サンプルプログラムソースコード

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <math.h>
4   #include <GLUT/glut.h>
5
6   #define NN 10000                // size of array
7
8   #define PI 3.14159265
9   #define echarge                 // charge of an elementary electron
10  #define ep0                     // Electric permittivity of free space
11  #define emass                   // mass of a free electron
12
13  /*Declaration of variables related to electron dynamics */
14  int PARTICLE_NUM;
15  double dt,time;
16  int step,total_step;
17  double side_x,side_y,side_z,sideh_x,sideh_y,sideh_z,Efield;
18  double epsilon;
19
20  static double cd[NN];
21  static double cd_draw[NN];
22  static double vl[NN];
23  static double fc[NN];
24  static double mass[NN];
25  static double kinenergy[NN];
26
27  /***** Declaration of variables related to OpenGL *****/
28  int stop_flg = 1;
29  double eye_len=220;
30  double trans[3] = {0.0, 0.0, 0.0};
31  double angle[3] = {0.0, 0.0, 0.0};
32  int mouse_l = 0;
33  int mouse_m = 0;
34  int mouse_r = 0;
35  int mpos[2];
36  double m_matrix[16];
37  double i_matrix[16];
38  /*******************************/
39
40  /** Declaration of the variables to open the file **/
41  FILE *fp_output;
42
43  /*Initialization of electron dynamics）*/
44  void init_dynamics(void)
45  {
46     int i,j,k;
47     int ix,iy,iz;
48     double vl2_sum;
49
50     /* Device size (in units of nano mater)*/
51     side_x=      ;
52     side_y=      ;
53     side_z=      ;
54
55     /*Half of the device size ( Don't change this part.)*/
56     sideh_x=side_x*0.5;
57     sideh_y=side_y*0.5;
58     sideh_z=side_z*0.5;
```

ライブラリ

物理定数

注意）行末にセミコロン（;）は必要ない

配列指定

電子のダイナミクスに関する初期条件の設定

導体のサイズの指定（nm 単位）

```
59
60      /*Number of electrons*/
61      PARTICLE_NUM=     ;
62
63      /*time step */
64      dt=          ;
65
66      /*time initialization*/
67      step = 1;
68
69      /*total number of simulation steps*/
70      total_step =          ;
71
72      /*electric field in units of [V/m]*/
73      Efield=          ;
74
75      /*file open command */
76      fp_output=fopen("output.dat","w");
77
78      /*mass of electrons */
79      for(i = 0; i < PARTICLE_NUM; i++)
80         {
81            mass[i]=              ;
82         }
83
84      /*dielectric constant*/
85         epsilon=                ;
86
87      /*initial positions of electrons in units of [m] */
88      for (i=0;i<PARTICLE_NUM;i++)
89         {
90            cd[i*3]     = ((double)rand()/RAND_MAX-.0)*side_x*1.0e-9;
91            cd[i*3+1] = ((double)rand()/RAND_MAX-.0)*side_y*1.0e-9;
92            cd[i*3+2] = ((double)rand()/RAND_MAX-.0)*side_z*1.0e-9;
93         }
94
95      /*initial velocities of electrons in units of [m/s] */
96      for(i = 0; i < PARTICLE_NUM; i++)
97         {
98            vl[i*3]    =        ;
99            vl[i*3+1]=          ;
100           vl[i*3+2]=          ;
101        }
102
103  }
104
105
106
107  /*electron dynamics */
108  void run_dynamics(void)
109  {
110     int i,j;
111     double dis,ld,md,nd,sumvz;
112
113     time+=dt;
114
115     /* update of forces acting on electrons*/
116     double dist;
```

PARTICLE_NUM=計算に使う粒子数

dt=時間の刻み幅

*step=計算回数．すなわち時刻 time=step*dt*

Efield は導体にかける電界

データ保存用に output.dat という名前のファイルを開く（作成する）

164 行目を参照のこと

材料の有効質量はここで指定する

材料の誘電率はここで指定する

→ x
→ y
→ z
（*初期配置座標*）

0~1 の乱数を発生させる組み込み関数（このまま使う）

→ v_x
→ v_y
→ v_z
（*初期速度*）

初期条件の設定に関するサブルーチンはここまで

実際に運動方程式を解く部分
（*run_dynamics を total_step=10000 回繰り返す*）

```
117     for(i = 0; i < PARTICLE_NUM; i++)
118       {
119         fc[i*3]=0.0;
120         fc[i*3+1]=0.0;
121         fc[i*3+2]=0.0;
122         for(j = 0; j < PARTICLE_NUM; j++)
123           {
124            if(j != i)
125              {
126                dist=sqrt(     (cd[i*3]-cd[j*3])*(cd[i*3]-cd[j*3])     +     (cd[i*3+1]-cd[j*3+1])*(cd[i*3+1]-cd[j*3+1])     +
127   (cd[i*3+2]-cd[j*3+2])*(cd[i*3+2]-cd[j*3+2]) );
128                fc[i*3]+=    0.0 ;
129                fc[i*3+1]+= 0.0 ;
130                fc[i*3+2]+= 0.0 ;
131              }
132           }
133       }
134
135     /*update of velocities*/
136     for(i = 0; i < PARTICLE_NUM; i++)
137       {
138         vl[i*3]=         ;
139         vl[i*3+1]=     ;
140         vl[i*3+2]=     ;
141       }
142
143     /*update of positions */
144     for(i = 0; i < PARTICLE_NUM; i++)
145       {
146         cd[i*3] =         ;
147         cd[i*3+1] =     ;
148         cd[i*3+2] =     ;
149       }
150
151
152     /*average velocity in z-direction*/
153     sumvz=0;
154     for(i = 0; i < PARTICLE_NUM; i++)
155       {
156       sumvz+=vl[i*3+2];
157       }
158       sumvz=sumvz/PARTICLE_NUM;
159
160
161
162     /*file output*/
163     fprintf(fp_output," %e %e ¥n",time,sumvz);
164     /*printf("average velocity= %e (m/s) ¥n",sumvz);*/
165
166
167
168     /* reflections at x,y boundaries and periodic b.c. at z boundaries */
169   /*    for(i = 0; i < PARTICLE_NUM; i++)
170       {
171         if (cd[i*3+2]>side_z*1.0e-9)
172           {
173             cd[i*3+2]=    ;
174           }
```

電子間クーロン反発力の計算 → f_x → f_y → f_z

式(9)

式(10)

全電子（100 個）の z 方向の平均速度(sumvz)を計算する

時刻と全粒子の平均速度を出力させる
・fprintf はファイルに保存
・printf は画面に表示
%e: 実数型で出力、%d: 整数型で出力
¥n は改行を与える

境界条件を与える部分

周期的境界条件（z 方向）

```
175             if (cd[i*3+2]<0)
176                {
177                   cd[i*3+2]=    ;
178                }
179             if (cd[i*3]>side_x*1.0e-9)
180                {
181                   vl[i*3]=    ;
182                   cd[i*3]=    ;
183                }
184             if (cd[i*3]<0)
185                {
186                   vl[i*3]=    ;
187                   cd[i*3]=    ;
188                }
189             if (cd[i*3+1]>side_y*1.0e-9)
190                {
191                   vl[i*3+1]=    ;
192                   cd[i*3+1]=    ;
193                }
194             if (cd[i*3+1]<0)
195                {
196                   vl[i*3+1]=    ;
197                   cd[i*3+1]=    ;
198                }
199          }*/
200
201
202
203    /* time increment*/
204       step++;
205
206       if(step == total_step)
207          {
208             fclose(fp_output);    // file close.
209             exit(1);                    // exit the dynamics time roop
210          }
211
212       glutPostRedisplay();       // OpenGL command
213    }
214
215
216
217
218
219    /*ここから以下はグラフィックスに関する部分*/
220
221    void draw_box(void)
222    {
223       glDisable(GL_LIGHTING);
224       glColor3f(1.0,1.0,1.0);
225
226       glBegin(GL_LINE_LOOP);
227       glVertex3f(      0,      0,0);
228       glVertex3f(side_x,      0,0);
229       glVertex3f(side_x,side_y,0);
230       glVertex3f(0,      side_y,0);
231       glEnd();
232
```

周期的境界条件（z 方向）

鏡面反射条件（x, y 方向）

electron dynamics に関するサブルーチンはここまで

```
233     glBegin(GL_LINE_LOOP);
234     glVertex3f(     0,     0,side_z);
235     glVertex3f(side_x,     0,side_z);
236     glVertex3f(side_x,side_y,side_z);
237     glVertex3f(0,     side_y,side_z);
238     glEnd();
239
240     glBegin(GL_LINES);
241     glVertex3f(     0,     0,     0);
242     glVertex3f(     0,     0,side_z);
243     glVertex3f(side_x,     0,     0);
244     glVertex3f(side_x,     0,side_z);
245     glVertex3f(     0,side_y,     0);
246     glVertex3f(     0,side_y,side_z);
247     glVertex3f(side_x,side_y,     0);
248     glVertex3f(side_x,side_y,side_z);
249     glEnd();
250
251
252     glEnable(GL_LIGHTING);
253   }
254   void draw_dynamics(void)
255   {
256     int i;
257
258     double color_level;
259     GLfloat color[4];
260
261     glTranslated(-sideh_x,-sideh_y,-sideh_z);
262     draw_box();
263
264     for(i = 0; i < PARTICLE_NUM; i ++){
265
266
267       cd_draw[i*3]=cd[i*3]*1.0e9;
268       cd_draw[i*3+1]=cd[i*3+1]*1.0e9;
269       cd_draw[i*3+2]=cd[i*3+2]*1.0e9;
270
271
272       glPushMatrix();
273       glTranslated(cd_draw[i*3],cd_draw[i*3+1],cd_draw[i*3+2]);
274
275
276       color[0]=0;
277       color[1]=1;
278       color[2]=0;
279       color[3]=1.0;
280       glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,color);
281       glutSolidSphere(0.2, 20, 10);
282
283       glPopMatrix();
284     }
285   }
286   void mat_inv(double a[4][4])
287   {
288     int i,j,k;
289     double t, u, det;
290     int n = 3;
```

粒子の動きを描画する部分

→ *Red*
→ *Green*
→ *Blue*

粒子を緑色に光らせる部分

0.2:*粒子の大きさを設定*

```
291
292         det = 1;
293         for(k = 0; k < n; k++){
294            t = a[k][k]; det *= t;
295            for(i = 0; i < n; i++) a[k][i] /= t;
296            a[k][k] = 1 / t;
297            for(j = 0; j < n; j++)
298               if(j != k){
299                  u = a[j][k];
300                  for(i = 0; i < n; i++)
301                     if(i != k) a[j][i] -= a[k][i] * u;
302                     else       a[j][i] = -u/t;
303               }
304         }
305      }
306      void init_gl(void)
307      {
308         GLfloat light_position[] = {1.0, 1.1, 1.2, 0.0};
309
310         glShadeModel(GL_SMOOTH);
311         glLightfv(GL_LIGHT0, GL_POSITION, light_position);
312
313         glMatrixMode(GL_MODELVIEW);
314         glGetDoublev(GL_MODELVIEW_MATRIX,m_matrix);
315         glGetDoublev(GL_MODELVIEW_MATRIX,i_matrix);
316      }
317      void display(void)
318      {
319         int i,j;
320         double d0,d1,d2,d3,d4,d5;
321         GLfloat color[4];
322
323         glClearColor(0.0, 0.0, 0.0, 1.0);
324         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
325
326         glEnable(GL_DEPTH_TEST);
327         glEnable(GL_CULL_FACE);
328         glEnable(GL_LIGHTING);
329         glEnable(GL_LIGHT0);
330         glCullFace(GL_BACK);
331
332         glLoadIdentity();
333         glPushMatrix();
334
335         gluLookAt(eye_len, 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0);
336
337         glTranslated(trans[0], trans[1], trans[2]);
338
339         glPushMatrix();
340         glLoadIdentity();
341         glRotatef( angle[0],1.0,0.0,0.0);
342         glRotatef( angle[1],0.0,1.0,0.0);
343         glRotatef( angle[2],0.0,0.0,1.0);
344         glMultMatrixd(m_matrix);
345         glGetDoublev(GL_MODELVIEW_MATRIX, m_matrix);
346         glPopMatrix();
347
348         for(i = 0; i < 16; i++)
```

```
349         i_matrix[i] = m_matrix[i];
350       mat_inv((double(*)[4])i_matrix);
351
352       glMultMatrixd(m_matrix);
353
354       if(mouse_l == 1 || mouse_m == 1 || mouse_r == 1){
355         angle[0] = 0;
356         angle[1] = 0;
357         angle[2] = 0;
358       }
359
360       draw_dynamics();
361
362       glPopMatrix();
363
364       glDisable(GL_DEPTH_TEST);
365
366       glDisable(GL_LIGHT0);
367       glDisable(GL_LIGHTING);
368       glDisable(GL_CULL_FACE);
369
370       glutSwapBuffers();
371     }
372     void reshape(int w, int h)
373     {
374       int i;
375
376       glViewport(0, 0, (GLsizei)w, (GLsizei)h);
377
378       glMatrixMode(GL_PROJECTION);
379       glLoadIdentity();
380
381       gluPerspective(30.0, (double)w / (double)h, 1.0, 800.0);
382       glMatrixMode(GL_MODELVIEW);
383     }
384     void mouse(int button, int state, int x, int y)
385     {
386       switch (button) {
387       case GLUT_LEFT_BUTTON:
388         if (state == GLUT_DOWN) {
389           mpos[0] = x;
390           mpos[1] = y;
391           mouse_l = 1;
392         }
393         if (state == GLUT_UP) {
394           mouse_l = 0;
395         }
396         break;
397       case GLUT_MIDDLE_BUTTON:
398         if (state == GLUT_DOWN) {
399           mpos[0] = x;
400           mpos[1] = y;
401           mouse_m = 1;
402         }
403         if (state == GLUT_UP) {
404           mouse_m = 0;
405         }
406         break;
```

```
407         case GLUT_RIGHT_BUTTON:
408             if (state == GLUT_DOWN) {
409                 mpos[0] = x;
410                 mpos[1] = y;
411                 mouse_r = 1;
412             }
413             if (state == GLUT_UP) {
414                 mouse_r = 0;
415             }
416             break;
417         default:
418             break;
419         }
420     }
421     void motion(int x, int y)
422     {
423         double d0;
424         double len = 10;
425
426         len = eye_len;
427
428         if(mouse_l == 1 && mouse_m == 1){
429             trans[0] += (double)(y-mpos[1])*len/150;
430             angle[0] = -(double)(x-mpos[0])*0.2;
431         } else    if(mouse_m == 1 || (mouse_l == 1 && mouse_r == 1)){
432             trans[1] += (double)(x-mpos[0])*len*.001;
433             trans[2] -= (double)(y-mpos[1])*len*.001;
434         } else if(mouse_r == 1){
435             trans[0] -= (double)(y-mpos[1])*len/150;
436             angle[0] =    (double)(x-mpos[0])*0.2;
437         } else if(mouse_l == 1){
438             d0 = len/50;
439             if(d0 > 1.0) d0 = 1.0;
440             angle[1] = (double)(y-mpos[1])*d0;
441             angle[2] = (double)(x-mpos[0])*d0;
442         }
443         if(mouse_l == 1 || mouse_m == 1 || mouse_r == 1){
444             mpos[0] = x;
445             mpos[1] = y;
446             glutPostRedisplay();
447         }
448     }
449     void keyboard(unsigned char key, int x, int y)
450     {
451         if( key == 'q' || key == 'Q') exit(0);
452         if(key == 's')
453             {
454                 if(stop_flg == 1)
455                     {
456                         stop_flg = 0;
457                         glutIdleFunc(run_dynamics);
458                     }
459
460                 else if(stop_flg == 0)
461                     {
462                         stop_flg = 1;
463                         glutIdleFunc(NULL);
464                     }
```

```
465
466         }
467     }
468     int main(int argc, char** argv)
469     {
470         init_dynamics();
471         glutInit(&argc, argv);
472         glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
473         glutInitWindowSize (1000, 1000);
474         glutInitWindowPosition (400, 100);
475         glutCreateWindow (argv[0]);
476         init_gl();
477         glutDisplayFunc(display);
478         glutReshapeFunc(reshape);
479         glutMouseFunc(mouse);
480         glutMotionFunc(motion);
481         glutKeyboardFunc(keyboard);
482         glutMainLoop();
483         return 0;
484     }
485
486
```

メイン文