Simple Model of Asymmetric Velocity

by Sven Nilsen, 2020

In this paper I present a simple physics model of asymmetric velocity.

Asymmetric Velocity Logic is a theory based on asymmetric velocity reference frames. It can be used to prove some common sense problems in physics. However, the logic describes physical systems in frozen time and does make not enough assumptions to model physics.

Therefore, I created a simple physics model to demonstrate Asymmetric Velocity Logic:

- All particles have same radius
- All particles have same mass
- No gravity
- No Einstein relativity
- Fixed time step
- A simple global friction coefficient
- N-dimensional (works in both 1D, 2D, 3D etc.)
- All interactions grounded to boundary between particles (specified by radius)

This model is designed for easy manipulation and to program from scratch without much experience.

Instead of keeping track of velocities, there are three lists of particle positions:

- List of previous particle positions `prev_pos`
- List of current particle positions `pos`
- List of next particle positions `next_pos`

The velocity is computed the following way:

```
vel(i : nat) ~ pos: [vector], prev_pos: [vector] = pos[i] – prev_pos[i]
```

This works because the time step is fixed.
Here, the `~` symbol introduces context variables into scope.

A pair of particles `i, j` has one of four relations with interaction depending on vector `d(i, j)`:

```
d(i : nat, j : nat) ~ pos: [vector] = pos[j] – pos[i]
```

- no particle interacts with itself
- stay (no interaction)
- push (interacts when `|d(i, j)| < 2 * rad`)
- pull (interacts when `|d(i, j)| > 2 * rad`)
- follow (always interact)

Pseudo code for interaction:

```
interacts(i : nat, j : nat) ~ rel: [[velocity_constraint]], rad: real =
    if i == j {false} else {
        (mi, ma) := if i < j { (i, j) } else { (j, i) }
        match rel[j][i] {
            stay => false,
            push => |d(i, j)| < 2 * rad,
            pull => |d(i, j)| > 2 * rad,
            follow => true,
        }
    }
```

Here, `rad` is the radius of particles. *Pro tip:* One can set the radius to `0.5` or `1` to optimize.

Relations are stored in a list of lists that increases in size for each particle:

```
rel[0] = []
rel[1] = [push]         // 0
rel[2] = [push, pull]   // 0, 1
…
```

This makes it possible to add new particles without needing to update existing lists of relations.

Notice that the implementation of relations can be optimized further (memory complexity).

Another important thing to notice: The relations that Asymmetric Velocity logic infers along a path, is **not** the relations specified here. These relations are **used** for inference, but the semantics of logical relations along a path is interpreted within the logic for reasoning, not for simulation.

The new position of a particle during a time step is computed the following way:

```
new_pos[i] = inv_friction * (vel(i) +
    ∑ j [0, n) { if interacts(i, j) { 0.5 * (|d(i, j)| - 2 * rad) * d(i, j) / |d(i, j)| }
})
```

The `inv_friction` variable is a number between `0` (100% friction) and `1` (0% friction).

Notice that the implementation of computing new positions can be optimized further (time complexity). One can also improve numberic stability by smoothing out the divisor `|d(i, j)|` when it goes to zero.

After new positions are computed, one can do the following swaps:

```
prev_pos => pos
pos => next_pos
next_pos => prev_pos
```

Pro tip: Swap pointers if the implementation programming language supports it.