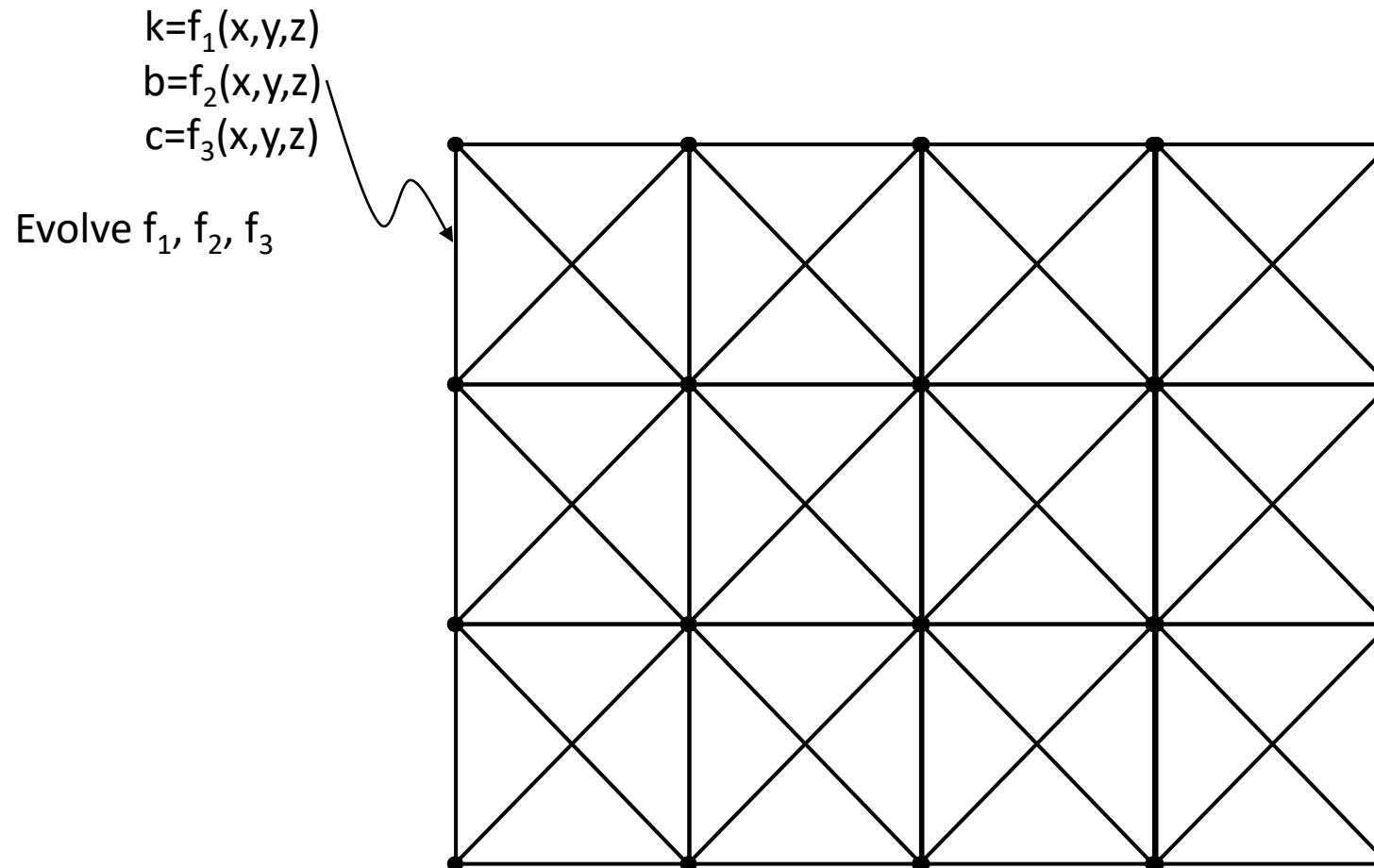
Muscle2: expand then contract



Bone: hard support



$$L_0 = a + b \cdot \sin(wt + c)$$



Flexible Muscle-Based Locomotion for Bipedal Creatures

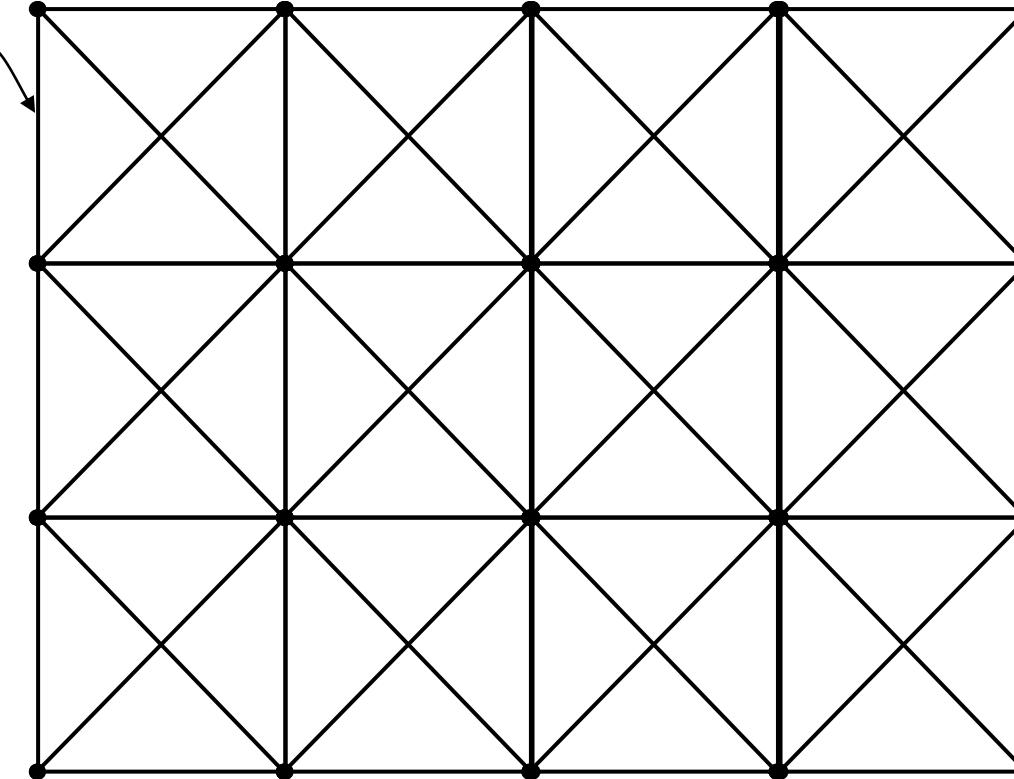
SIGGRAPH ASIA 2013

**Thomas Geijtenbeek
Michiel van de Panne
Frank van der Stappen**

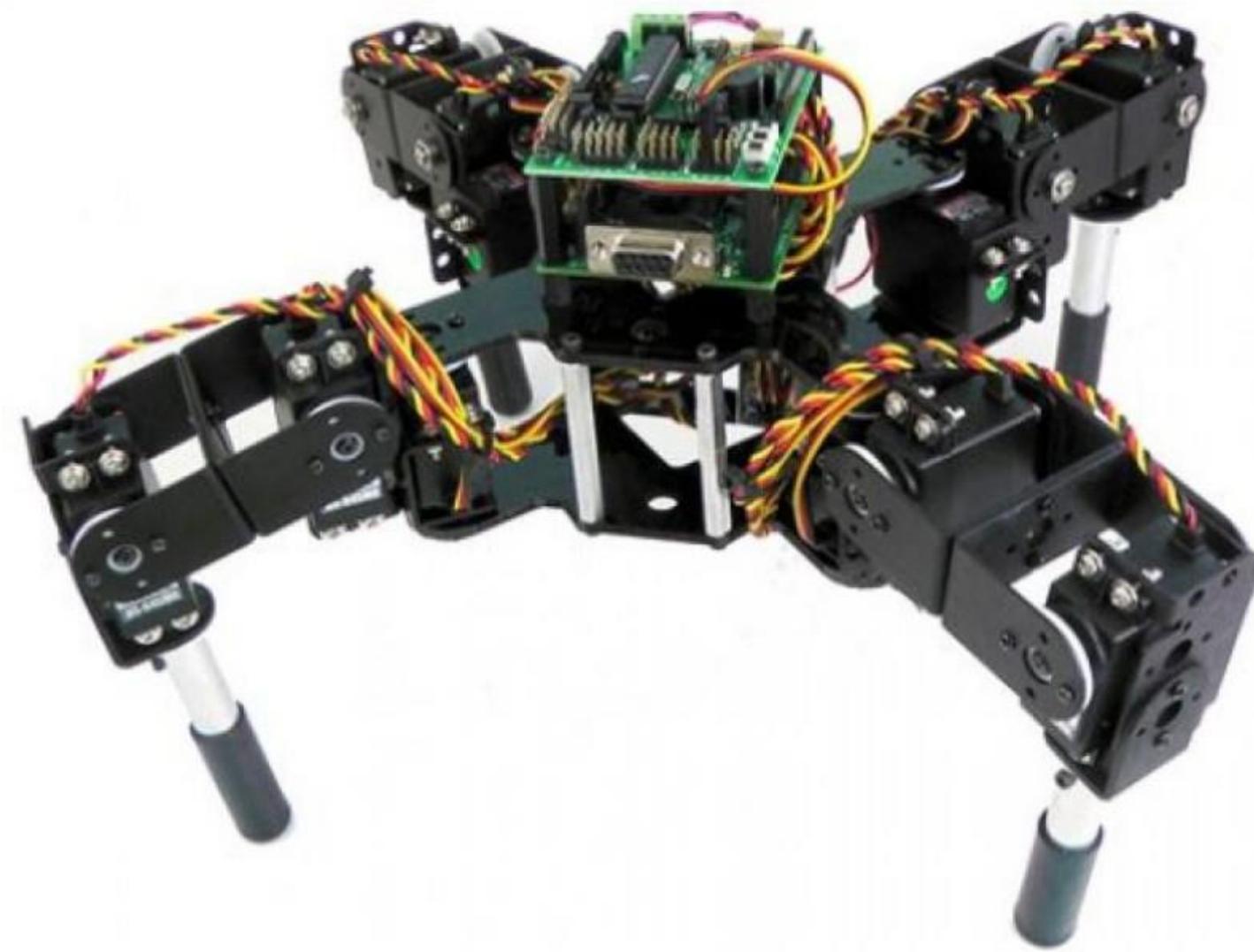
$$L_0 = a + b * \sin(wt + c)$$

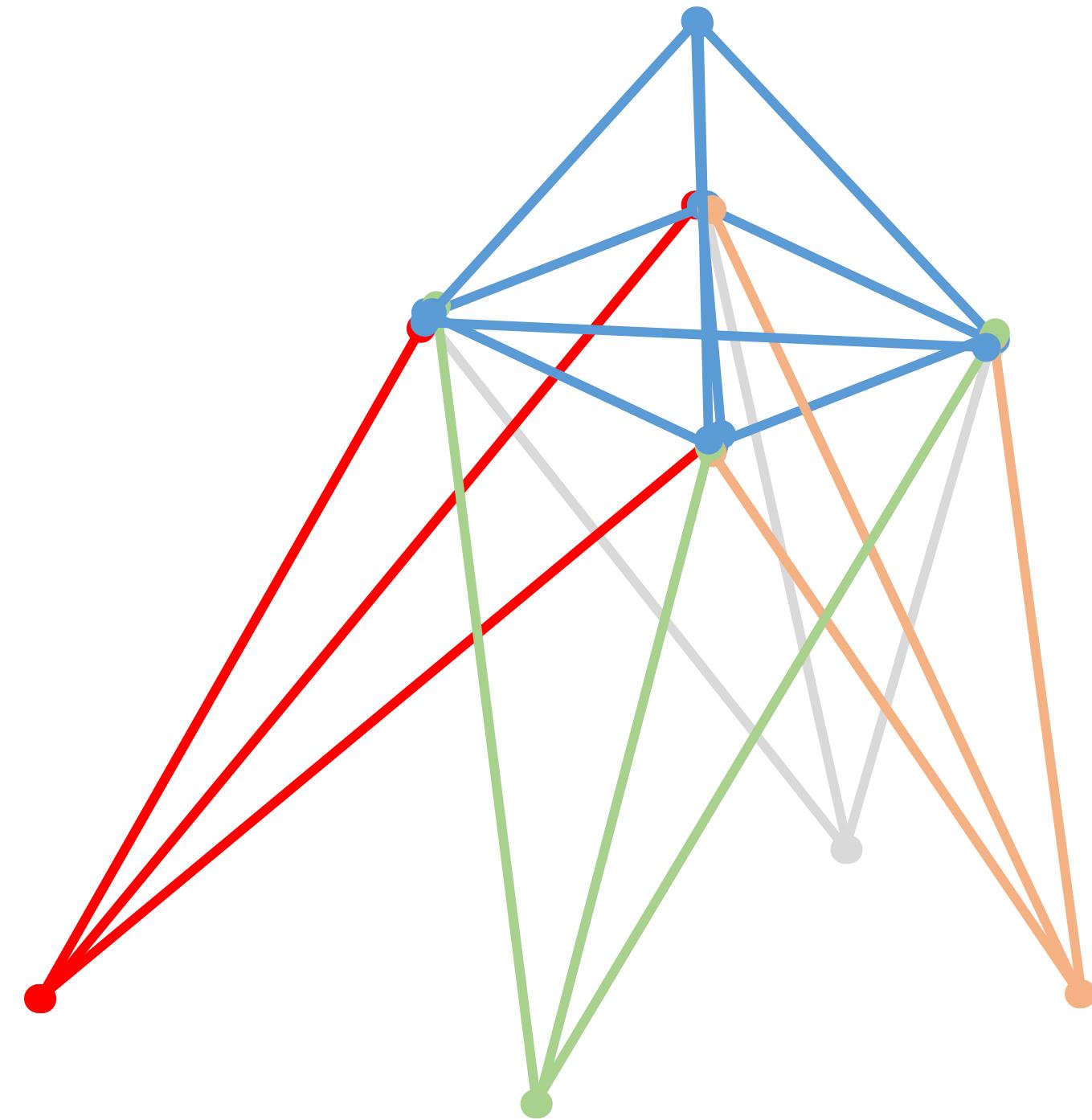
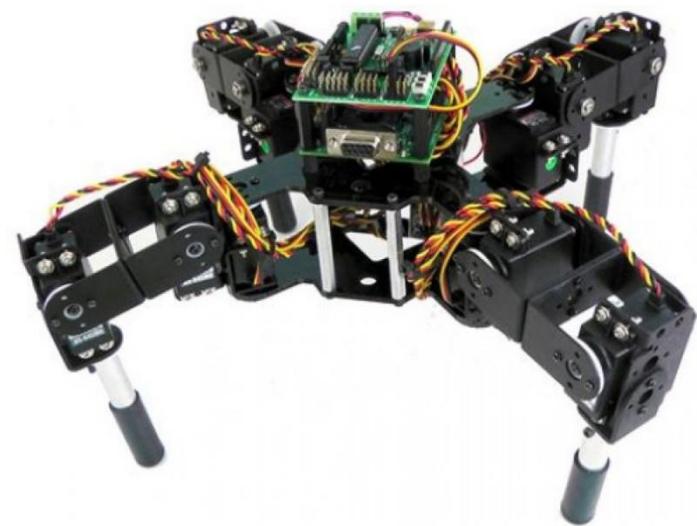
$$L_0 = f(x, y, z, t)$$

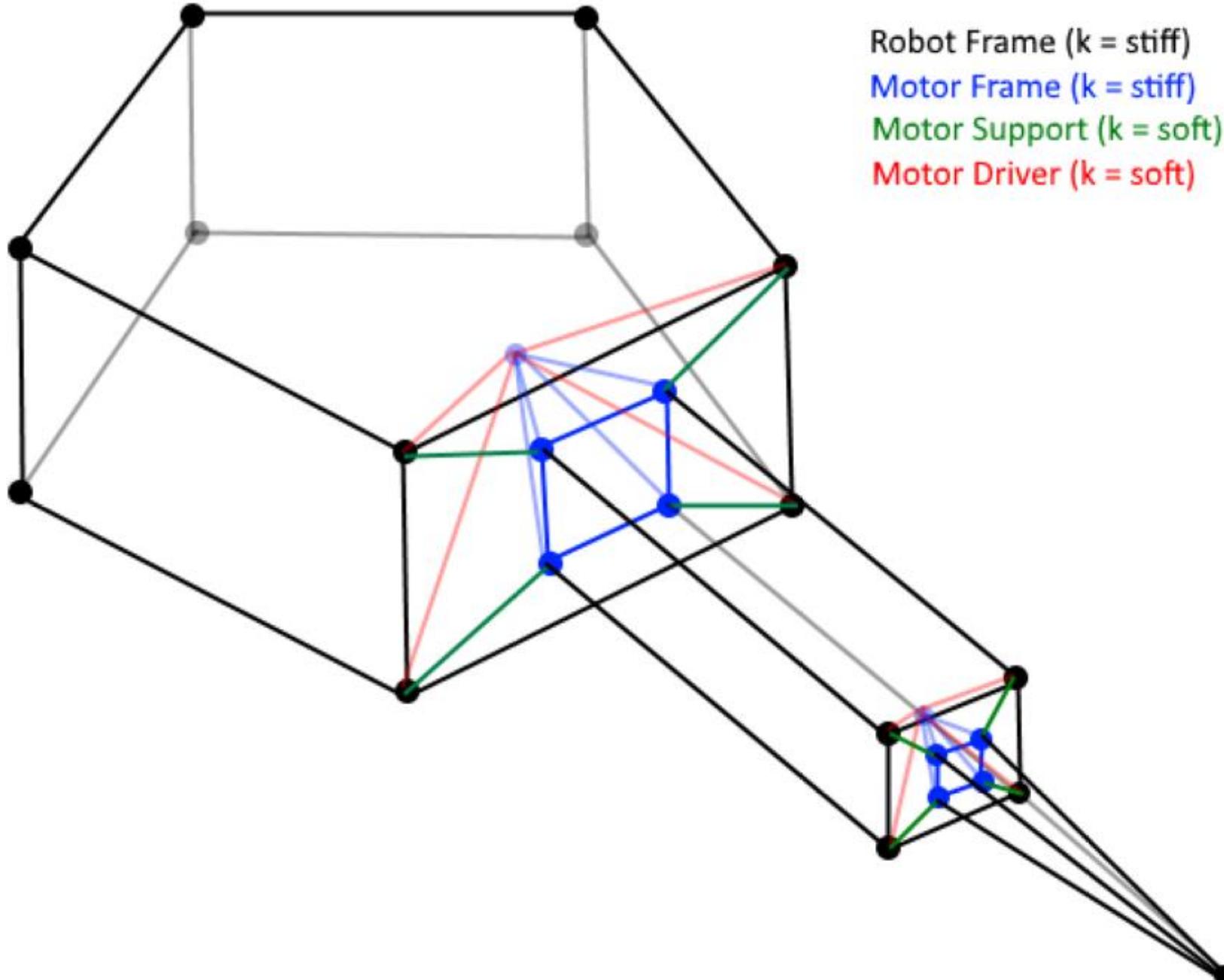
Evolve f

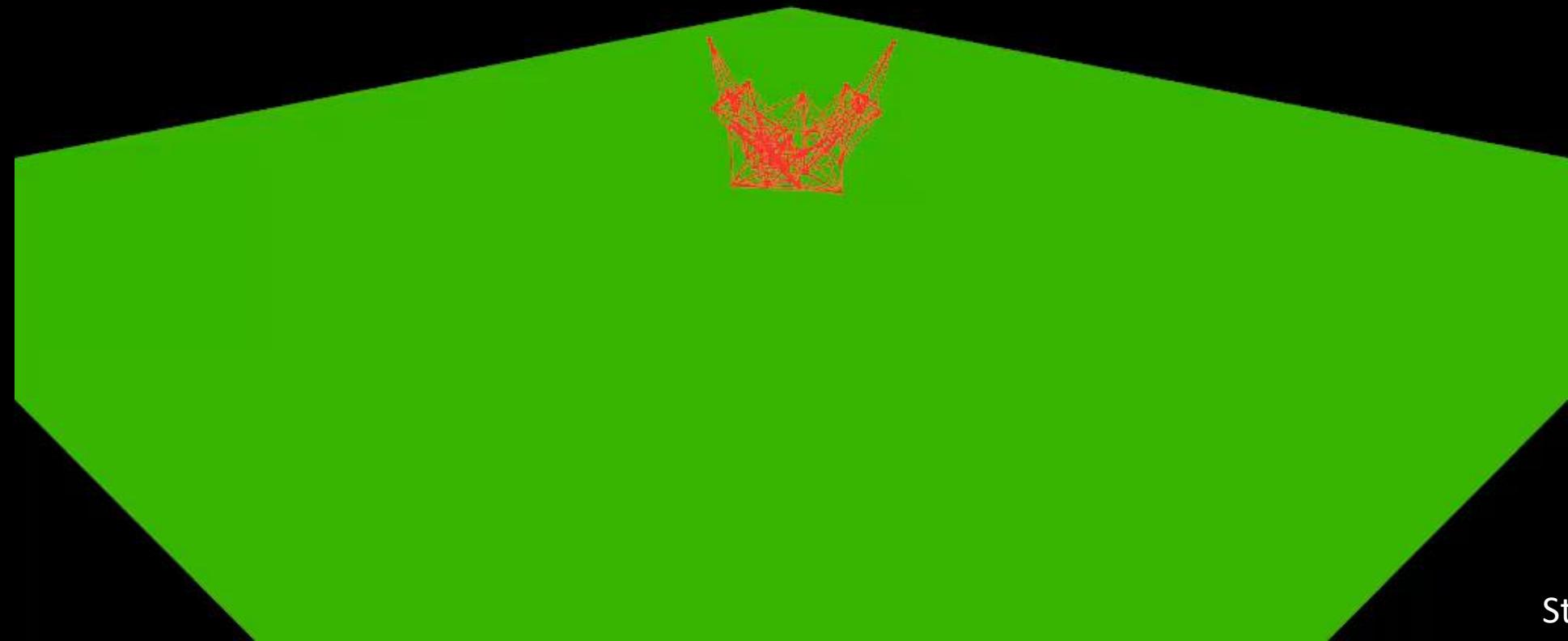


$f(x, y, z, t)$ can be an analytical function or a neural network

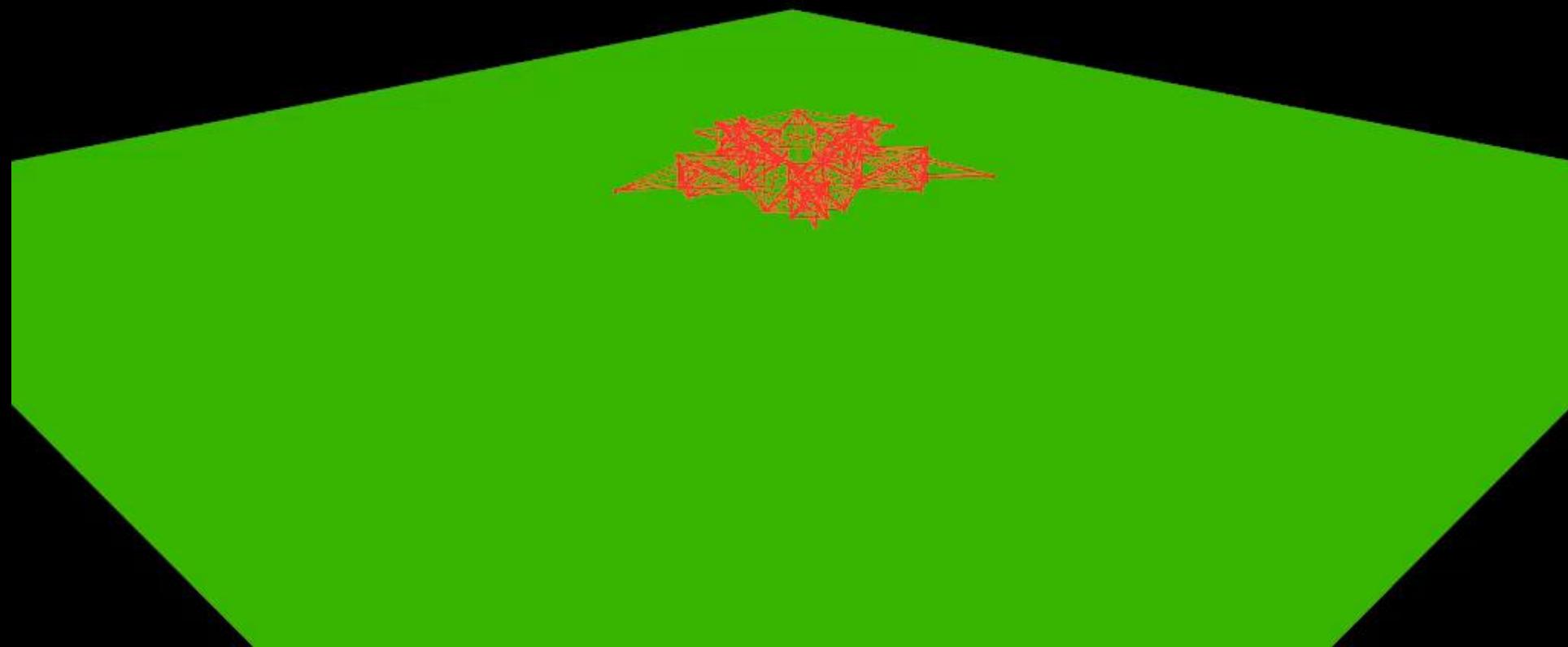




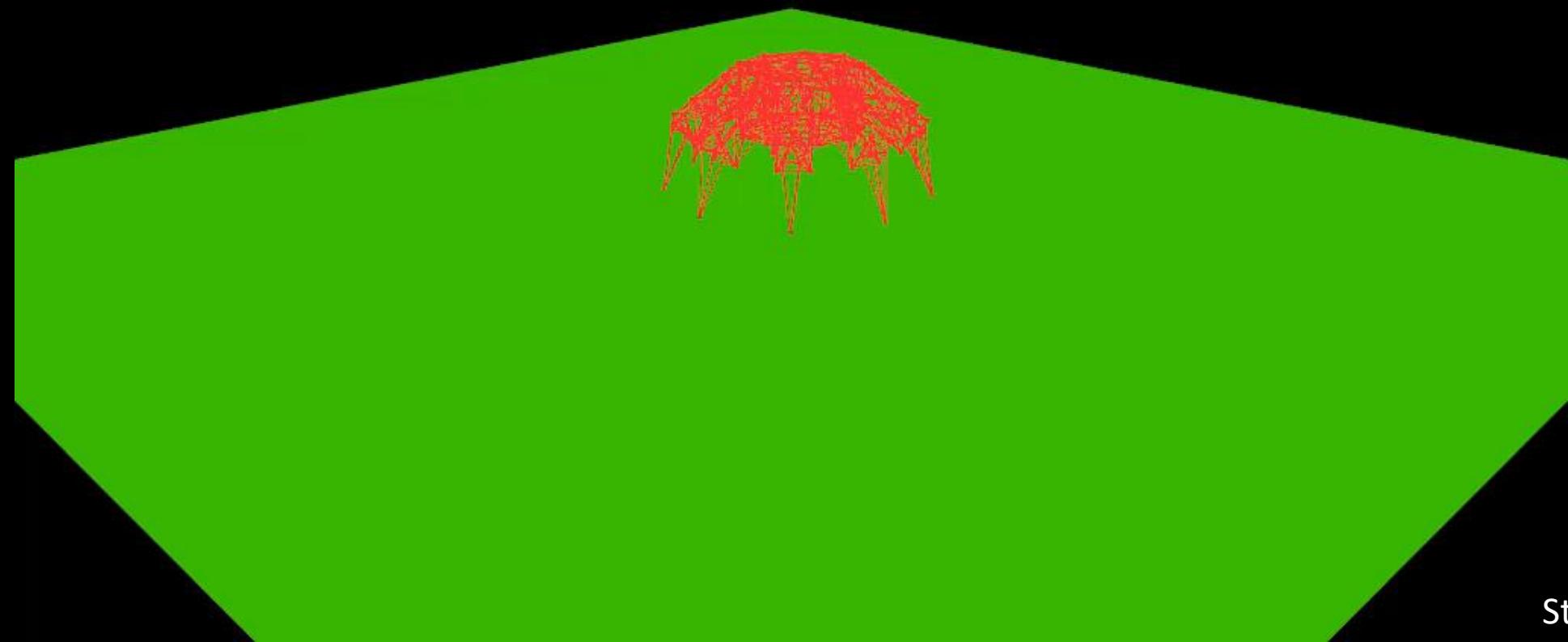




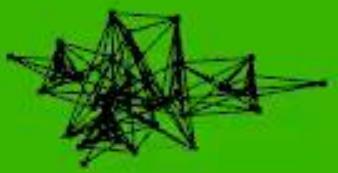
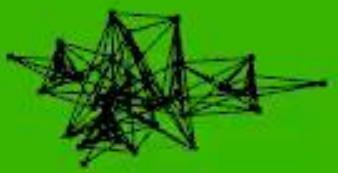
Steven Mazzola



Steven Mazzola



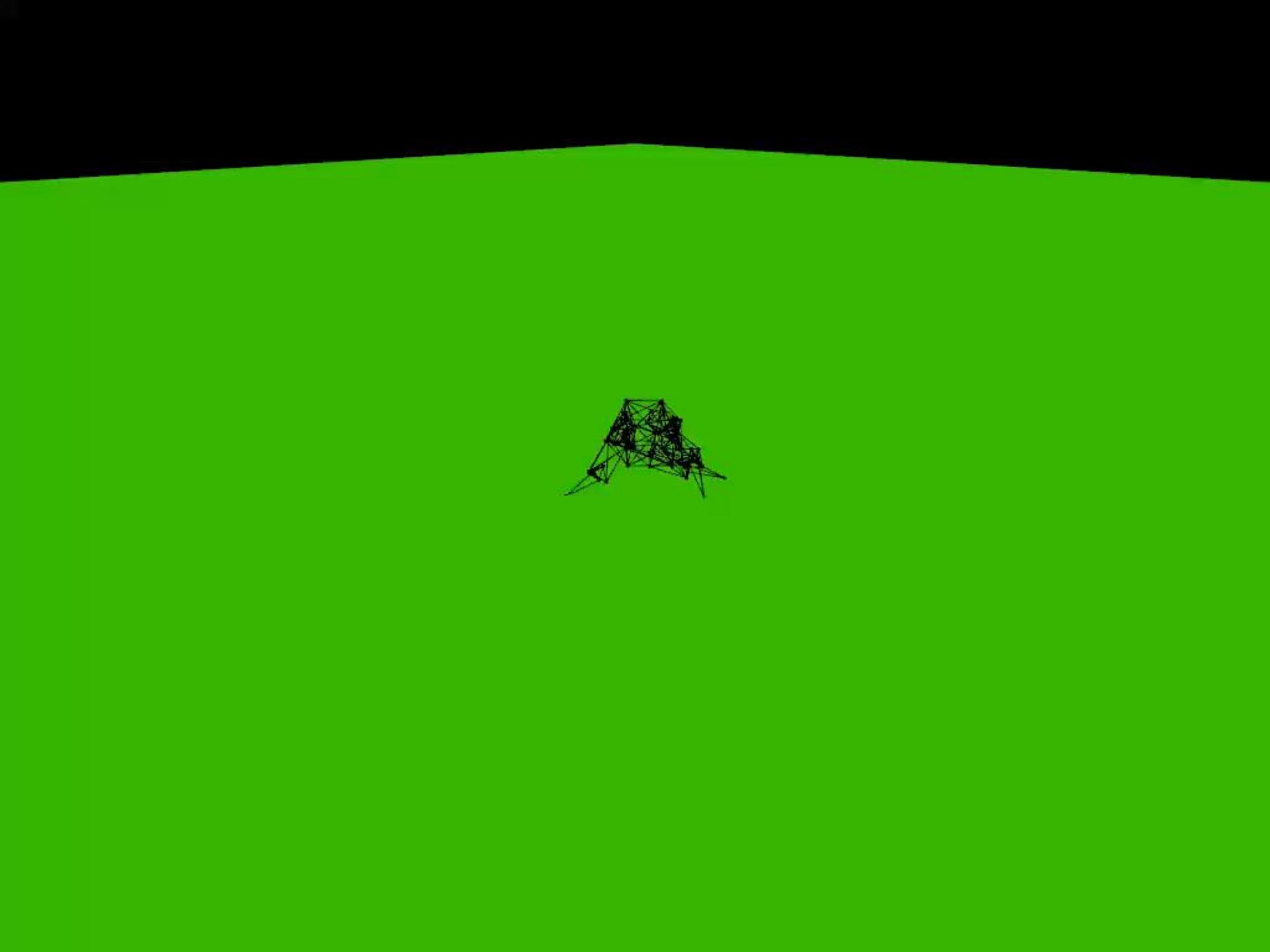
Steven Mazzola

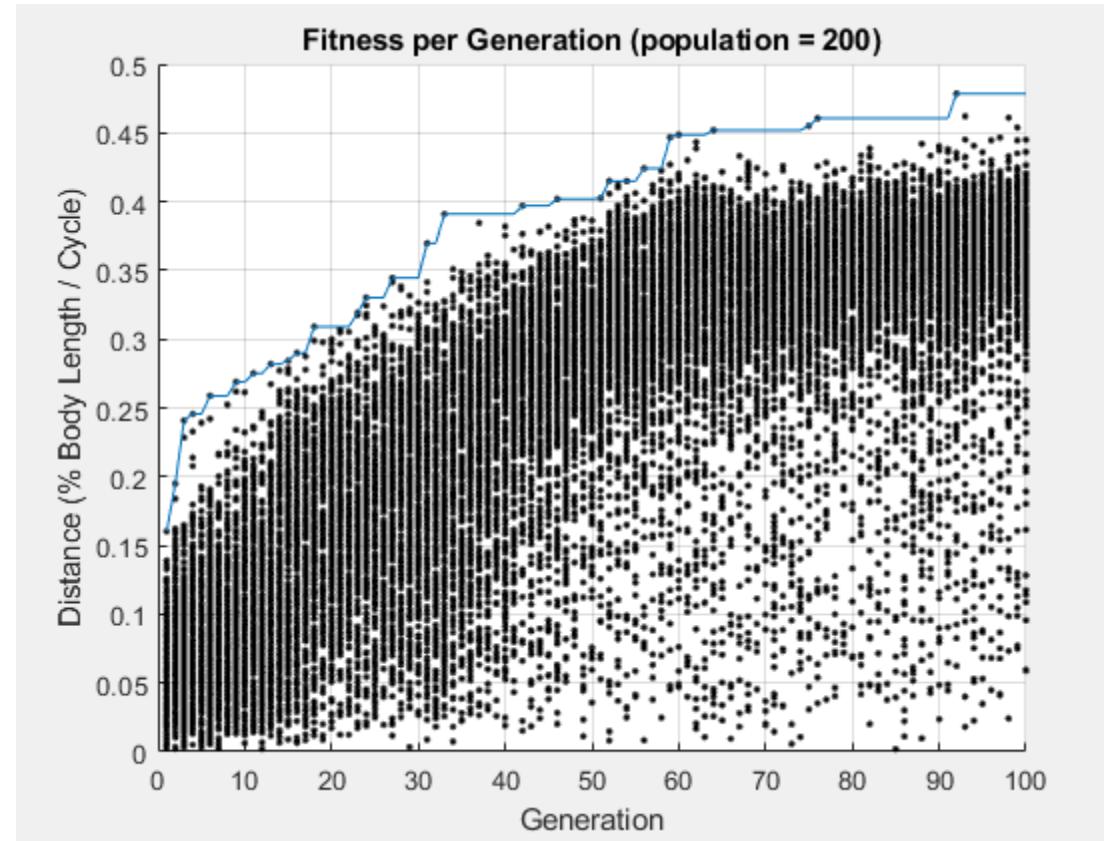


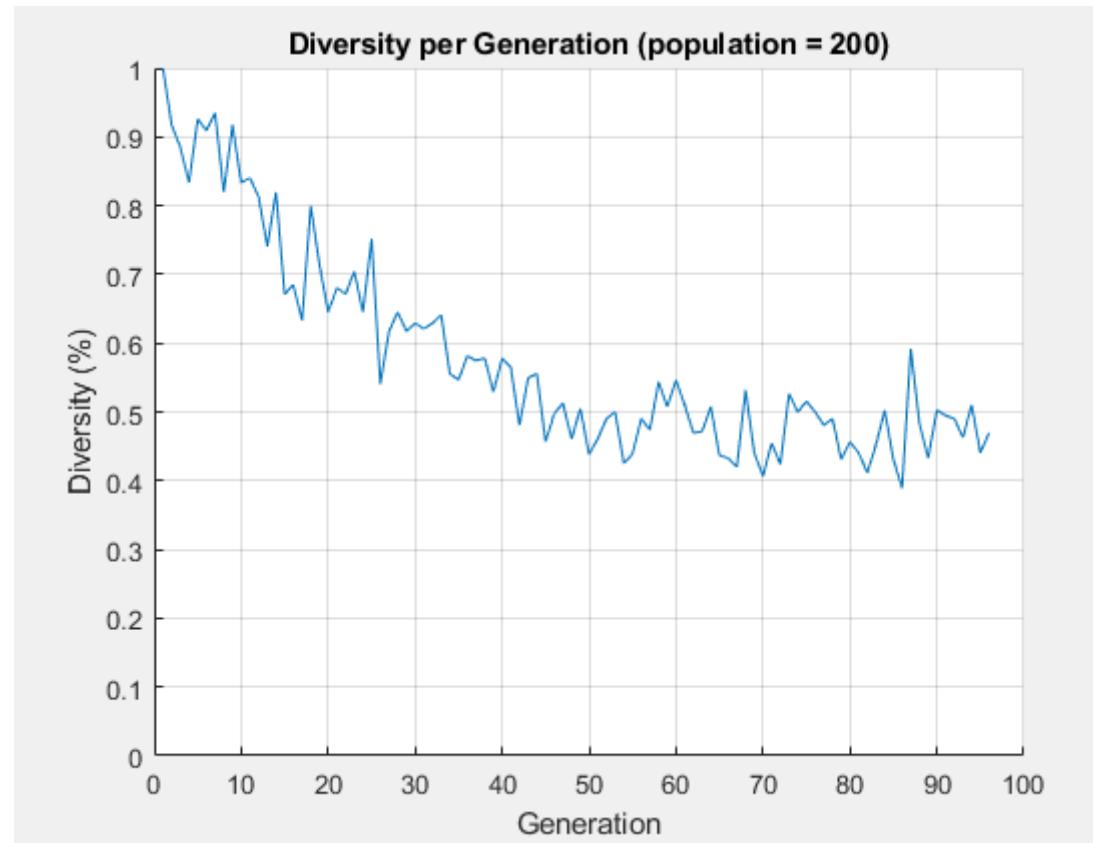
300K Evaluations



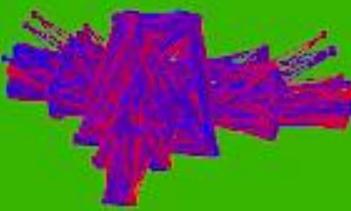
Steven Mazzola



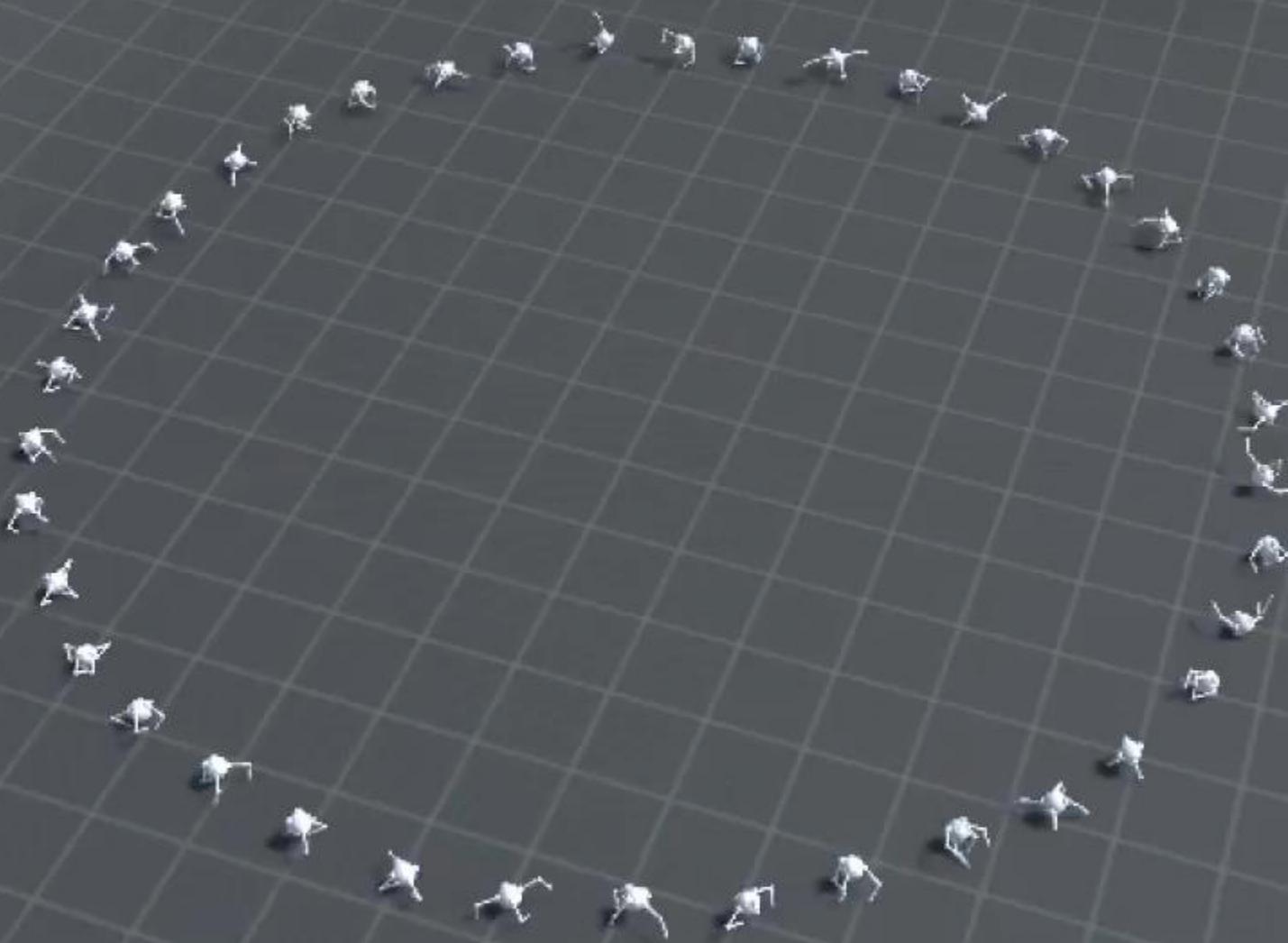




Many robots in one world

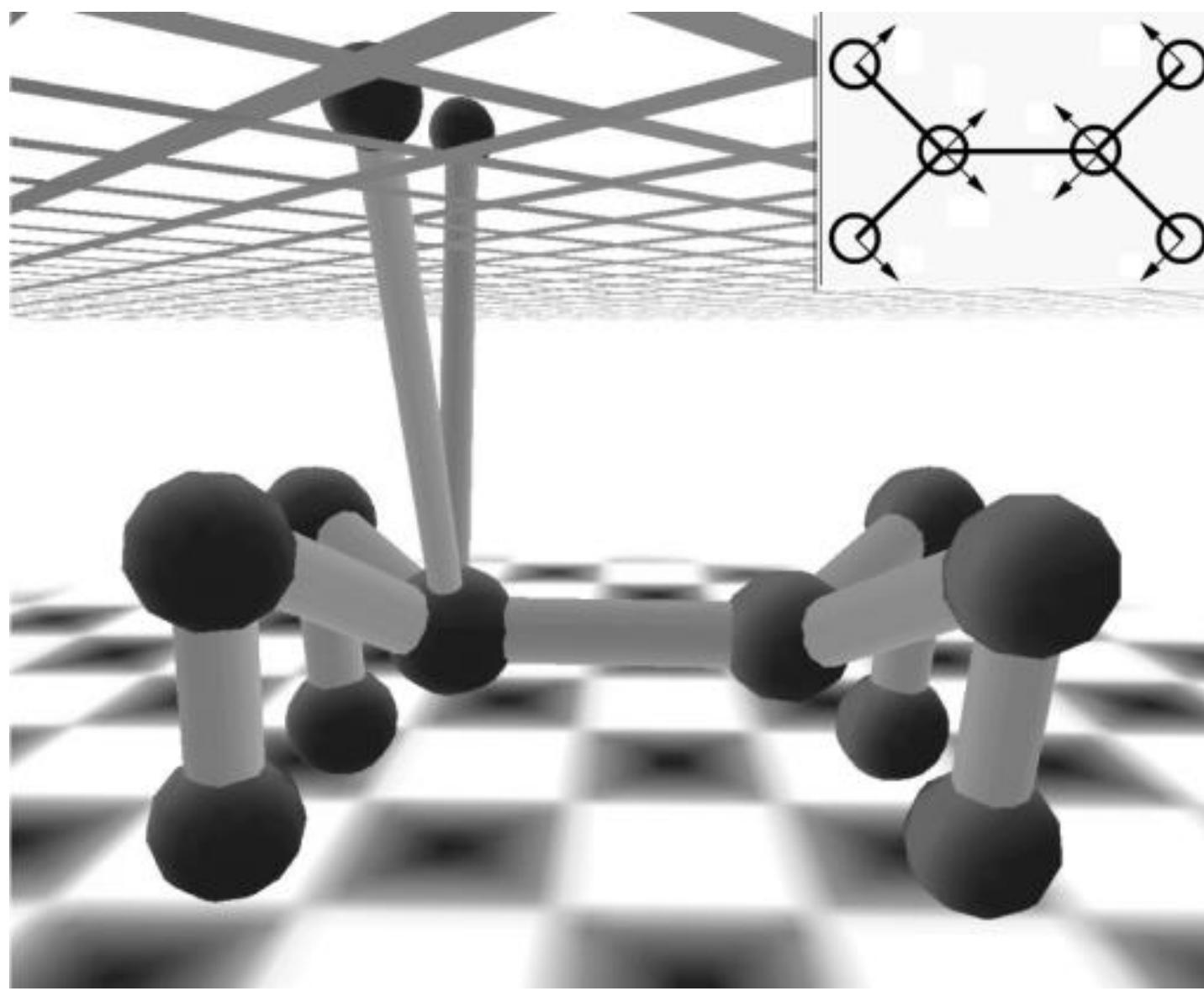


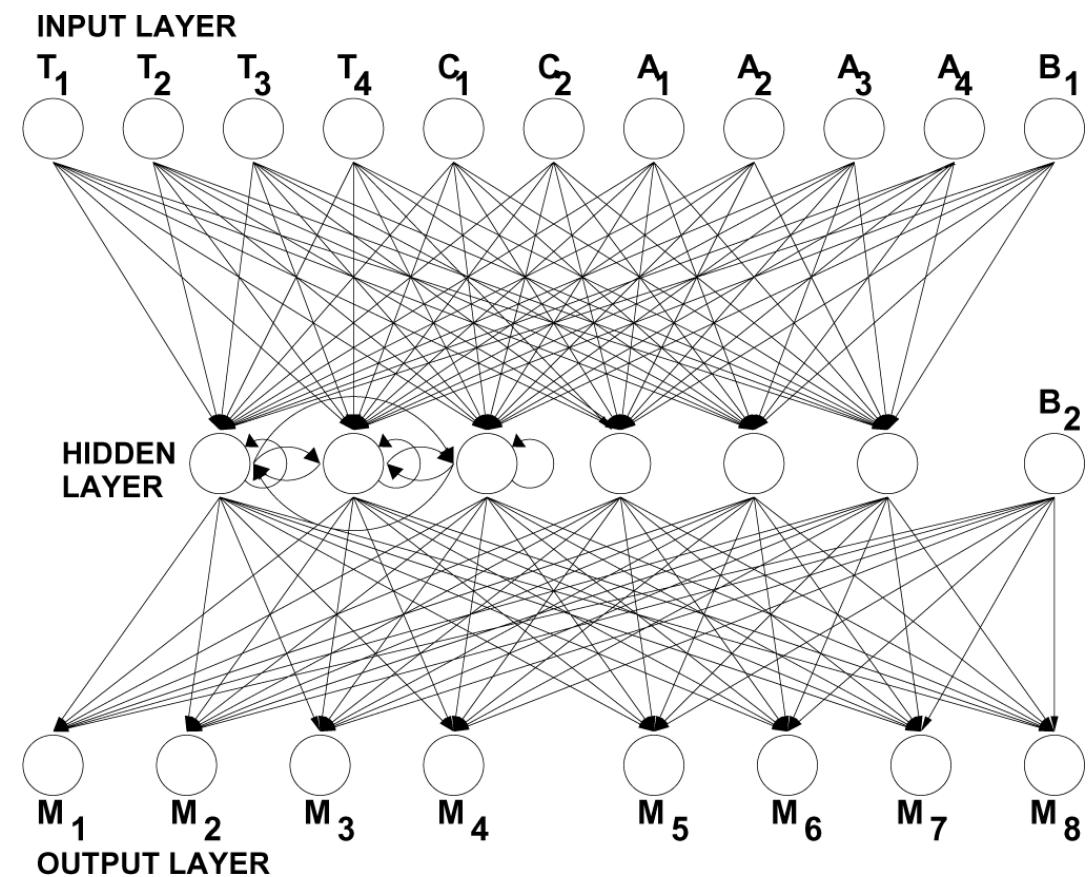
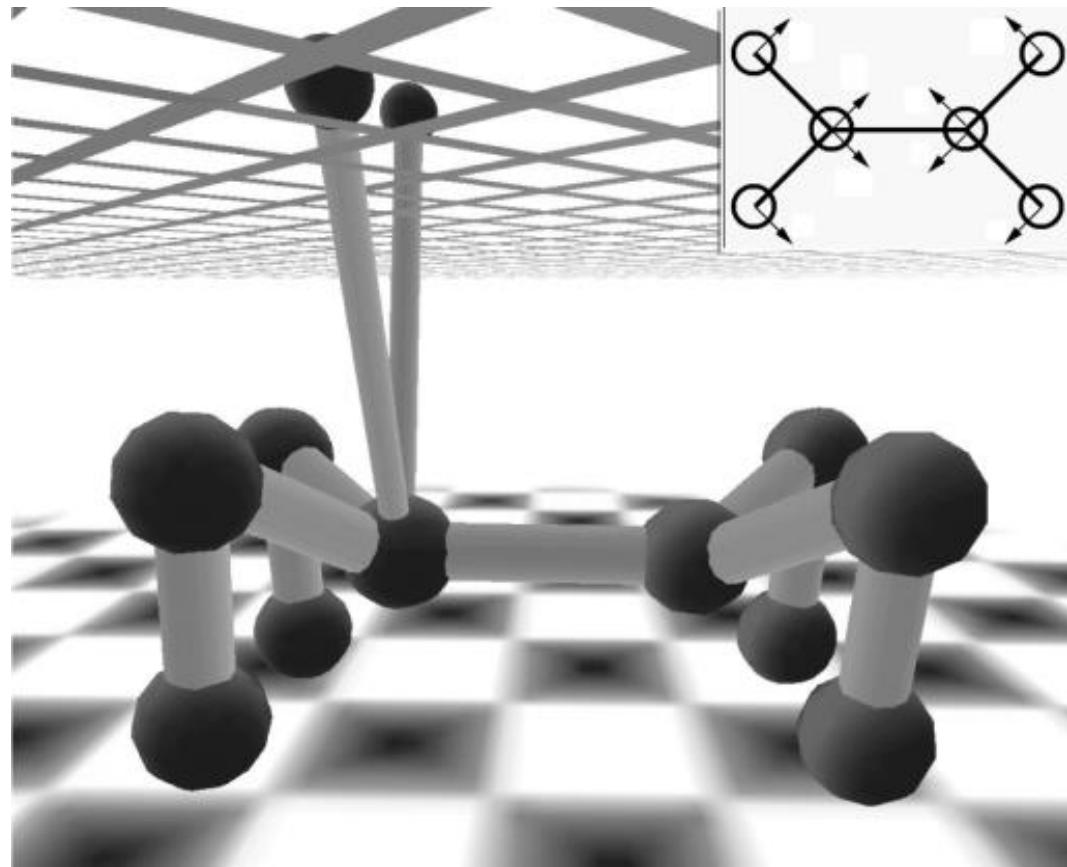
Gen 1

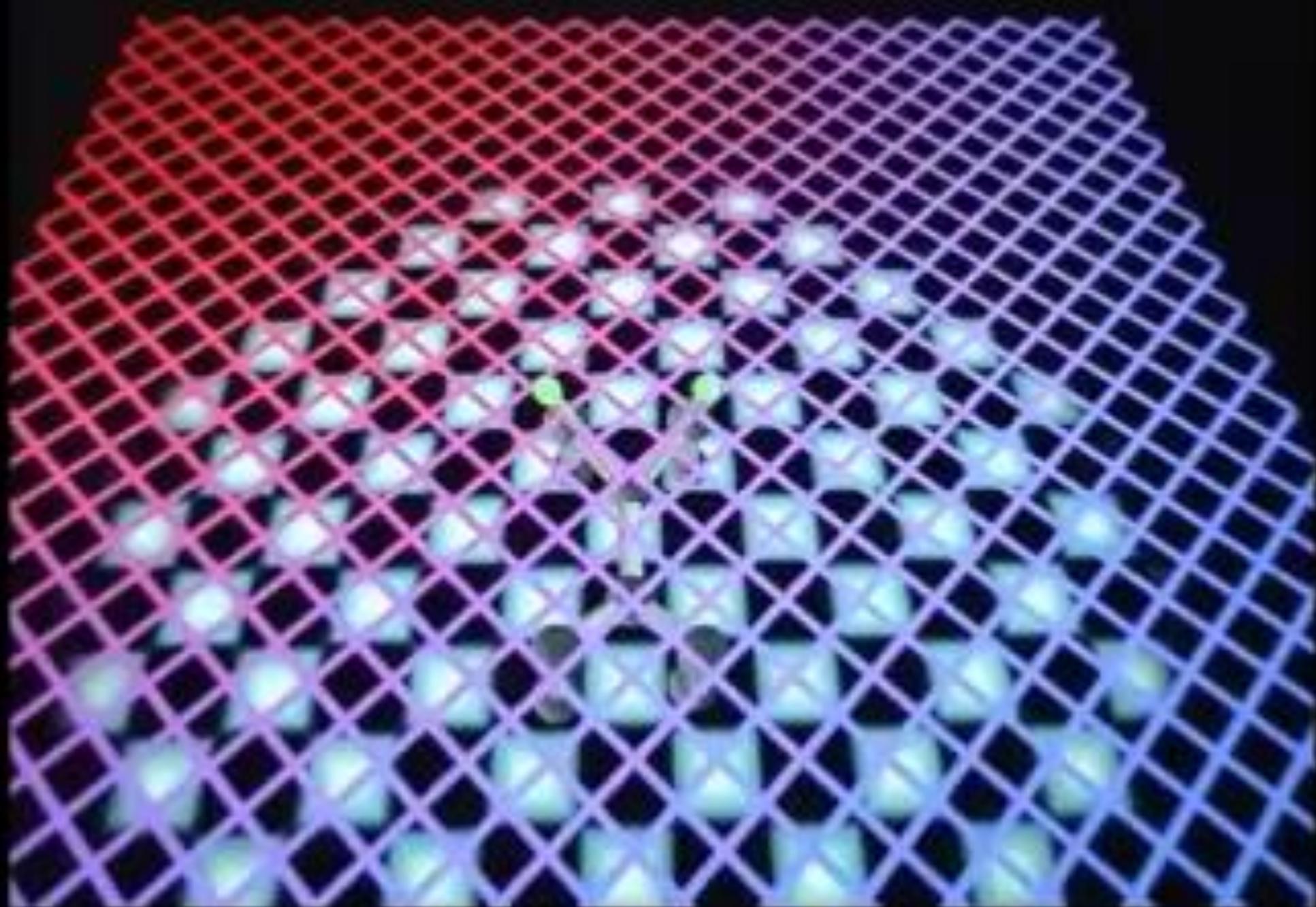


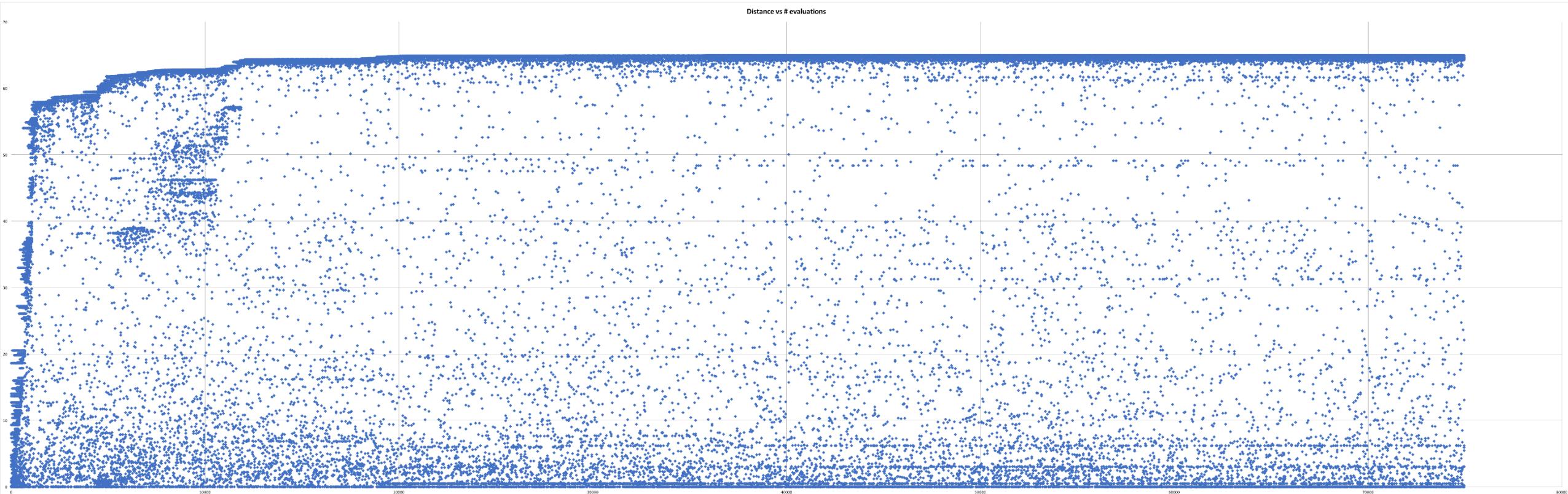
Incorporating Sensors

- You can change the length L_0 of springs also as function of sensors
- Sensor types:
 - External, e.g.
 - light level
 - Distance to goal
 - Internal (proprioceptive)
 - Actual length of spring
 - Ground force (Whether mass is on or below ground)
 - Speed
 - ...









More complex functions

- $L_0 = a + b * \sin(\omega t + c)$
- $L_0 = a + b * \sin(\omega t + c) + d * \sin(2\omega t + e)$
- $L_0 = a + b * \sin(\omega t + c) + d * \sin(2\omega t + e) + f * \sin(3\omega t + g)$
- ...
- $L_0 = f(\sin(\omega t)) \leftarrow \text{Evolve } f \text{ directly}$

Assignment 3c

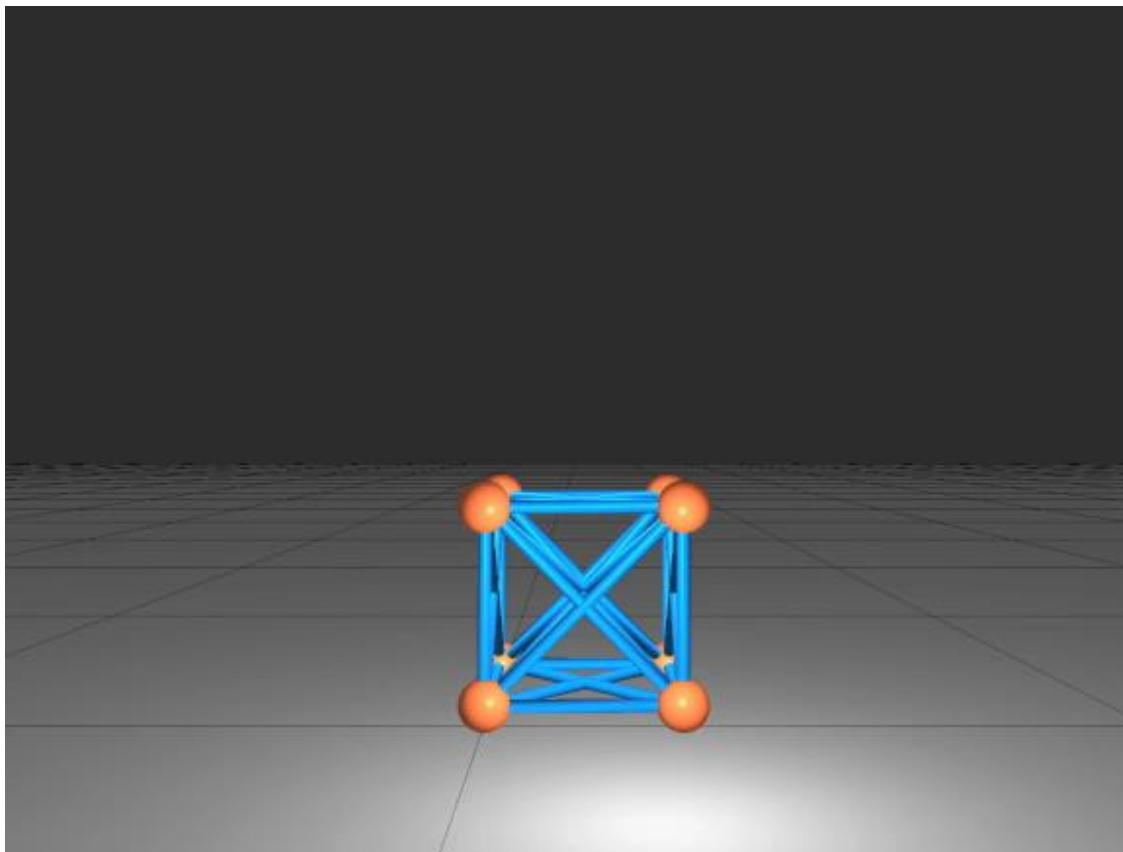
Evolve a robot with a variable morphology

Evolve a robot with a variable morphology

1. Cover page includes all information
2. General quality of the report (grammar, layout)
3. Video of fastest robot with at least three cycles shown
4. Video of fastest robot bouncing on the ground
5. Robot zoo – images of nine different robots (in a 3x3 panel)
6. Nice rendering of the robot (bars, spheres, ground grid, shaded faces)
7. Multiple robots showed in a single video
8. Description of all simulation parameters
9. Description of all robot parameters
10. Description of all evolutionary parameters
11. Learning curve with error bars
12. Populating dot plot
13. Populating Diversity chart
14. Discussion of what works and did not work
15. Performance (speed of robot) in robot maximum diameter per cycle.
16. Innovative robot(s) shown
17. Alternative representations tried
18. Post video of robot on piazza (provide screenshot and link)

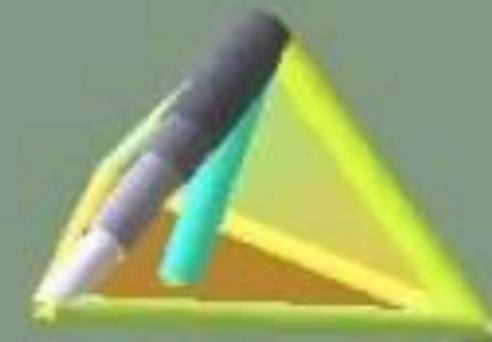
Assignment 3c

- Evolve morphology

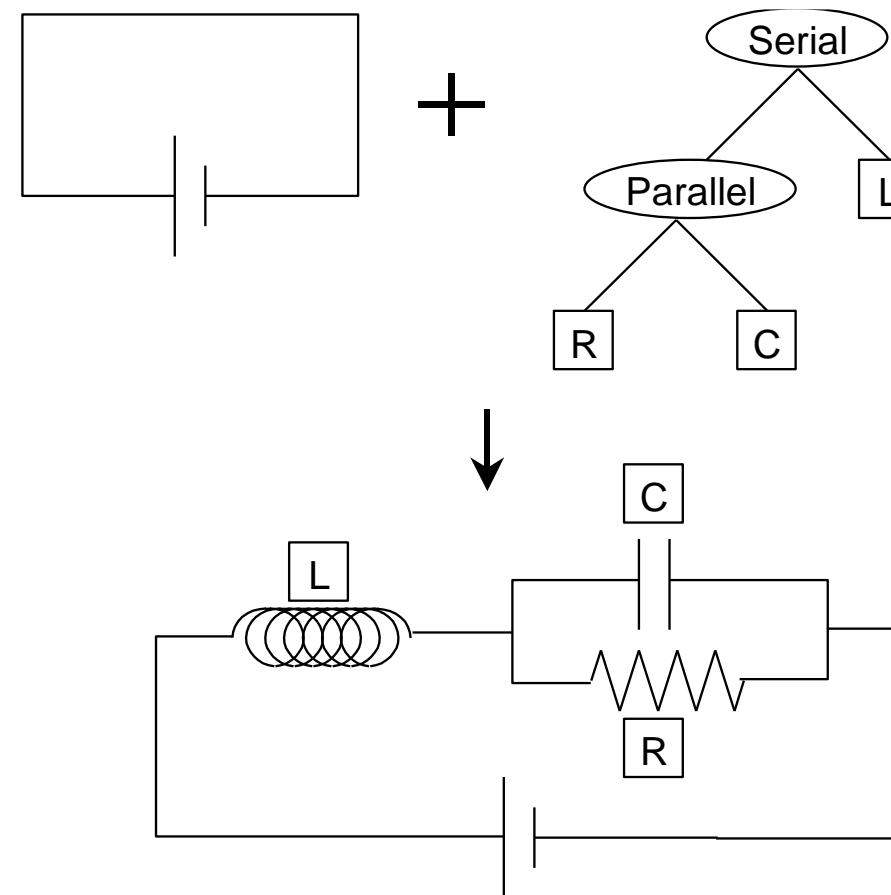


Direct morphological operators

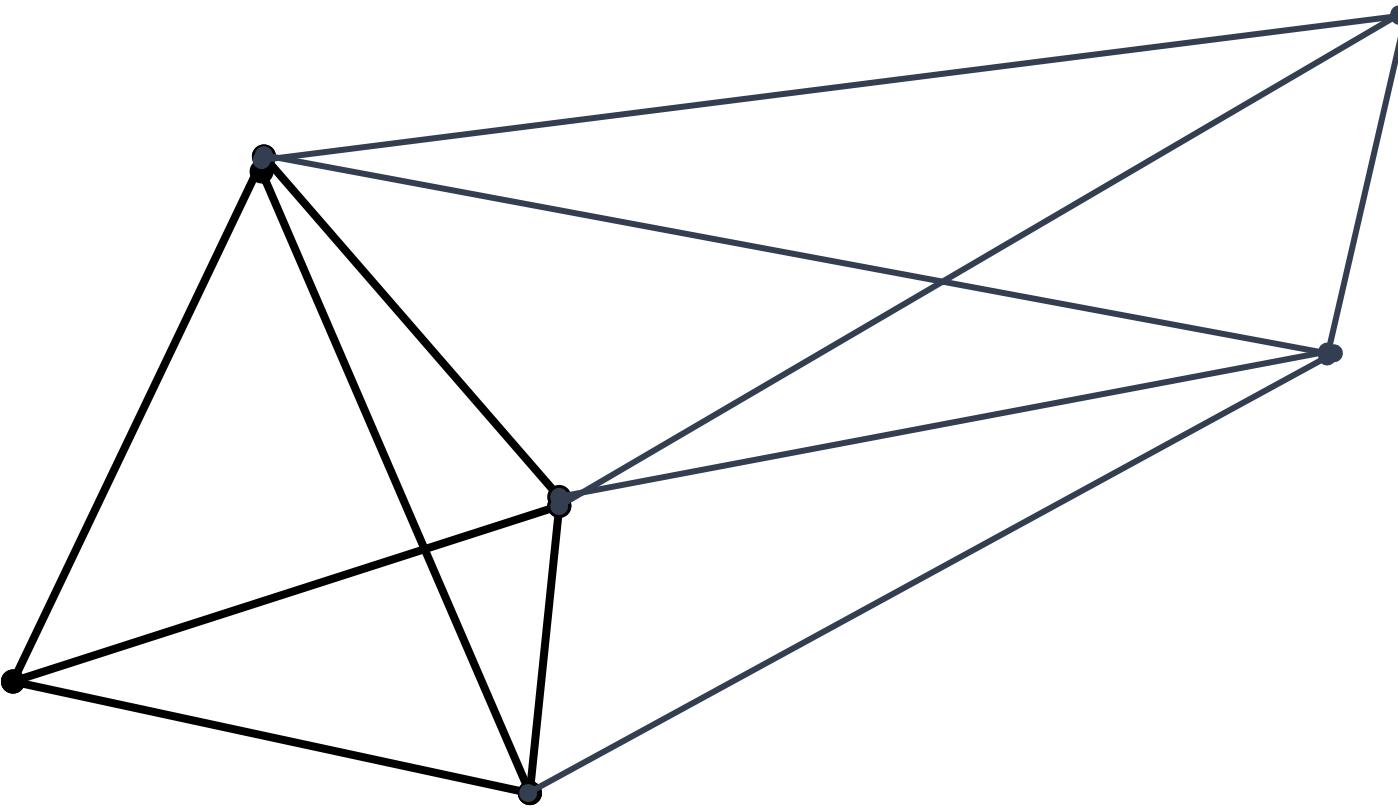
- Control changes
 - Change actuation parameters a, b, c, k
- Morphological changes
 - Add/remove spring between two exiting masses
 - Add/remove mass
 - Change mass distribution (but keep total mass constant)
 - Change rest length of existing spring
- Some operators require “cleanup” and bookkeeping
 - For example, after removing a mass you may need to remove dangling springs and update indices of other masses

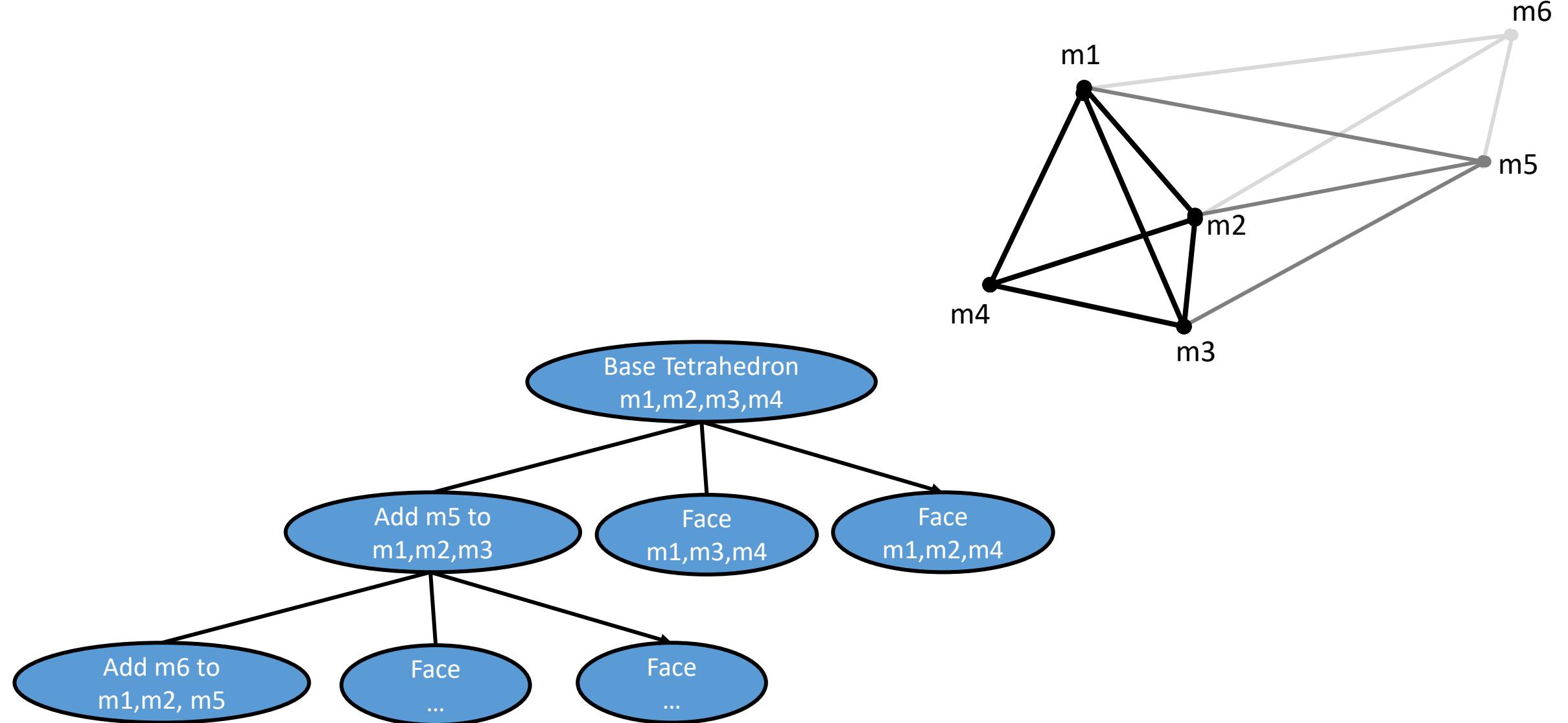


Developmental encoding



Developmental encodings

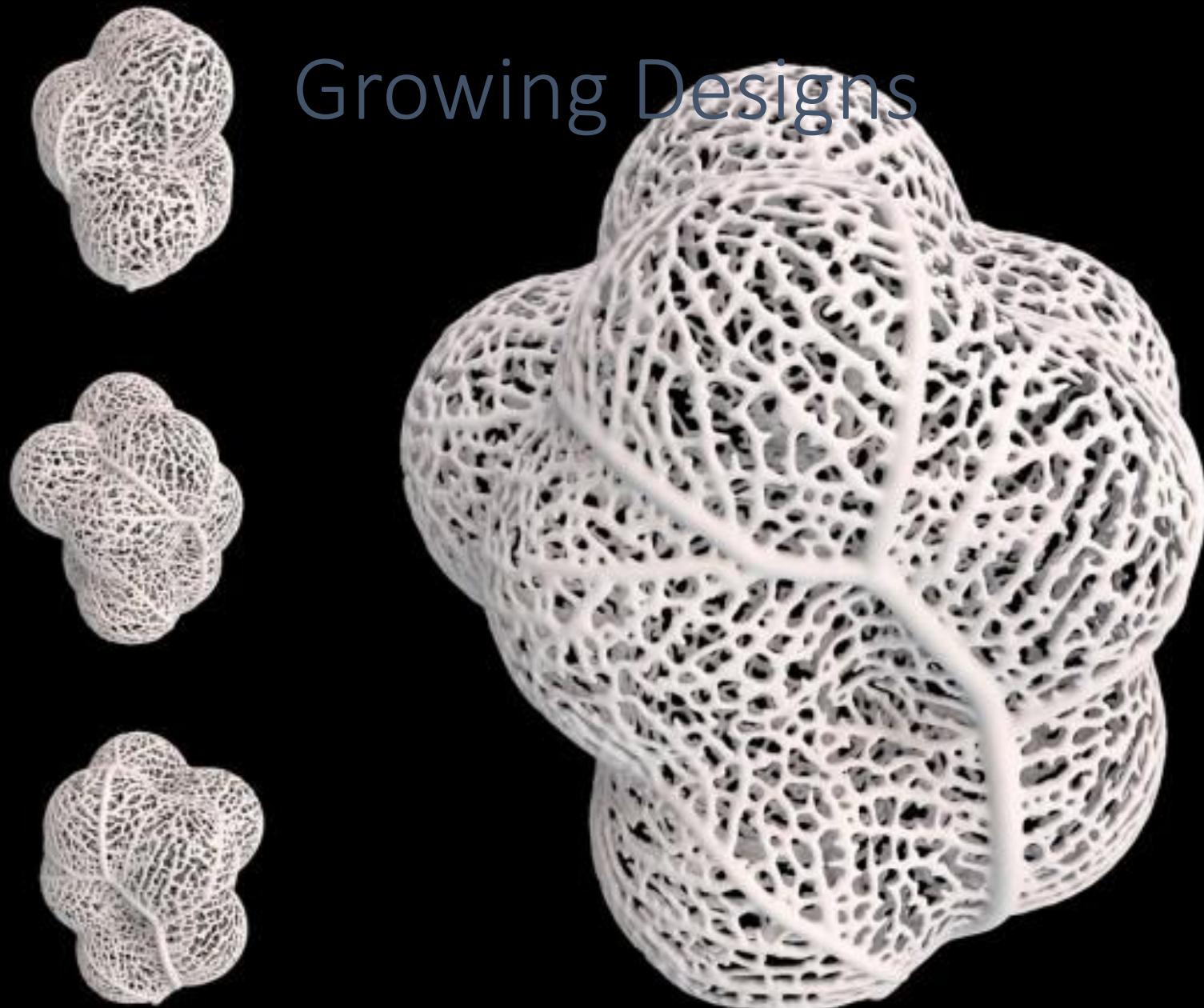




RoboGrammar: Graph Grammar for Terrain-optimized Robot Design

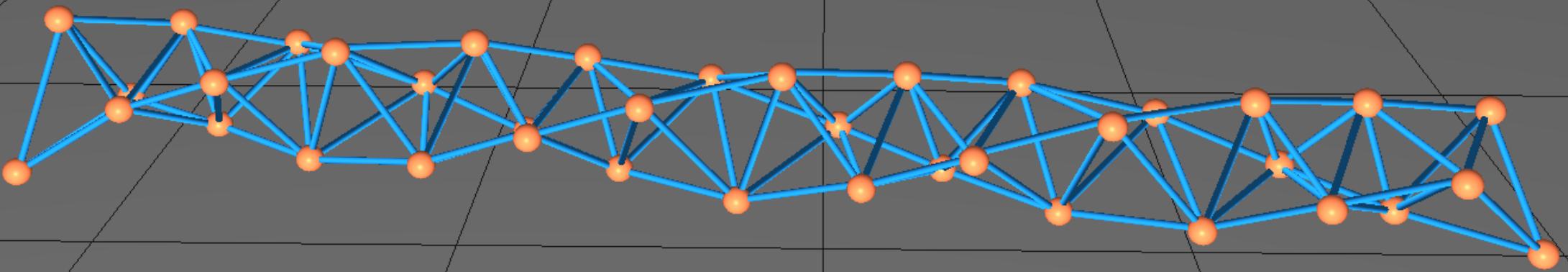


Growing Designs



Generative Blueprints

- $A \rightarrow B$ $B \rightarrow AB$
 - A
 - B
 ↓
 - AB
 ↓
 - BAB
 ↓
 - ABBAB
 ↓
 - BABABBBAB
 ↓
 - ABBABBABABBAB



Final Presentation

1. Cover page includes all information
2. Video of best robot shown
3. Multiple robot videos shown
4. Single video with multiple robot is shown
5. Robot zoo – images of nine different robots (in a 3x3 panel)
6. Nice rendering of the robot (bars, spheres, ground grid, shaded faces)
7. Post video of robot on your online portfolio (provide screenshot and link)