Nature Inspired Computing

K.Dimopoulos

Forces and Newton

What is a force

So far we have seen that objects in our simulations have a **location** in the virtual world they exist. A change in location occurs if the objects has **velocity**. A change in velocity occurs if the object has **acceleration**. But what causes an object to have acceleration? Newton gave an answer to this: A **force**!

A force is a vector that causes an object with mass to accelerate.

Force → acceleration → velocity → location

Newton's 3 laws translated in P5.JS

1. **An object at rest stays at rest and an object in motion stays in motion.**
    - An object's Vector velocity will remain constant if no forces are acted on the object
2. **The sum of all forces acted on an object equals its mass times its acceleration.**
    - The sum of all Vectors that are forces in an object divided by the object's mass determined the object's acceleration
3. **For every action there is an equal and opposite reaction.**
    - If we calculate a Vector f that is a force of object A on object B, we must also apply the force Vector.mult(f,-1); that B exerts on object A.

A word on units of measurement

So far we have avoided mentioning this, but overall we have been simulating physical quantities without defining (explicitly) the unit of measurement. However for all the physical quantities so far was a unit even if that was not physical:

- **Location** was measured in **pixels**
- **Time** was measured in **frames**
- **Velocity** in **pixels per frame**

It is easy to map pixel to distance (e.g. mms) and frames to time (e.g. secs)

What about mass?

Mass in our simulation seems arbitrary. We could tie mass to the size of the shape of the object. The bigger the size the more mass it has!

But a small iron ball does not have the same weight as a ball of styrofoam of the same dimensions! What is the difference? Both occupy the same volume but have different densities.

mass = ρ * volume / density

We can make our simulation as realistic as we want, but for now we will ignore density.

Types of forces

- **External forces**: Are acted on the objects from external factors.
  - Wind, gravity, friction, collisions with other objects, etc.
- **Internal forces**: Are acted on an object by the object itself.
  - E.g. A cars, gas pedal *forces* the car to accelerate.
  - Although these are also affect the environment we will ignore them. Example if we simulate a rowboat, we will simulate only the force that is acted on the boat because of the paddles pushing against the water, an not vise versa.
- Weight is an external force that is acted on an object because of the object's mass. Do not confuse weight with mass. (After all an objects has a different weight on the surface of the Earth that on the surface of the Moon, even though the mass of the object is the same)

Calculating the acceleration of an object

The sum of all Vectors that are forces in an object divided by the object's mass determined the object's acceleration

$$F1+F2+...+Fn = mass * acceleration \Leftrightarrow acceleration = (F1+F2+...+Fn) / mass \Leftrightarrow$$

$$acceleration = F1 / mass + F2/ mass +...+ Fn / mass$$

Everytime in a frame that a force is applied to an object, we need to divide the force by the object's mass and add the result to the object's acceleration.

It is important to **reset the acceleration to 0 at the start of each frame**!

Particle Life

Particle life

All life started as interactions of particles (atoms). In 2017 digital artist Jeffrey Ventrella created "Clusters", An asymmetrical particle system with ambiguous entities

A bit later Tom Mohr used the idea to come up with "Particle Life"

In this system particles have three main properties: color (type), position and velocity. Of course there is also acceleration involved.

There are many types of particle. However there are rules that define attraction and repulsion forces between different types of particles within a certain range (r_{max}).

Two rules are universal:

1. Particles of any type are always repulsed from each other if the distance between them is very small (so that they do not collapse on each other).
2. Attraction/repulsion force between two particles increases with distance up to a point, after which it decreases until it reaches the maximum range from which point on it is 0.

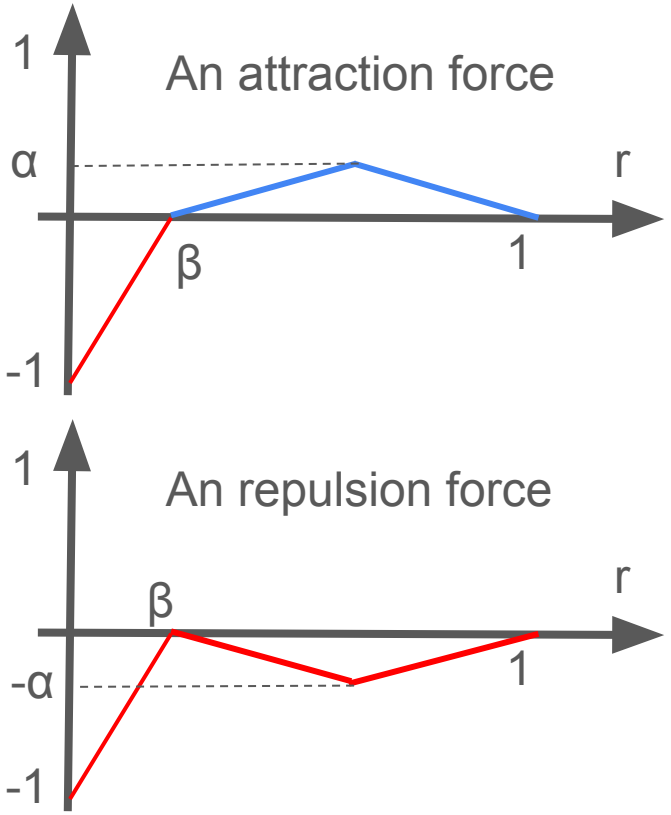
Force function

The force function starts at -1 and until distance β linearly decreases. From that point on it increases linearly until α and then from α until r decreases

This mean that to determine the force function we need to define two values α and β (assuming r is the same across all particles).

We need to define these values for every pair of particles.

Notice that we break Newton's 3rd law, by making the forces asymmetrical!



Accumulating forces

For every particle in our simulation calculate the distance to every other particle.

For every other particle that is in a distance less than a predefined radius, calculate the force and accumulate it to the particle, changing its acceleration.

We can define a force function that accepts the distance to the other particle, and the α and β values, and returns the force. However the distance needs to be scaled from 0 to 1.

Then using the calculated accelerations update velocities and then positions.

What happens if a particle exceeds the limits of the screen?

Other considerations

The force function needs to get the scaled distance to the particle, we can do that by dividing the distance with r_{max}. Then to counter the scaling, we need to multiply the accumulated force with r_{max}.

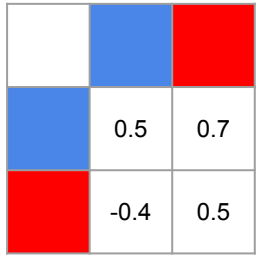
As we break Newton's third law we are constantly adding energy to the system. Eventually this will break our simulation. To mitigate this, we can add **friction**. That is a portion of the velocity is constantly lost.

We typically set β the same value for all forces (e.g. 0.3)

We keep the α values for each particle pair in a 2D matrix.

In the example next we have two particles blue and red

Blue is attracted to red (0.7), but red is repelled to blue (-0.4)



Extending the idea to multiple particles

Things become more complicated when we add more particle types (colors) and more interaction between them:

Blue is attracted to red, red is attracted to yellow, yellow is attracted to blue but repulsed by red, red is repulsed by blue, blue is repelled by yellow, etc.

From simple rules, complexity arises. This is a common theme in life and ALife

	1	0.8	0.3	-0.5
	-0.5	1	0.8	-0.8
	0.8	-0.5	1	0.8
	0.8	-0.4	-0.5	1

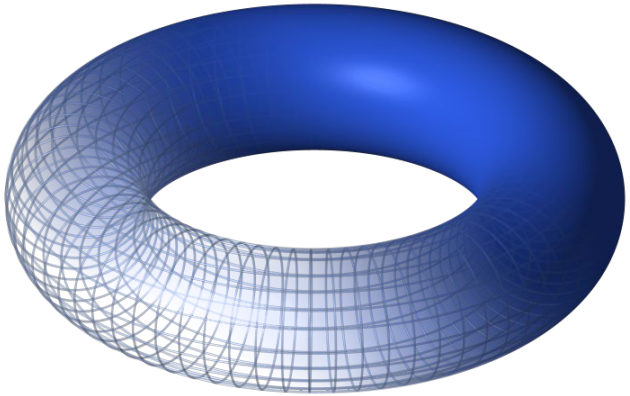
Periodic Boundary Conditions

What to do with the edges?

As we discussed earlier about Cellular Automata, when it comes to the edges there are three options: (no update, special rules, wrap the left to the right).

In 2D worlds like a particle life world the options are:

1. Edges are walls: Particles can not pass them and are repealed by them
2. Edges wrap to the opposite (left to right, top to bottom) creating a toroid



Typically we use the second option: if a particle's position exceeds an edge, reset the position to the opposite side. This has a side-effect - it is like teleporting the particle: suddenly a particle exists at a different location.

If particles experience forces from neighbour particles, then what happens at the edges?

This problem has been solved in molecular dynamics simulations with the concept of periodical boundary conditions

The obvious solution

Assume an area of width w and height h (say 400 by 400), and a particle with (x, y) coordinates. The particle travels from left to right (increasing its x).

What happens when x>400? Easy answer is to set x to x-w. Similarly we can generalise this in all directions.

However say that a particle is at (10, 50) and another at (390, 50). What is the minimum distance between them? If we use the Euclidean distance, that is 380.

However, because we wrapp left to right, the actual minimum distance is wrapped as well and it is 20!

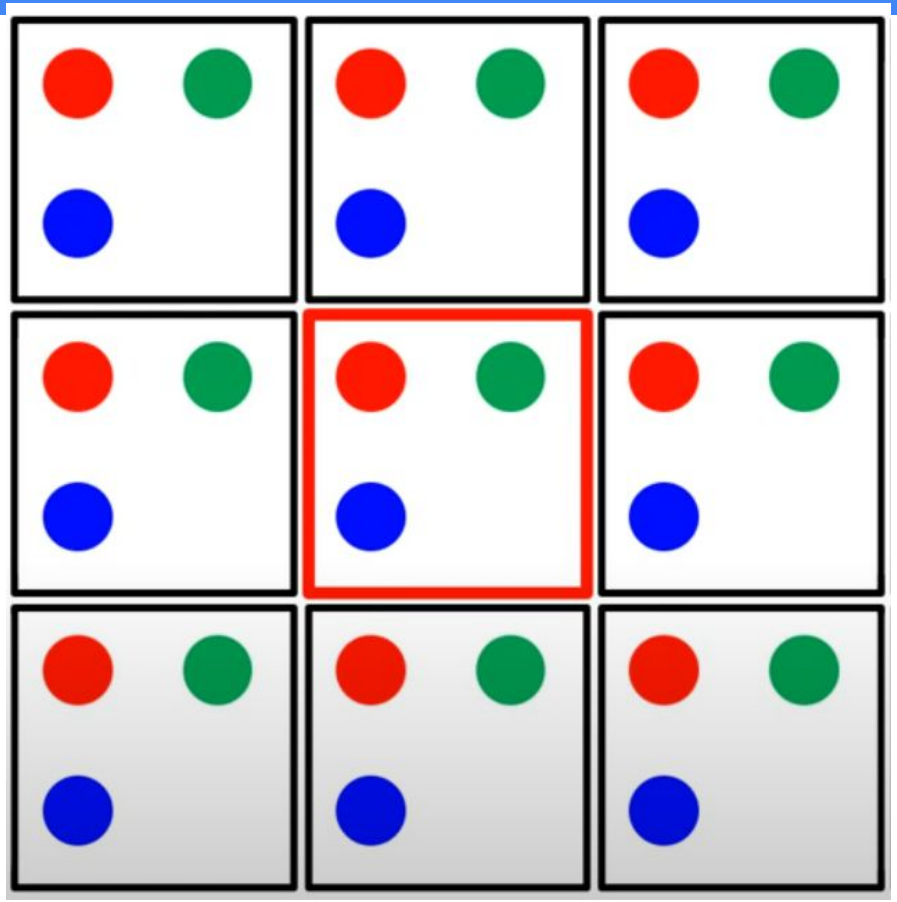
The actual solution is **Periodic Boundary Conditions** (PBC)

The problem from the molecular dynamics side of view

Molecular computation is about creating simulation of molecular systems. However for the simulation to be accurate it needs to model over 10E23 atoms. A moderate computer simulation can simulate about 10E5 to 10E6 atoms.

The solution that Periodic Boundary Conditions (PBC) offer is to model only a small area but repeat it to all directions at the same time. The simulated area is called the primary cell, and it is surrounded by copies (shadows) in all directions.

If something changes in the primary cell, it is copied directly to the other cells.



Wrapping the positions

We do not need to keep multiple copies of the particles.

Instead we can calculate the wrapped location of each particle with the following equations / code:

- on boundaries [0, 1):

```
x = x - floor(x); y = y - floor(y);
```

- on generic boundaries [a, b):

```
x = x - floor((x-a)/(b-a)); y = y - floor((y-a)/(b-a));
```

Calculating the shortest distance

Determine the shortest connection of two positions in your periodic space. The connection can wrap around boundaries. First, compute the connection in the usual way:

```
dx = x2 - x1; dy = y2 - y1;
```

Then, wrap the connection:

- on boundaries [0, 1):

  ```
  dx = dx - round(dx); dy = dy - round(dy);
  ```

- on generic boundaries [a, b):

  ```
  dx = dx - round((dx)/(b-a)); dy = dy - round((dy)/(b-a));
  ```

The shortest vector connecting the two particles:

```
v = createVector(dx, dy);
```

Then the distance is calculated with `d = v.mag();`

Learn more about Periodic Boundary Conditions

https://tommohr.dev/

https://www.youtube.com/watch?v=IpdTtx98AWQ&t=606s&ab_channel=MoBioChem