K会2012年度冬季講習　情報講座

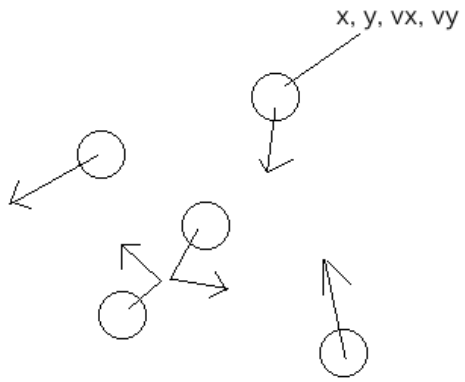
リピーターコース　Part4　シミュレーションCG

目次

4-1. 粒子法

（１）粒子法の基本

生物の群れ、軍隊、といった集団の動きをシミュレーションする代表的な手法が粒子法である。粒子法では個体一人ひとりを{x, y, vx, vy, … }といった変数の組で表し、その時間変化を追跡する。水や布、あるいは固体を結合した粒子の集まりだとしてシミュレーションする手法もある（4-7 で扱う）。



大事なのは時間変化なので、１ステップに各変数がどう変化するのかが重要である。おなじみの物体の位置を x, y で、速度を vx, vy で表せば、まっすぐに進む物体の１ステップ（時間が dt 進むとする）での動きは、

$$\begin{array}{ll} x & x + dt * vx \\ y & y + dt * vy \\ vx \quad \rightarrow & vx \\ vy & vy \end{array}$$

となる。Part1 で作った雪のプログラムはこのような粒子法の一つである。ここから先はこのプログラムをベースにして作っていくので、右ページにその例を載せておいた。

なお、class との相性が良いので、class を使える人は使ったほうが綺麗に書けるかもしれない。

```
int N = 100;
float[] x, y, vx, vy;
float dt = 1.0;

void setup(){
  size(400, 400);
  colorMode(RGB, 100);

  x = new float[N];
  y = new float[N];
  vx = new float[N];
  vy = new float[N];

  for(int i=0; i<N; i++){
    x[i] = random(0, width);
    y[i] = random(0, height);
    vx[i] = random(-1, 1);
    vy[i] = random(-1, 1);
  }
}

void timestep(){
  // 物体を移動させる
  for(int i=0; i<N; i++){
    x[i] += dt * vx[i];
    y[i] += dt * vy[i];
  }

  // 端に行った物体を反対側の端へ
  for(int i=0; i<N; i++){
    if(x[i]<0) x[i]=width-1;
    if(x[i]>width) x[i]=1;
    if(y[i]<0) y[i]=height-1;
    if(y[i]>height) y[i]=1;
  }
}
void draw(){
  timestep();

  background(100);
  for(int i=0; i<N; i++){
    ellipse(x[i], y[i], 10, 10);
  }
}
```

4-2. 力

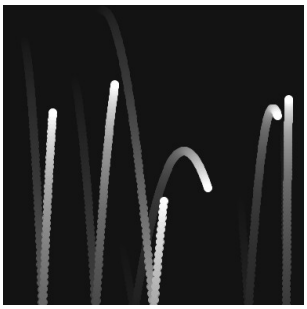
力を受けた物体はその速度を変化させる。ニュートンの理論によれば、物体の速度の変化（加速度という）は、受けた力に比例する。加速度を ax, ay とすれば、

```
x          x + dt * vx
y          y + dt * vy
vx   →     vx + dt * ax
vy         vy + dt * ay        ax, ayは力に比例
```

これから力を(fx, fy)と書くことにする。これは(ax, ay)と比例しているので、プログラムでは k を何か定数として、ax = k*fx, ay = k*fy として計算すれば良い。

（１）一定の力

```
fx = 一定
fy = 一定
```



力が一定のときの運動は、(fx, fy)の方向に常に加速され続ける運動となる。地球上での重力はその一例である。右図は下方向に一定の力をかけた時の例（下の面で反射させている）。

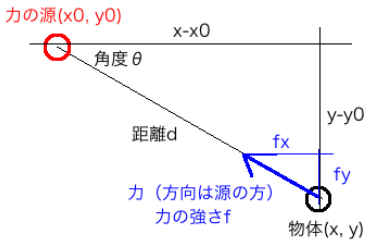
（２）速度に比例した力

```
fx = -a * vx
fy = -a * vy     aは定数
```

このように速度に比例し、速度と逆向きの力は抵抗力の一種である。速度が速いほど、速度とは反対の向きに力がかかるので、さっきの重力と組み合わせると右図のようにだんだん抵抗を受けてエネルギーを失っていく物体の運動が描ける。

（３）位置で決まる力

力の源がある位置を(x0, y0)、源と物体との距離を d=sqrt((x-x0)*(x-x0)+(y-y0)*(y-y0))とする。



三角形の相似から
fx = f * (x-x0) / d
fy = f * (y-y0) / d

⑴源との距離に比例する力

力の強さ f が d に比例

```
fx = a * d * (x-x0) / d = a * (x-x0)
fy = a * d * (y-y0) / d = a * (y-y0)
```

力の源の位置を画面の中心とした時の物体の動き。源を中心とするだ円を動く。

源からバネで繋がれているような運動をし、源から遠ざかるほど強い力がかかるという性質がある。

⑵距離に反比例

力の強さ f が 1/d に比例

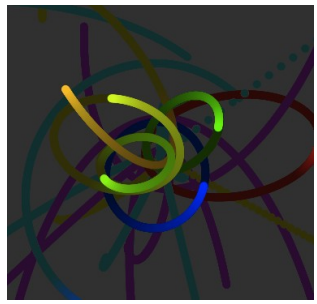
```
fx = a * (x-x0) / d / d
fy = a * (y-y0) / d / d
```

源から遠ざかるほど力は弱まるが、どれだけ離れていても中心の周りをぐるぐると回る。決まった軌道は持たない。

力の強さ f が 1/d/d に比例

```
fx = a * (x-x0) / d / d / d
fy = a * (y-y0) / d / d / d
```

これは太陽のまわりを回る惑星などと同じ運動をする。楕円形の決まった軌道をとる（同じ所を回り続ける）ものと、一度離れると永久に戻ってこないものがいる。



（４）課題

物体たちがマウスに集まってきたり、逃げていくようなプログラムを書こう。

4-1 の timestep 命令を次のようにすると、マウスとの距離に比例した引力が働き、物体とマウスとがバネで繋がれているかのように動く。

・マウスとの距離に反比例して引き寄せられるようにしてみよう

・反発するようにしてみよう

```
void timestep(){
  float[] fx, fy;
  fx = new float[N];
  fy = new float[N];
  // 力
  for(int i=0; i<N; i++){
    // マウスへよる力
    fx[i] = - 0.001 * (x[i]-mouseX);
    fy[i] = - 0.001 * (y[i]-mouseY);
    // 抵抗力
    fx[i] -= 0.01 * vx[i];
    fy[i] -= 0.01 * vy[i];
  }
  // 物体を移動させる
  for(int i=0; i<N; i++){
    x[i] += dt * vx[i];
    y[i] += dt * vy[i];
    vx[i] += dt * fx[i];
    vy[i] += dt * fy[i];
  }

  // 端に行った物体を反対側の端へ
  /*
  for(int i=0; i<N; i++){
    if(x[i]<0) x[i]=width-1;
    if(x[i]>width) x[i]=1;
    if(y[i]<0) y[i]=height-1;
    if(y[i]>height) y[i]=1;
  }
  */
}
```

・物体同士が反発しあうプログラムを書こう。すべての物体について、その物体が他の物体から受ける力を全て計算し、合計する。物体に複数の力が働いているときは、その合計が、最終的にその物体が受ける力となる。

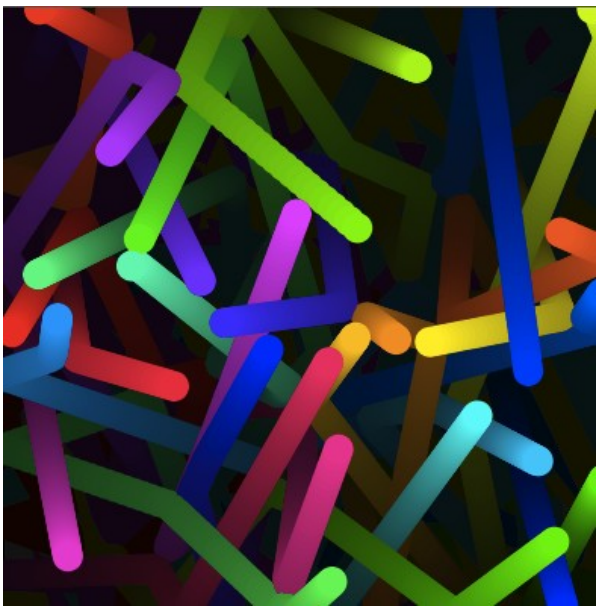
　物体同士がある距離 d0 以内に近づいたら距離に比例した反発力が働くようにする。そのとき

```
fx = - a * (d-d0) * (x[i]-x[j]) / d
fy = - a * (d-d0) * (y[i]-y[j]) / d

d: x[i]とx[j]の距離。sqrt((x[i]-x[j])^2 + (y[i]-x[j])^2)
d0: 反発力が働き始める距離
```

となる。

物体が吹っ飛んでいってしまう時は、抵抗力を加えると良い。



4-3. 群れ

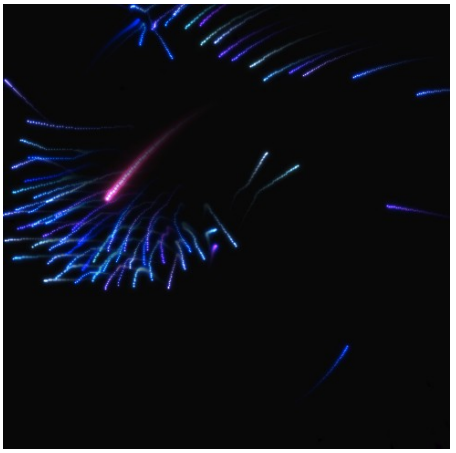
魚の群れや編隊飛行する戦闘機などの動きをシミュレーションする。
粒子に次のような力をかけると実現できる。

（１）抵抗力
　動きが速くなり過ぎないように速度に比例した抵抗をかける。ただし、4-2 でのやりかただと動きが止まってしまうので、速度 v=sqrt(vx*vx+vy*vy)に対し、-(v-v0)に比例するような力をかける（v0 は目標速度）。

（２）反発力
　物体同士がある程度以上近づかないようにする。物体同士の距離を d, これ以上近づくと反発するという距離 d0 として、d<d0 のときのみ(d-d0)に比例するような力をかけると、反発が実現できる。

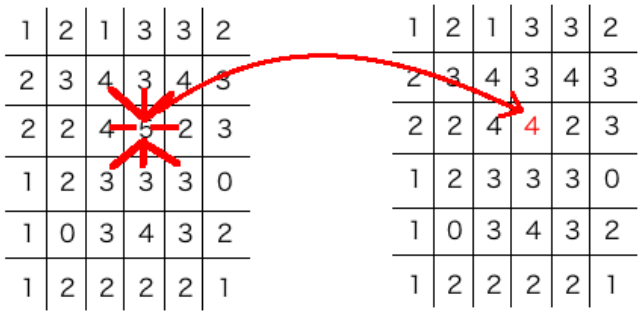
（３）同調圧力
　近くにいる物体と同じ速度になろうとする力。物体 i, j について、速度差 -(vx[i]-vx[j])に比例し、かつ距離 d=sqrt(x*x+y*y)に反比例するようにすると良い。

下の図が完成目標。右のは赤玉から逃げるようになっている。3D で作ってみても面白い。



4-4. 格子法

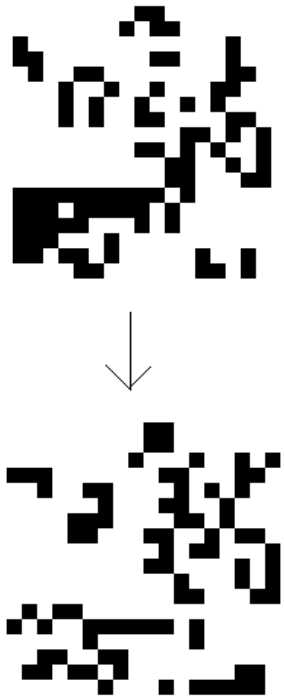
連続的な空間の動き（波とか）をシミュレーションするには、その空間を配列上にとって（2 次元空間なら 2 次元配列）、その各点 x[i][j]の時間変化を計算する。格子法がその真価を発揮するのは、x[i][j]の時間変化が、その周囲の点 x[i+1][j]などのみから計算できる場合である。離れた点同士が相互作用する場合（x[i][j]と x[i+50][j-30]とか）は、さっきの粒子法を使った方がいい。



格子法の一つのシンプルな例がライフゲームである。

あるマスが次に白になるか黒になるかはその周囲 3*3 マスの状態だけで決定される。（詳しいルールは初級編テキストの最後を参照、あるいは wikipedia とか）下はライフゲームの timestep と、盤面の変化の例

```
void timestep(){
  for(int i=1; i<NX-1; i++){
    for(int j=1; j<NY-1; j++){
      int s = life[i+1][j+1]+life[i+1][j]+life[i+1][j-1];
      s += life[i][j+1]+life[i][j-1];
      s += life[i-1][j+1]+life[i-1][j]+life[i-1][j-1];
      if(life[i][j]==1){
        if(s==2 || s==3){
          nlife[i][j] = 1;
        }else{
          nlife[i][j] = 0;
        }
      }else if(life[i][j]==0 && s==3){
        nlife[i][j] = 1;
      }else{
        nlife[i][j] = 0;
      }
    }
  }
  for(int i=1; i<NX-1; i++){
    for(int j=1; j<NY-1; j++){
      life[i][j] = nlife[i][j];
    }
  }
}
```



4-5. 炎

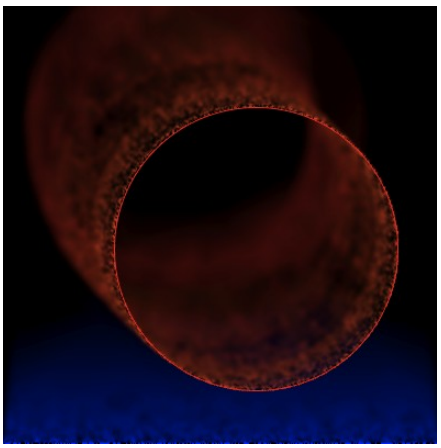
　格子法を使うと炎を簡単に作ることができる。毎ステップに次のような処理を行う。

・一番下のマスに、ランダムに 0 か 1 を入れる。
・各マスの値を、次の比率で混ぜ合わせた値に上から更新していく

0	0.5	0
0.125	0.25	0.125

z[i][j] = 0.5*z[i][j] + 0.125*z[i+1][j-1] + 0.25*z[i+1][j] + 0.125*z[i+1][j+1]

一番下のマスだけでなく、マウスを中心とした円上にランダムに 0, 1 をつけると、次のような炎の輪になる。



混ぜ合わせる比率や方法を変えるともっと綺麗に見えるようになるかも知れない

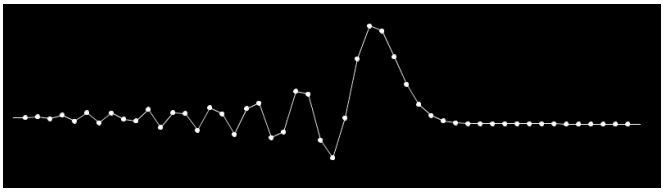
4-6. 波

　波のシミュレーションは、格子点上に並べられ、隣り合った粒子がバネで繋がれているような模型を計算すると実現できる。格子上の各点が位置と速度 z[i][j], vz[i][j]を持つと考える。隣り合った粒子との z[i][j]の差に比例した、差をなくそうとする方向の力がかかるとすると、

```
z[i][j]    →   z[i][j] + dt * vz[i][j]
vz[i][j]       vz[i][j] + dt * a * ((z[i+1][j]-z[i][j]) + (z[i-1][j]-z[i][j])
                                   + (z[i][j+1]-z[i][j]) + (z[i][j-1]-z[i][j]))
```

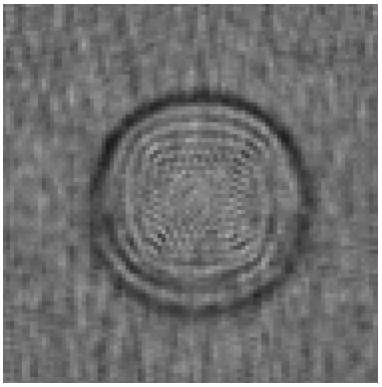
のように二次元配列の値を更新するといいことがわかる。
ただし、この問題は計算順序によっては発散（答えが徐々に増幅されて無限大に吹っ飛ぶ）することが知られている。
　z[i][j]を全て更新する　→　vz[i][j]を全て更新する　→　z[i][j]を　…
の順序でやればうまくいくだろう。
　また、抵抗力を加えると波が時間とともに穏やかになっていくので良い。

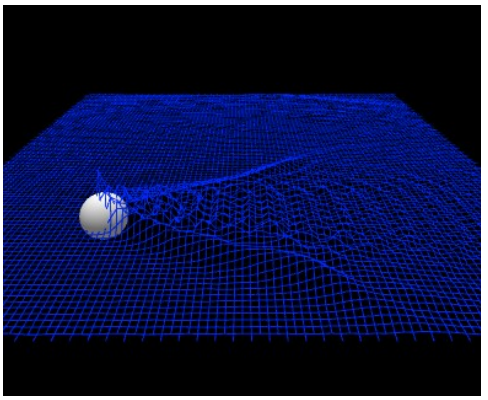
　マウスの位置にも力がかかるようにすると、波を自分で起こせるようになる。



1 次元の波
(これは 1D 配列で)

２次元の波。左：濃淡で表示、右： 3D の高さで表示





4-7. 弾性体

　消しゴムやボールのような弾力をもった物体の動きをシミュレーションする。硬いものでも強い力を受けると歪むので、映画で大きなものが爆破されるシーンなどにも使えたりする。
　格子法と粒子法をハイブリッドする。まず格子法のように二次元配列で物体の情報を表現する。ただし、粒子法のように物体の各点 x[i][j]などは速度 vx[i][j]を持ち空間内を動くことができる。力は格子法的に周囲のマスからのみ受ける。固体のシミュレーションを考えているので、物体の大きすぎる変形により「隣り合ってたマスが隣り合わなくなってしまう」とかの現象はないものと考える。

　隣り合った物体同士の及ぼす力は、4-2 などでやったような、物体間の距離に比例したものとする。ただし、ある好みの距離 d0 があり、そこから外れると、戻るような力がかかるようにする。この d0 が物体間の結合距離となる。

```
fx =  a * (d-d0) * (x[i]-x[j]) / d
fy =  a * (d-d0) * (y[i]-y[j]) / d
```

ここで、ナナメの結合は横の結合距離 d0 の√2 倍になることに注意。
重力をかけると、うまくいくと次のように地面で跳ねる。
　布や 3 次元の弾性体も作れる。布は二次元配列のままで作れるので意外と楽。