How To Implement a Pressure Soft Body Model

by
Maciej Matyka

maq@panoramix.ift.uni.wroc.pl

March 30, 2004

Abstract

Since a lot of people ask me about details of my Pressure Soft Body model implementation I decided to write this paper. Paper contains detailed description of how to start the simplest program for simulation of soft bodies. Details about physics behind Pressure Soft Body Model can be find in [1], which is available online[1]. There paper I will show how to organize the code and write first simple working soft body simulator.

1 General Information

(It's a working copy, English haven't been checked, mail[2] me if you will find any bugs)
We will consider an example of Soft Ball build of a two dimensional shape of material points (MP). MP are connected by a linear springs but we use connection between neighbour MP only (see figure 1). We are not using any special structural springs at all.

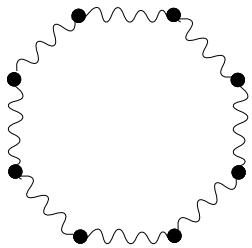

Figure 1: Simple two dimensional mesh.

It is rather obviuos that simulation of a spring-mass (SM) model presented in the figure (1) in a gravity field will be useless. A shape will collapse and nothing interesting will happens. My goal in a pressure model is to apply one additional force into force accumulator which will keep the shape of the body to be more or less similar to initial one. That means that deformation of the body is allowed, but an energy minimum will be there where a ball looks like an initial one. What we will do is to apply a pressure force (see [1] for details) to an object. It means, physically that we consider "closed shape" without any holes which has a gas inside. So, we will pump a gas into the body and calculate three forces - gravity force, linear spring forces (body shape) and pressure force which keeps a shape of the body more or less constant. Let us describe now details of an algorithm, then I will describe details of all algorithm parts with ansi c code samples. In last section you will find full c code of working two dimensional soft body simulator.

2 Data Structures

It is always important to plan how the simulation will be kept in a computer memory. For my purposes of simple simulation program I use two 'c style' structures which keeps informations about Point and Springs in a computer memory (see figure 2). Those structures keeps all needed informations about physics objects in the model.

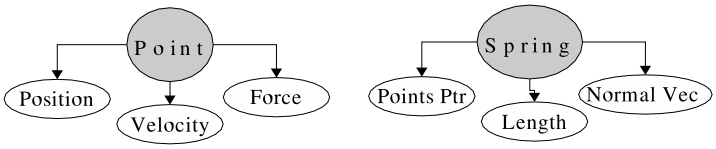

Figure 2: Structures used in c version of the code.

All mesh points will be put into a one dimensional table:

```
CPoint2d myPoints[NUMP];
```

where NUMP is a number of mesh points which will be created.

Also all springs will be put into a one dimensional table:

[1] http://panoramix.ift.uni.wroc.pl/∼maq
[2] maq@panoramix.ift.uni.wroc.pl

```
CSpring myPoints[NUMS];
```

where NUMS is equal to a number of springs ($NUMS = NUMP + 1$).

2.1 Material Point

For computer representation of a material points we will use an ansi c structure which contains general informations about the point - its position, velocity and force which act on it. Ansi C Implementation of this structure is very simple:

```
typedef struct
{
    float    x,y;        // position
    float    vx,vy;      // velocity
    float    fx,fy;      // force accumulator
} CPoint2d;
```

As we see MP structure[3] keeps information about actual position and velocity of the point. Also force accumulator have been placed here. There are vector values of course, so that is a reason why we seperated x and y values in a structure. We will rewrite all mathematical expressions seperately for x and y axis. Please note that use of c++ objects fit here very well.

2.2 Linear Spring

For a spring we use following structure:

```
typedef struct
{
    int       i,j;        // points indexes
    float     length;     // rest length
    float     nx,ny;      // normal vector
} CSpring;
```

In a CSpring structure we keep an index of first (i) and second (j) point. At a level of mesh creation we will also calculate rest length of the spring to use it in a subroutine of linear spring force calculation. Also a normal vector[4] to the spring will be calculated and kept in a CSpring structure.

[3]It is rather intuitive to use here a class hierarchy of vector - point - material point, but I decided to present c solution instead of c++.

[4]Normal vectors to springs will be needed in Pressure Force calculation subroutine.

3 Algorithm Outline

An algorithm contains 5 main steps and is exactly the same like for simple SM model. Only one difference is that we will calculate three forces instead of two which we calculate for SM. An additional Pressure Force require to calculate some additional properties of the whole body, but we will consider it later, first let us take a look on a general algorithm of our simulation program (figure 3).

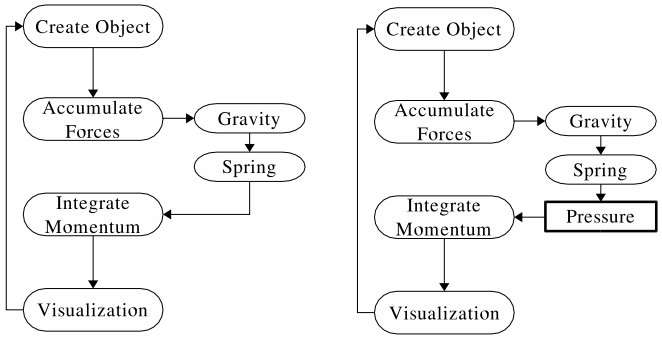

Figure 3: Pressure Soft Body Model Algorithm (right) vs simple spring-mass model (left).

To show how easy an algorithm is we present comparison with an algorithm of well known SM model (left in the figure). Only one difference of Pressure Soft Body model is that we compute one additional force (pressure force). In a first step of the algorithm we init a mesh of an object. It is up to us how the mesh will be build. For our purposes we will use a parametric representation of two dimensional circle (big words for simple things :). In a second step we calculate all forces in the model. Forces will be calculated for all points on the surface. We apply three forces on the model - one external (gravity force) and two internal object forces (linear spring force and pressure force). Then simple integration of momentum equation (second Newton Law + 1st order explicit Euler integration) will be done. After those steps we will visualize our body on the screen (GLUT will be used here) and we will back to step two, where forces are calculated...

As we can see the algorithm is rather simple and our implementation will be also as simple as possible. I decided to use simlple mesh (a 2d ball), simple integration algorithm (an Euler one) and simple visualization procedures (GL Toolkit - GLUT) to make solution as simple as possible[5].

[5]See [1] to get some ideas about enhancements in an implementation

4 Create Object

Function which creates a simulation object simply get a point (BALLRADIUS,0) and rotate it with a specified angle ($= 1/NUMP$ part of $2 * \pi$). It gives us a set of points which have to be connected by linear springs. It is simple procedure:

```
void CreateBall(void)
{
    int i;

    // create points
    for(i=1 ; i <= NUMP ; ++i)
    {
        myPoints[i].x = BALLRADIUS *
          sin(i * (2.0 * 3.14) / NUMP );
        myPoints[i].y = BALLRADIUS *
          cos(i * (2.0 * 3.14) / NUMP ) +
          SCRSIZE/2;
    }

    // create springs
    for(i=1 ; i <= NUMP ; ++i)
        AddSpring(i,i,i+1);
        AddSpring(i-1,i-1,1);
}
```

An *AddSpring* function use mySprings[] table (mentioned before) to store springs. It simply get index in a spring table (pi) and indexes of first and second spring point (those indexes are just pointers in myPoints[] table). Function calculates length of the spring and put all informations in actual (pi) place in mySprings[] table:

```
void AddSpring(int pi, int i, int j)
{
    mySprings[pi].i = i;
    mySprings[pi].j = j;
    mySprings[pi].length =
      sqrt(
       (myPoints[ i ].x - myPoints[ j ].x) *
       (myPoints[ i ].x - myPoints[ j ].x) +
       (myPoints[ i ].y - myPoints[ j ].y) *
       (myPoints[ i ].y - myPoints[ j ].y)
      );
}
```

Now an object is created and we are able to start main part of algorithm.

5 Force Accumulation

After the object has been created we start simulation loop. First we will accumulate forces which act on all particles of the soft body object. As it was mentioned before - we consider three main types of forces. Two standard SM forces - gravity and linear spring force are applied. Then third, special - pressure force is computed for every body face and distributed over particles. Let us go trough implementation of all the forces, where special effort will be made to describe an implementation of pressure force calculation subroutine.

5.1 Gravity Force

We know from simple physics that on MP which has a mass m a gravity force is equal to $\vec{F} = m \cdot \vec{g}$, where $\vec{g}$ is a gravity vector and m is MP mass. In our two dimensional space we simply update all the forces of all particles to be equal to $(0, g_y)$

```
/* gravity */
for(i=1 ; i <= NUMP ; ++i)
{
    myPoints[i].fx = 0;
    myPoints[i].fy = MASS * GY *
      (Pressure - FINAL_PRESSURE >= 0);
}
```

5.2 Spring Linear Force

After accumulation of external gravity force, we are moving into first internal body forces - linear spring force. Linearity means in that case that we derive expression directly from a Hooke's law [6] with a constant elasticity factor k_s. Additional term with additional factor k_d is a damping term.

$$\vec{F}_{12}^{s} = (|\vec{r}_1 - \vec{r}_2| - r_l) * k_s + (\vec{v}_1 - \vec{v}_2) * \left(\frac{\vec{r}_1 - \vec{r}_2}{|\vec{r}_1 - \vec{r}_2|} \right) * k_d [N] \quad (1)$$

where $\vec{r}_1$ is position of first spring point, $\vec{r}_2$ is a position of second spring point, r_l is rest length of the spring (length, where $\vec{F}_{12}^{s} = 0$) stored in the spring structure. k_s and k_d are spring and damping factors.
Implementation of linear spring accumulation procedure for one spring is straighforward. First we calculate distance between two points of the spring (start - end point) and check if it is ! = than 0. If distance is equal to 0 - we simply skip calculation, since that distance gives us non

[6] If interested - please refer to any cloth / spring mass model to find derivation of that expression

continuality in the spring force expression. Then we compute difference of velocities which is needed for damping term calculation. After that all calculated quantities are collected to compute the force value. At the end - force value is multiplied by vector "from point 1 to point 2" to get force vector and added to force accumulator of first (1) and substracted from accelerator of second (2) point. A part of accumulation procedure, responsible for linear force calculation is presented below.

```
/* loop over all springs */
for(i=1 ; i <= NUMS ; ++i)
{
  // get positions of spring start & end points
  x1 = myRelPoints[ mySprings[i].i ].x;
  y1 = myRelPoints[ mySprings[i].i ].y;
  x2 = myRelPoints[ mySprings[i].j ].x;
  y2 = myRelPoints[ mySprings[i].j ].y;

  // calculate sqr(distance)
  r12d = sqrt (
         (x1 - x2) *(x1 - x2) +
         (y1 - y2) * (y1 - y2) );

  if(r12d != 0)    // start = end?
  {

  // get velocities of start & end points
  vx12 = myRelPoints[ mySprings[i].i ].vx -
        myRelPoints[ mySprings[i].j ].vx;

  vy12 = myRelPoints[ mySprings[i].i ].vy -
        myRelPoints[ mySprings[i].j ].vy;

  // calculate force value
   f = (r12d -  mySprings[i].length) * KS +
       (vx12 * (x1 - x2) +
        vy12 * (y1 - y2)) * KD / r12d;

  // force vector
  Fx = ((x1 - x2) / r12d ) * f;
  Fy = ((y1 - y2) / r12d ) * f;

  // accumulate force for starting point
  myRelPoints[ mySprings[i].i  ].fx -= Fx;
  myRelPoints[ mySprings[i].i  ].fy -= Fy;

  // accumulate force for end point
  myRelPoints[  mySprings[i].j  ].fx += Fx;
  myRelPoints[  mySprings[i].j  ].fy += Fy;

  }

  // Calculate normal vectors to springs
  mySprings[i].nx =  (y1 - y2) / r12d;
  mySprings[i].ny =  -(x1 - x2) / r12d;
}
```

In above procedure not only spring force has been calculated. We compute also normal vectors to springs and store them in springs structures. Why we do that? It will be used further in procedure of pressure force calculation. When spring forces for all springs has been calculated we are ready to make last step of force accumulation procedure - to calculate pressure force. To this point all the procedures were some kind of standard for spring-mass models. From now I will describe something brand new - an implementation of specific type of force which appear only in pressure model of soft bodies. Please refer to [1] for details about physics background and derivation of that force.

5.3 Pressure Force

We are now ready to start calculation of pressure force. Derived governing equation:

$$\vec{P} = \frac{1}{V} \cdot A \cdot P \cdot \hat{n}[N] \tag{2}$$

where V is a body volume, A is a face field (or edge size if we talk about two dimensional case), P is a pressure value[7], $\hat{n}$ is normal vector to the face (to the edge in two dimensions)[8]

5.3.1 Volume of The Body

In the paper [1] we introduced simple technique for calculation of soft objects volume Bounding objects are simple and straighforward way to do it. You simply define x_{min}, x_{max} and y_{min}, y_{max} of all body points, the bounding box (rectangle in two dimensions) gives you first approximation of the body volume. When I had a talk on SIGRAD'03 conference in UMEA (Sweden), after my talk Jos Stam[9] told me that he know easy way to fast calculation of the volume of the body. Yeah.. almost 5 years of physics studies and I forgot about Gauss theorem - it was in my head when he told me this idea. Idea is simple, but you need to have some knowledge in academic math. If

[7] We are not interested in exactly physics value of pressure, however if you are interested, you can derive it from fundamental physics principles, use Clausius-Clapeyron equation etc.

[8] There were questions about $[N]$ in equation (2) - it is just a dimension of the force, here dimension is $[N]$ what means just Newton. Newton is a dimension of the force.

[9] Jos S. develop real time smoke dynamics simulations.

you heard about Gauss theorem - with closed shapes we are able to replace integration over volume by integration over surface of the body. In one of appendixes you will find more formal derivation of an idea of the body integration.

However for our purposes we use simply expression for body volume[10]:

$$V = \sum_{i=1}^{NUMS} \frac{1}{2} abs(x_1 - x_2) \cdot n_x \cdot dl \qquad (3)$$

where: V is body volume, $abs(x_1 - x_2)$ is an absolute difference of x component of the spring start and end points, n_x is normal vector x component and finally dl is spring length. We do a sum over all springs in the model (sum from $i = 1$ to $NUMS$), since spring is a model of an edge in that model. Implementation of that procedure is very easy and straighforward, that simple loop is presented below.

```
/* Calculate Volume of the Ball
(Gauss Theorem) */

for(i=1 ; i<=NUMS-1 ; i++)
{
  x1 = myPoints[ mySprings[i].i ].x;
  y1 = myPoints[ mySprings[i].i ].y;
  x2 = myPoints[ mySprings[i].j ].x;
  y2 = myPoints[ mySprings[i].j ].y;

  // calculate sqr(distance)
 r12d = sqrt (
         (x1 - x2) *(x1 - x2) +
         (y1 - y2) * (y1 - y2) );

  volume += 0.5 * fabs(x1 - x2) *
          fabs(mySprings[i].nx) * (r12d);

    }
```

Now we have done two steps of pressure force calculation. Normal vector calculation and volume of the body calculation. What we have left is to calculate final force and distribute it over all points in the model.

5.3.2 Pressure Force Distribution

From an equation (2) we calculate vector components of pressure force. A P scalar value is taken experimental and does not fit any real physics model. Of course there is a field for some further research to find properly values which will be relative to some real situations, but let us assume that we are interested in properly behavior of soft body, not in its physics properties.

Following part of accumulation procedure calculates value of pressure force:

```
for(i=1 ; i<=NUMS-1 ; i++)
{
  x1 = myRelPoints[ mySprings[i].i ].x;
  y1 = myRelPoints[ mySprings[i].i ].y;
  x2 = myRelPoints[ mySprings[i].j ].x;
  y2 = myRelPoints[ mySprings[i].j ].y;

  // calculate sqr(distance)
  r12d = sqrt (
         (x1 - x2) *(x1 - x2) +
         (y1 - y2) * (y1 - y2) );

  pressurev = r12d * Pressure * (1.0f/volume);

  myPoints[ mySprings[i].i ].fx +=
   mySprings[ i ].nx * pressurev;

  myPoints[ mySprings[i].i ].fy +=
   mySprings[ i ].ny * pressurev;

  myPoints[ mySprings[i].j ].fx +=
   mySprings[ i ].nx * pressurev;

  myPoints[ mySprings[i].j ].fy +=
   mySprings[ i ].ny * pressurev;
}
```

As we can see in above procedure it is very simple to calculate pressure force. A smart idea of volume of the body calculation with simple expression for pressure gives us little piece of code which have to be added to spring-mass model to get soft body behavior.

6 Integrate Newton's Equations

We finished force accumulation procedure and we are now able to start integration of equations of motion for all particles. To make the solution as simple as possible I decided to show procedure which does integration with simple Euler 1st order integrator. In appendix B you will find description of how to implement second orded semi-implicit predictor-corrector Heun integrator which allows to set bigger time steps or add more elasticity to simulated objects. However, for two dimensional simulation even first order Euler is enough to do working simulation.

[10] Actually in two dimensional case we are not talking about volume, but about figure field, but we I will call it volume, because of it relation to three dimensional soft body model.

For our purposes instead of integrating second order ODE[11]:

$$\frac{d^2\vec{r}_i}{dt^2} = \frac{\vec{F}_i}{m_i} \tag{4}$$

We use relation:

$$\frac{d\vec{r}_i}{dt} = \vec{v}_i \tag{5}$$

And integrate first order ODE:

$$\frac{d\vec{v}_i}{dt} = \frac{\vec{F}_i}{m_i} \tag{6}$$

Let us take a look on the Euler integration procedure which integrate above two equations for all $i = 1, 2, \ldots, NUMP$ points:

```
void IntegrateEuler()
{
  int i;
  float dry;

  for(i=1 ; i <= NUMP ; ++i)
  {
    /* x */
     myPoints[i].vx = myPoints[i].vx +
      ( myPoints[i].fx / MASS )* DT;
     myPoints[i].x = myPoints[i].x +
        myPoints[i].vx * DT;

     /* y */
     myPoints[i].vy = myPoints[i].vy +
                   myPoints[i].fy * DT;

     dry = myPoints[i].vy * DT;

     /* Boundaries  Y */
     if(myPoints[i].y + dry < -SCRSIZE)
     {
       dry = -SCRSIZE - myPoints[ i ].y;
       myPoints[i].vy = - 0.1 *myPoints[i].vy;
     }

     myPoints[i].y = myPoints[i].y + dry;
  }
}
```

As we can see in above code in an integration procedure we also do simple collision test with an enviroment. We just test point y component if it is less than $-SCRSIZE$ then we reflect y component of point velocity and we reflect point movement. To make solution more stable and accurate I propose you to use different kind of integrator - see Appendix B, where semi-implicit Heun predictor-corrector scheme has been described.

[11] Ordinary Differential Equation

7 Visualization

For OpenGL visualization of the two dimensional soft body I used GL_QUADS type which gave me simply way to draw filled body. Below a complete GLUT callback procedure which draw a body in a GLUT window is presented:

```
void Draw(void)
{
  int i;
  glClearColor(1,1,1,0);
  glClear(GL_COLOR_BUFFER_BIT);

  glBegin(GL_QUADS);
  for(i = 1 ; i <= NUMS-1 ; i++)
  {
    glColor3f(1,0.4,0.4);
    glVertex2f(myPoints[ mySprings[i].i ].x,
               myPoints[ mySprings[i].i ].y);
    glVertex2f(myPoints[ mySprings[i].j ].x,
               myPoints[ mySprings[i].j ].y);

    glVertex2f(myPoints[ NUMP -
                         mySprings[i].i + 1].x,
               myPoints[ NUMP -
                         mySprings[i].i + 1].y);
    glVertex2f(myPoints[ NUMP -
                         mySprings[i].j + 1].x,
               myPoints[ NUMP -
                         mySprings[i].j + 1].y);
  }
  glEnd();

  glutSwapBuffers();
}
```

As we see in above code we simply draw QUADS (i,j,NUMP-i+1,NUMP-i+1), it is done as simply as possible and of course some nice features can be added here easly (normal/velocity vectors drawing, color differences while pressure changes etc.).

8 Source Code

Complete source code has been included with that paper. Generally it bases on procedures presented in that paper. As you can see thoat code has been written as simple as possible. One additional thing which has been added is a user interaction with mouse and motion GLUT callbacks (to make whole thing more fun). Hope with that paper and included codes you will be able to write your own Soft Body simulation. Good Luck!

9 Results

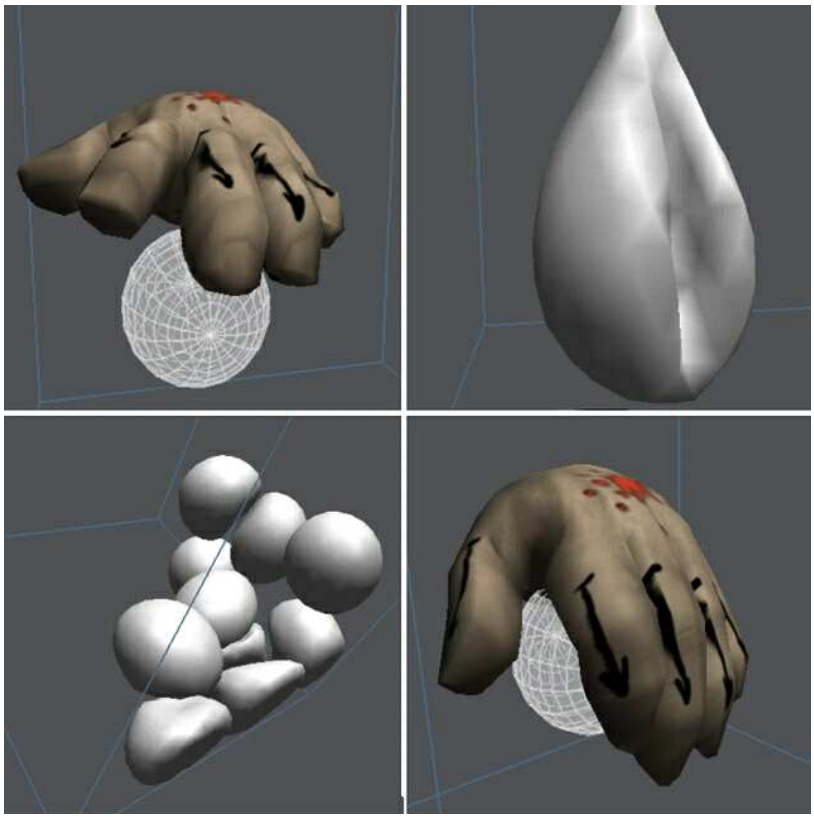

Figure 4: Three Dimensional version of Pressure Soft Body Model.

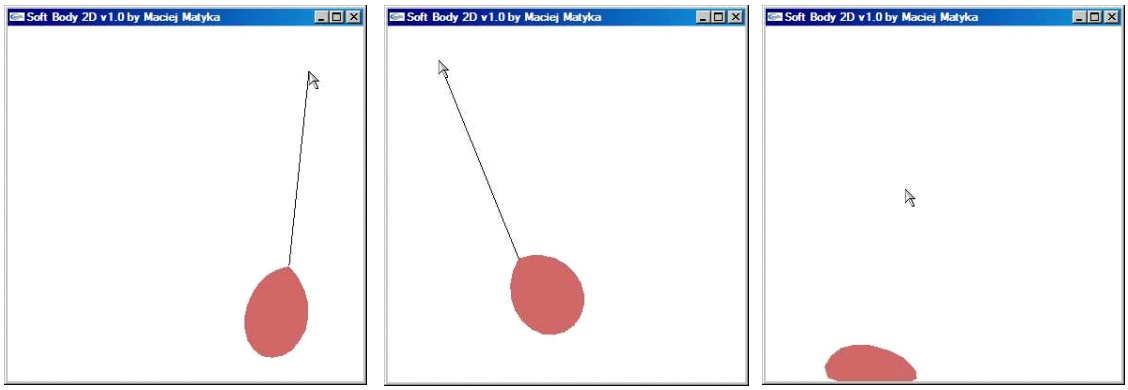

Figure 5: Result of two dimensional code from that paper. User is playing with 2d soft ball.

A Volume Calculation (Gauss Theorem)

A mathematical expression which describe Gauss theorem can be written in following form:

$$\int\int\int_V (\vec{\nabla}\cdot\vec{F})d\tau = \oint_S \vec{F}d\vec{s} \tag{7}$$

Where $\vec{F}$ is a vector field, V is whole body volume and S is body surface. Let us rewrite it in a case of two dimensional object:

$$\int\int_V (\vec{\nabla}\cdot\vec{F})d\tau = \oint_L \vec{F}d\vec{l} \tag{8}$$

Where V is body field, $\vec{F}$ is any vector field and L is body edge. If we assume $\vec{F} = (x, 0)$ then because of:

$$\vec{\nabla}\cdot\vec{F} = 1 \tag{9}$$

and

$$\vec{F}\cdot d\vec{l} = (x,0)\cdot\hat{n}\cdot dl = x\cdot n_x\cdot dl \tag{10}$$

we will get a simple expression which describe body field value (our "volume" calculated in the code):

$$V = \int\int_V dxdy = \sum_{i=1}^{NUMS} x\cdot n_x\cdot dl \tag{11}$$

B Heun Predictor - Corrector Integration

Euler integration is the simplest possible integration of motion equations and after implementing it you will find that small δt values are allowed. Problems with stiffness occurs too and it is well known disadvantage of low order schemes of ODE integration. To make solution more accurate and stable you should consider using more complex integrator. It may be an explicit scheme from Runge-Kutta family (second order Mid-Point method could be good choice) or one of unconditionaly stable implicit schemes (i.e. Backward Euler). Besides I am not a fan of implicit schemes (complex solution + huge computational cost) I propose to use Heun Predictor-Corrector scheme. It is rather easy to implement (if you understand Euler integration) and easy to understand (it is also important to know how your integrator works, not only how your integrator is implemented).

We consider problem of ODE:

$$\frac{dy}{dt} = f(y,t) \tag{12}$$

Detailed description and discussion about semi-implicit schemes you can find in great book by M. G. Ancona [2]. Heun's predictor/corrector scheme consist of two main steps, a predictor:

$$\hat{y}_{n+1} = y_n + \Delta t\cdot f(y_n, t_n) \tag{13}$$

and corrector step:

$$y_{n+1} = y_n + \frac{\Delta t}{2}\cdot[f(y_n,t_n) + f(\hat{y}_{n+1}, t_{n+1})] \tag{14}$$

Whole procedure is quite easy - first we compute forces, then a simply Euler step is done (prediction). Result of that is substituted into corrector as an argument of force computation procedure. Then we use correction central scheme to update and finalize integration. It is easy, cost us only one additional step of force computation and gives semi implicitness which result in second order accuracy of the solution.

References

[1] Matyka, M., Ollila, M. "A Pressure Model for Soft Body Simulation", Proc. of Sigrad, UMEA, 2003

[2] Ancona, M. G. "Computational Methods for Applied Science and Engineering: An interactive Approach", Rinton Press, 2002