# morpho-particles

## Installation

Place `morpho-particles.morpho`, `DictlikeSetOps.morpho`, `LinAlgTools.morpho`, and `RandomPoints.morpho` in your current working directory. Import `morpho-particles.morpho` and `RandomPoints.morpho` with the `import` command:

```
import "morpho-particles.morpho"
import "RandomPoints.morpho"
```

## Usage

### Basic Example

Set up a mesh. For example, a flat plane:

```
var verts = [
    [-1,1,0] ,[0,1,0] ,[1,1,0],
    [-1,0,0] ,[0,0,0] ,[1,0,0],
    [-1,-1,0],[0,-1,0],[1,-1,0]
]
var edges = [
    [0,1],
    [1,2],
    [0,3],
    [0,4],
    [1,4],
    [1,5],
    [2,5],
    [3,4],
    [4,5],
    [3,6],
    [3,7],
    [4,7],
    [4,8],
    [5,8],
    [6,7],
    [7,8]
]
var faces = [
    [0,1,4],
    [0,3,4],
    [1,2,5],
    [1,4,5],
    [3,4,7],
    [3,6,7],
    [4,5,8],
    [4,7,8]
]
var mb = MeshBuilder()
for (v in verts) mb.addvertex(v)
for (e in edges) mb.addedge(e)
for (f in faces) mb.addface(f)
var plane = mb.build()
var mesh = plane
```

Initialize a `Substrate` object using the surface mesh.

```
var s = Substrate(mesh)
```

Initialize a cloud of projectable points. You might use a method from `RandomPoints.morpho`, such as `GenParametrizedLine` to generate points along a parametrized vector function.

```
var func = fn (t) (Matrix([-0.5,t,1]))
var ptsArr = GenParametrizedLine(
    func,
    tStart = -1,
    tEnd = 1,
    numPts = 20
    )
```

From the point cloud, build a disconnected mesh of vertices.

```
var mbb = MeshBuilder()
for (x in ptsArr) mbb.addvertex(x)
var particleMesh = mbb.build()
```

Initialize a `Particles` object with a particle mesh argument:

```
var p = Particles(particleMesh)
```

One of the key functions of the `Particles` object is to project points onto a subtrate surface. This is accomplished by calling the `project` method of the `Particles` object with a `Substrate` object argument.

```
p.project(s)
```

The coordinates of these projected points can be accessed via the `projectedPointLocs` method.

To move the particles on the mesh surface, we need to define a force function. The `morpho-particles` module has some helpful methods to do this. The user can specify a constant force for which all particles experience the same force.

```
var e1 = Matrix([1,0,0])
var e2 = Matrix([0,1,0])
var e3 = Matrix([0,0,1])
var f=ConstantForce((1*e1))
```

`f` is defined as a force with magnitude 1 in the **x** direction The `SinglePositionForce`

allows the user to define a function of only each particle's position.

```
var f=SinglePositionForce(fn (x) Matrix([-x[1], x[0], 0])
```

The function above would generate an counterclockwise angular force on each particle.

Now, the user can call a movement method to apply the force function on a specified substrate. `moveAll` calculates all of the movement vectors at once then applies them sequentially one time.

```
p.moveAll(s, f)
```

It is also useful to employ a movement loop by which the force is calculated and applied in small steps. `moveAllLoop` accomplishes this for a specified number of steps and step size, which scales the calculated force vectors by that scalar value.

```
p.moveAllLoop(s, f, 1000, stepsize=0.00001)
```

A built-in plotting object, `ParticlePlotter` is also included to faciliate plotting the groups of projectable points, projected point, and moved points. Any of the preceding groups are omitted by default.

```
var pp = ParticlePlotter()
pp.plot(
    p,
    s,
    substrateGrade=[2],
    projectablePoints=true,
    projectedPoints=true,
    movedPoints=true
    )
```

## Torus Example

We can also explore particle movement on a more complex surface like a torus.

```
var r=1
var a=0.35
var impl = ImplicitMeshBuilder(
    fn (x,y,z) (x^2+y^2+z^2+r^2-a^2)^2 - 4*r^2*(x^2+y^2)
    )
var torus = impl.build(
    start=Matrix([1,0,0.5]),
    stepsize=0.25,
    maxiterations=400
    )
torus.addgrade(1)
var mesh = torus
```

Instead of a line, we can also generate a random cloud of projectable points.

```
var ptsArr = GenRandomRectangle(
    1000,
    xBounds=[-2,0],
    yBounds=[-2,0],
    zBounds=[-2,2]
    )
var mbb = MeshBuilder()
for (x in ptsArr) mbb.addvertex(x)
var particleMesh = mbb.build()
```

This function generates a randomly distributed cloud of points within a rectangular prism of the specified bounds.

As before, we can apply a force.

```
var p = Particles(particleMesh)
p.project(s)

var e1 = Matrix([1,0,0])
var e2 = Matrix([0,1,0])
var e3 = Matrix([0,0,1])

var f = ConstantForce(1*e1+1*e2)
p.moveAll(s, f)
```

Notice that the particles remain constrained to the mesh.