Homework 5: Feynman problem

David Jedynak

May 10, 2018

Abstract

We show how an angled tube can be created in an integer lattice gas method simulation. This angled tube is used with particle sources to perform an experiment to provide insight on the Feynman problem. The momentum transferred from the particles to the tube will be estimated to understand the response of the tube to different densities. The influence of density ratios inside and outside the tube on the force will be observed. The influence of the tube geometry on momentum will also be tested.

1 Introduction

The Feynman problem presents the question of what will occur when an angled tube attached to a pivot point has fluid pushed out or sucked into the tube. Will the tube rotate clockwise or counter clockwise? My prediction is that the tube will rotate clockwise when air is pushed out of the tube and counter clockwise when air is sucked in, if the experiment is set up as in the figures 1 - 6 below.

The Lattice Gas simulation where this experiment will be executed has constraints that particle momentum and the number of particles must be conserved. The collisions and resulting velocities can be modeled using random processes that can lower the computation cost from prior simulations that would calculate interactions between every particle based on distance, charge, mass, and fields. This can be done because the prior simulations exhibit this random behavior.

2 Walls

The Feynman experiment will require 2 Dimensional walls to create a right angled tube. Particles must be reflected if they hit a wall. Particles cannot pass through walls. To accomplish this requirement, velocities in different directions can be swapped. The lattice gas simulation is composed of many cells. Each cell has 9(3 by 3 grid) different values to describe particles motion in a cell.

Here are the indexes for each velocity. The value behind each index is the magnitude of the velocity in that direction

Indexes			
	0	1	2
	3	4	5
	6	7	8

Velocities corresponding to Indexes above

$(-v_x, +v_y)$	$(0, +v_y)$	$(+v_x, +v_y)$
$(-v_x, 0)$	$(0, 0)$	$(+v_x, 0)$
$(-v_x, -v_y)$	$(0, -v_y)$	$(+v_x, -v_y)$

2.1 Walls as Links

A wall will be composed of links. The links will have an x and y coordinates that will be the same as the lattice cell they are interacting with. Additionally these links will also store the index of the velocity position they will be swapping. Horizontal walls will be switching [0,1,2] will be switched with [8,7,6] respectively. Vertical walls will be switching [0,3,6] will be switched with [8,5,2] respectively. Only one of these sets for each Vertical and Horizontal walls will need to be stored because both will be able to derive the second set's indexes from the first set. Code defining Horizontal Walls:

Note: The horizontal walls link 0, 1, and 2 with 8, 7, and 6 respectively.

```
//horizontal walls
  for (int x=x0; x<x1+1; x++){
            links[linkcount][0] = x; //x-position
            links[linkcount][1] = yy2;
            links[linkcount][2] = 0;
            linkcount++;
            links[linkcount][0] = x; //x-position
            links[linkcount][1] = yy2;
            links[linkcount][2] = 1;
            linkcount++;
            links[linkcount][0] = x; //x-position
            links[linkcount][1] = yy2;
            links[linkcount][2] = 2;
            linkcount++;
  }
```

Code Defining Vertical Walls:
This wall in particular is also the opening for the tube that can be disabled or enabled. Note: The vertical walls link 0, 3, and 6 with 8, 5, and 2 respectively.

```
  //vertical walls
if(close_tube == 1){
  for (int y=yy0; y<yy1+1; y++){
        links[linkcount][0] = x2; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 0;
        linkcount++;
        links[linkcount][0] = x2; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 3;
        linkcount++;
        links[linkcount][0] = x2; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 6;
        linkcount++;
  }
}
```

Additional code added to prevent the tube from leaking particles at several vertices.

```
//2 links added to prevent leaking on the x0,yy2 and x2,yy0 squares
        links[linkcount][0] = x0; //x-position
            links[linkcount][1] = yy2;
            links[linkcount][2] = 0;
            linkcount++;
        links[linkcount][0] = x2; //x-position
            links[linkcount][1] = yy0;
            links[linkcount][2] = 0;
            linkcount++;
```

Code Defining How particles reflect off of walls (Both horizontal and vertical)

```
void bounceback(){
  tot_vx =0;
  tot_vy =0;
  for (int lc=0; lc<linkcount; lc++){
    //quantity of partices in a given link
    int x=links[lc][0];
    int y=links[lc][1];
    //velocity of the particles in a given link
    int v=links[lc][2];
    int vx=v%3-1;
    int vy=1-v/3;
    int tmp= n[x+vx][y+vy][v];
```

```
    //summing all momemtums
    tot_vx += -2*vx*(n[x][y][8-v]-tmp);
    tot_vy += -2*vy*(n[x][y][8-v]-tmp);
    //swapping the particles trying to enter
    //and leave to have the effect of a wall
    n[x+vx][y+vy][v]= n[x][y][8-v];
    n[x][y][8-v]=tmp;
  }
  //measure routine stores values for plotting
  Measure();
}
```

3 Calculating Momentum

The momentum of the tube can be calculated by relating the quantity and velocity of the particles being reflected by the tube. The momentum can be described using 2 dimensions x and y. The tube momentum in either direction will be proportional to 2 times the opposite momentum of the difference of the particles attempting to leave or enter the tube/wall. To find the total momentum, the sum of all these particle momentum will be taken over every single "link" or way particles can travel from one lattice cell to another.

$$(\rho_x, \rho_y) = -2 * \sum_{link_0}^{link+1} (vel_x * (\Delta particles_x), vel_y * (\Delta particles_y))$$

Accomplishing this in the program requires a calculating in the bounce back routine previously discussed in section 2.1 to iterate through all links and sum this calculation.

```
    int tmp= n[x+vx][y+vy][v];
    //summing all momemtums
    tot_vx += -2*vx*(n[x][y][8-v]-tmp);
    tot_vy += -2*vy*(n[x][y][8-v]-tmp);
```

4 Force as a Function of Density Ratios

As the difference in the source and drain densities increase, the magnitude of the force and momentum should increase. The particles are going to try and find the route of least resistance from source to drain. On this path they will probably collide with the tube and exert force on the tube. This is measured by observing the tube momentum as described in section3.

5 Results

Note: For figures 1 through 7, the Collisions parameter is set to 100, the source is placed inside the bent tube and the drain is placed at the coordinates (x,y) = (50,50). Both source and drain are the same size.

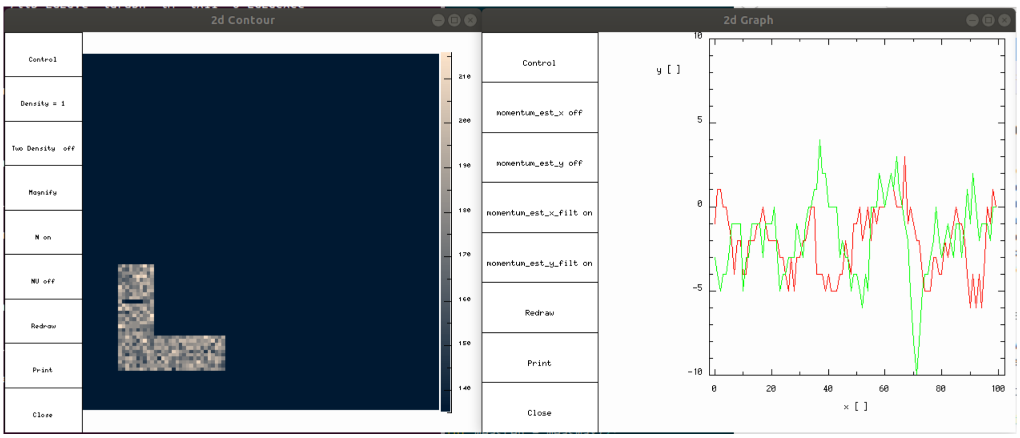

Figure 1: ρ_x is green and ρ_y is red. Image of fully closed tube with source located inside. light color high desity dark color low to zero density. The graph to the left shows the averaged x and y momentums centered around 0, so this is a good check that the box is sealed, beacuse otherwise there would be non zero net momentum.

5.1 Influence of source and drain densities on momentum

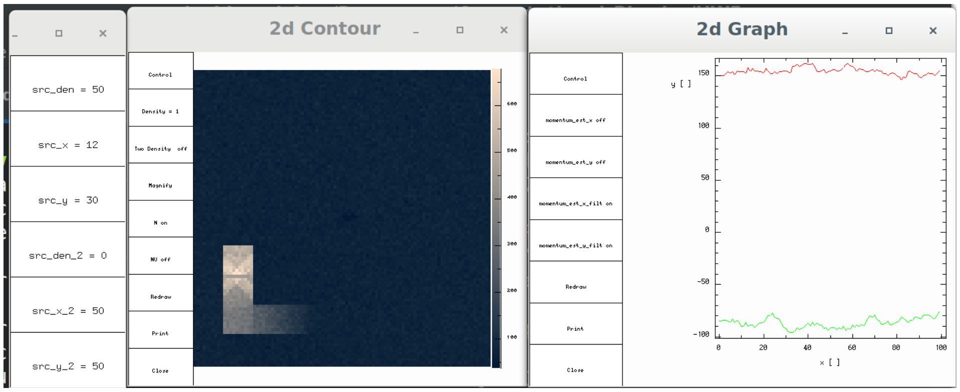

Figure 2: ρ_x is green and ρ_y is red. Source density is 50 and Drain density is 0. The tube momentum is in the positive y and negative x.

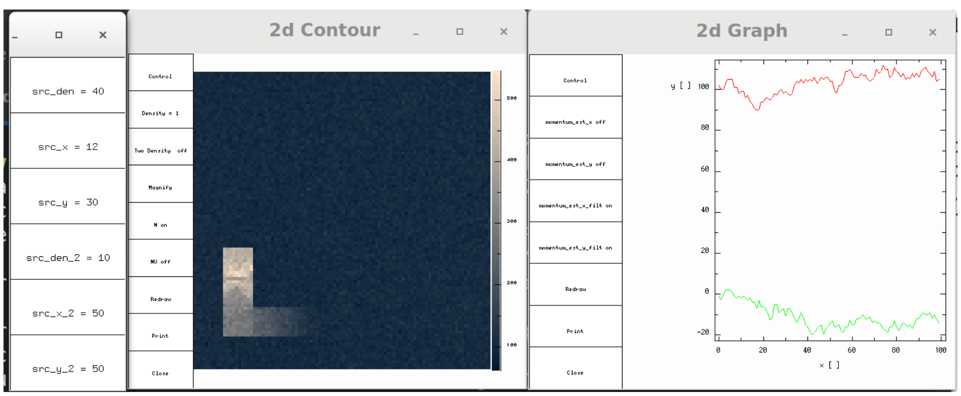

Figure 3: ρ_x is green and ρ_y is red. Source density is 40 and Drain density is 10.

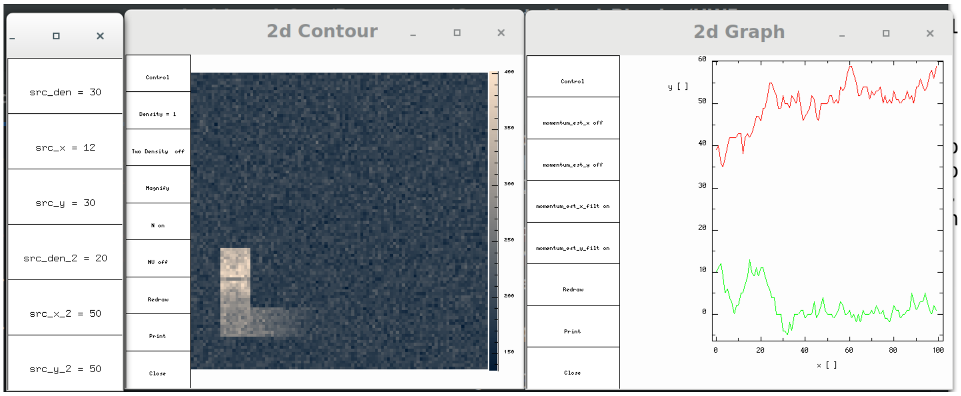

Figure 4: ρ_x is green and ρ_y is red. Source density is 30 and Drain density is 20.

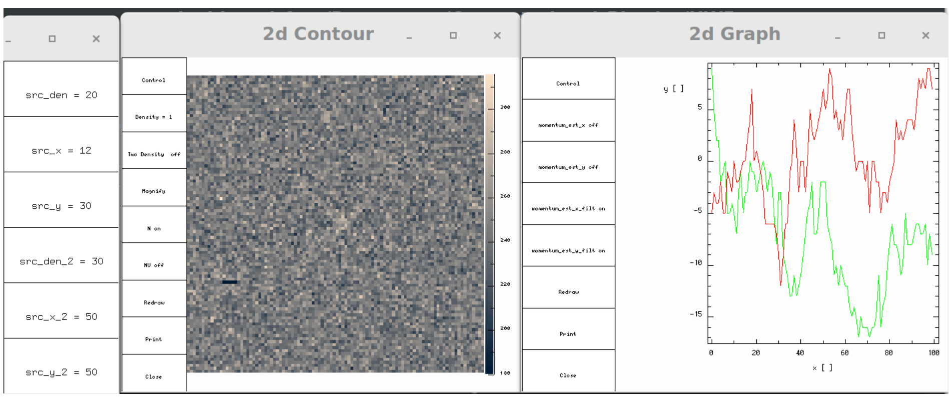

Figure 5: ρ_x is green and ρ_y is red. Source density is 20 and Drain density is 30.

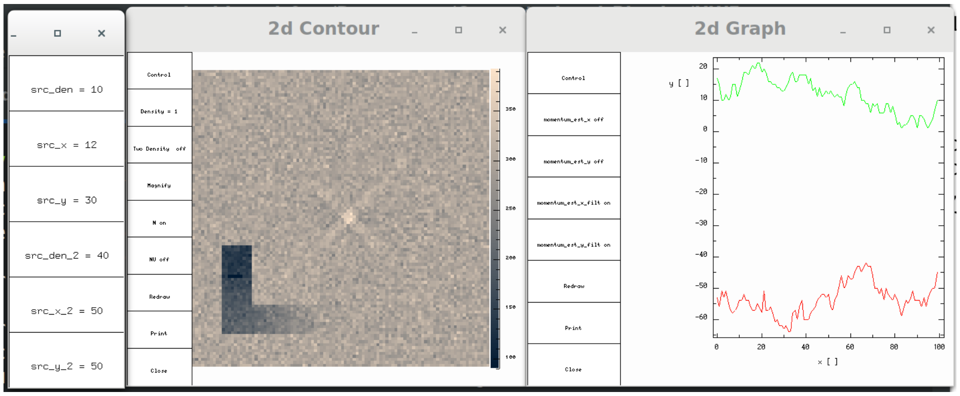

Figure 6: ρ_x is green and ρ_y is red. Source density is 10 and Drain density is 50.

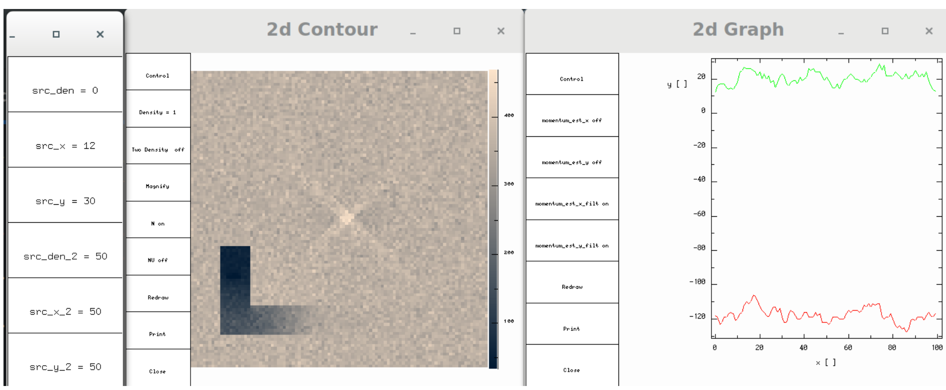

Figure 7: ρ_x is green and ρ_y is red. Source density is 0 and Drain density is 50. The tube momentum is in the negative y and positive x.

5.2 Force as a function of density ratios

Observing changes in momentum due to changing outlet dimensions. Collisions are set to 100. the source is set to 15 and the drain is set to 0.

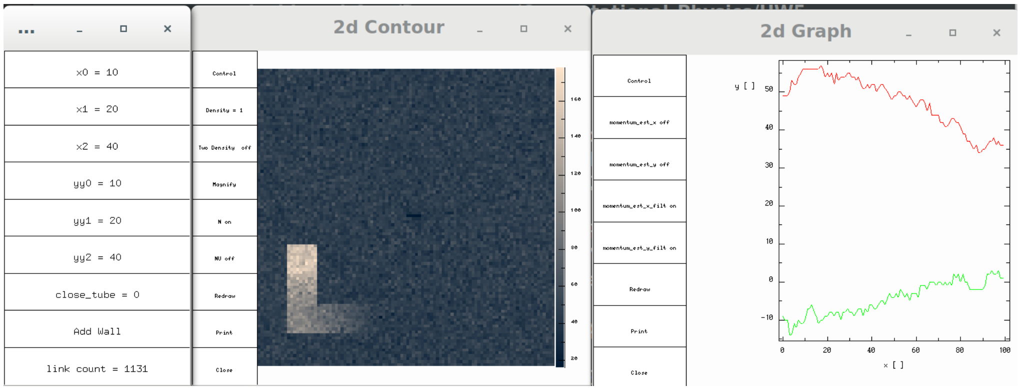

Figure 8: ρ_x is green and ρ_y is red. Outline size is 10

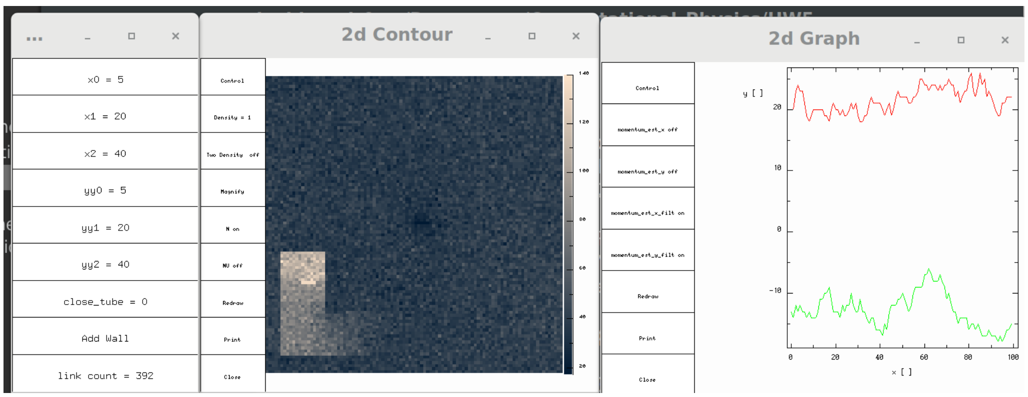

Figure 9: ρ_x is green and ρ_y is red. Outlet size is 15.

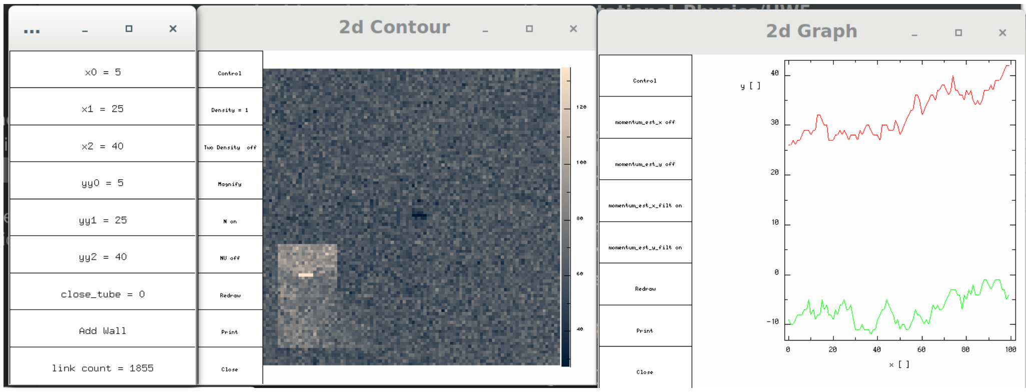

Figure 10: ρ_x is green and ρ_y is red. Outlet size is 20.

6 Conclusions

Figure 1 shows a tube fully sealed with no leaking, this is an important step because if there were leaks in the tube, this would cause error in the Feynman experiment momentum measurements.
By observing the figures 2 through 7, it can be seen that as the source and drain densities switch, the resulting momentum change for x and y do not appear to be equivalent. One notable change is that when source=50 and drain=0 the y momentum was around 150 and the y momentum was around -90. When the source=0 and the drain=50, the y momentum was around -120 and the x momentum was around 20. Additionally, in figure 3 and 4 where source and drain switch from being 30 and 20 to 20 and 30, they are extremely different in appearance. Perhaps this test required more time to reach steady state, but this was also observed using source and drain densities of 3 and 2 to 2 and 3.
As the outlet size increased, the momentum in both x and y dimensions decreased. This can be observed in figures 8 through 10. Y momentum changes from 50 to 20, X did not change very much -20 to -10. This hypothesized characteristic might be due to noise and potentially invalid. Figure 10 shows the y going back up to 40. Running the simulation will low source densities around 0 to 5 yielded very noisy data. The data from the 0 to 50 source densities yielded much more distinct changes in x and y momentum.

A Code

```
// Lattice gas code in 1d.
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <math.h>
```

```
#include <mygraph.h>
#include <string.h>

#define xdim 100
#define ydim 100
#define V 9       //number of velocities
#define MeasMax 200
int XDIM=xdim,YDIM=ydim;
int C=100;
double N[xdim][ydim],NU[xdim][ydim][2];
int Nreq=0,NUreq=0;
int n[xdim][ydim][V];

// Boundary variables
#define LINKMAX 10000
int x0=10,x1=20,x2 = 40,y2=20,yy0 = 10,yy1 = 20, yy2 = 40,close_tube = 0;
int linkcount=0,links[LINKMAX][3];
//variables for measuring tube momentum
int tot_vx=0,tot_vy=0;
int tot_vx_list[10],tot_vy_list[10];
//particle source parameters
int src_x = 12,src_y = 30,src_len = 5,src_den = 15;
int src_x_2 = 50,src_y_2 = 50,src_len_2 = 5,src_den_2 = 5;
int var1 =10;

//graphing
double momentum_est_x[MeasMax],momentum_est_y[MeasMax];
double momentum_est_x_filt[MeasMax],momentum_est_y_filt[MeasMax];
int MeasLen = MeasMax/2;
int range_val = 20;
int val[] = {0,0};
//added a running average to smooth out the graphs
void average(int range){
        val[0] = 0;
        val[1] = 0;
        for(int i = 0; i < range;i++){
                val[0] += momentum_est_x[i];
                val[1] += momentum_est_y[i];
                }
        val[0] = val[0]/range;
        val[1] = val[1]/range;
}
void Measure(){
  memmove(&momentum_est_x[1],&momentum_est_x[0],(MeasMax-1)*sizeof(int));
  momentum_est_x[0]=tot_vx;
  memmove(&momentum_est_y[1],&momentum_est_y[0],(MeasMax-1)*sizeof(int));
```

```
  momentum_est_y[0]=tot_vy;
  //filtered data
  average(range_val);
  memmove(&momentum_est_x_filt[1],&momentum_est_x_filt[0],(MeasMax-1)*sizeof(int
  momentum_est_x_filt[0]= val[0];
  memmove(&momentum_est_y_filt[1],&momentum_est_y_filt[0],(MeasMax-1)*sizeof(int
  momentum_est_y_filt[0]=val[1];
}
void FindLink(){
        //2 links added to prevent leaking on the x0,yy2 and x2,yy0 squares
        links[linkcount][0] = x0; //x-position
        links[linkcount][1] = yy2;
        links[linkcount][2] = 0;
        linkcount++;
        links[linkcount][0] = x2; //x-position
        links[linkcount][1] = yy0;
        links[linkcount][2] = 0;
        linkcount++;
  //horizontal walls
  for (int x=x0; x<x1+1; x++){
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy2;
        links[linkcount][2] = 0;
        linkcount++;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy2;
        links[linkcount][2] = 1;
        linkcount++;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy2;
        links[linkcount][2] = 2;
        linkcount++;
  }
    for (int x=x1; x<x2+1; x++){
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy1;
        links[linkcount][2] = 0;
        linkcount++;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy1;
        links[linkcount][2] = 1;
        linkcount++;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy1;
        links[linkcount][2] = 2;
        linkcount++;
```

```
  }
    for (int x=x0; x<x2+1; x++){
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy0;
        links[linkcount][2] = 0;
        linkcount++;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy0;
        links[linkcount][2] = 1;
        linkcount++;
        links[linkcount][0] = x; //x-position
        links[linkcount][1] = yy0;
        links[linkcount][2] = 2;
        linkcount++;
  }
  //vertical walls
if(close_tube == 1){
  for (int y=yy0; y<yy1+1; y++){
        links[linkcount][0] = x2; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 0;
        linkcount++;
        links[linkcount][0] = x2; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 3;
        linkcount++;
        links[linkcount][0] = x2; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 6;
        linkcount++;
  }
}
  for (int y=yy1; y<yy2+1; y++){
        links[linkcount][0] = x1; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 0;
        linkcount++;
        links[linkcount][0] = x1; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 3;
        linkcount++;
        links[linkcount][0] = x1; //x-position
        links[linkcount][1] = y;
        links[linkcount][2] = 6;
        linkcount++;
  }
```

```
  for (int y=yy0; y<yy2+1; y++){
         links[linkcount][0] = x0; //x-position
         links[linkcount][1] = y;
         links[linkcount][2] = 0;
         linkcount++;
         links[linkcount][0] = x0; //x-position
         links[linkcount][1] = y;
         links[linkcount][2] = 3;
         linkcount++;
         links[linkcount][0] = x0; //x-position
         links[linkcount][1] = y;
         links[linkcount][2] = 6;
         linkcount++;
  }

}

void bounceback(){
  tot_vx =0;
  tot_vy =0;
  for (int lc=0; lc<linkcount; lc++){
    //quantity of partices
    int x=links[lc][0];
    int y=links[lc][1];
    //velocity
    int v=links[lc][2];
    int vx=v%3-1;
    int vy=1-v/3;
    int tmp= n[x+vx][y+vy][v];
    //summing all momemtums
    tot_vx += -2*vx*(n[x][y][8-v]-tmp);
    tot_vy += -2*vy*(n[x][y][8-v]-tmp);
    //swapping the particles trying to enter and leave to have the effect of a u
    n[x+vx][y+vy][v]= n[x][y][8-v];
    n[x][y][8-v]=tmp;
  }
  //measure routine stores values for plotting
  Measure();
}

void setrho(){
  for (int x=src_x; x<src_x+src_len; x++){
    int y=src_y;
    n[x][y][0]=src_den;
    n[x][y][1]=src_den;
    n[x][y][2]=src_den;
```

```
    n[x][y][3]=src_den;
    n[x][y][4]=src_den;
    n[x][y][5]=src_den;
    n[x][y][6]=src_den;
    n[x][y][7]=src_den;
    n[x][y][8]=src_den;
  }
  for (int x=src_x_2; x<src_x_2+src_len_2; x++){
    int y=src_y_2;
    n[x][y][0]=src_den_2;
    n[x][y][1]=src_den_2;
    n[x][y][2]=src_den_2;
    n[x][y][3]=src_den_2;
    n[x][y][4]=src_den_2;
    n[x][y][5]=src_den_2;
    n[x][y][6]=src_den_2;
    n[x][y][7]=src_den_2;
    n[x][y][8]=src_den_2;
  }
}

void init(){
  for (int x=0; x<xdim; x++){
    for (int y=0; y<ydim; y++){
      if ((abs(xdim/2-x)<25)&&(abs(ydim/2-y)<25)){
        n[x][y][0]=0;
        n[x][y][1]=0;
        n[x][y][2]=0;
        n[x][y][3]=0;
        n[x][y][4]=0;
        n[x][y][5]=0;
        n[x][y][6]=0;
        n[x][y][7]=0;
        n[x][y][8]=0;
      }
      else
        {
          n[x][y][0]=0;
          n[x][y][1]=0;
          n[x][y][2]=0;
          n[x][y][3]=0;
          n[x][y][4]=0;
          n[x][y][5]=0;
          n[x][y][6]=0;
          n[x][y][7]=0;
          n[x][y][8]=0;
```

```
        }
      }
    }
  }
void initShear(){
  for (int x=0; x<xdim; x++){
    for (int y=0; y<ydim; y++){
      if (x<xdim/2){
        n[x][y][0]=20;
        n[x][y][1]=30;
        n[x][y][2]=20;
        n[x][y][3]=20;
        n[x][y][4]=10;
        n[x][y][5]=20;
        n[x][y][6]=0;
        n[x][y][7]=10;
        n[x][y][8]=0;
      }
      else {
        n[x][y][0]=0;
        n[x][y][1]=10;
        n[x][y][2]=00;
        n[x][y][3]=20;
        n[x][y][4]=10;
        n[x][y][5]=20;
        n[x][y][6]=20;
        n[x][y][7]=30;
        n[x][y][8]=20;
        }
    }
  }
}

void iterate(){
  int NI=0;
  int rate=RAND_MAX/8.;
  for (int x=0; x<xdim; x++){
    for (int y=0; y<ydim; y++){
      NI=0;
      for (int v=0; v<V; v++) NI+=n[x][y][v];
       // first implementation: random collisions
      if (NI>0){
        for (int c=0; c<C; c++){
          //        int r1=1.0*NI*rand()/RAND_MAX;
          //        int r2=1.0*NI*rand()/RAND_MAX;
          int r1=rand()%NI;
```

```
int r2=rand()%NI;
if (r1!=r2){
  int v1=0;
  for (; r1>=n[x][y][v1];v1++)
    r1-=n[x][y][v1];
  int v2=0;
  for (; r2>=n[x][y][v2];v2++)
    r2-=n[x][y][v2];
  // now we picked two particles with velocity v1 and v2.
  int v1x=v1%3-1;
  int v1y=v1/3-1;
  int v2x=v2%3-1;
  int v2y=v2/3-1;
  // x-collision
  if ((v1x==-1)&&(v2x==1)){
    v1x=0;
    v2x=0;
  }
  else if ((v1x==1)&&(v2x==-1)){
    v1x=0;
    v2x=0;
  }
  else if ((v1x==0)&&(v2x==0)){
    int r=rand();
    if (r<rate){
      if (r<rate/2){
        v1x=-1;
        v2x=1;
      }
      else {
        v1x=1;
        v2x=-1;
      }
    }
  }
  else {
    int tmp=v1x;
    v1x=v2x;
    v2x=tmp;
  }
  // y-collision
  if ((v1y==-1)&&(v2y==1)){
    v1y=0;
    v2y=0;
  }
  else if ((v1y==1)&&(v2y==-1)){
```

```
            v1y=0;
            v2y=0;
          }
          else if ((v1y==0)&&(v2y==0)){
            int r=rand();
            if (r<rate){
              if (r<rate/2){
                v1y=-1;
                v2y=1;
              }
              else {
                v1y=1;
                v2y=-1;
              }
            }
          }
          else {
            int tmp=v1y;
            v1y=v2y;
            v2y=tmp;
          }
          n[x][y][v1]--;
          n[x][y][v2]--;
          v1=(v1y+1)*3+(v1x+1);
          v2=(v2y+1)*3+(v2x+1);
          n[x][y][v1]++;
          n[x][y][v2]++;
        }
      }
    }
  }
}
// move the particles in x-direction
{
  int ntmp[ydim];
  for (int c=0; c<3; c++){
    for (int y=0; y<ydim; y++) ntmp[y]=n[xdim-1][y][c*3+2];
    for (int x=xdim-1; x>0; x--){
      for (int y=0; y<ydim; y++)
        n[x][y][c*3+2]=n[x-1][y][c*3+2];
    }
    for (int y=0; y<ydim; y++) n[0][y][c*3+2]=ntmp[y];
  }
  for (int c=0; c<3; c++){
    for (int y=0; y<ydim; y++) ntmp[y]=n[0][y][c*3];
    for (int x=0; x<xdim-1; x++){
```

```
          for (int y=0; y<ydim; y++)
            n[x][y][c*3]=n[x+1][y][c*3];
        }
        for (int y=0; y<ydim; y++) n[xdim-1][y][c*3]=ntmp[y];
      }
    }
    // move the particles in y-direction
    {
      int ntmp[xdim];
      for (int c=0; c<3; c++){
        for (int x=0; x<xdim; x++) ntmp[x]=n[x][ydim-1][c];
        for (int y=ydim-1; y>0; y--){
          for (int x=0; x<xdim; x++)
            n[x][y][c]=n[x][y-1][c];
        }
        for (int x=0; x<xdim; x++) n[x][0][c]=ntmp[x];
      }
      for (int c=0; c<3; c++){
        for (int x=0; x<xdim; x++) ntmp[x]=n[x][0][c+6];
        for (int y=0; y<ydim-1; y++){
          for (int x=0; x<xdim; x++)
            n[x][y][c+6]=n[x][y+1][c+6];
        }
        for (int x=0; x<xdim; x++) n[x][ydim-1][c+6]=ntmp[x];
      }
    }
    bounceback();
}

void getdata(){
  if (NUreq||Nreq){
    NUreq=0;
    Nreq=0;
    for (int x=0; x<xdim; x++)
      for (int y=0; y<ydim; y++){
        N[x][y]=0;
        NU[x][y][0]=0;
        NU[x][y][1]=0;
        for (int v=0; v<V; v++){
          N[x][y]+=n[x][y][v];
          NU[x][y][0]+=n[x][y][v]*(v%3-1);
          NU[x][y][1]+=n[x][y][v]*(1-v/3);
        }
      }
  }
}
```

```
void debug(){
  int NUX=0,NUY=0,NN=0;
  for (int x=0; x<xdim; x++)
    for (int y=0; y<ydim; y++){
      for (int v=0; v<V; v++){
        NN+=n[x][y][v];
        NUX+=n[x][y][v]*(v%3-1);
        NUY+=n[x][y][v]*(1-v/3);
      }
    }
  printf("N=%i; NUX=%i; NUY=%i;\n",NN,NUX,NUY);
}

void main(){
  int done=0,sstep=0,cont=0,repeat=10;
  init();
  Measure();
  DefineGraphNxN_R("N",&N[0][0],&XDIM,&YDIM,&Nreq);
  DefineGraphNxN_RxR("NU",&NU[0][0][0],&XDIM,&YDIM,&NUreq);
  DefineGraphN_R("momentum_est_x",&momentum_est_x[0],&MeasLen,NULL);
  DefineGraphN_R("momentum_est_y",&momentum_est_y[0],&MeasLen,NULL);
  DefineGraphN_R("momentum_est_x_filt",&momentum_est_x_filt[0],&MeasLen,NULL);
  DefineGraphN_R("momentum_est_y_filt",&momentum_est_y_filt[0],&MeasLen,NULL);
  StartMenu("LG",1);
  DefineFunction("init",init);
  DefineFunction("init shear",initShear);
  StartMenu("Measure",0);
  DefineInt("range_val",&range_val);
  DefineGraph(curve2d_,"Measurements");
  DefineInt("tot_vx",&tot_vx);
  DefineInt("tot_vy",&tot_vy);
  EndMenu();
  StartMenu("Wall",0);
  DefineInt("x0", &x0);
  DefineInt("x1", &x1);
  DefineInt("x2", &x2);
  DefineInt("yy0", &yy0);
  DefineInt("yy1", &yy1);
  DefineInt("yy2", &yy2);
  DefineInt("close_tube",&close_tube);
  DefineFunction("Add Wall",FindLink);
  DefineInt("link count",&linkcount);
  EndMenu();
  StartMenu("Particle Source",0);
  DefineInt("src_den",&src_den);
```

```
  DefineInt("src_x",&src_x);
  DefineInt("src_y",&src_y);
  DefineInt("src_den_2",&src_den_2);
  DefineInt("src_x_2",&src_x_2);
  DefineInt("src_y_2",&src_y_2);
  EndMenu();
  DefineGraph(contour2d_,"Graph");
  DefineInt("C", &C);
  DefineInt("repeat",&repeat);
  DefineBool("sstep",&sstep);
  DefineBool("cont",&cont);
  DefineBool("done",&done);
  EndMenu();


  while (!done){
    Events(1);
    getdata();
    DrawGraphs();
    if (cont || sstep){
      sstep=0;

      for (int i=0; i<repeat; i++) {
        iterate();
        setrho();
      }
    } else sleep(1);
  }
}
```