# A Word On Noise Flow Fields

Bhaswar Chakraborty, Second Semester

A flow field is simply a kind of vector field. In other words, it's a collection of vectors with a given magnitude and direction. It can be visualised by a plane having a collection of arrows with various lengths, pointing at various directions. All of these vectors are a list of two numbers, the x-component and the y-component. This doesn't seem much, but these vectors can be thought of as individual forces. So if we sprinkle a few particles in this imaginary force field, they would be influenced by these forces and move around the plane according to that. That will not be very interesting to look at, though, because at this point we just have a bunch of vectors with random magnitudes and directions. This is a great place to use something called Perlin Noise. Perlin Noise is an algorithm to generate a collection of pseudorandom numbers, that are in hindsight very organic and smooth, rather than being completely chaotic. In this case we are going to need the 2-dimensional Perlin Noise. So basically, we are going to use it to set the magnitudes and the directions of the vectors present in our flow field. For simplicity's sake, let's make all the vectors of unit length and change only the direction. Let's try to implement it then. I am going to be using JavaScript syntax for the code snippets.
Each vector is a list of two numbers, so it could be an array, or in this example, an object. So the field is essentially an array of the vector objects. But how do we get the x and y components of the vector? Remember that we are considering the vectors as unit vectors having an angle according to the 2D Perlin Noise. For a given angle $\theta$, the components of a unit vector would be $(cos\,\theta,\ sin\,\theta)$, as the magnitude, $x^2 + y^2 = 1$. So, in code:

```
const flowfield = [];
for (let j = 0; j < rows; j++) {
  for (let i = 0; i < cols; i++) {
    let angle = noise(i * 0.1, j * 0.1) * TWO_PI // from perlin noise
    let v = { x : cos(angle), y : sin(angle)}
    flowfield.push(v)
  }
}
```

We have now created our flow field. But it's still basically a bunch of vectors. We need some particles to visualise it. Let's make an array of coordinate vectors representing a bunch of particles.

```
for (let i = 0; i < n; i++){
  points[i] = { x : Math.random() * width, y : Math.random() * height }
}
```

Now we just need to apply the nearest vector from the field to the particle.

```
for (let p of points) {
  let x = floor(p.x / scale), y = floor(p.y / scale);
  let index = x + y * cols;
  let force = flowfield[index];
  p = vectorAddition(p, force);
}
```

And that's it, if we trace the positions of each point through time, we can see how the particles are influenced by our flow field. You can make another flow field with a different distribution of directions, and it would be radically different from the first one. The possibilities are endless.