Diffusion in Three Dimensions — Monte Carlo

This is our twenty-second notebook. It would be straightforward now that you have seen one- and two-dimensional diffusion solved with NDSolve[] to take the same approach in three dimensions. *So let's not do that.*

Instead we are going to do something completely different: we will use Monte Carlo methods. This takes us all the way back to the very first demonstration I did, which was called "Heads or Tails." What better way to simulate random processes, than to let the computer make random steps?

Also, I did a fair amount of the two-dimensional case below. Your job is going to be to finish that which is not a big deal, but then then upgrade the two-dimensional code to be three-dimensional.

A Swarm

We need a list that holds a swarm of particles. The particle properties could get fancy. They could have mass, temperature, and composition. We are just starting out, so we are going to deal with swarms of particles that have no properties. Each particle will only have a position. Let's make it easy to add a particle to the swarm:

In[1623]:=
```
trailLength = 500;
addToSwarm[s_, position_] := Append[s, Table[position, trailLength]]
```

As particles diffuse away from the source, they will eventually get sufficiently far that we no longer care about their future movements. If we don't stop to prune such particles, our computers are going to bog down computing the positions of an ever-enlarging swarm. Let's make it easy to prune particles from the swarm:

In[1625]:=
```
retainCondition[trail_] := Module[{latestPosition = trail[[1]]},
   -1 ≤ latestPosition[[1]] ≤ 5 && -3 ≤ latestPosition[[2]] ≤ 3];
pruneSwarm[s_, retainCondition_] := Select[s, retainCondition]
```

Evolution Step for a Particle

Now we need an evolution step that we will repeatedly apply to each particle:

In[1627]:=
```
windComponentX = 0.002;
windComponentY = 0.00;
steadyWind = TORRENTIAL;

horizontalSigma = 0.005;
horizontalMixingGaussian = NormalDistribution[0, horizontalSigma];
randomMotion := DOWNPOURS;

particleStep[particleTrail_] := (
  previousPosition = particleTrail[[1]];
  newParticleTrail = Drop[particleTrail, -1];
  newPosition = MOVE BOULDERS;
  newParticleTrail = AND TREES;
  newParticleTrail
 )
```

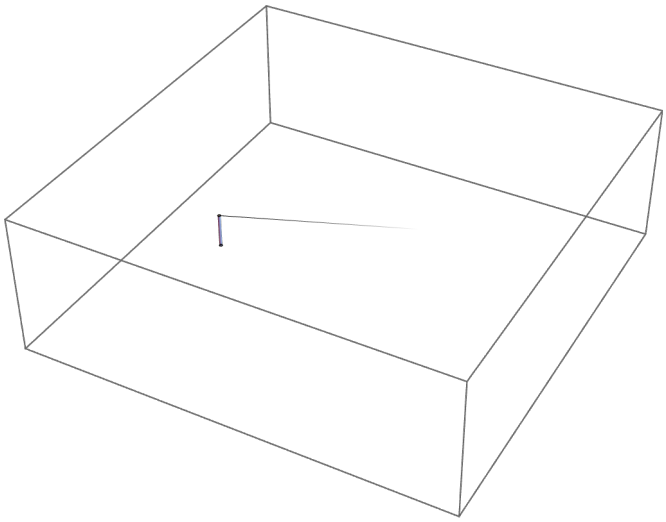
Rendering the Swarm

Anticipating that we will want to go to three dimensions by the end of this notebook, even though at the moment our swarm is two-dimensional, let's render it in 3-D so that we don't have as much rewriting to do later:

In[1634]:=
```
renderSwarm[s_] := Module[{smokestack = Cylinder[{{0, 0, 0}, {0, 0, 0.5}}, 0.02],
    points = Table[{PointSize[0.001], Opacity[(trailLength - i + 1) / trailLength],
        Point[{#[[i]][[1]], #[[i]][[2]], 0.5}]}, {i, 1, trailLength}] & /@ s},
  Graphics3D[Flatten[{smokestack, points}],
   PlotRange → {{-1, 5}, {-3, 3}, {0, 2}}, BoxRatios → {6, 6, 2}]]
```

In[1635]:=
```
(* Test our rendering code with a single-particle swarm:) *)
renderSwarm[{Table[{ 0.005 i, 0.002 i}, {i, 0, trailLength - 1}]}]
```

Out[1635]=



Evolution Step for the Swarm

We need a convenient way to apply the evolution rule to every particle in the swarm, but that is so simple, we hardly even need to define a function to do it:

In[1636]:=
```
swarmStep[s_] := particleStep /@ s
```

Evolving the Swarm

In[1637]:=
```
totalAnimationTime = 20;
renderedFramesPerSecondOfAnimation = 5;
totalFrames = totalAnimationTime renderedFramesPerSecondOfAnimation + 1;
stepsPerFrame = 50;
```

In[1641]:=

```
(* We want a function that nests swarmStep stepsPerFrame times,
then prunes the swarm, *)
(* and then adds to the swarm a new particle at the origin: *)
compoundSwarmStep[s_] := addToSwarm[
   pruneSwarm[Nest[swarmStep, s, stepsPerFrame], retainCondition], {0.0, 0.0}];
 (* Then we want to use NestList to compound that
  function to make an evolution that is totalFrames long: *)
 evolution = NestList[compoundSwarmStep, {}, totalFrames - 1];
 (* Note that due to the way NestList works,
 you get one more frame than expected, *)
 (* so we asked for one less. *)
```

Animating the Evolution

In[1642]:=

```
(* This thing animates pretty slowly,
but we can help some by pre-rendering the frames: *)
renderedFrames = Table[renderSwarm[evolution[[i]]], {i, 1, totalFrames}];

Animate[renderedFrames[[i]], {i, 1, totalFrames, 1},
 DefaultDuration → 5 totalAnimationTime, AnimationRunning → False]
```

Out[1643]=