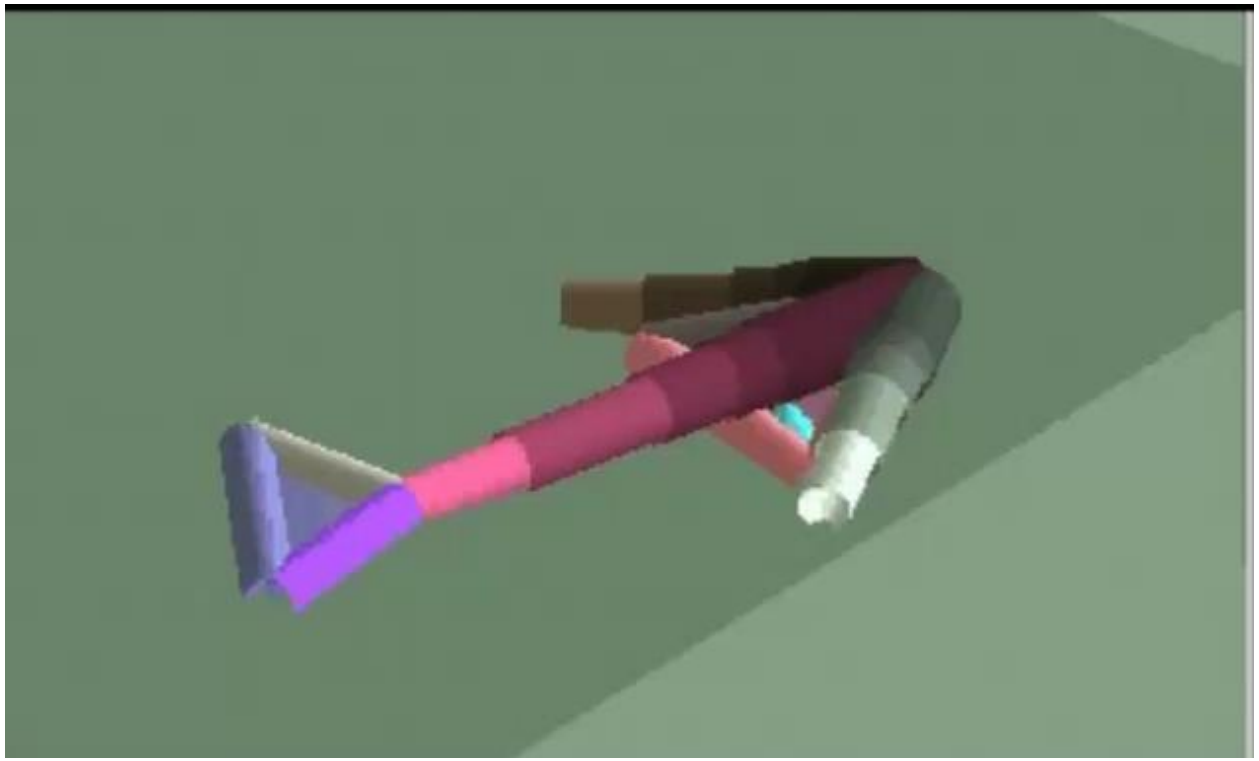
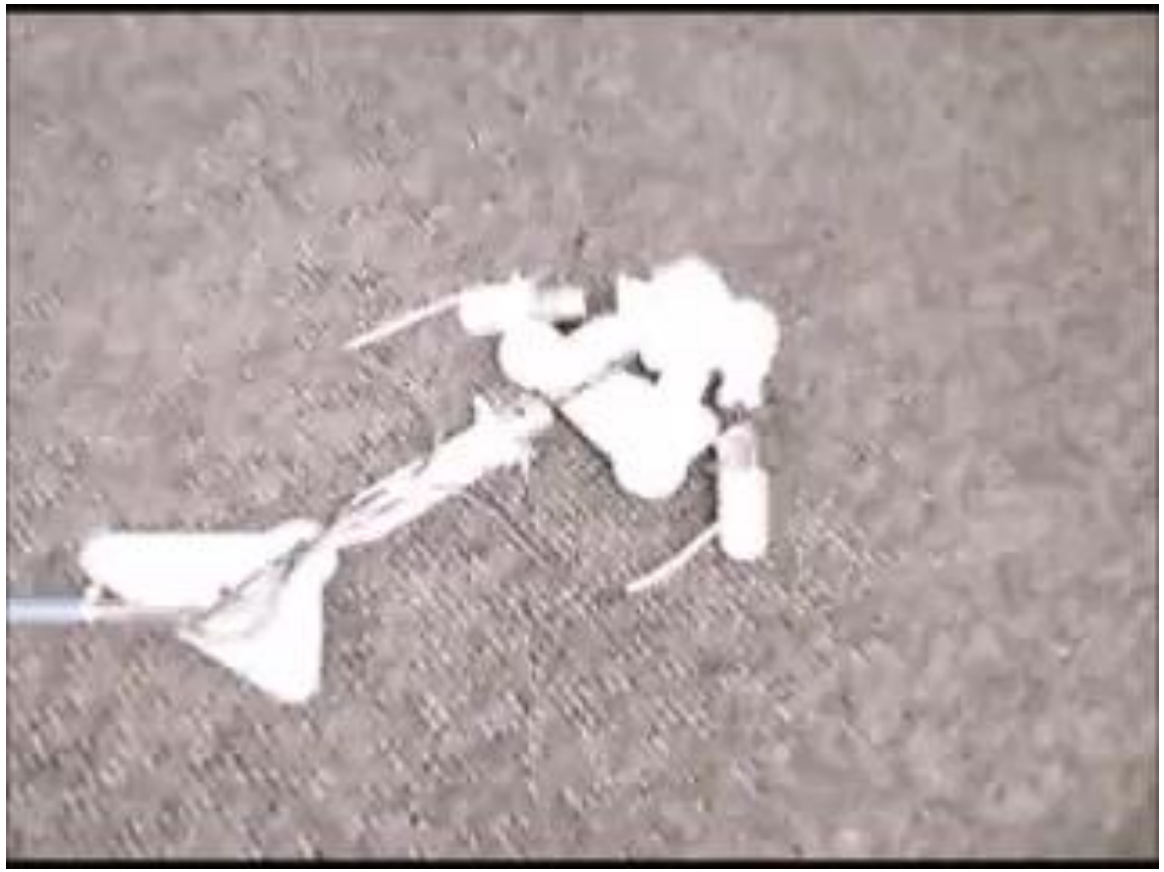
Assignment #3
Project

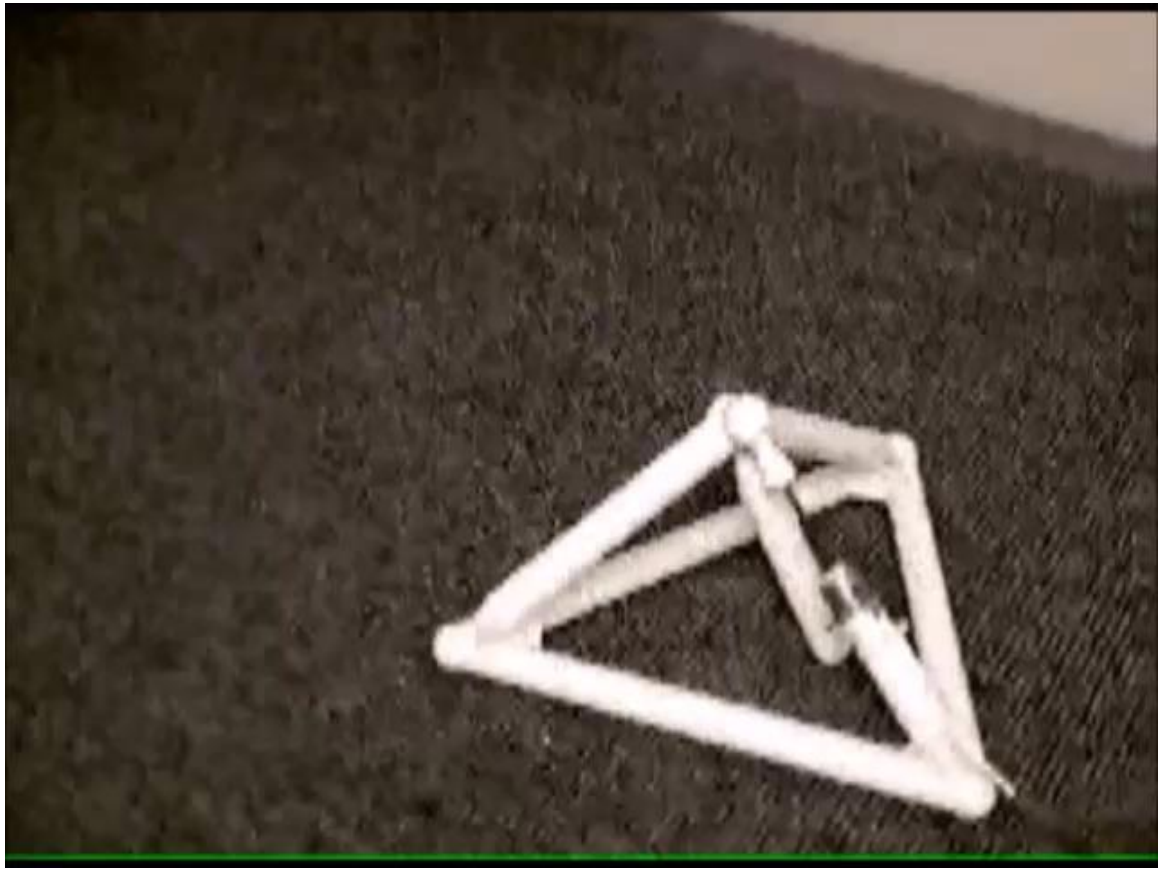
MECS 4510

Evolutionary Computation

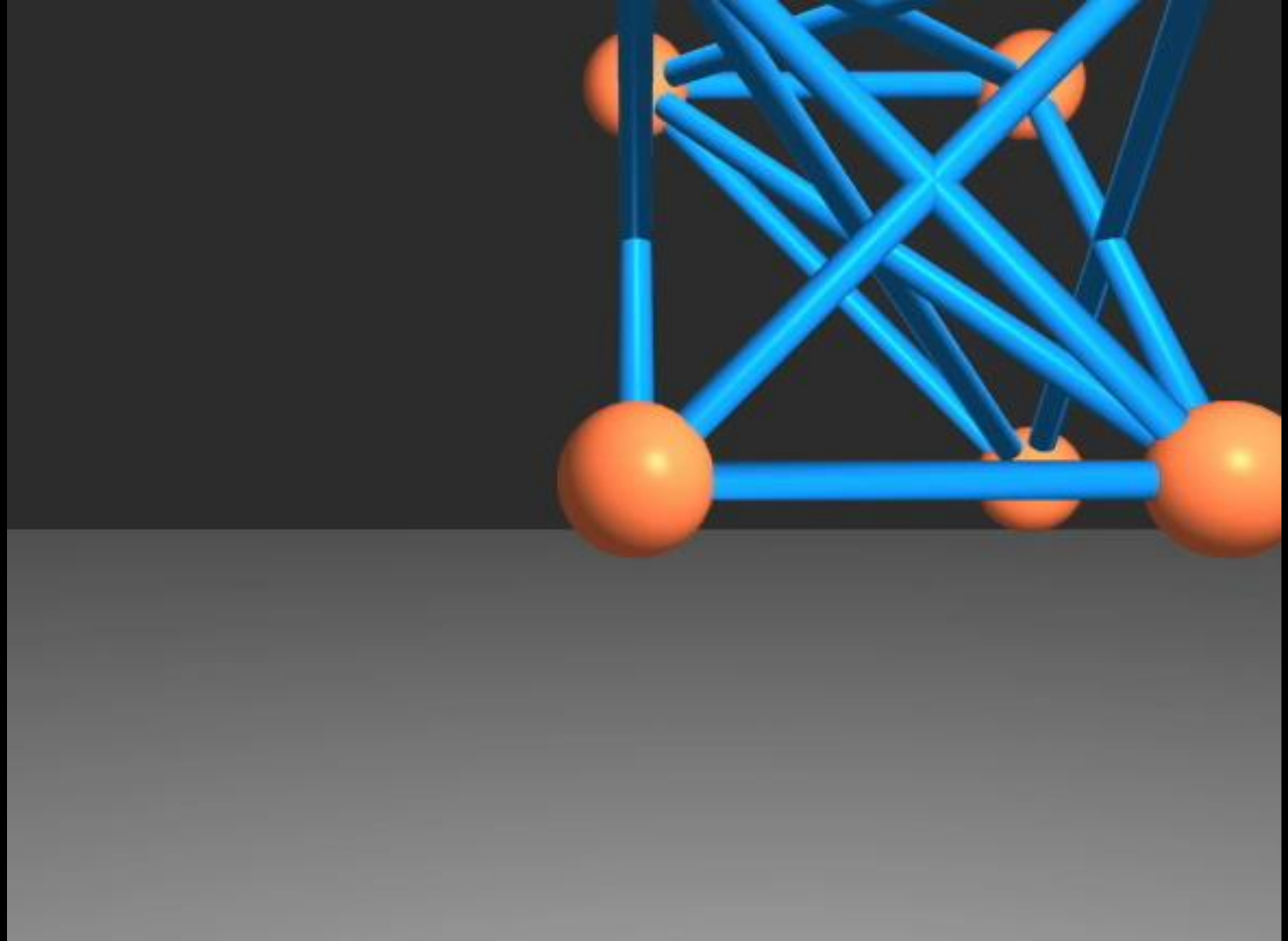
Hod Lipson







Lipson & Pollack, Nature 406, 2000



How to build a 3D simulator

Basic steps

Basic primitives

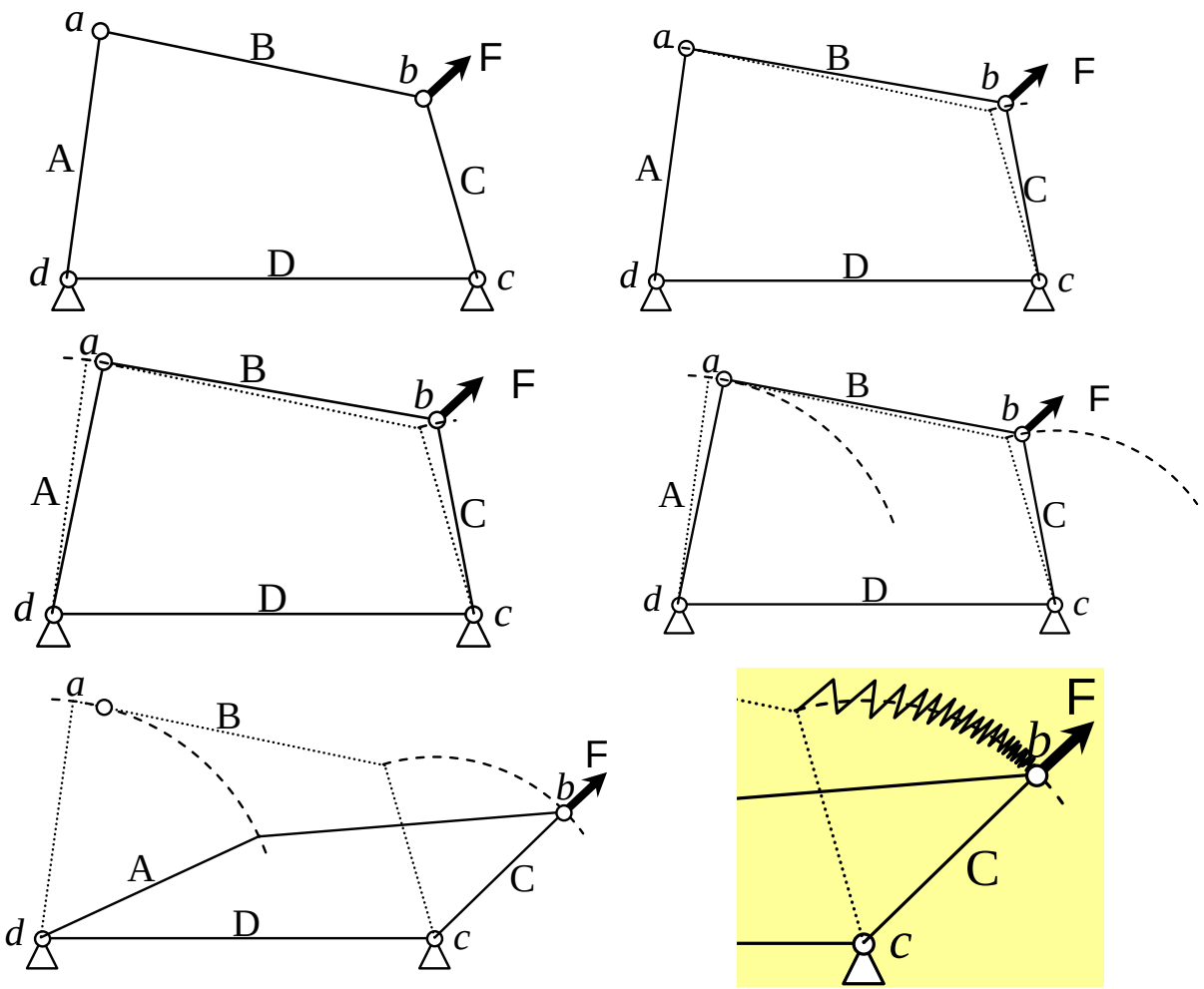
- Masses
  - Point masses (no volume)
  - Have position, velocity and acceleration
- Springs
  - Massless
  - Apply restoring forces

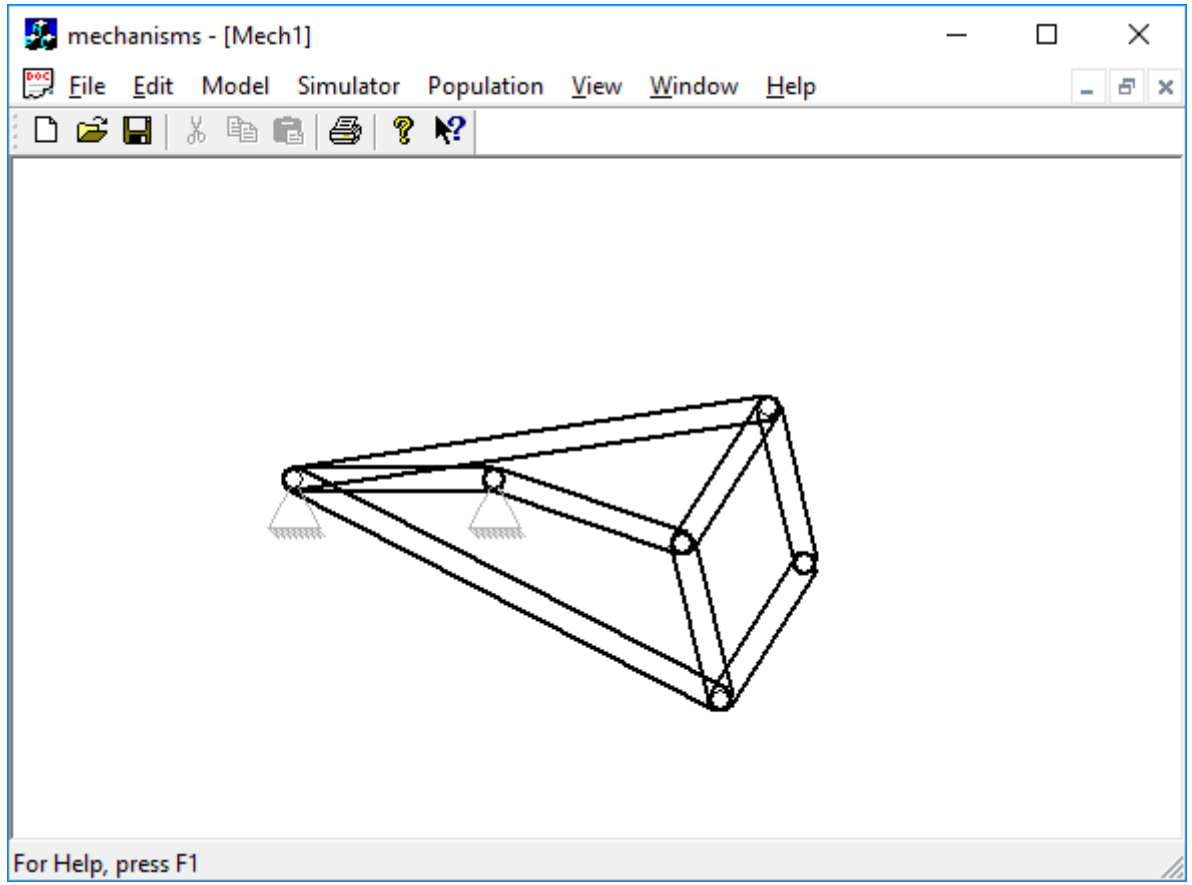
Basic simulator

- Choose a discrete time step ***dt***
  - no more 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 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




mechanisms - [Mech1]
File Edit Model Simulator Population View Window Help
For Help, press F1

First: Data structures

- Mass
  - Mass m (scalar)
  - Position $\boldsymbol{p}$ (3D Vector)
  - Velocity $\boldsymbol{v}$ (3D Vector)
  - Acceleration $\boldsymbol{a}$ (3D Vector)
  - External forces $\boldsymbol{F}$ (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)

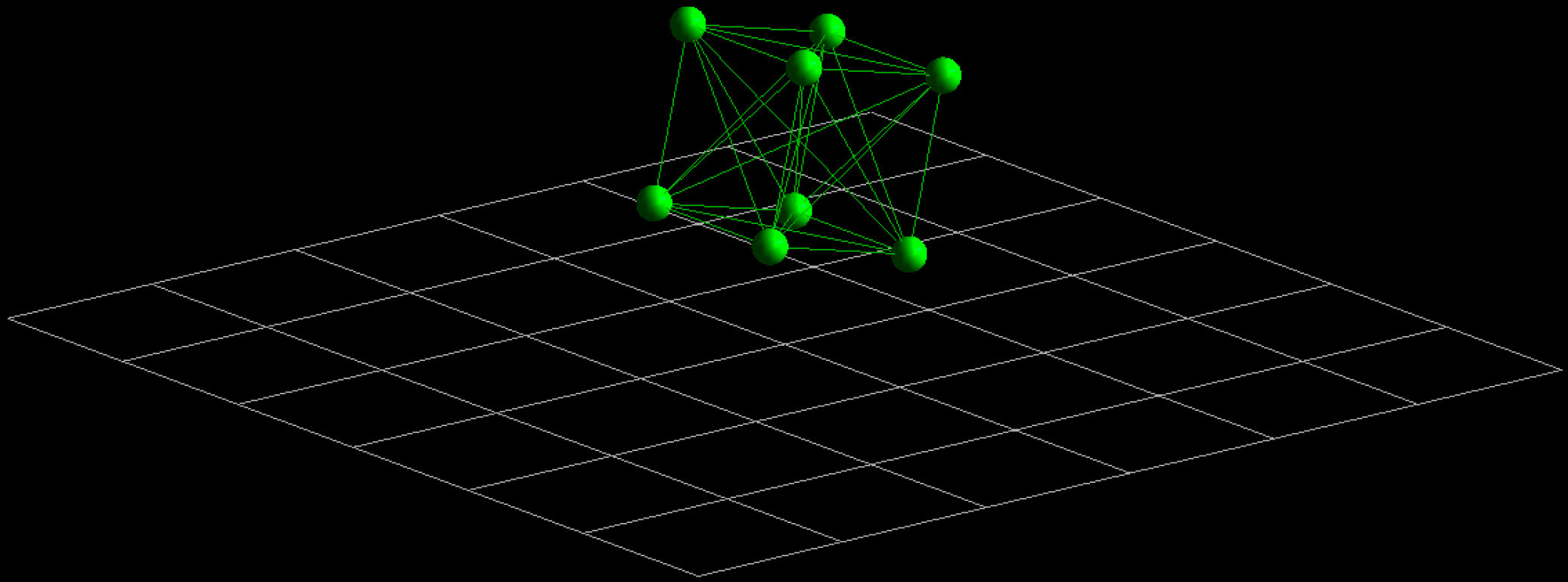
Dynamic = real time; geometry, mass, forces, velocities, accelerations

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

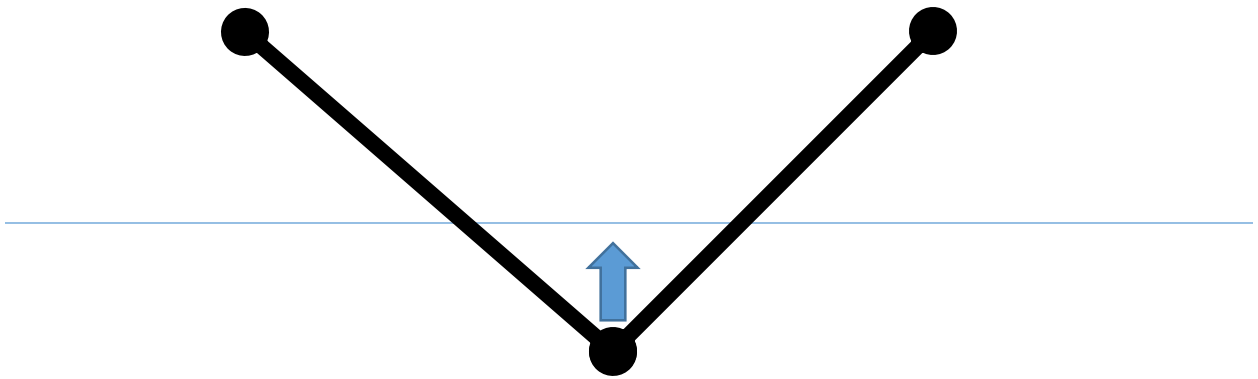
dt = simulation timestep (0.001)

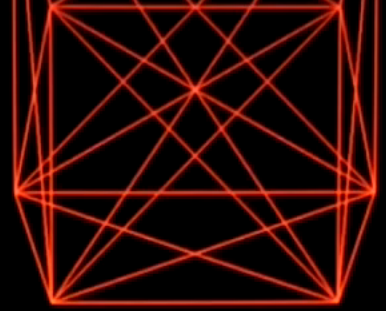
Dynamic = real time; geometry, mass, forces, velocities, accelerations



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

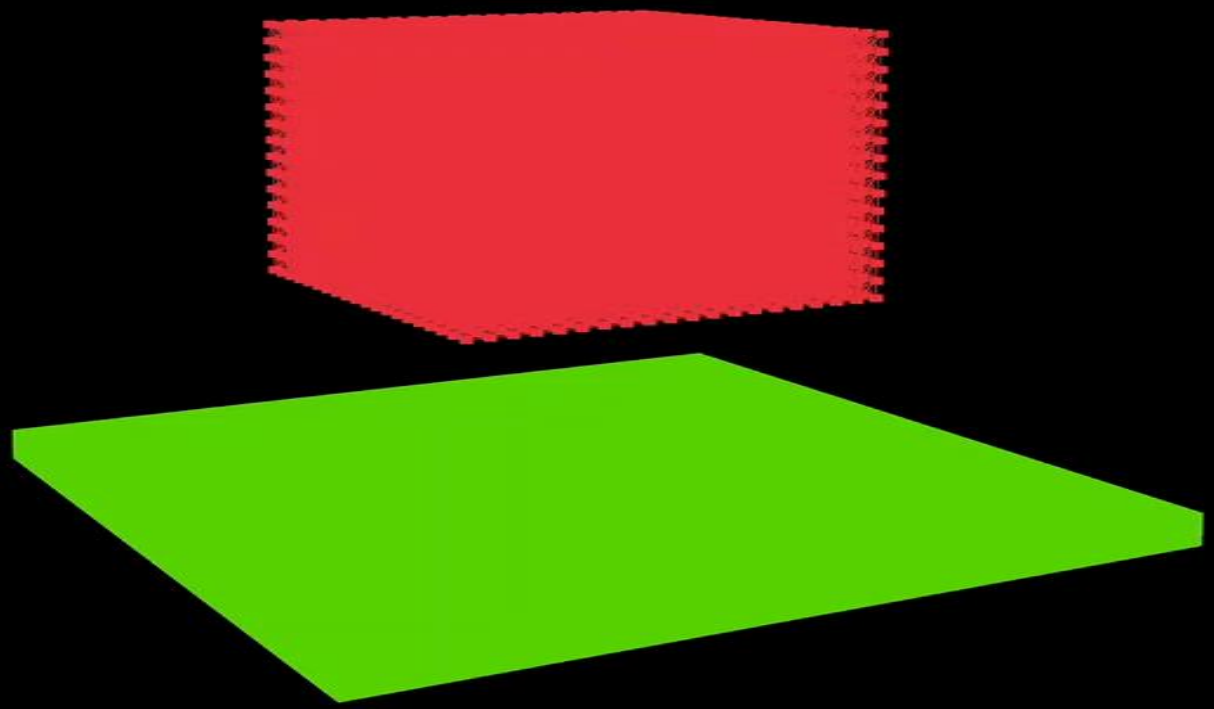




```
const double GRAVITY = 9.81;
const double damping = 0.9999;
const double DT = 0.0008; // simulation timestep
const double friction_mu_s = 1; // friction coefficient rubber-concrete
const double friction_mu_k = 0.8;// friction coefficient rubber-concrete
const double k_vertices_soft = 2000; // spring 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



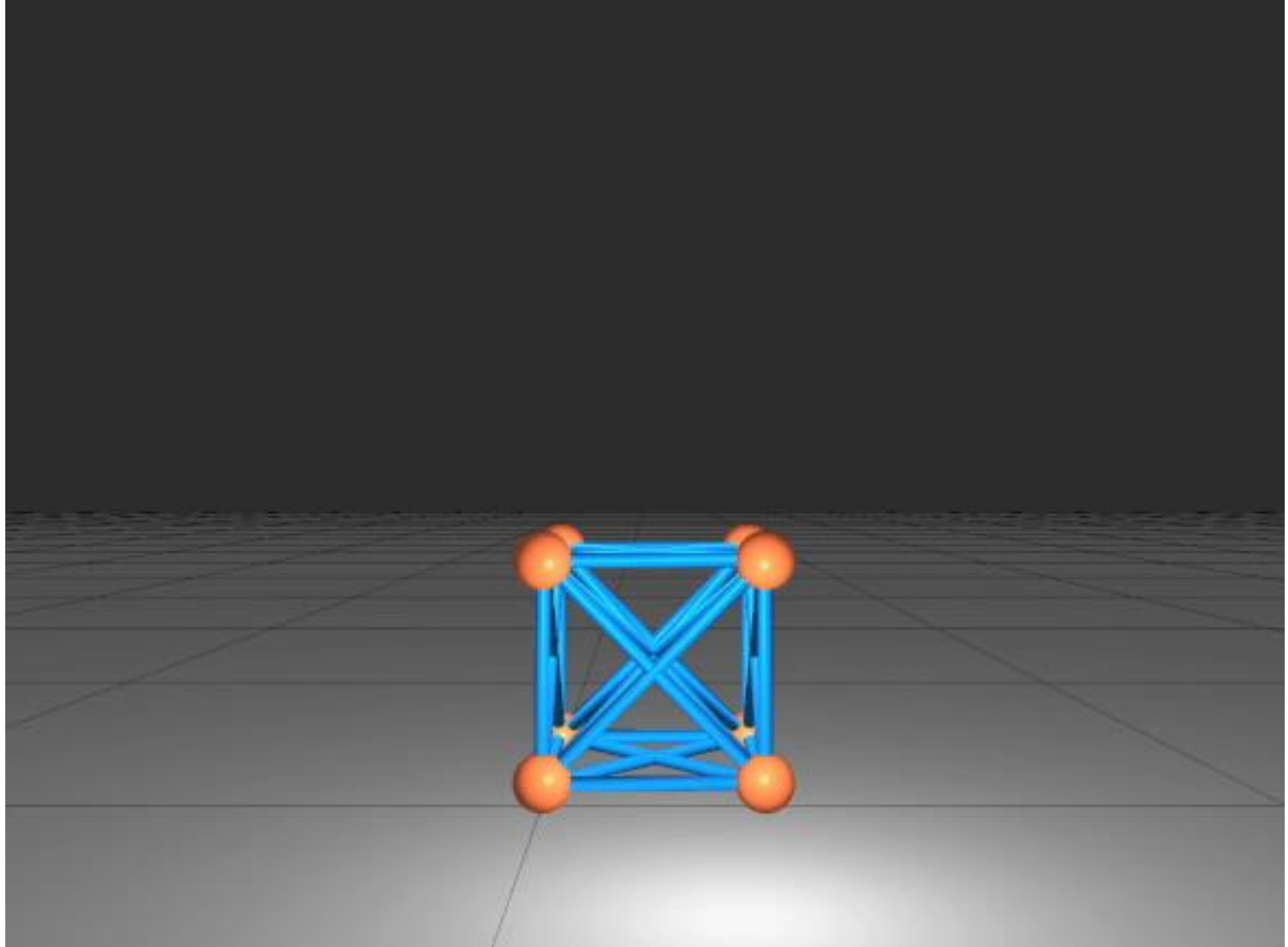
GPU- Accelerated Physis Simulations with CUDA

- 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;
}
```

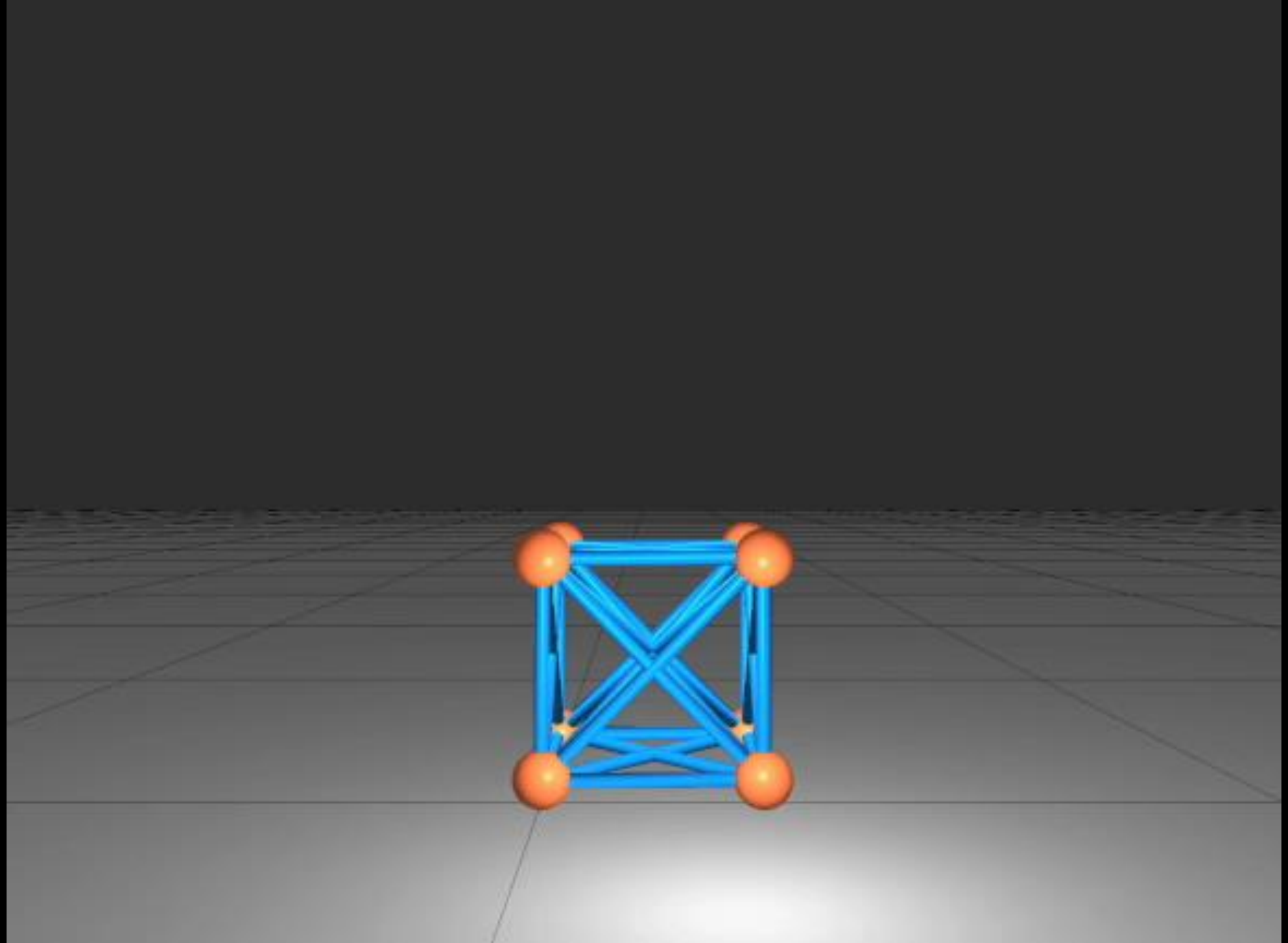
Titan

Interested in CUDA library? Contact jacob.austin@columbia.edu and Rafael Corrales Fatou rc2997@columbia.edu



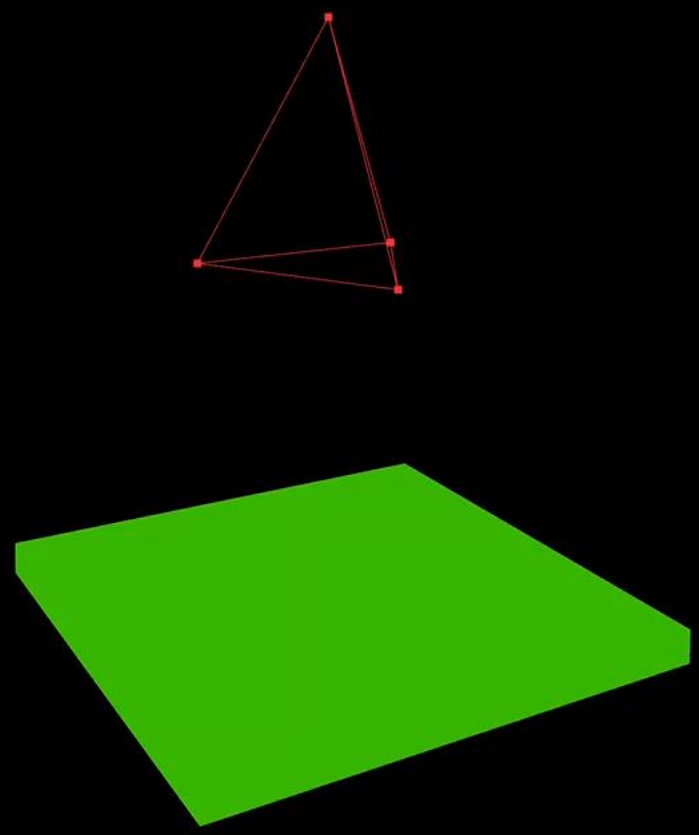
Create animated structures

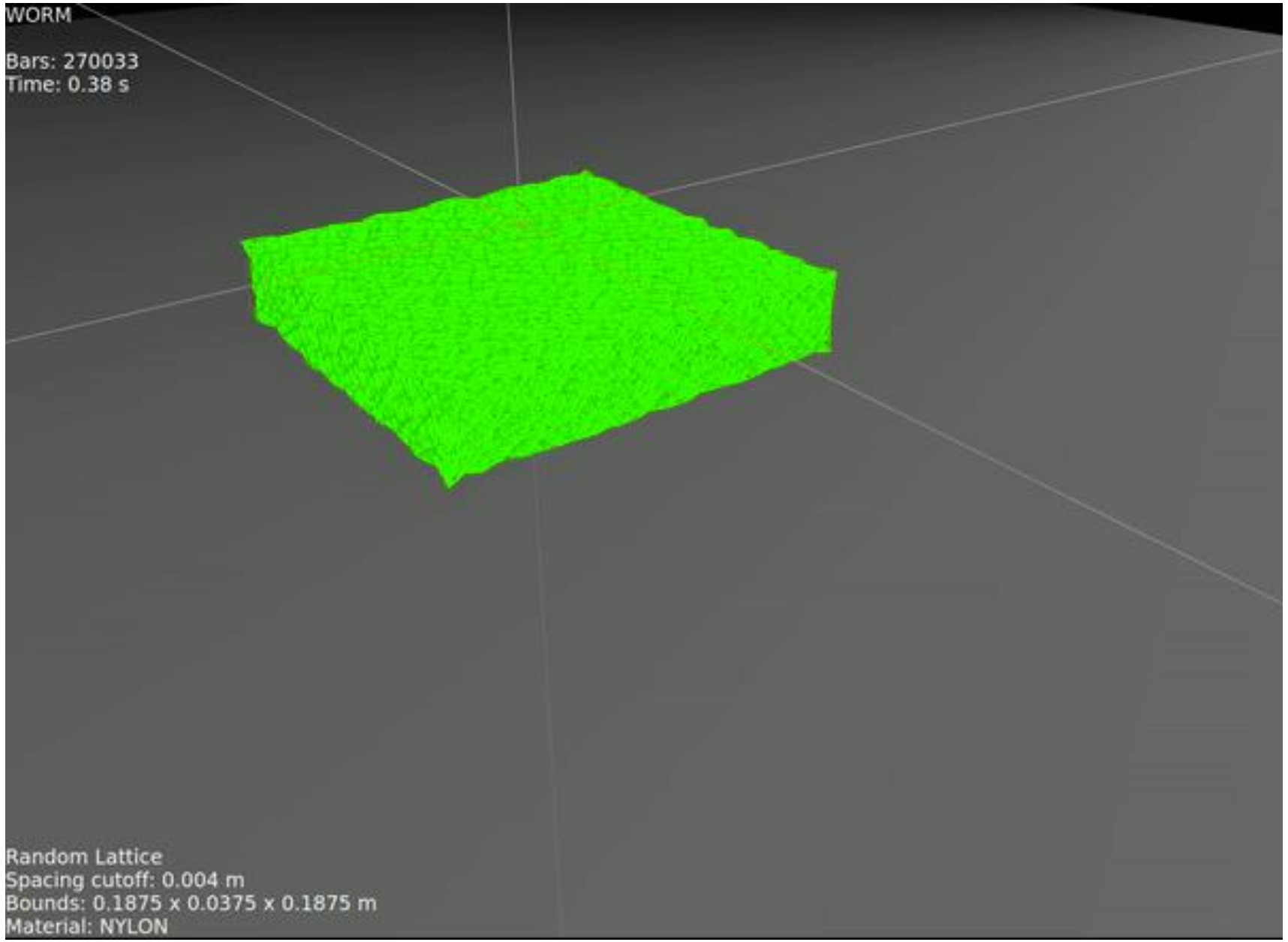
- Dynamically change L_0 of various springs
  - For example, change
    - $L_0=a+b*sin(\omega T+c)$
- The coefficient ω is a global constant that determines the frequency of oscillations.
  - Setting ω =1 will result in a period of 2π sec
- Chose *a,b,c* randomly within a reasonable range, for each spring.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

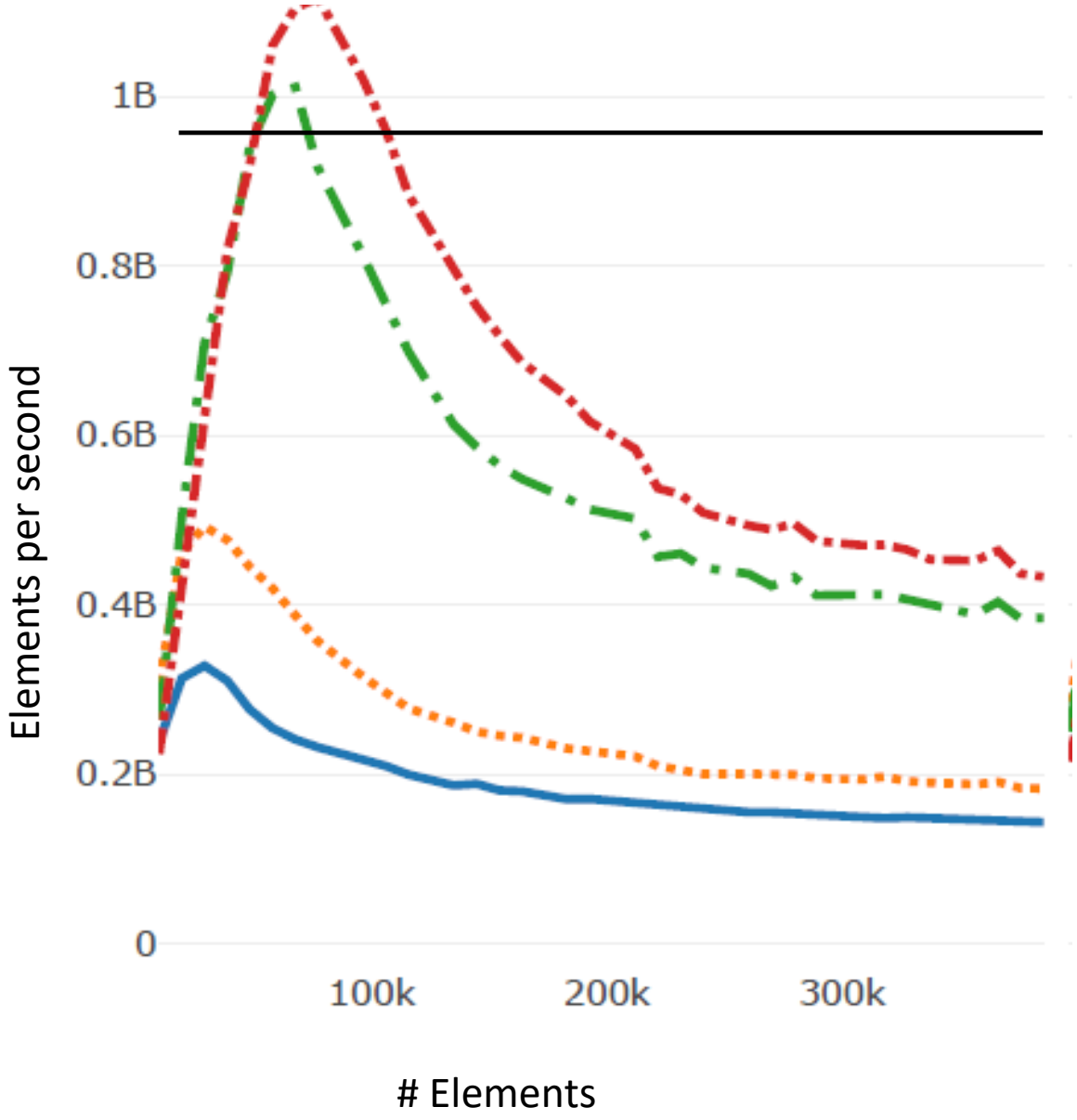


Simulation efficiency

- 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

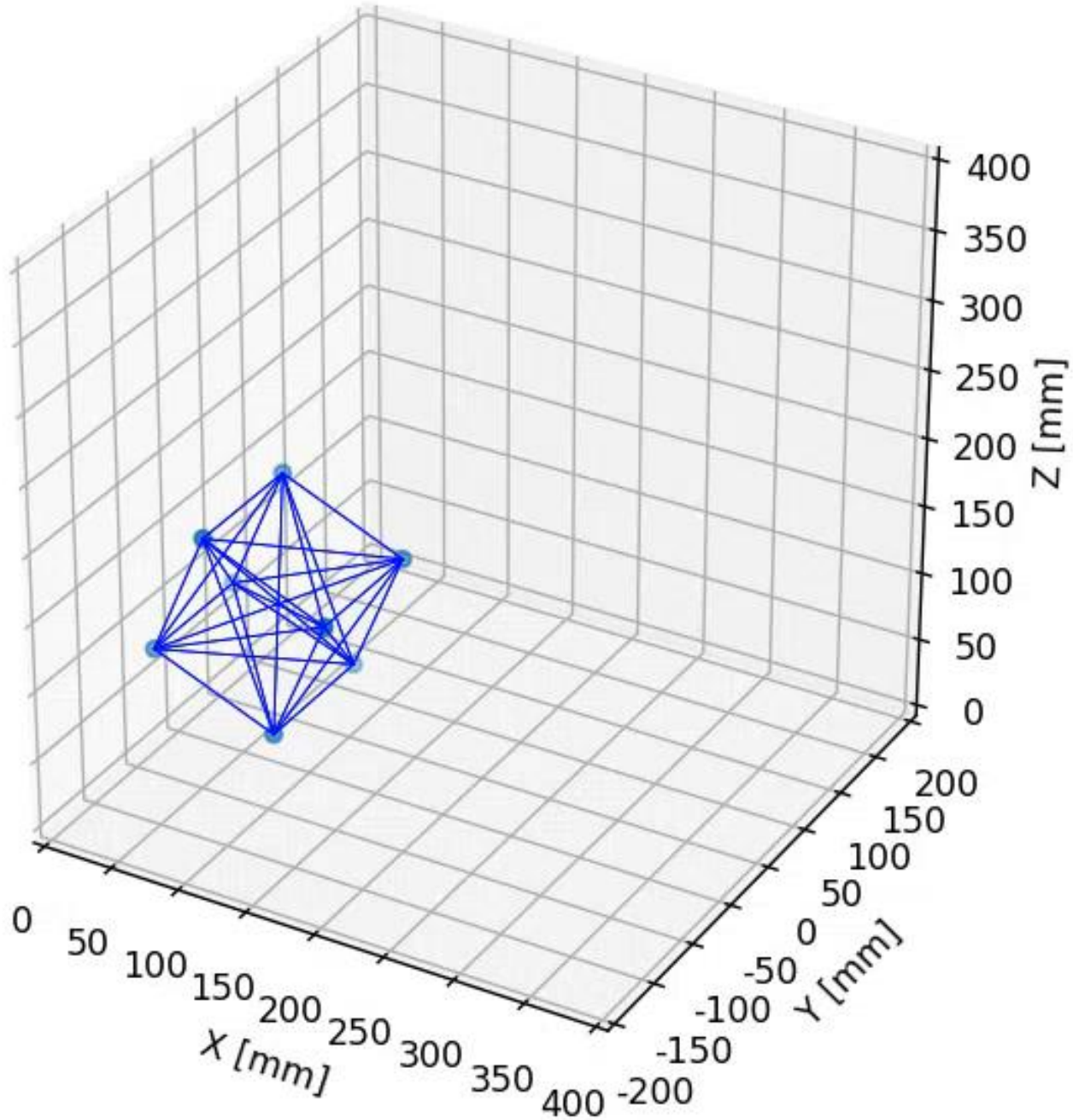

1B
0.8B
0.6B
0.4B
0.2B
0
Elements per second
100k
200k
300k
Elements

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


400
350
300
250
200
150
100
50
0
Z [mm]
200
150
100
50
0
-50
-100
-150
-200
Y [mm]
0
50
100
150
200
250
300
350
400
X [mm]

Boxi Xia -

vpython.org/

VPython

3D Programming for Ordinary Mortals

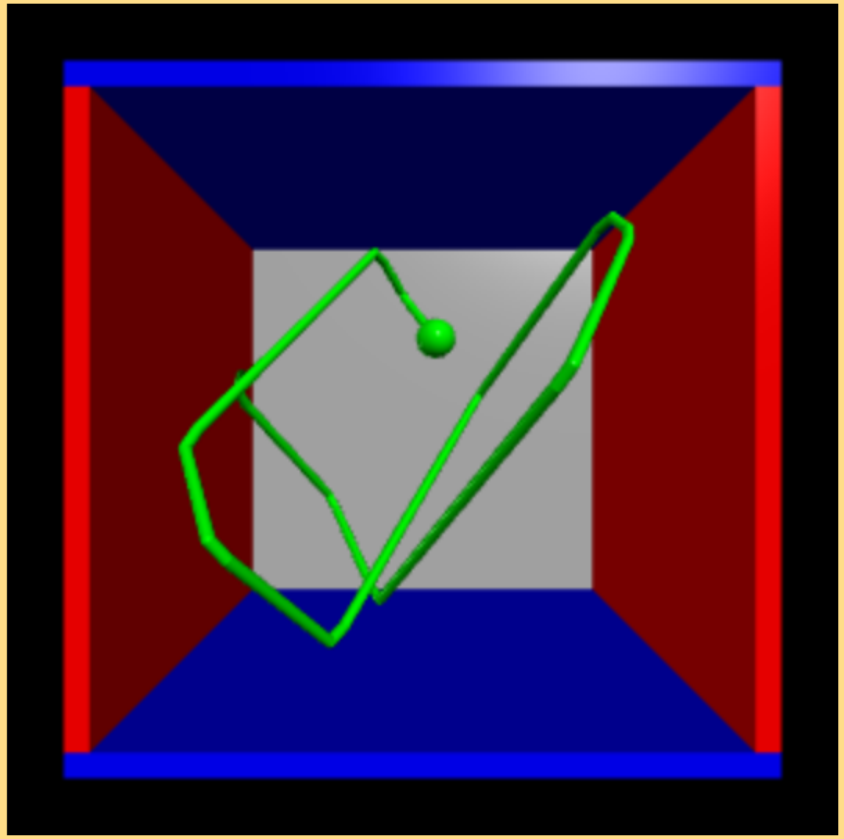
Examples

Documentation

GlowScript VPython User Forum

VPython 7 User Forum

glowscript.org

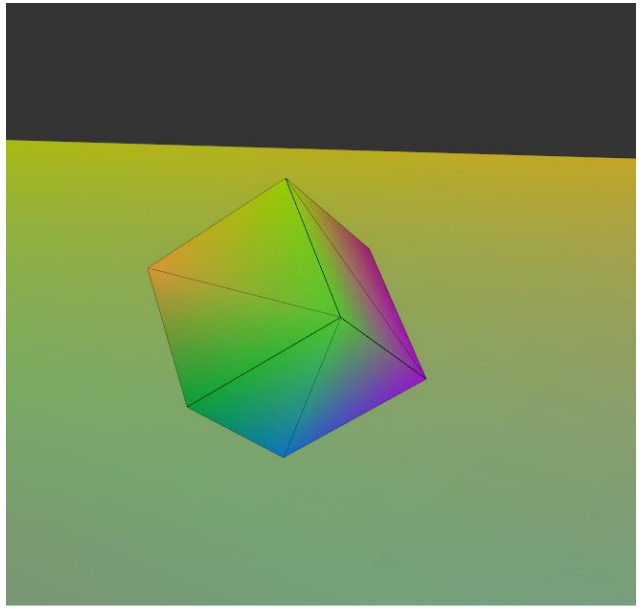


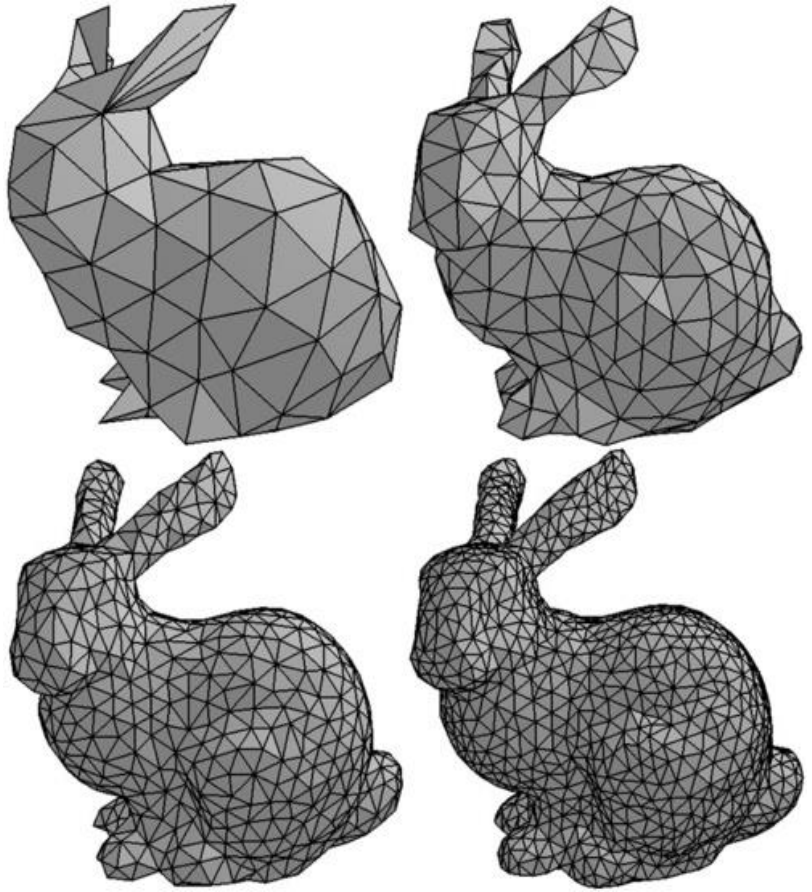
Pause

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.

Solid appearance

- You scan 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

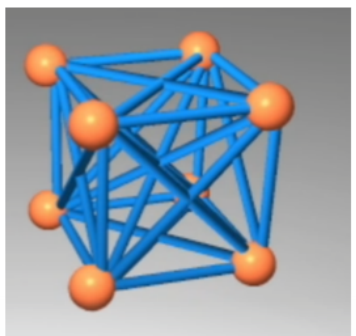
These instructions explain how to create a basic spring-mass physics simulation. You can implement this in any language. You will need to be able to plot some 3D graphics (lines and points). Assume that ***x,y*** plane is horizontal and ***z*** axis is pointing upwards.

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/s^2)
    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)



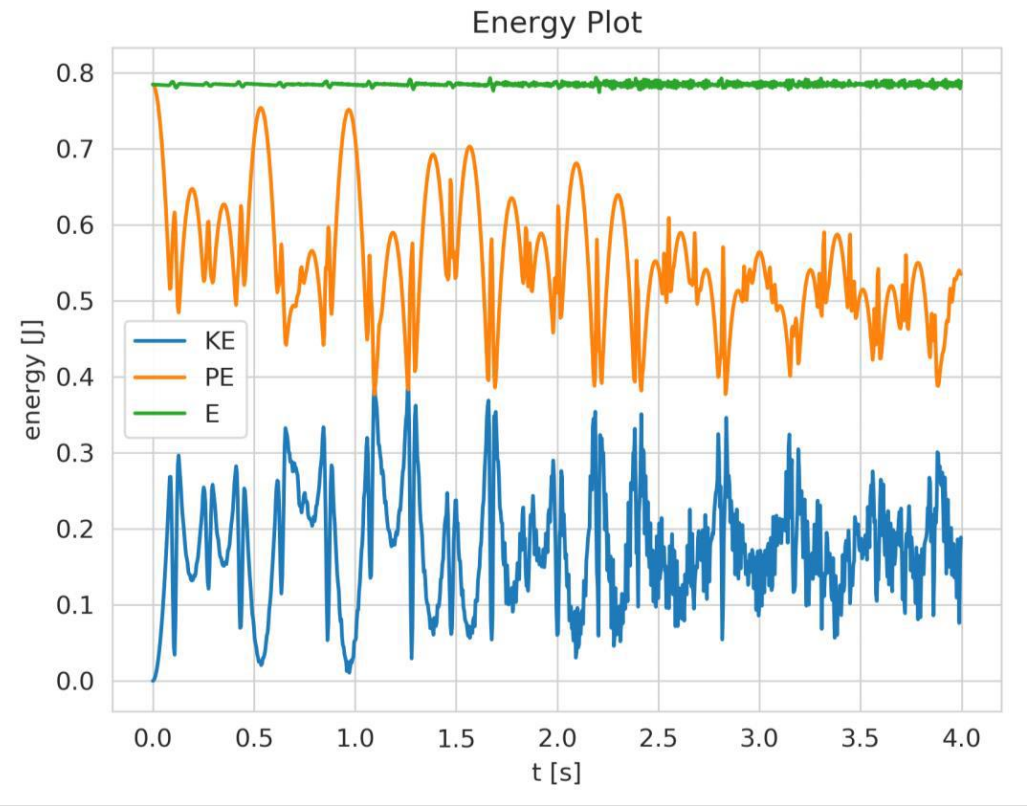
3. Set Global variables of the simulation:

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.

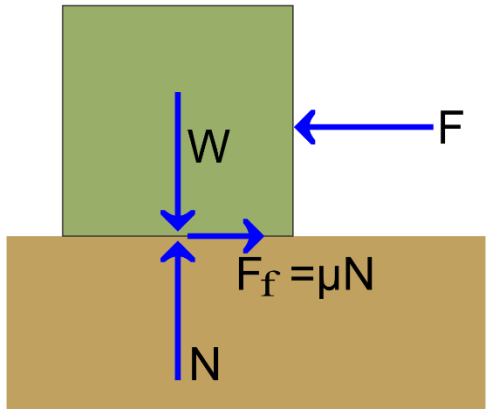
Validation

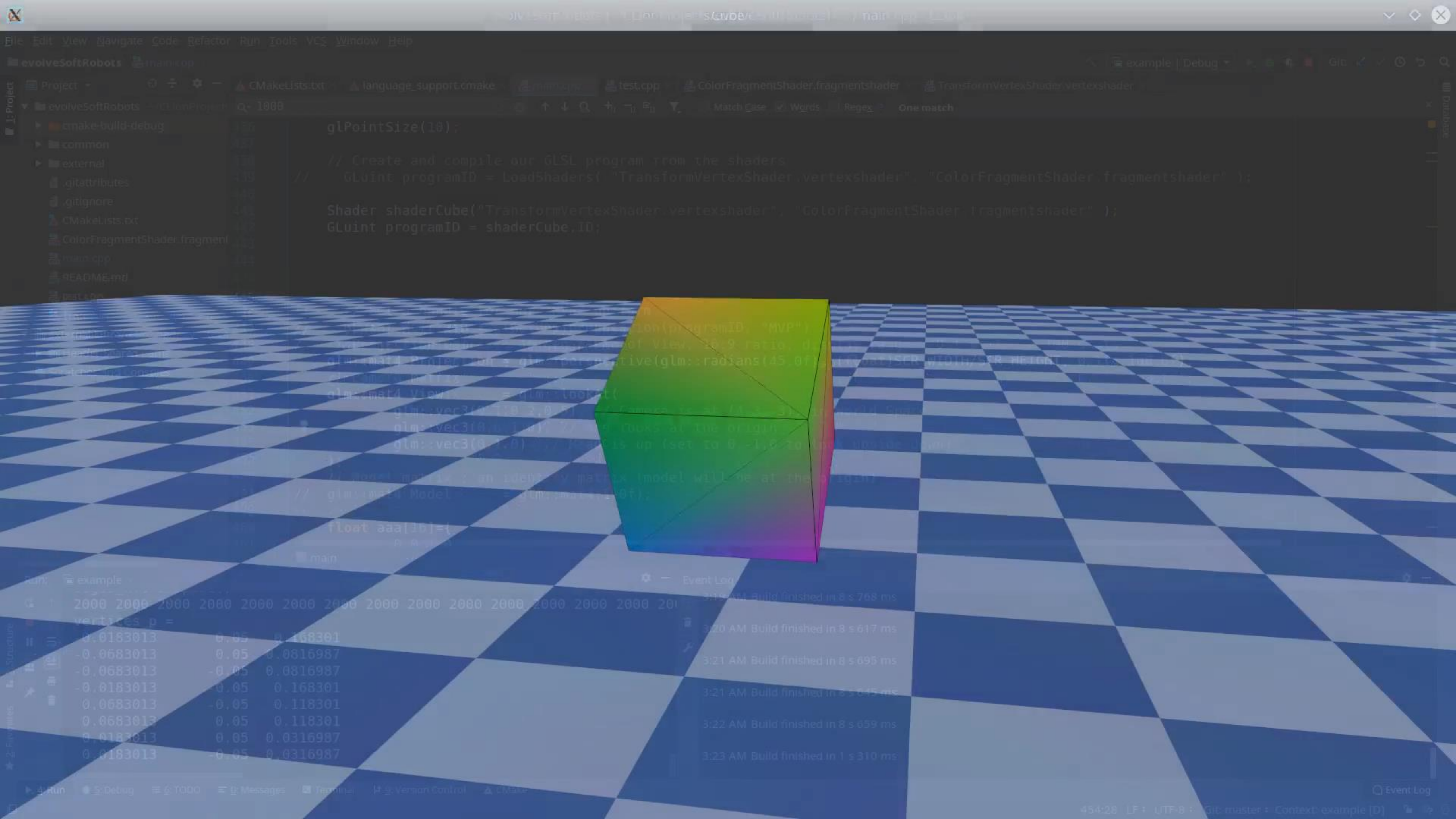
- If there is no damping or friction, energy should remain constant
  - Plot the total energy of the cube as function of time (kinetic energy, potential energy due to gravity, potential energy in the springs, as well as the energy related to the ground reaction force). The sum should be nearly constant. 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 $F_H=\text{sqrt}(F_x^2+F_y^2)$ 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 μ_k*F_n
- Update the mass velocity based on the total forces, including both friction and ground reaction forces



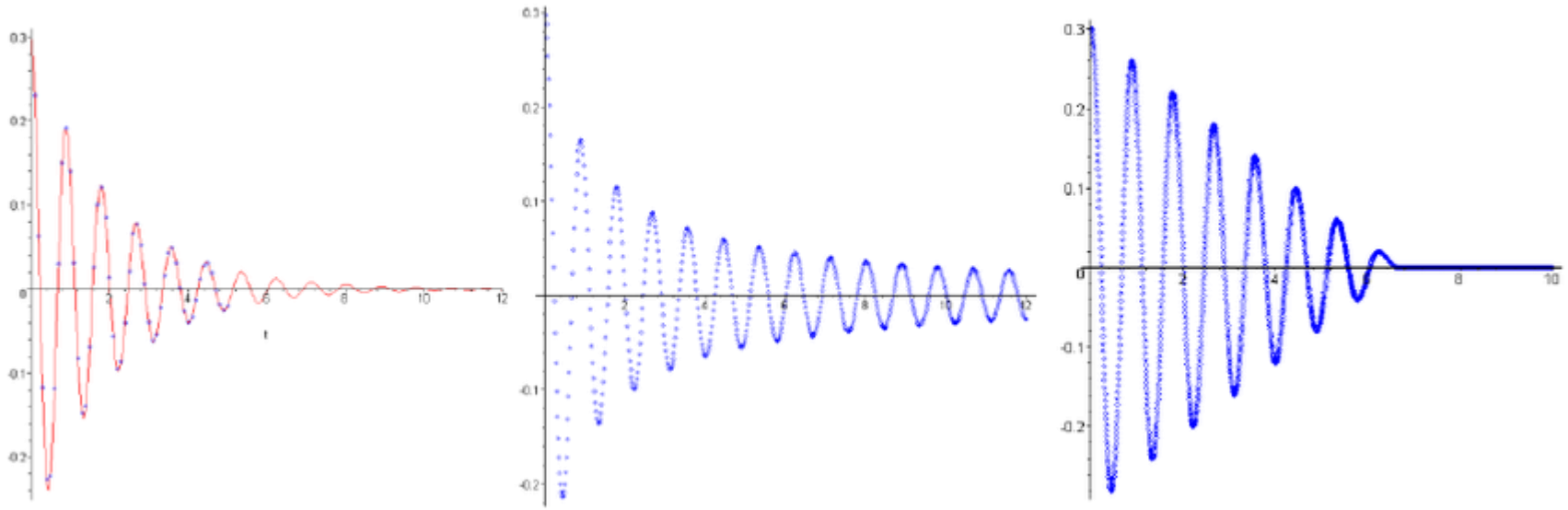

Cube
File Edit View Navigate Code Refactor Run Tools VCS Window Help
evolveSoftRobots
example | Debug
Project
CMakeLists.txt
language_support.cmake
test.cpp
ColorFragmentShader.fragmentshader
TransformVertexShader.vertexshader
1000
Match Case
Words
Regex
One match
cmake-build-debug
common
external
.gitattributes
.gitignore
CMakeLists.txt
ColorFragmentShader.fragment
README.md
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
Run:
example
Event Log
2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 20
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
3:20 AM Build finished in 8 s 617 ms
3:21 AM Build finished in 8 s 695 ms
3:22 AM Build finished in 8 s 659 ms
3:23 AM Build finished in 1 s 310 ms
4: Run
5: Debug
6: TODO
0: Messages
Terminal
9: Version Control
CMake
Event Log

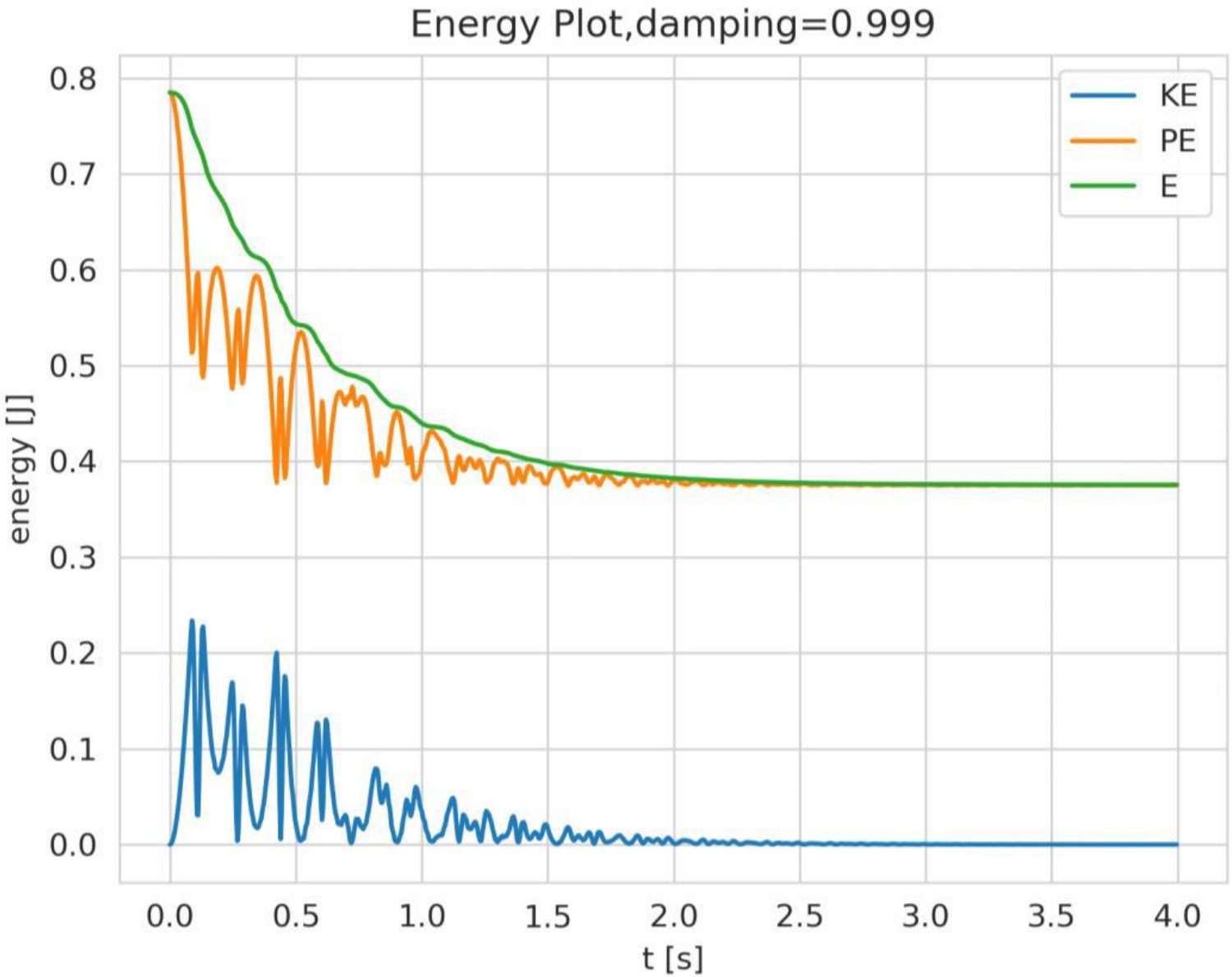
Coefficients of Friction for Common Surfaces

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 **V** by 0.999 (or similar) each time step
  - Helps reduce vibration, wobbling, and snapping

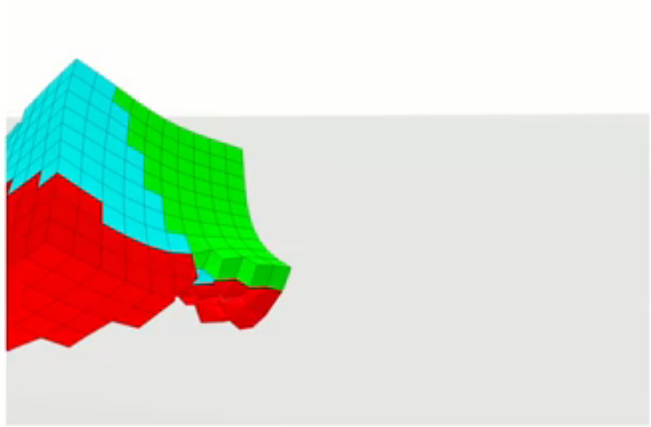



Energy Plot,damping=0.999
KE
PE
E
energy [J]
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
t [s]

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

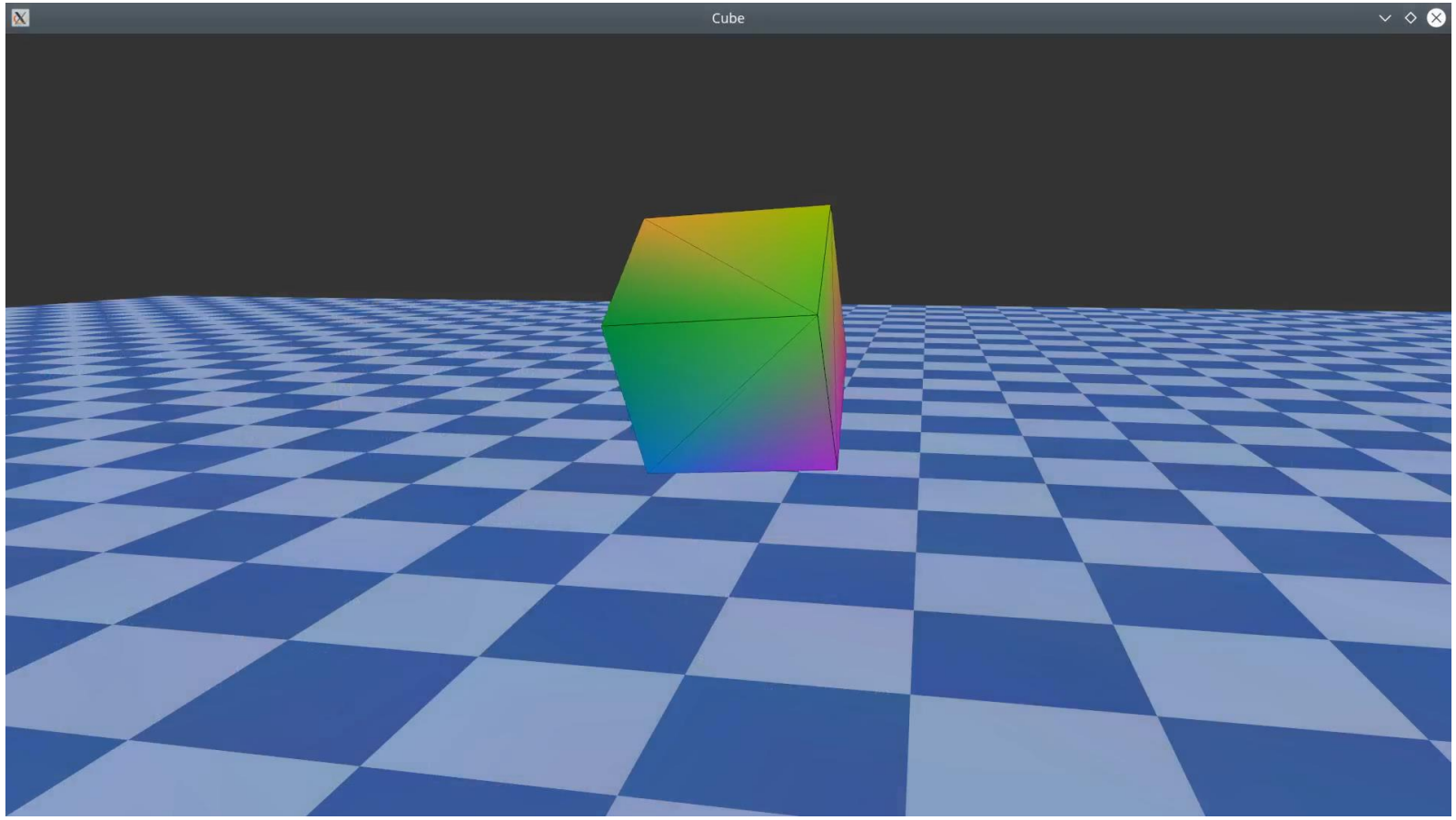


Cheney et al., 2013

Corucci et al., 2016

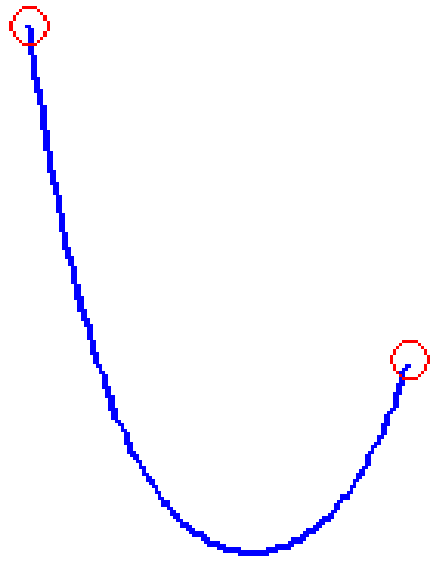
Anchors

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



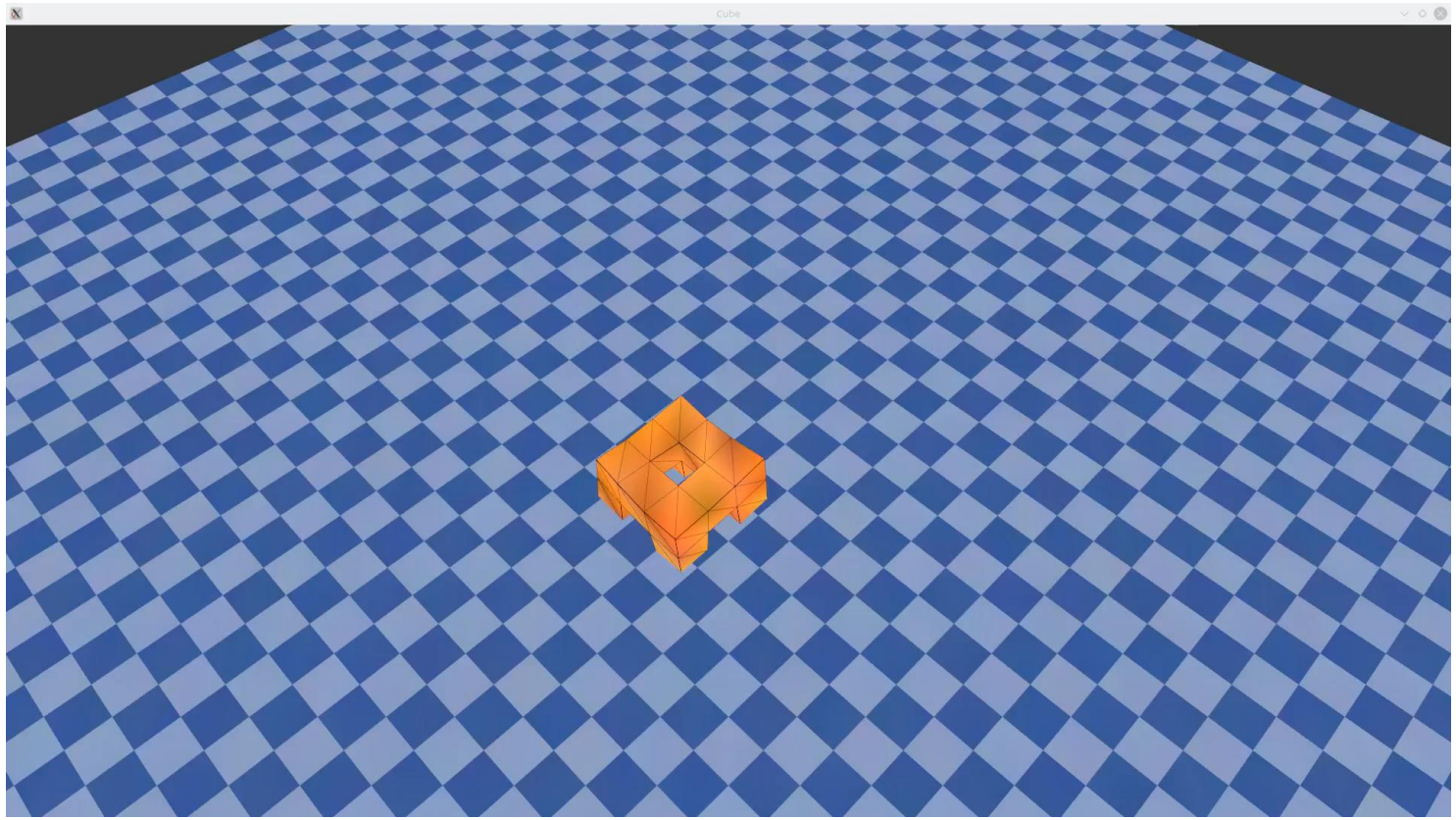
Add nonlinear behavior

- Replace $F=k(L-L_0)$ with some nonlinear behavior
- Cable
  - $F=k(L-L_0)$ if L>L0
  - F=0 if L<= 0



Assignment 3b

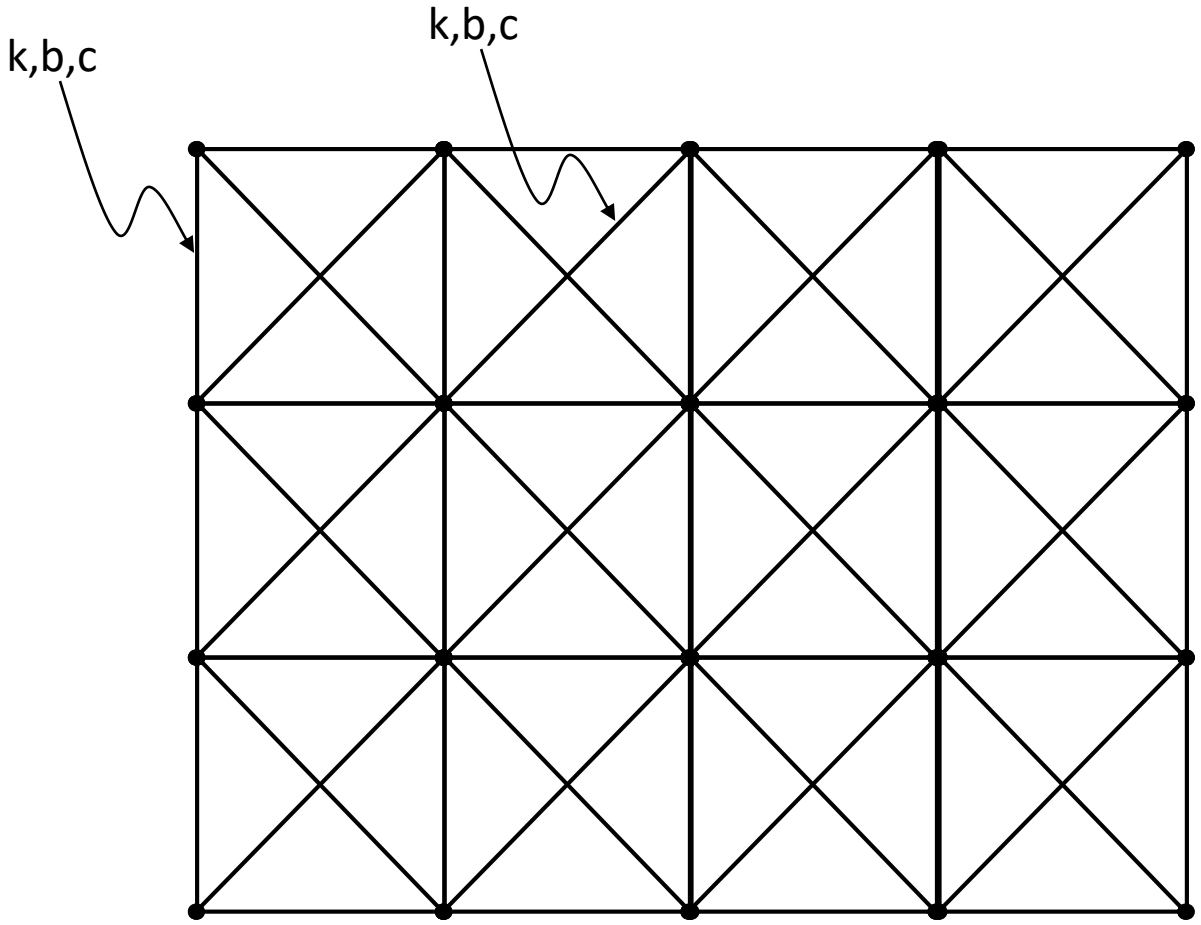
- Parametrically change spring rest lengths to make robot walk

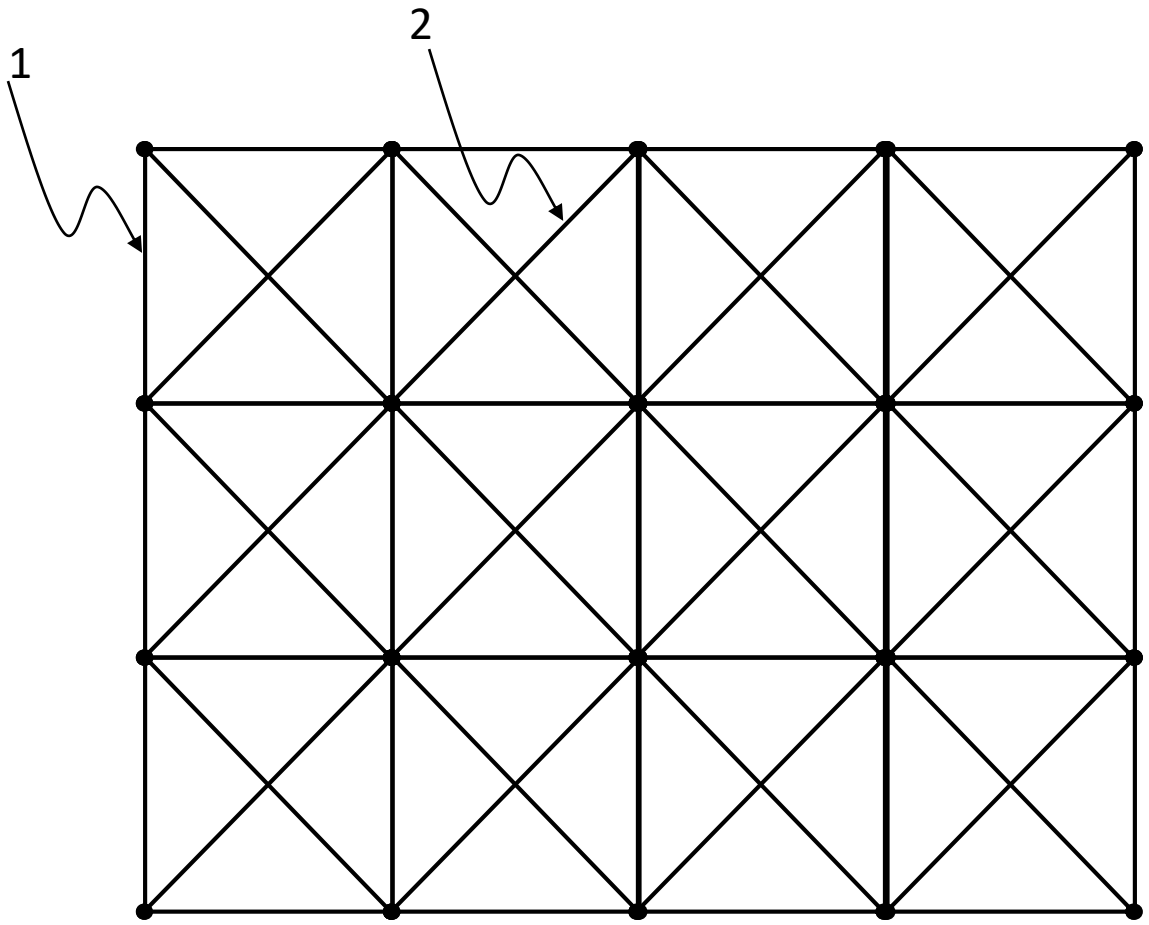


Evolving Motion

- For each spring set k and L_0 = a+b*sin(wt+c)
  - w is global frequency. *a,b,c,k* are spring parameters (*k* is spring coef)
- Evolve locomotion
  - Fitness: net distance travelled by center of mass
- Representation
  - Direct encoding: Chromosome with *a,b,c,k* for each spring
  - Indirect encoding:
    - Each spring has index into a table of materials. Each materials has k,a,b,c
    - Each cube has index into a table of materials. Each materials has k,a,b,c
    - Each region has index into a table of materials. Each materials has k,a,b,c
    - …

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

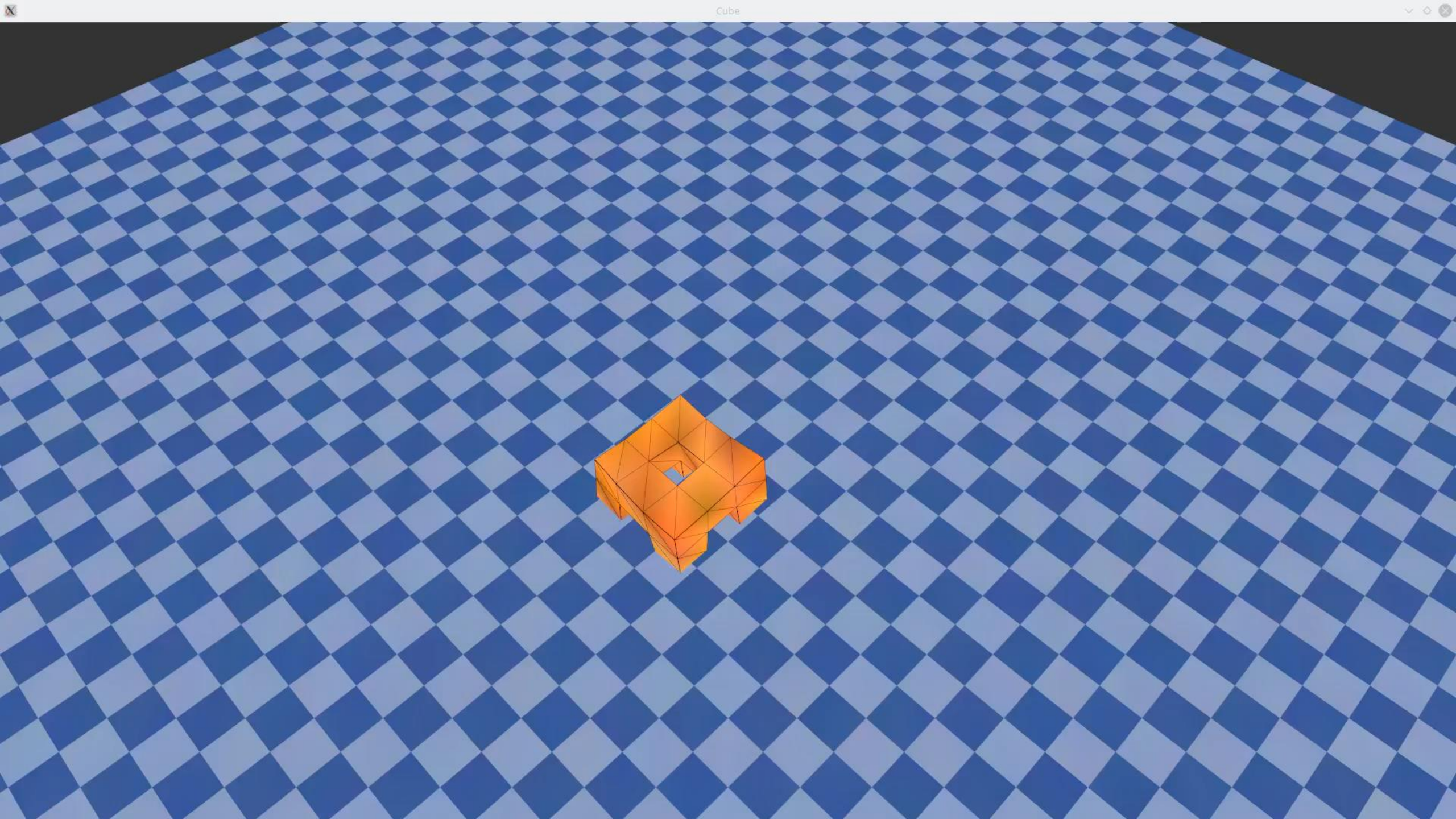


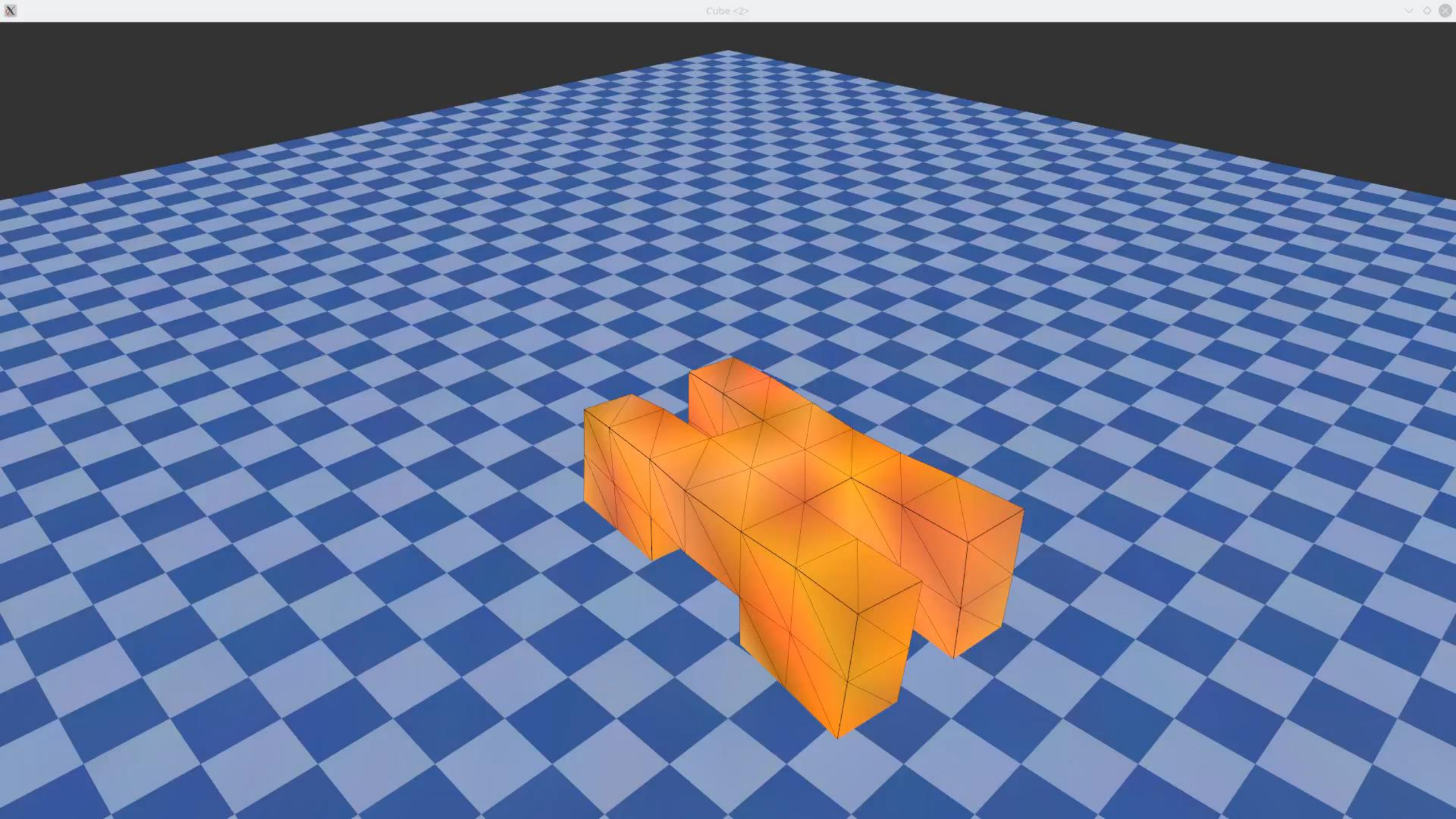


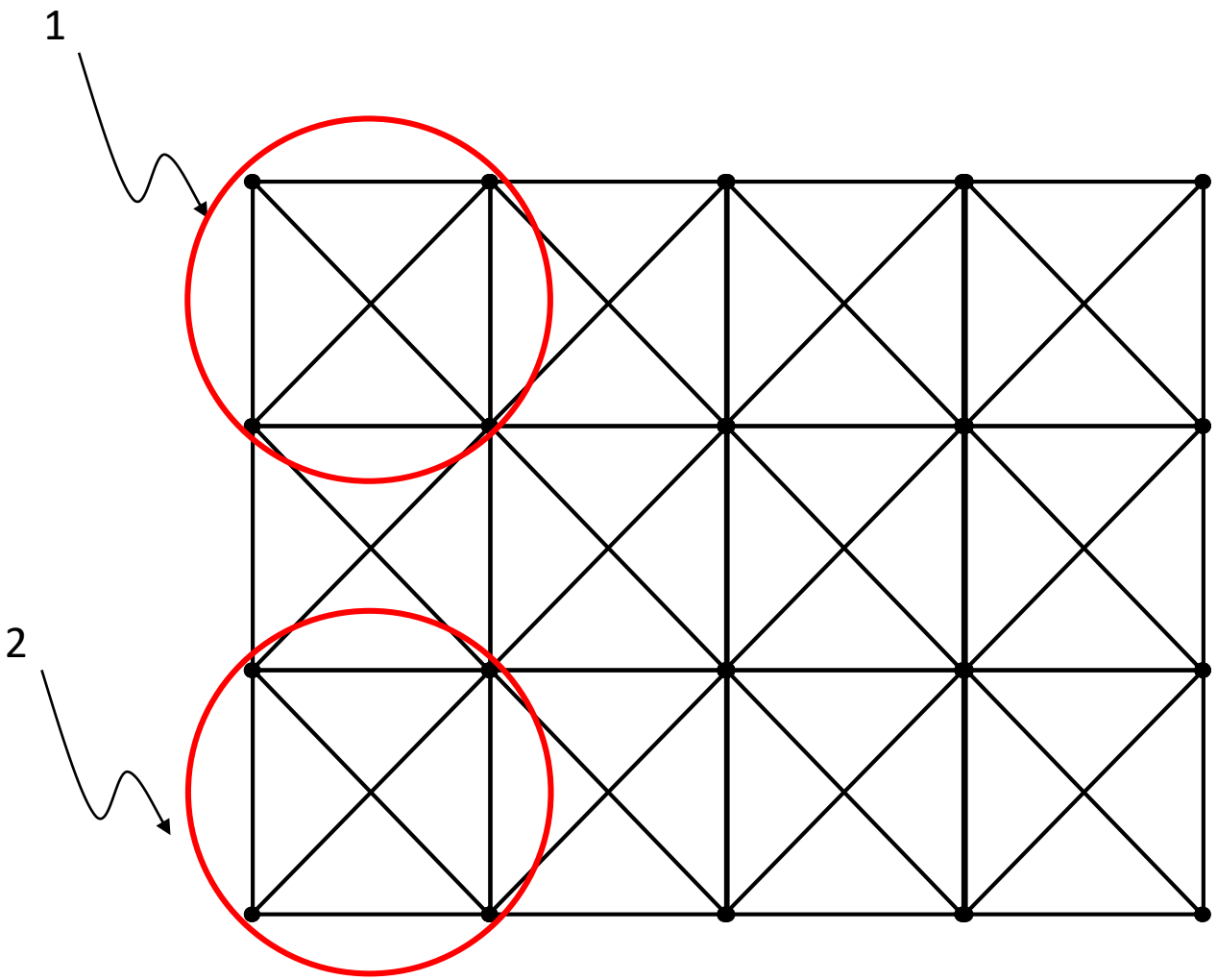
$L_0 = L_{0\_int}*[1+b*\sin(wt+c)]$

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=π

w=2π


Cube

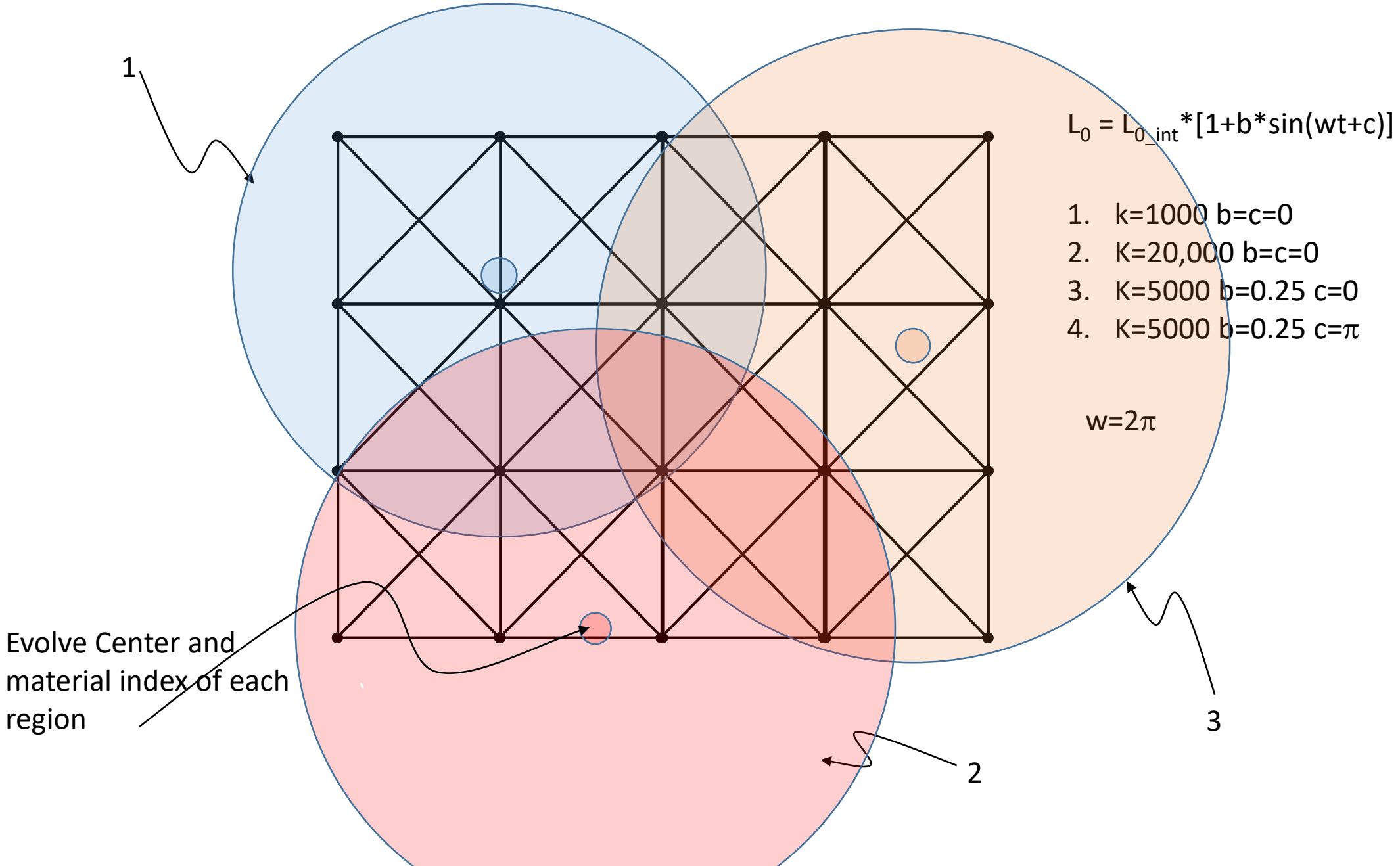

Cube <2>



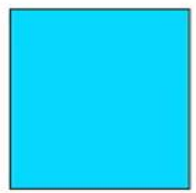
$L_0 = L_{0\_int}*[1+b*\sin(wt+c)]$

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=π

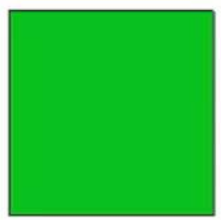
w=2π


1
$L_0 = L_{0\_int}*[1+b*\sin(wt+c)]$
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=π
w=2π
Evolve Center and material index of each region
2
3

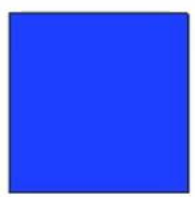
Muscle: contract then expand



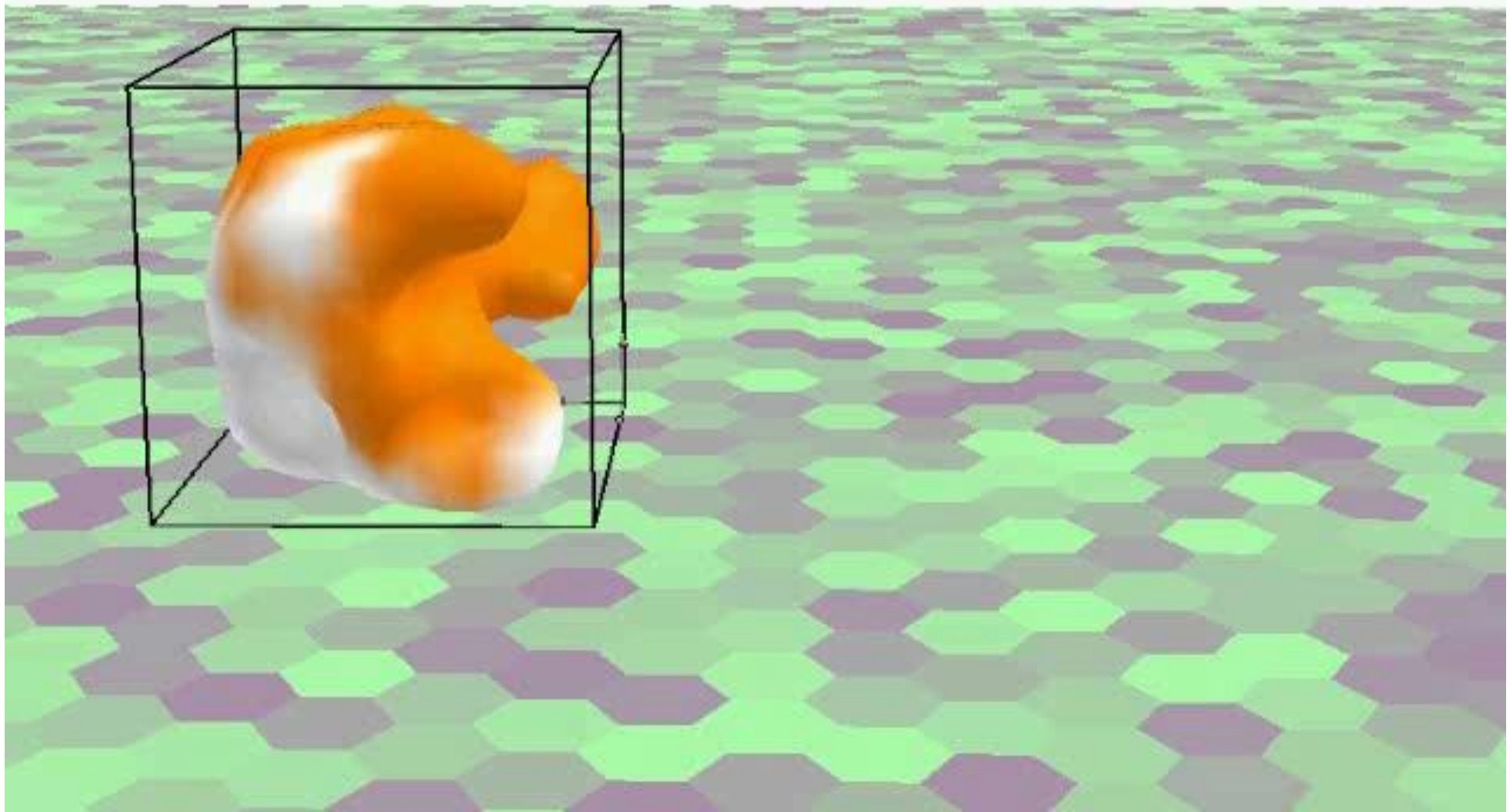
Tissue: soft support



Muscle2: expand then contract



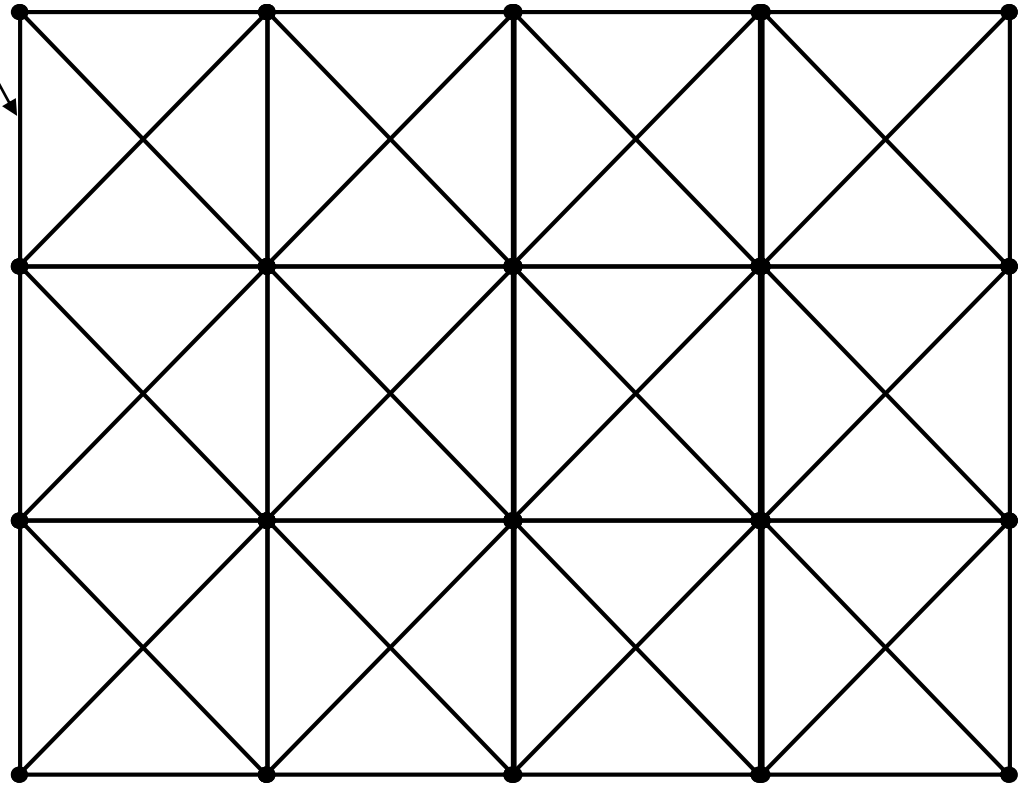
Bone: hard support

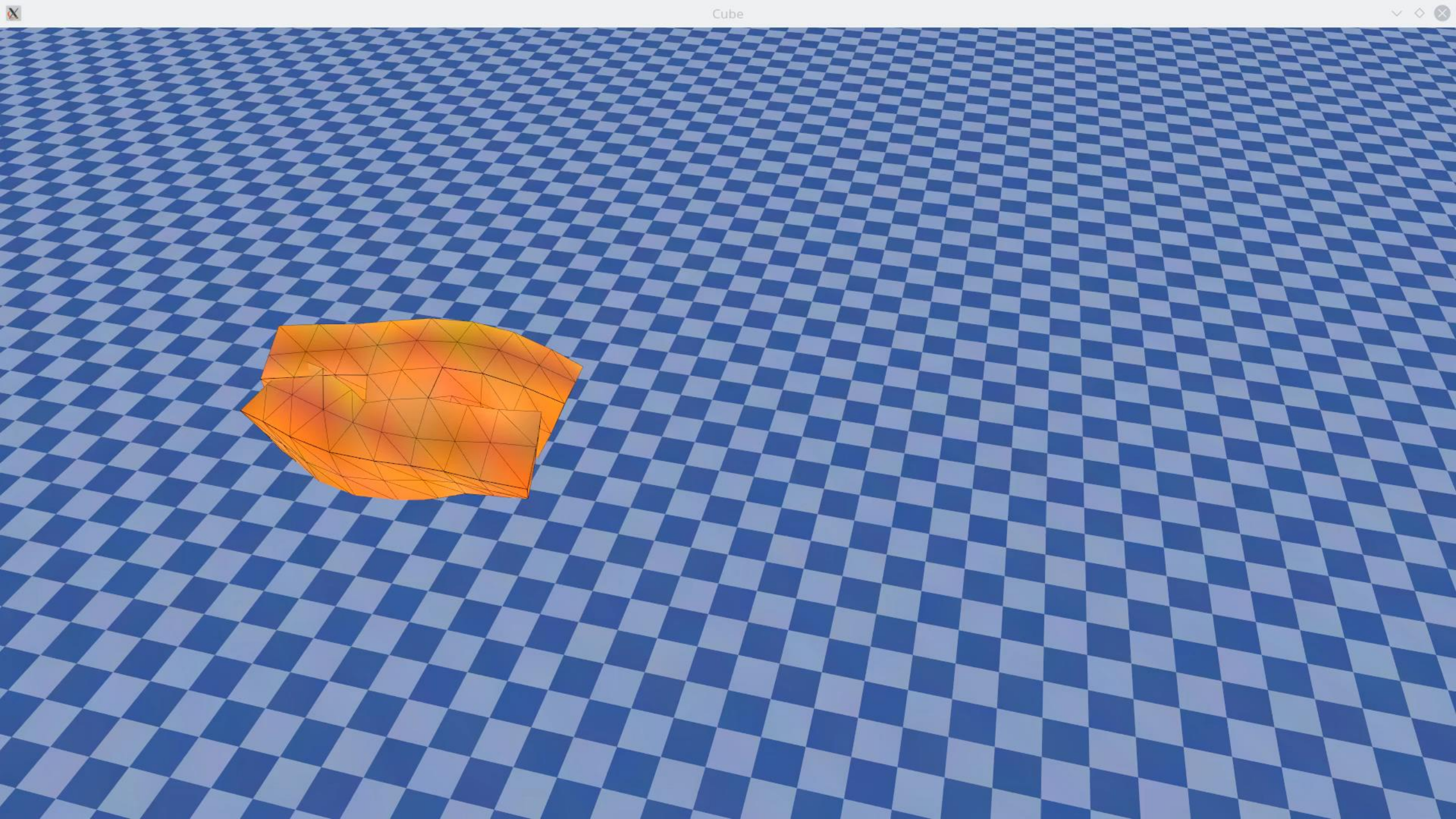


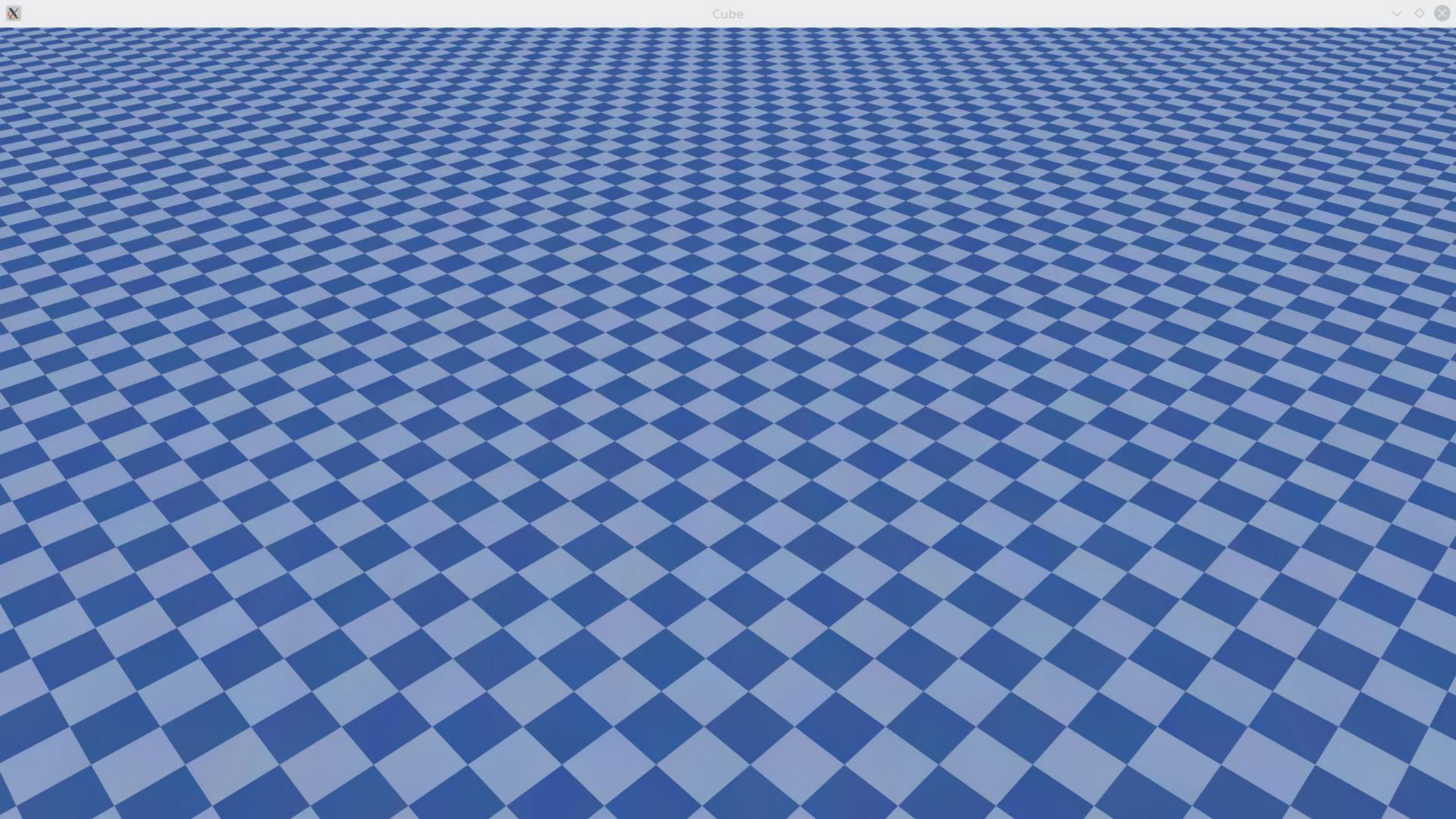
$L_0 = L_{0\_int}*[1+b*\sin(wt+c)]$

$k=f_1(x,y,z)$
$b=f_2(x,y,z)$
$c=f_3(x,y,z)$

Evolve f_1, f_2, f_3




Cube


Cube

Flexible Muscle-Based Locomotion for Bipedal Creatures

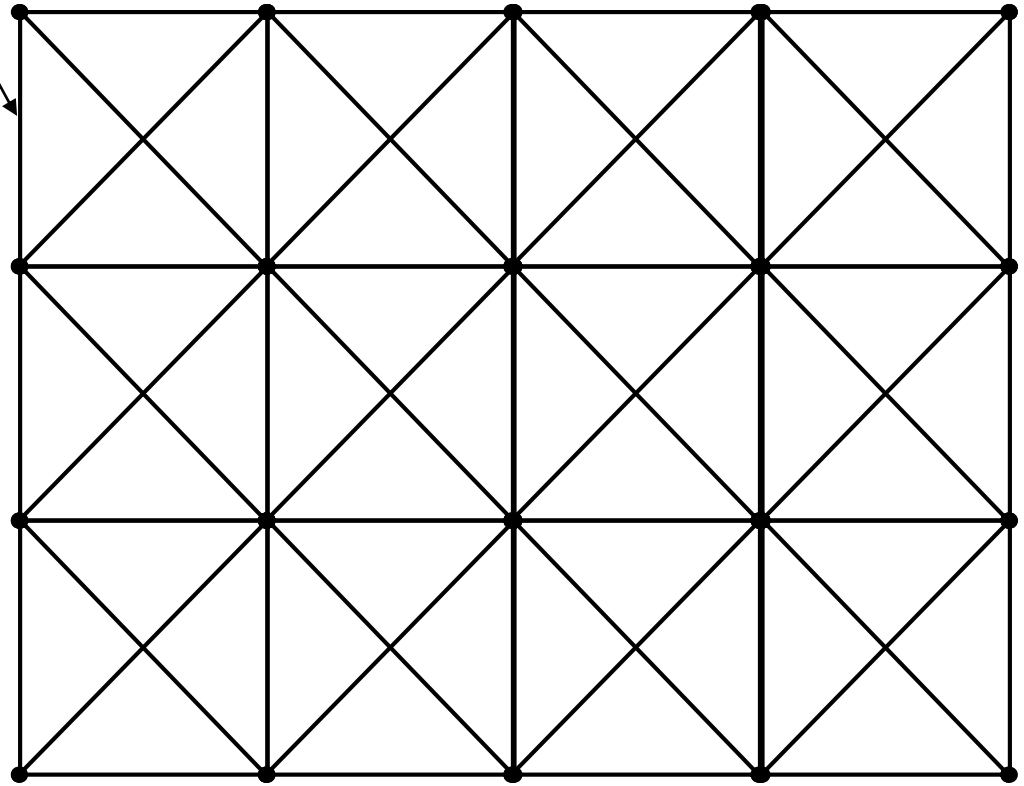
SIGGRAPH ASIA 2013

Thomas Geijtenbeek
Michiel van de Panne
Frank van der Stappen

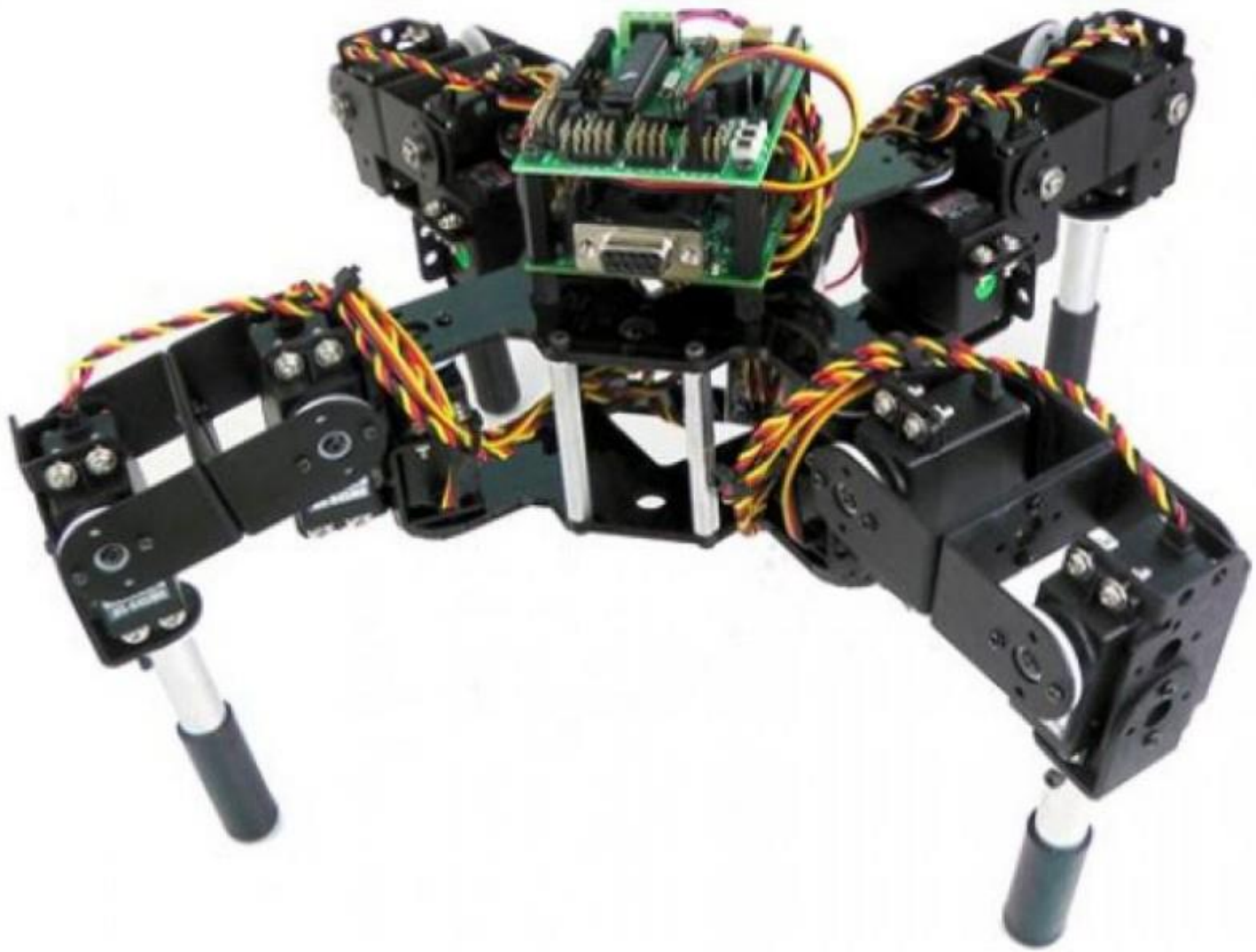
~~$L_0 = L_{0\_int} * [1 + b * \sin(wt + c)]$~~

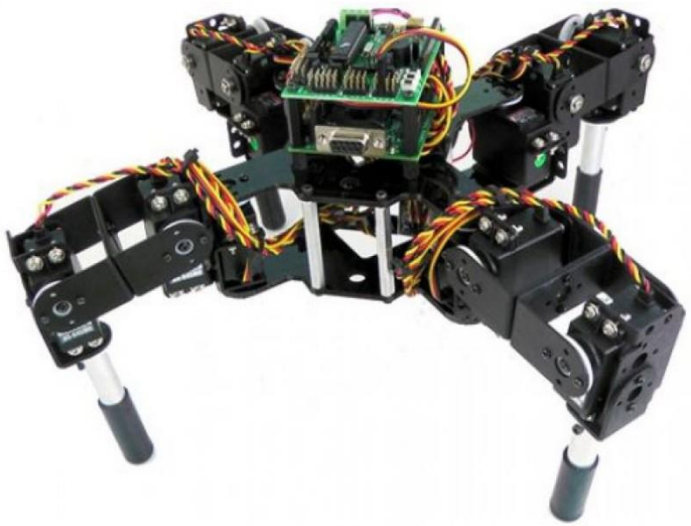
$L_0 = f(x,y,z,t)$

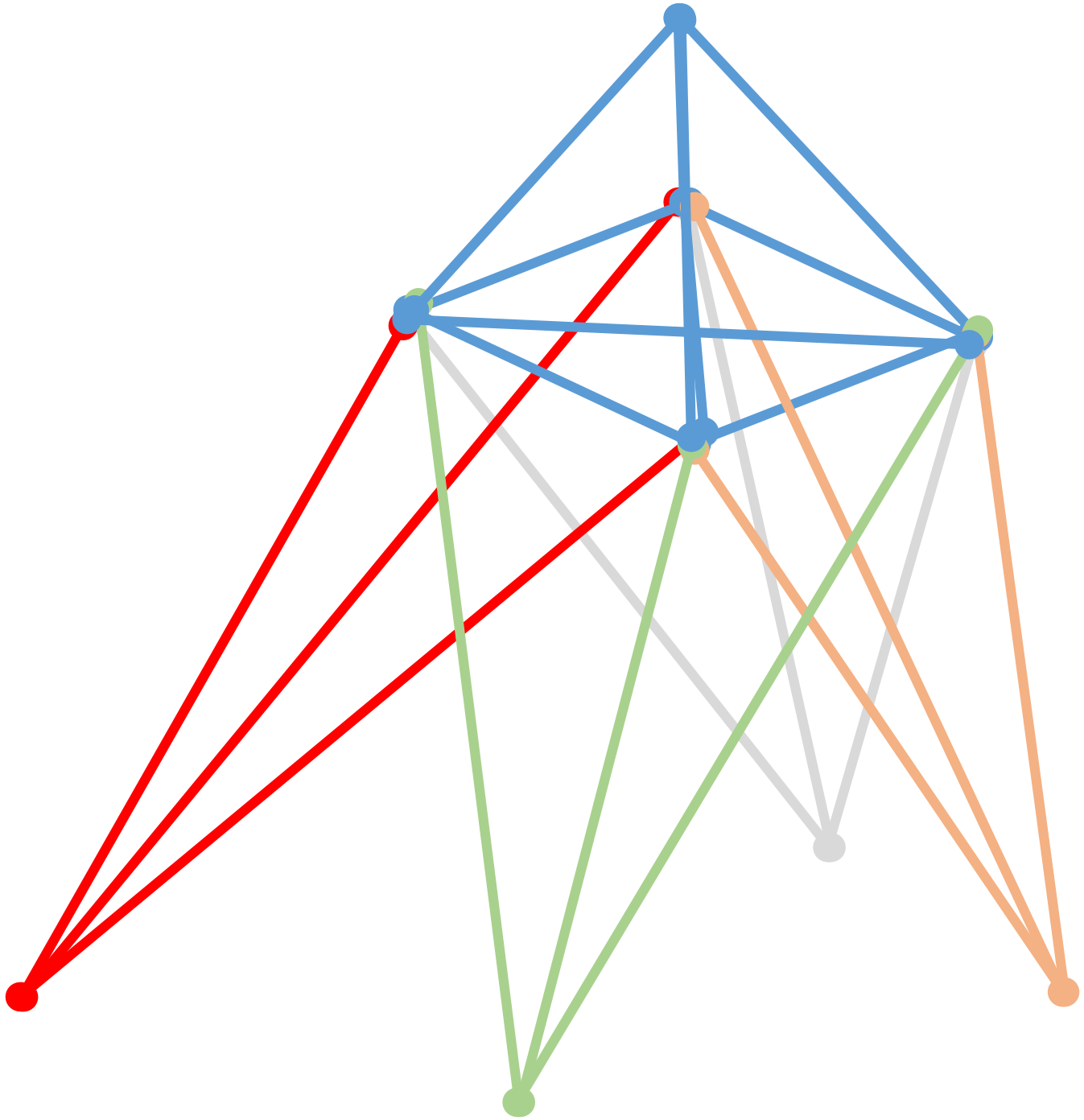
Evolve f

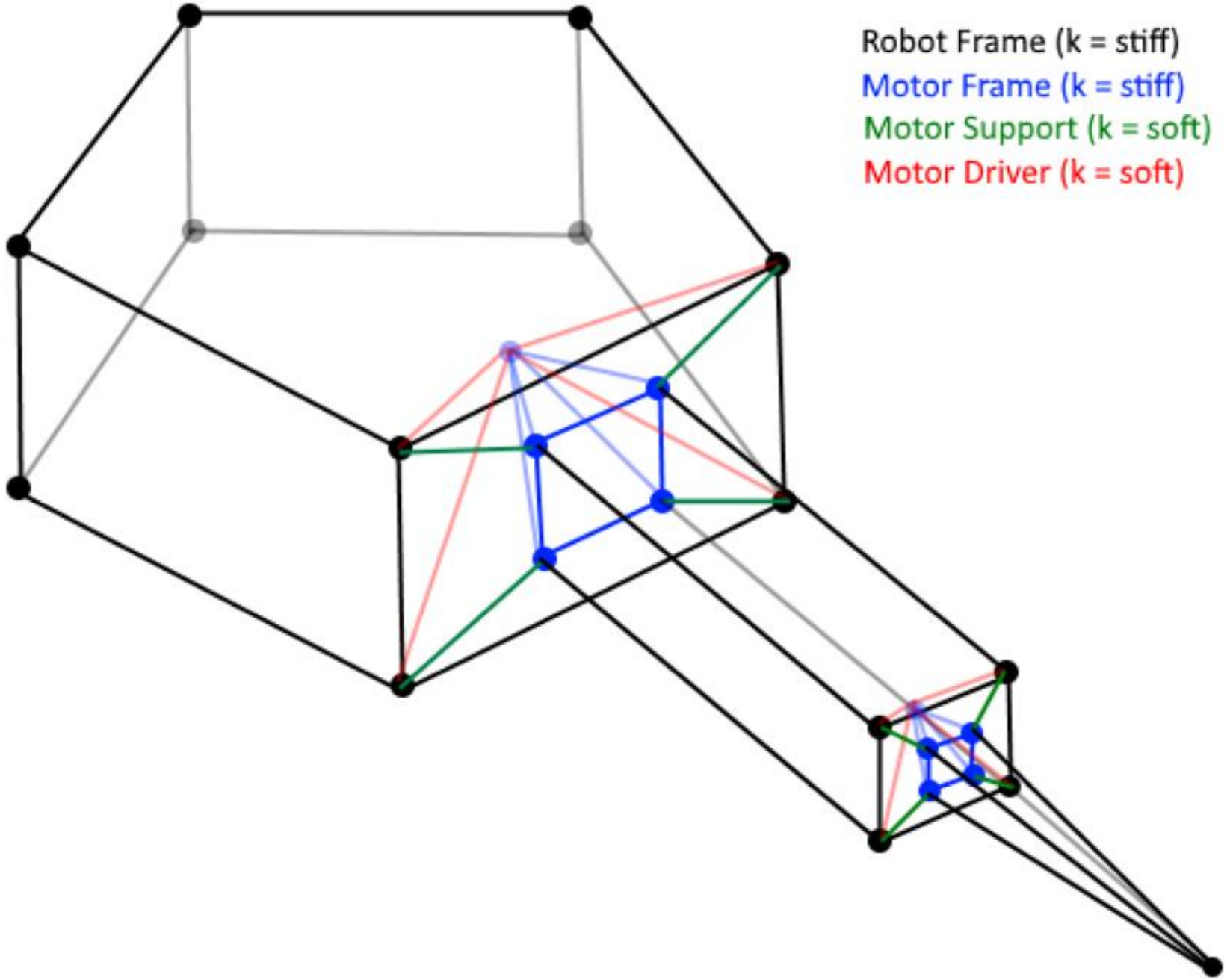


f(x,y,z,t) can be an analytical function or a neural network

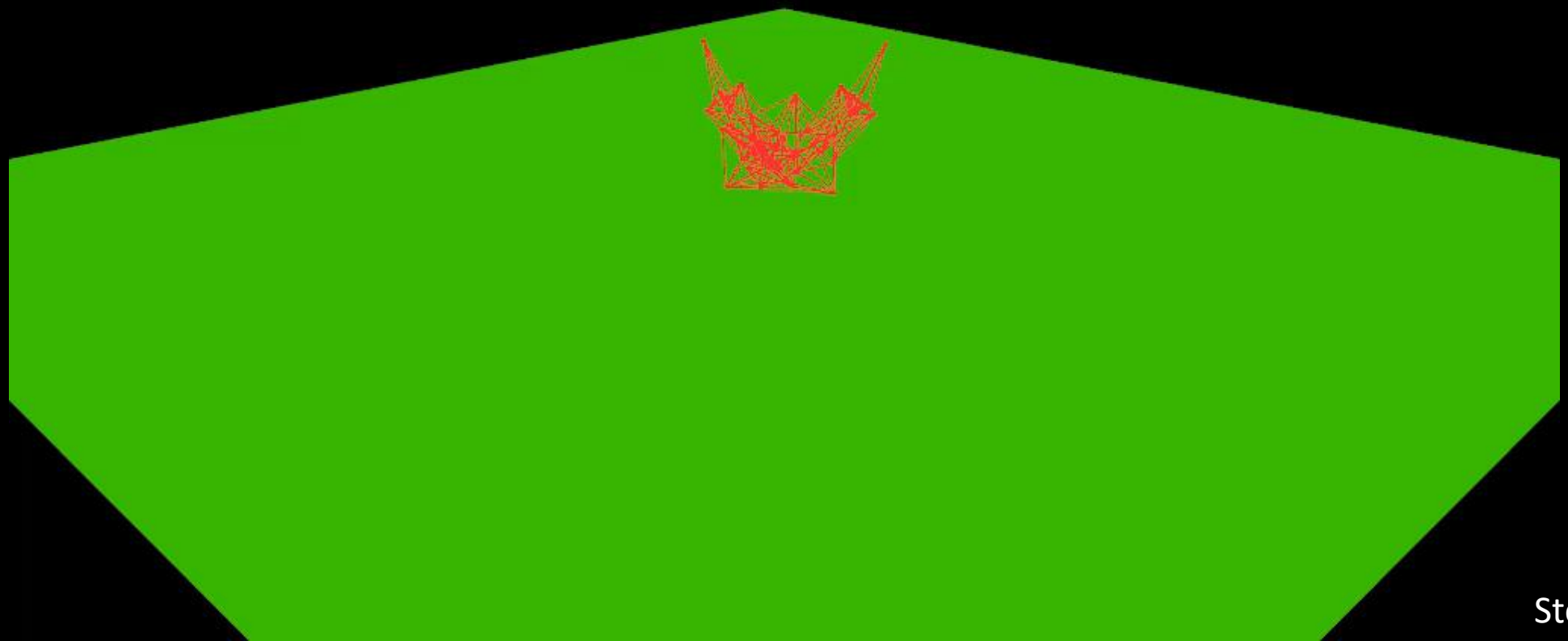




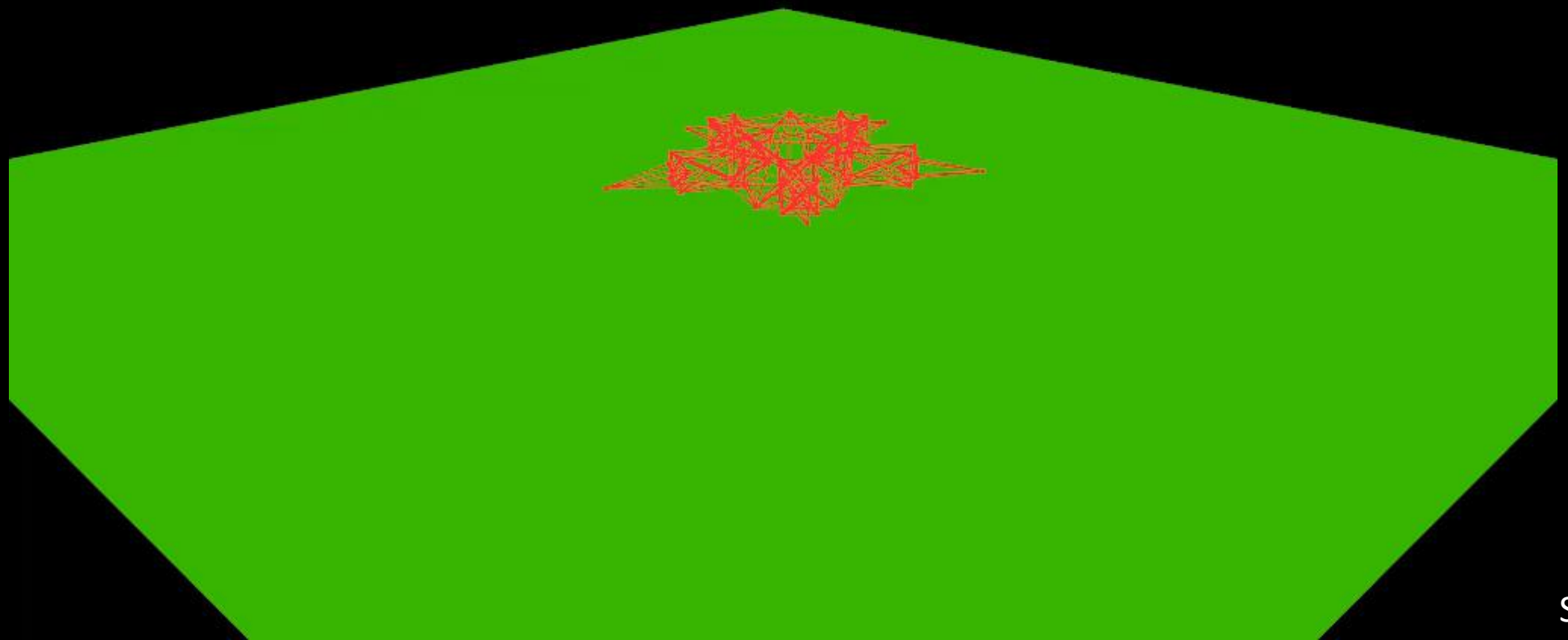




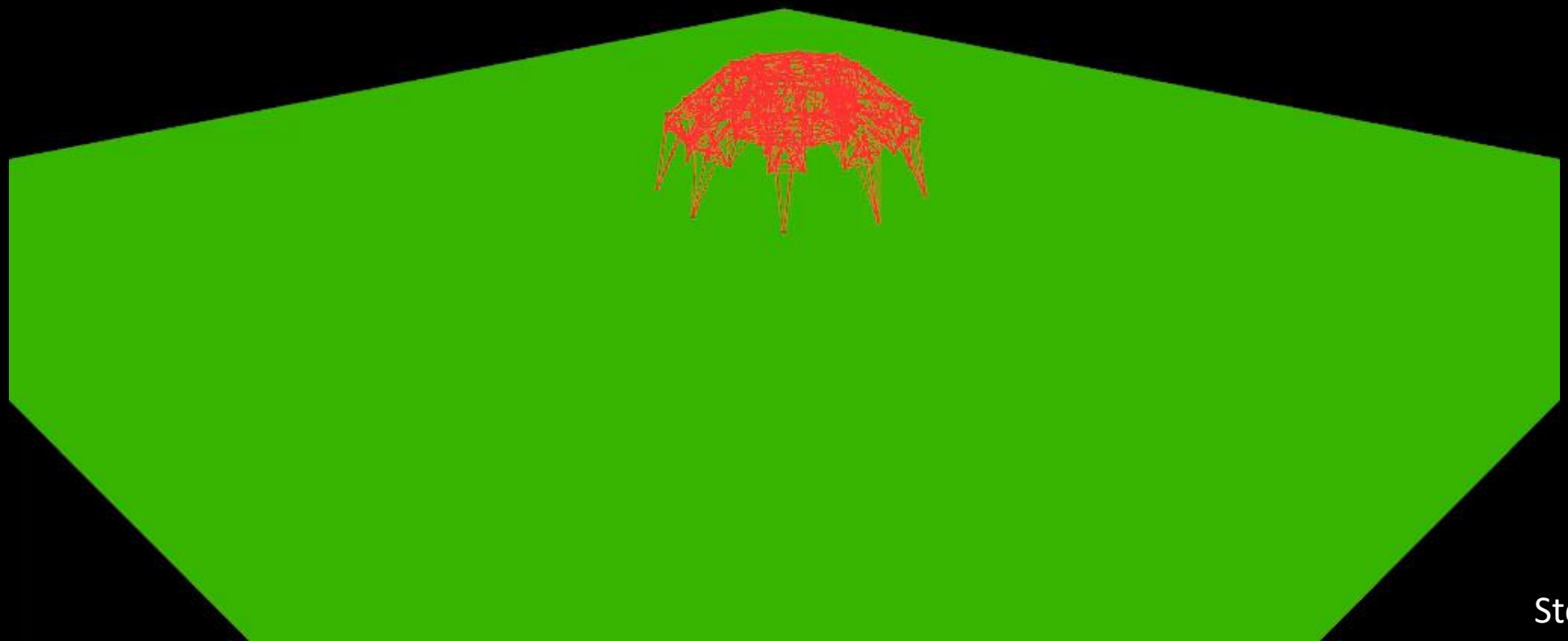
Steven Mazzola



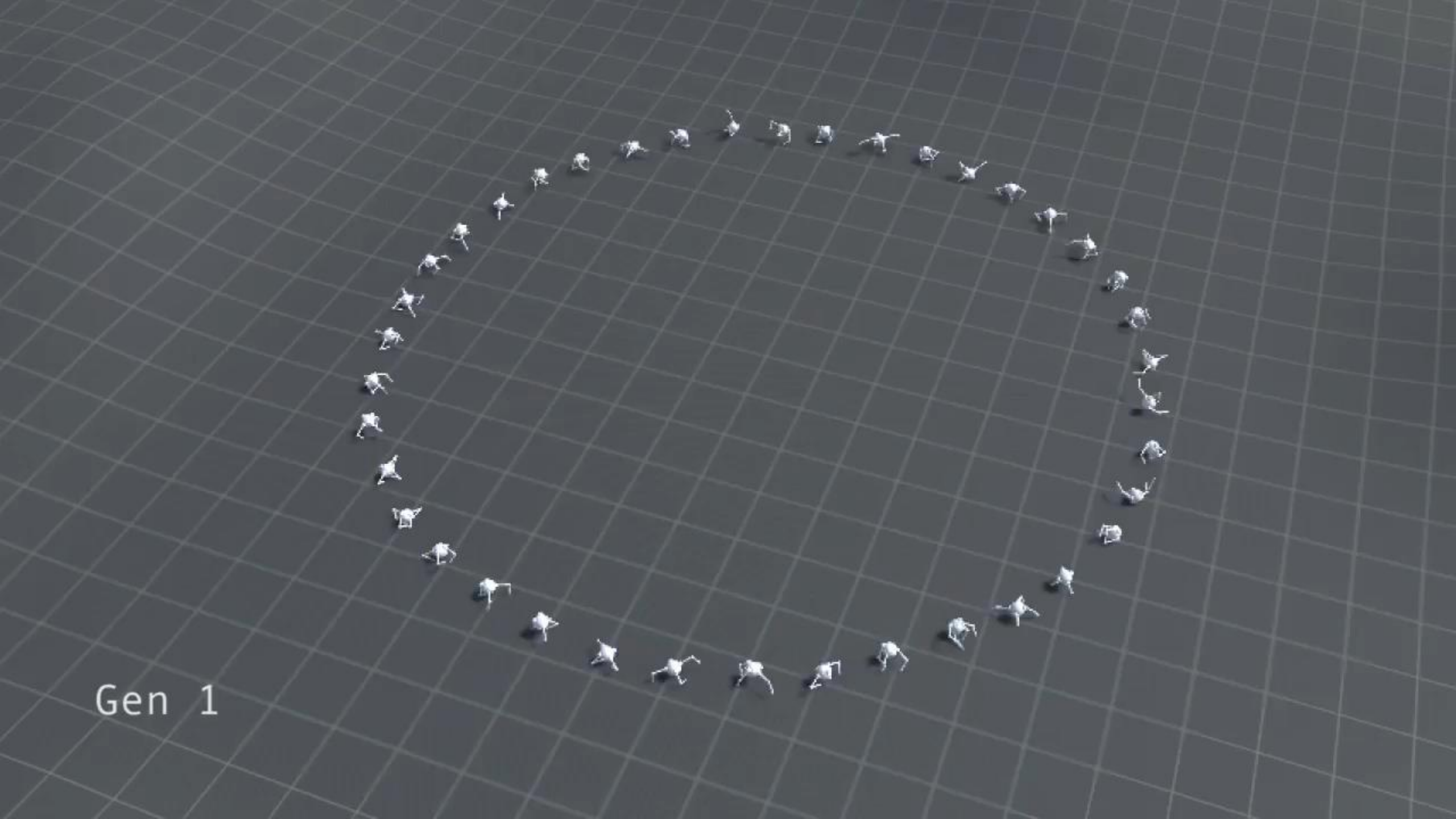
Steven Mazzola



Steven Mazzola

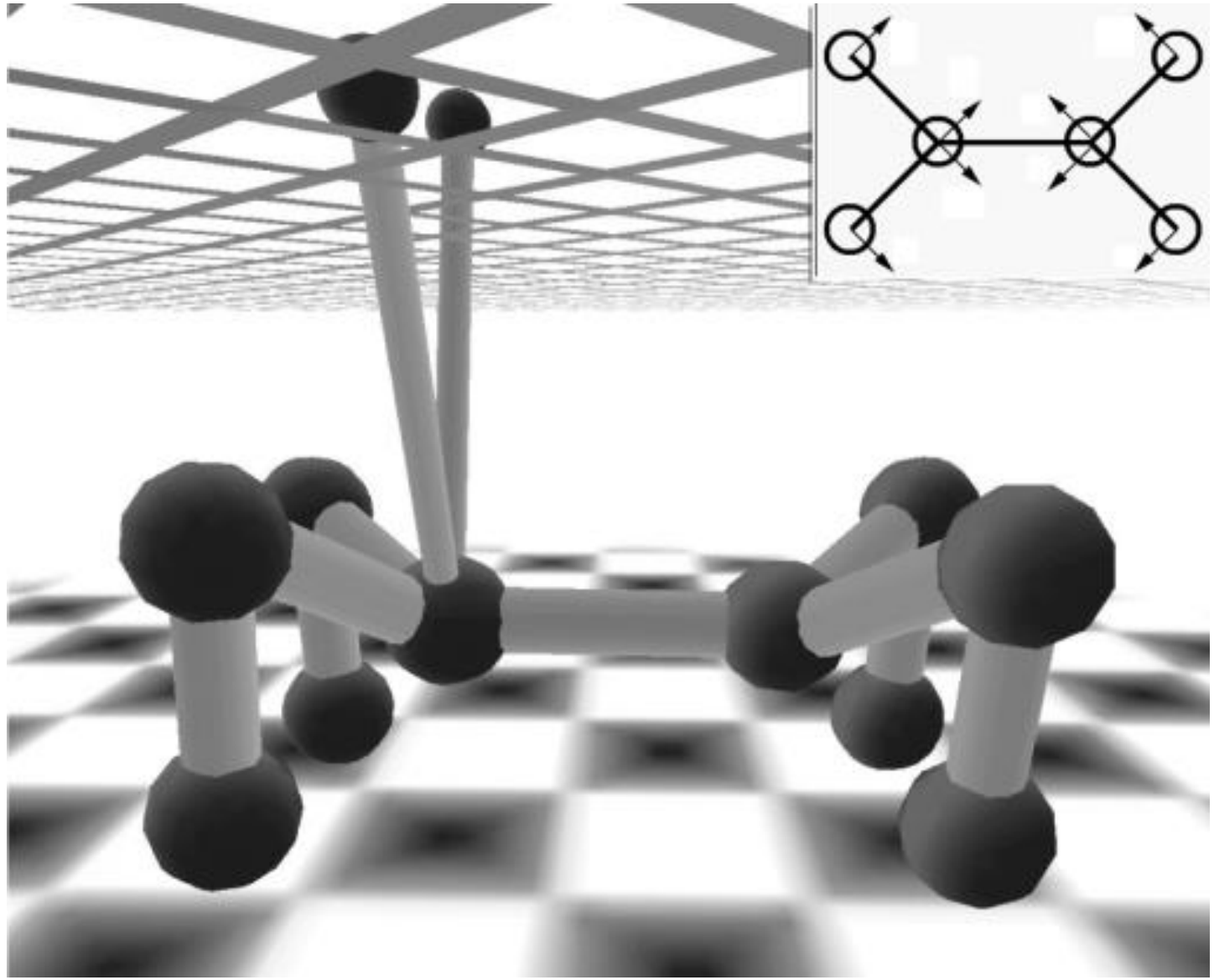


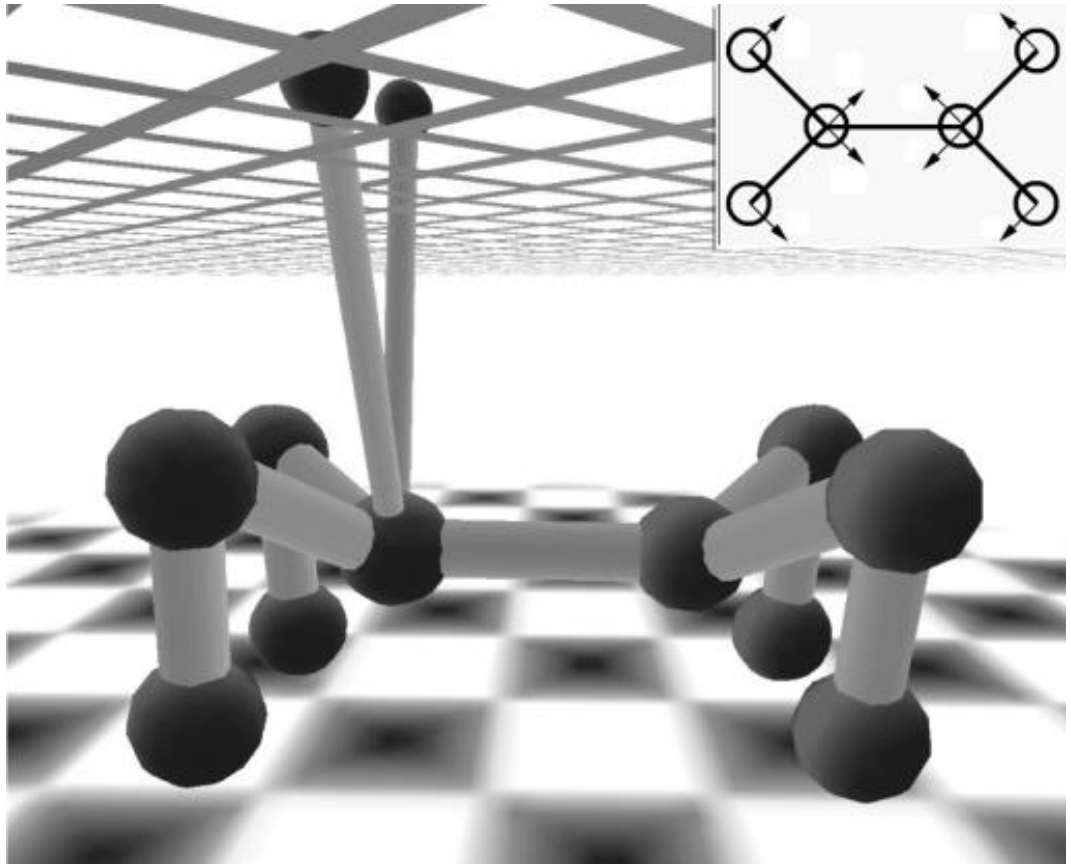
Steven Mazzola

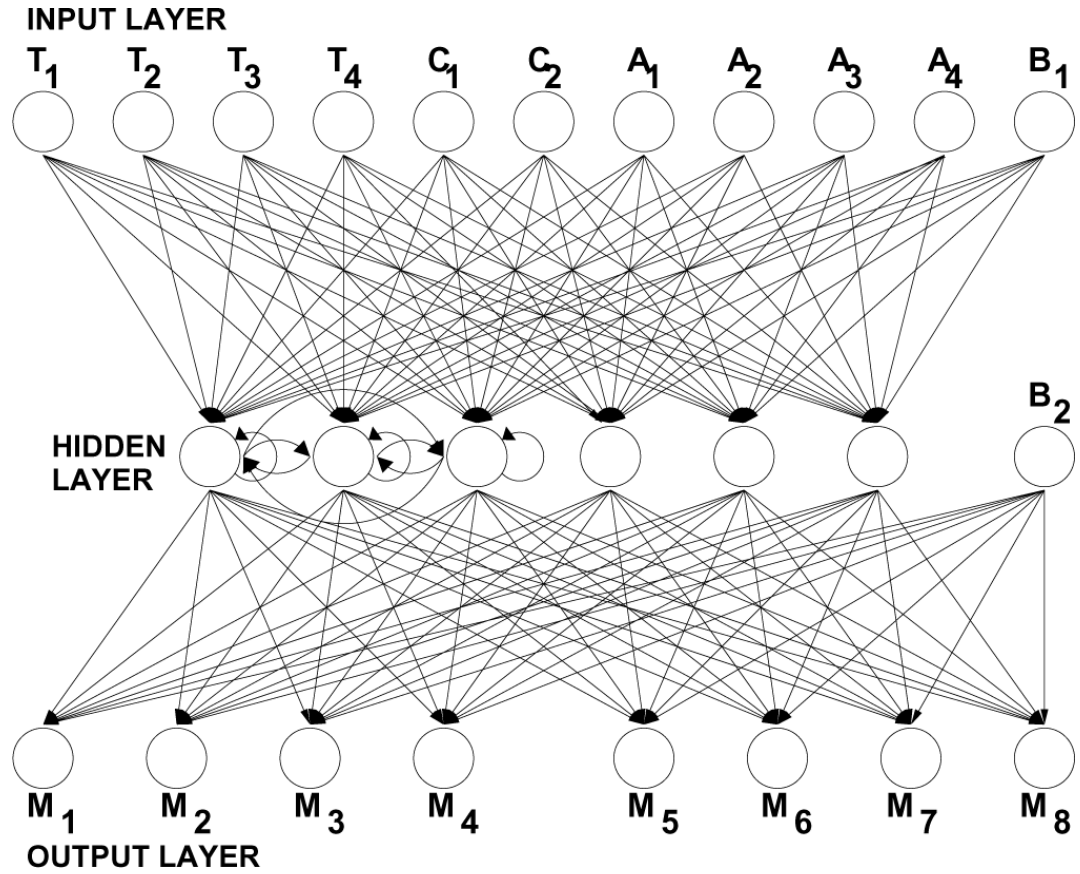

Gen 1

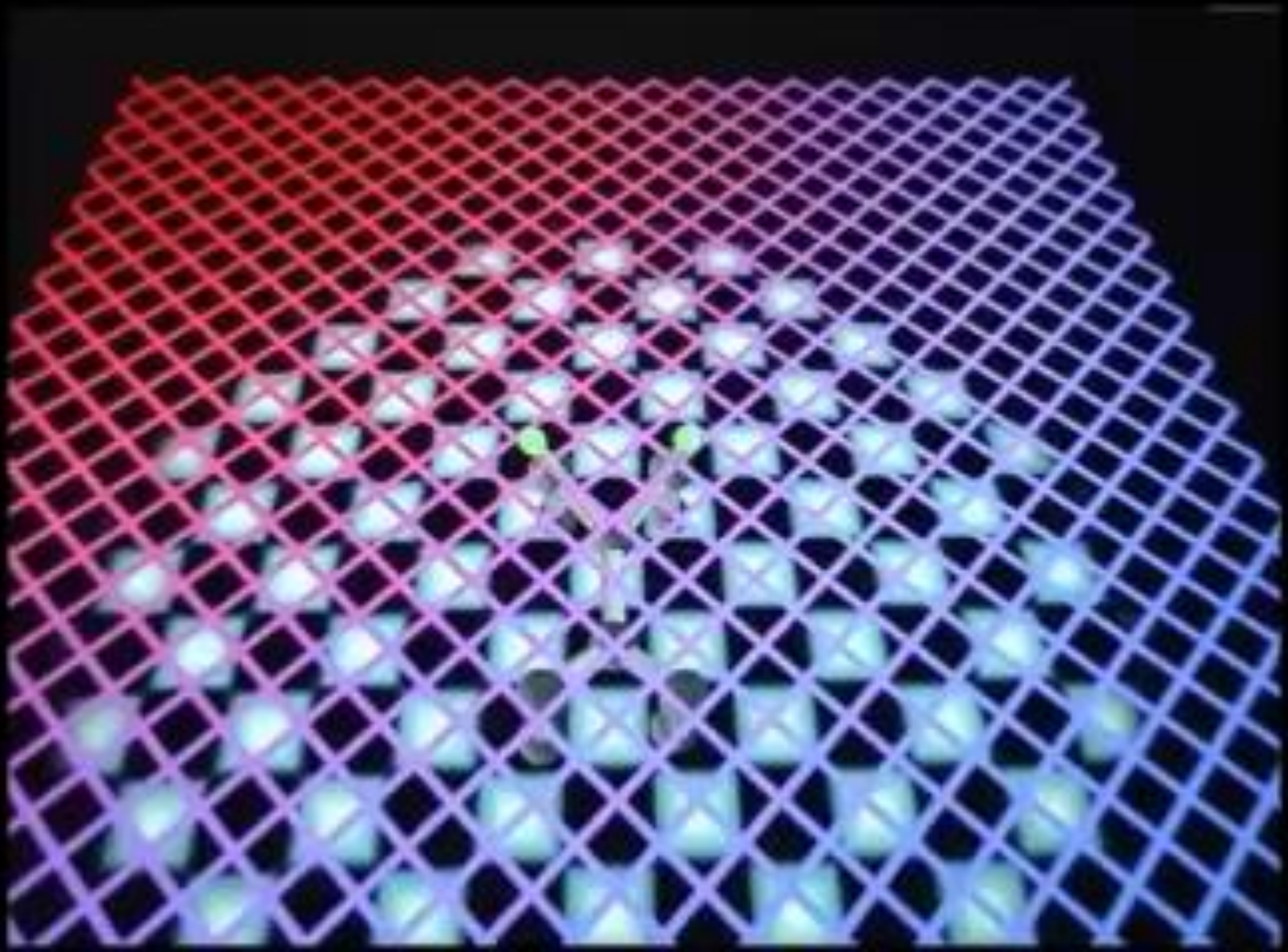
Incorporating Sensors

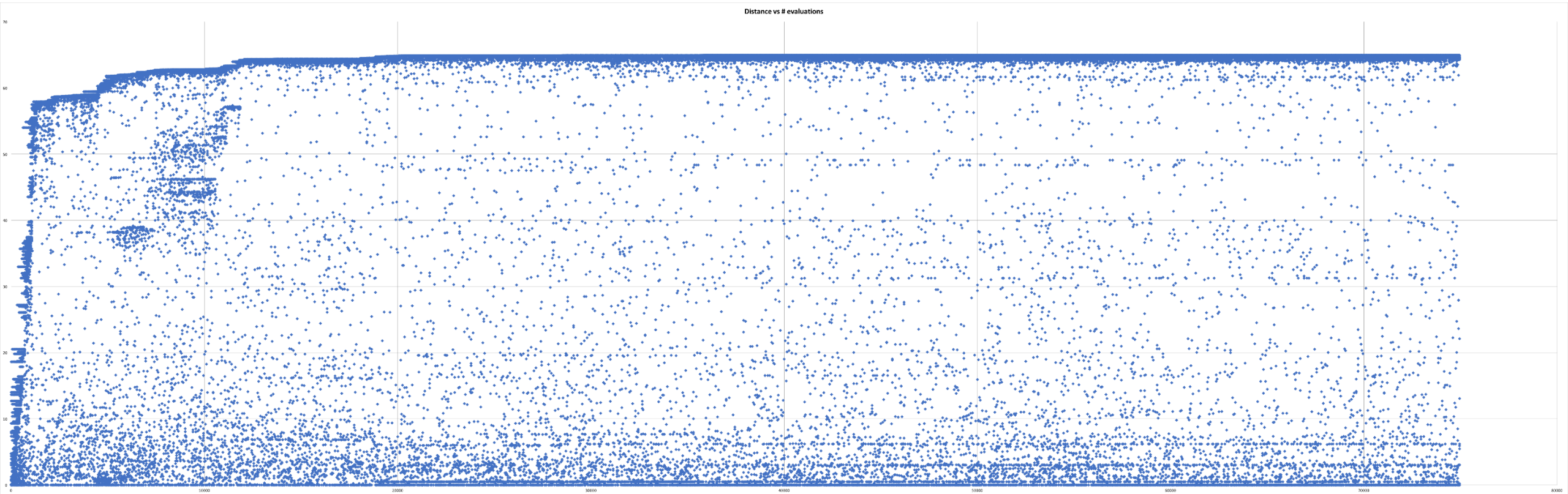
- You can change the length L0 of springs also as function of sensors
- Sensor types:
  - External, e.g.
    - light level
    - Distance to goal
  - Internal (proprioceptive)
    - Actual length of spring
    - Ground force (Whether mass is on or below ground)





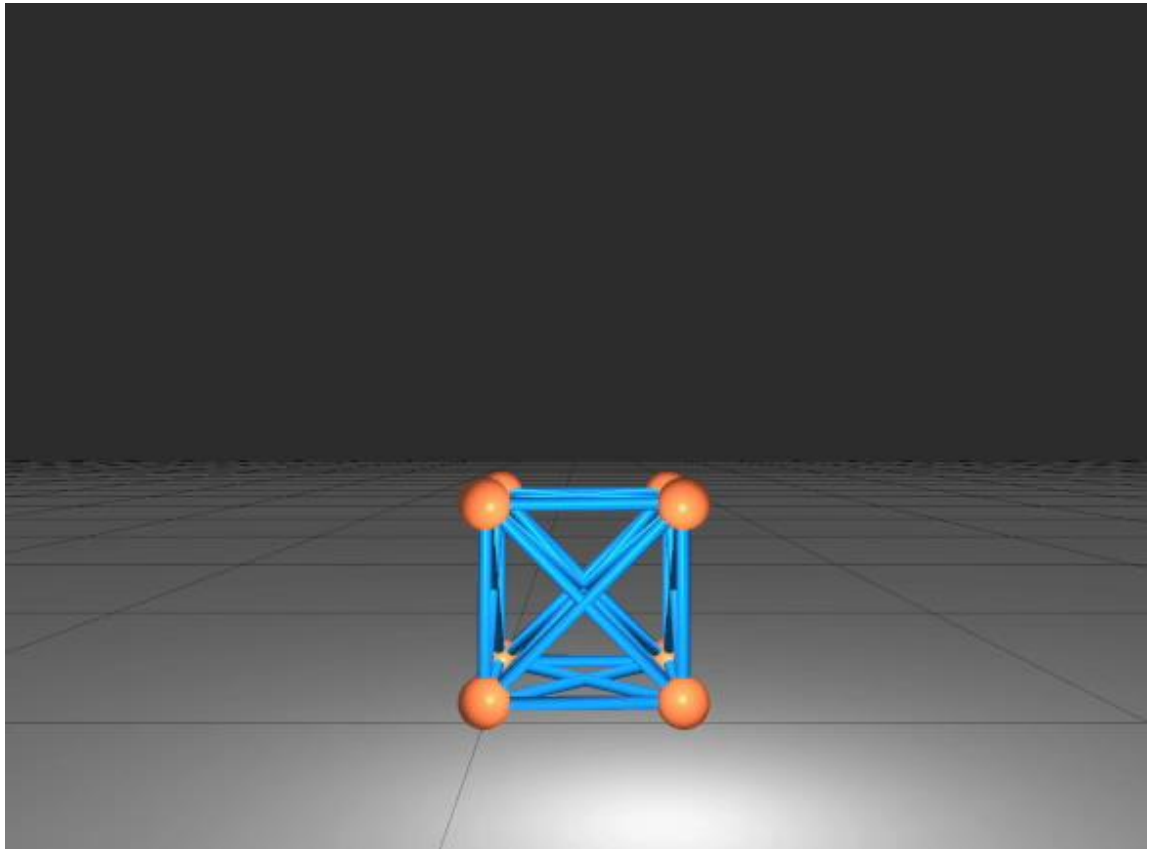

INPUT LAYER
T1 T2 T3 T4 C1 C2 A1 A2 A3 A4 B1
B2
HIDDEN LAYER
M1 M2 M3 M4 M5 M6 M7 M8
OUTPUT LAYER




Distance vs # evaluations
70
60
50
40
30
20
10
0
0
10000
20000
30000
40000
50000
60000
70000
80000

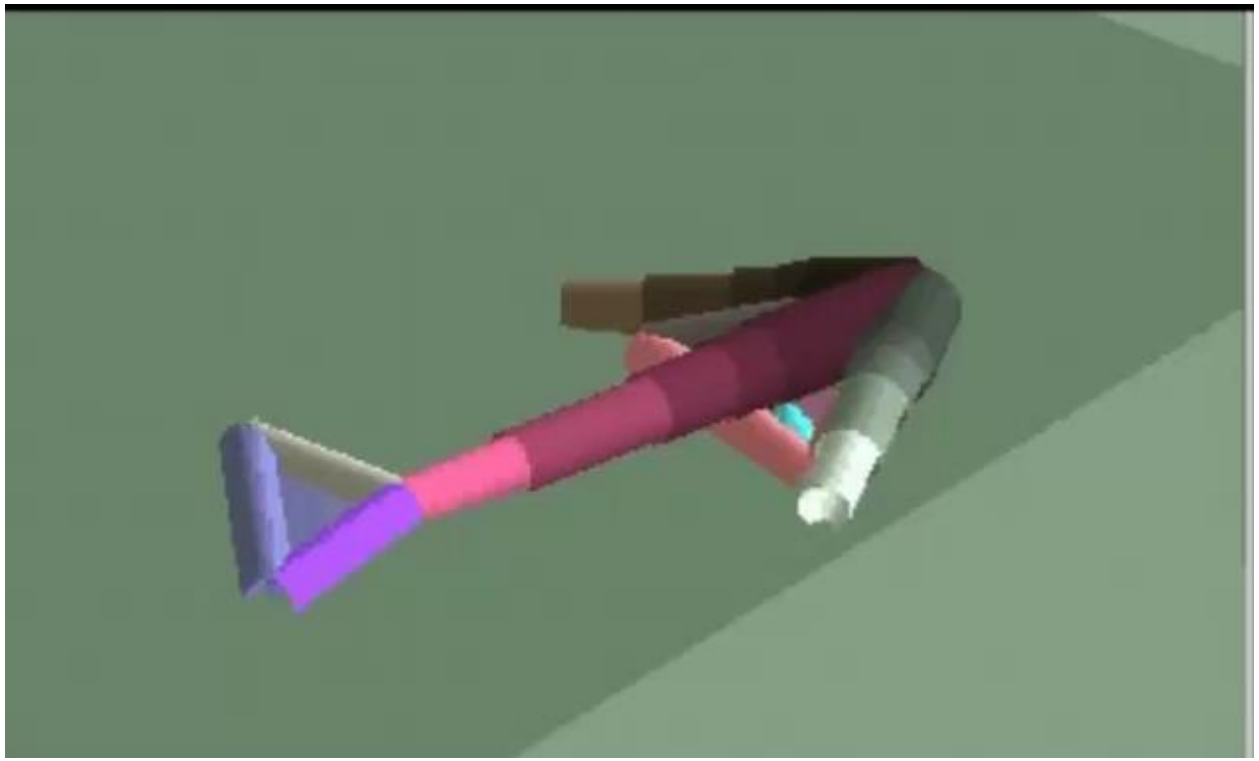
Assignment 3c

- Evolve morphology

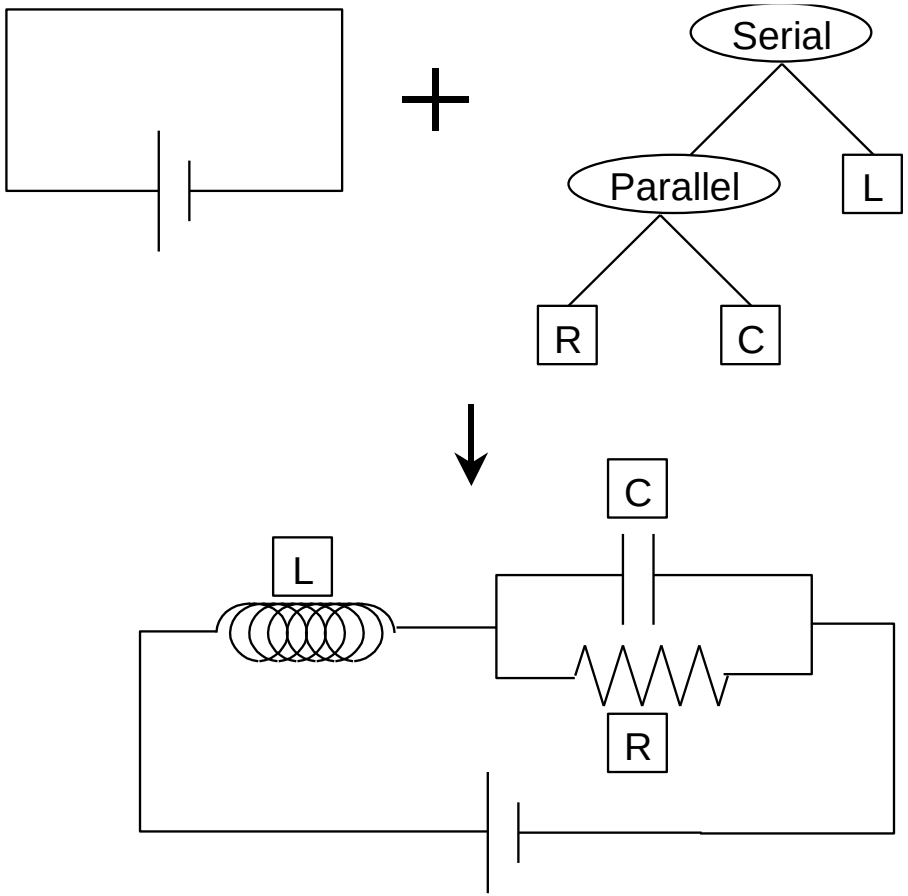


Direct morphological operators

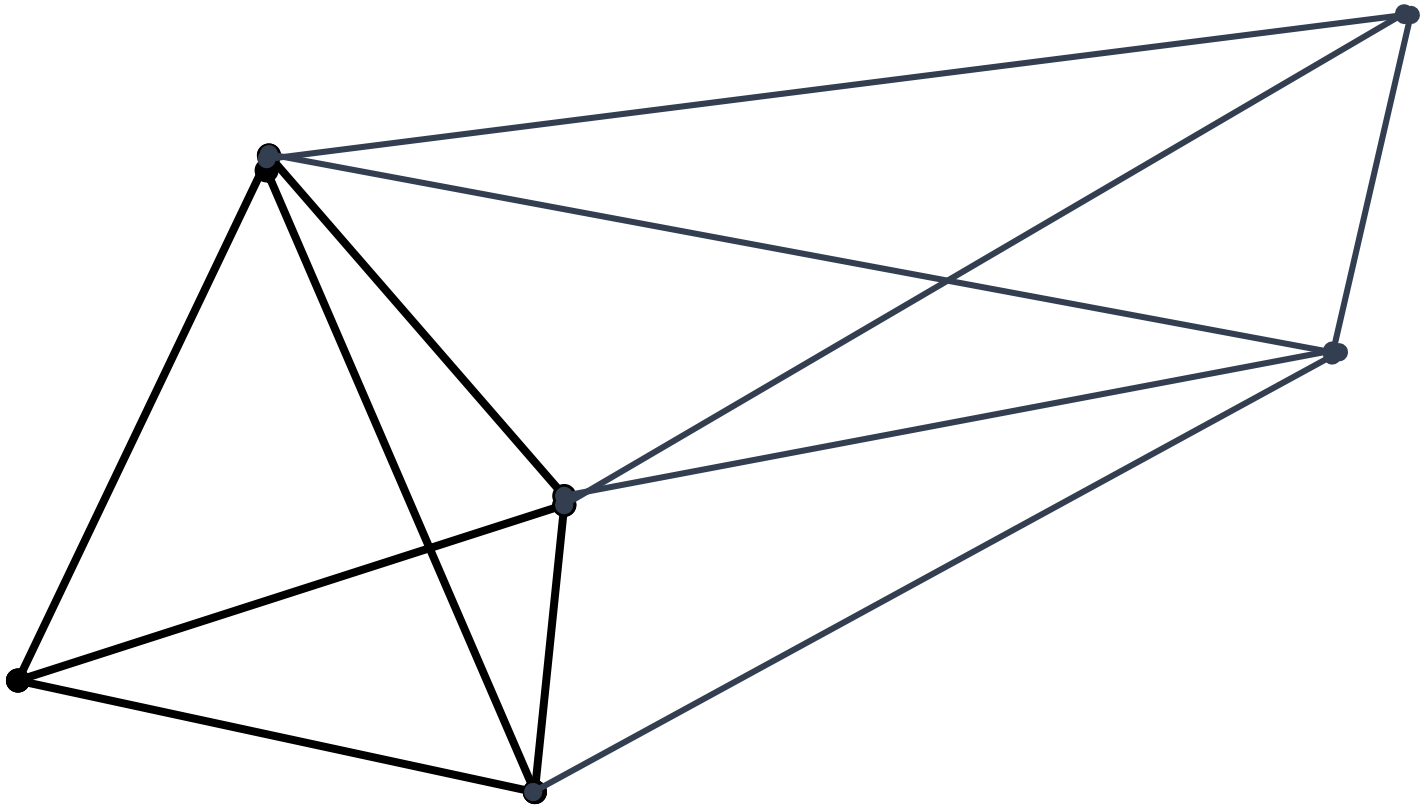
- Control changes
  - Change actuation parameters
- Morphological changes
  - Add/remove spring between two exiting masses
  - Add/remove mass
  - Change masss
  - Change rest length of existing spring
- Some operators require cleanup

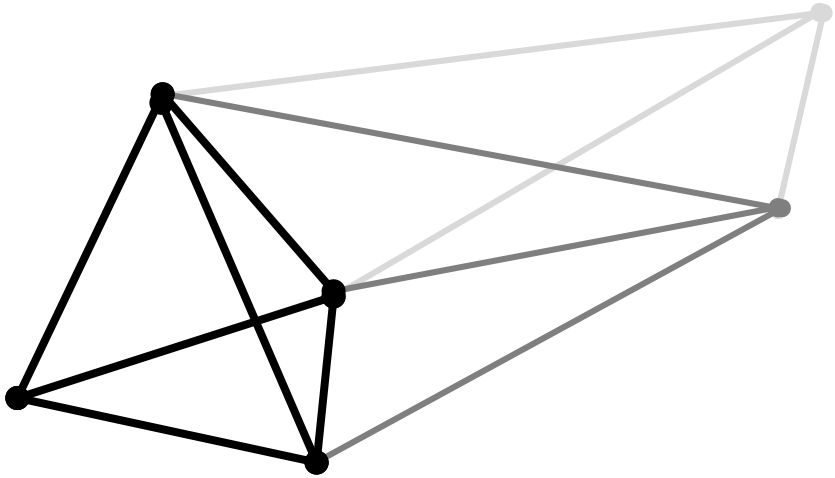


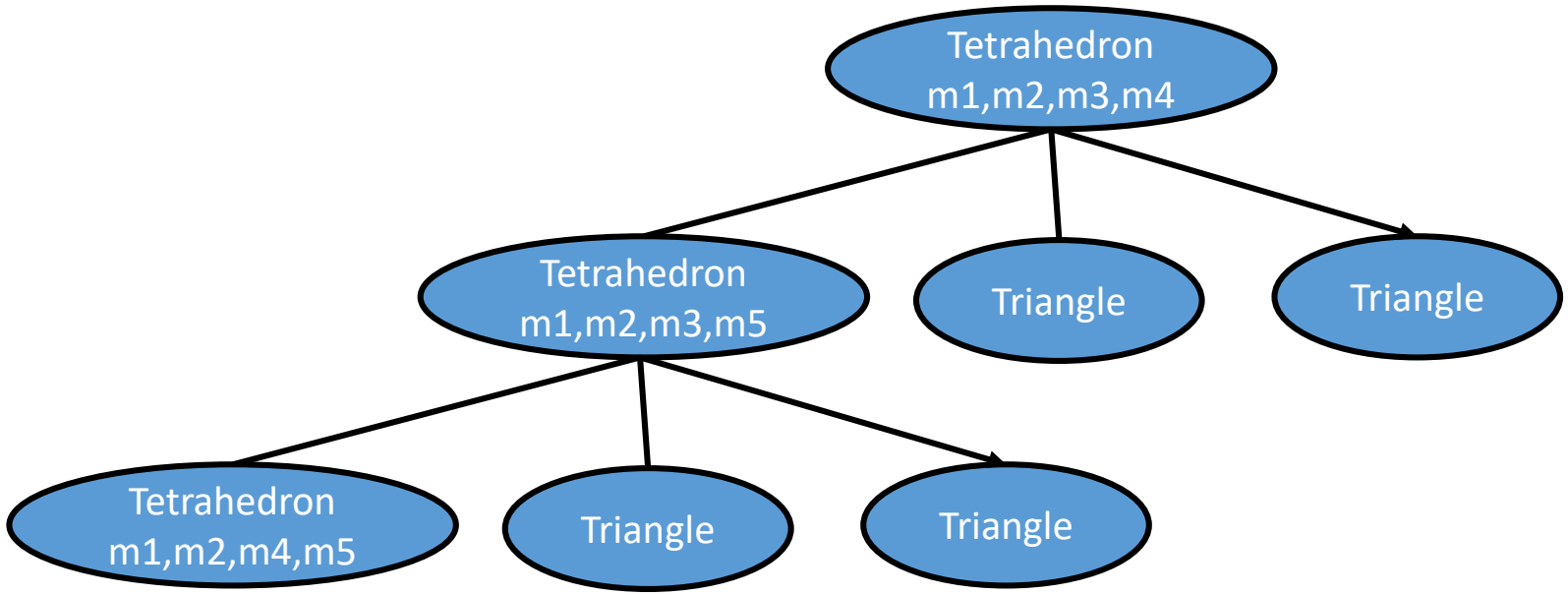
Developmental encoding



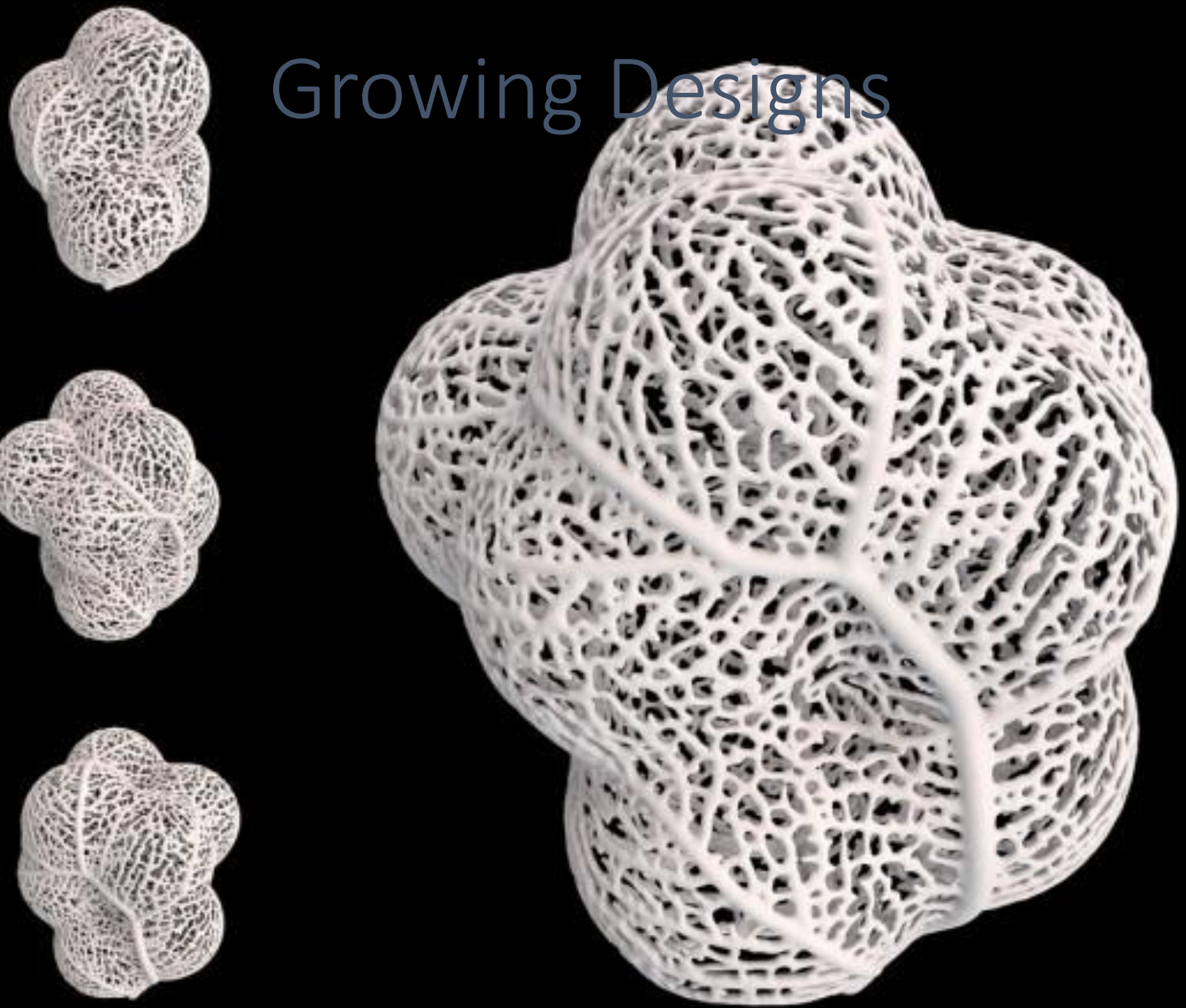
Developmental encodings






Tetrahedron
m1,m2,m3,m4
Tetrahedron
m1,m2,m3,m5
Triangle
Triangle
Tetrahedron
m1,m2,m4,m5
Triangle
Triangle

Growing Designs



Generative Blueprints

- A→B B→AB
  - A
  - B
  - AB
  - BAB
  - ABBAB
  - BABABBAB
  - ABBABBABABBAB