Theory Manual for JuliaS2B code

hpjeon

November 10, 2015

Chapter 1

Introduction

In early 90's, those who had impression on SPH came to invent new numerical scheme for structural analysis. They are T. Belytschko, W. Liu, S. Jun and so on[1, 2]. For about ten years, this method which is called as meshless method or meshfree method has summoned great sensation in computational mechanics. Still now, this method is applied to structural mechanics, biomechanics, field analysis and fluid dynamics. Still development is on going. Recently, 3-dimensional explicit shear band problem by RKPM on massively parallel processors was done successfully[3]. It shows that meshfree method is being fusioned with parallel architecture like other computational methods. Also multiscale problems have been analyzed with meshfree methods. At new century, meshfree method will be one of the leading computational schemes.

I had been looking for some numerical tricks for hydrodynamics which was used in terminal ballistics. I found this meshfree method about four years ago, when a lot of research was done. From then, I have collected related papers and reports. But I have not found precisely fitting branch for my research area. So I came to implement my own in-house code. One years ago, I implemented simple 1-dimensional explicit codes which are called as Faye, Julia. Faye is a plain FEM code. Julia is based on EFG method which was suggested by Belytschko. EFG and other meshfree methods suffer from essential boundary condition treatment. Recently, I found some trick which uses singular term. I think it can be another exit. So I modified Julia as JuliaS which used singular term as mentioned and this showed stable result with respect to other previous results, Faye, Julia. Recently I implemented new stress integration scheme. So it became Julia2B. Brief history of my in-house code.

Let me comment for this code. First, this code adopts improved EFGM for shape function generations. And Bucket-Cell Integration scheme for evaluating weights of nodes. Explicit time integration is used cause this is dynamic code. Contact treatment uses Triangular Contact Check algorithm. The code is still under development. But I think I must leave documentation for developing code not to forget or confuse later. This report consists of following orders. Shape function construction which uses improve Element Free galerkin methods, nodal integration for mass and internal force term and explicit time integration scheme. Then contact and nonlinear material behavior description are following. These works will be bases for constructing the Kay and M.Zak project.

Chapter 2

Shape function

In ealy 90's, Belytschko et al[1] suggested EFGM which is based on moving least square[4]. This method is simpler than RKPM which was suggested by Liu[2] and is much used in many rooky meshfree-developers. In common, meshfree methods which are based on moving least square interpolation. Like fig. 2.1, moving least square interpolation estimates nodal point approximately, not exactly. So it suffers from essential boundary condition treatment. So many researchers have developed treatment like lagrangian multiplier, interconnection to plain FEM, and so on[1, 5]. For dynamic problem collocation method[6] or D'Alembert method has been used[7]. In late 90's, modified EFG method was introduced by Saigal and Kaljevic[8]. This method, which is called as improved EFG, uses singular term for evaluation of weigting factor. Through this method, essential boundary condition is

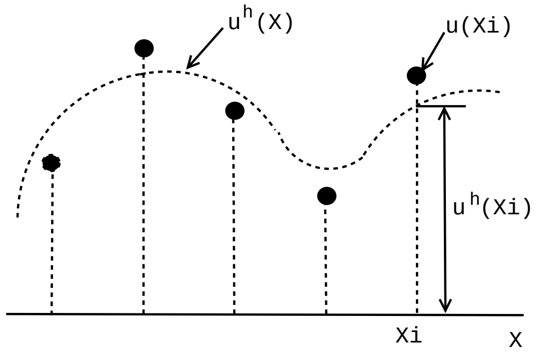

Figure 2.1: Comparison of moving least square interpolation and real data

satisfied and flow of code is same to plain FEM. There's no need to insert other routine for essential boundary condition treatment. But calculation course is much complex and requires more calculation time. Detail description is following.

In general meshless method(I mean moving least square), basis vector is

$$\tilde{p}^T(\tilde{x}) = [p^{(1)}(\tilde{x}) \quad p^{(2)}(\tilde{x}) \quad p^{(3)}(\tilde{x})] = [1 \quad x \quad y] \qquad (\mathrm{m}=3) \tag{2.1}$$

but in iEFGM(improved EFGM, you know), "$\tilde{u}$" is used rather than "$\tilde{p}$" [8]. Basis $\tilde{u}$ of iEFGM is

$$\tilde{u}^T(\tilde{x}) = [u^{(1)}(\tilde{x}) \quad u^{(2)}(\tilde{x}) \quad u^{(3)}(\tilde{x})] \tag{2.2}$$

Each component is described as following.

First, weight functions are obtained by radius concept.

$$w(\tilde{x},\tilde{x}_i) = \begin{cases} \left(\frac{r_d}{|\tilde{x}-\tilde{x}_i|}\right)^2 - 2\left(\frac{r_d}{|\tilde{x}-\tilde{x}_i|}\right) + 1 & , \quad |\tilde{x}-\tilde{x}_i| \leq r_d \\ 0 & , \quad |\tilde{x}-\tilde{x}_i| > r_d \end{cases} \tag{2.3}$$

Here, $\tilde{x}$ means arbitrary point of which we want to get some information, $\tilde{x}_i$ is descretized point like nodal point. Upper representation looks somewhat confused. So let's write like

$$w^{(i)}(\tilde{x}) = \begin{cases} \left(\frac{r_d}{r}\right)^2 - 2\left(\frac{r_d}{r}\right) + 1 & , \quad r \leq r_d \\ 0 & , \quad r > r_d \end{cases} \tag{2.4}$$

$$r = |\tilde{x}-\tilde{x}_i| = \sqrt{(x-x_i)^2 + (y-y_i)^2} \tag{2.5}$$

And derivatives of weight functions are

$$\begin{aligned} \frac{dw}{dx} = \frac{dw}{dr}\cdot\frac{dr}{dx} &= \frac{dw}{dr}\cdot(\Delta x^2 + \Delta y^2)^{-1/2} \\ &\quad \cdot(x-x_i)\cdot\mathrm{sign}(x-x_i) \\ &= \frac{dw}{dr}\frac{1}{r}|x-x_i| \end{aligned} \tag{2.6}$$

$$
\begin{aligned}
\frac{dw}{dy} = \frac{dw}{dr}\cdot\frac{dr}{dy} &= \frac{dw}{dr}\cdot(\Delta x^2+\Delta y^2)^{-1/2} \\
&\cdot(y-y_i)\cdot\mathrm{sign}(y-y_i) \\
&= \frac{dw}{dr}\frac{1}{r}|y-y_i| \qquad (2.7)
\end{aligned}
$$

$$
\frac{dw}{dr} = -\frac{2r_d^2}{r^3}+\frac{2r_d}{r^2} \qquad (2.8)
$$

For basis, term $\tilde{v}$ must be calculated and stored. And each term of basis is calculated. N is the total number of nodal points.

First component of basis vector is

$$
u^{(1)}(\tilde{x}) = \frac{1}{\sqrt{\sum_{j=1}^{N} w^{(j)}(\tilde{x})}} \qquad (2.9)
$$

$$
v^{(j)}(\tilde{x}) = \frac{w^{(j)}(\tilde{x})}{\sum_{l=1}^{N} w^{(l)}(\tilde{x})} \qquad (2.10)
$$

$$
\frac{dv^{(j)}(\tilde{x})}{dx} = \frac{\frac{dw^{(j)}}{dx}\cdot\sum_{l=1}^{N} w^{(l)}(\tilde{x}) - w^{(j)}(\tilde{x})\cdot\sum_{l=1}^{N}\frac{dw^{(l)}}{dx}}{\left\{\sum_{l=1}^{N} w^{(l)}(\tilde{x})\right\}^2} \qquad (2.11)
$$

$$
\frac{dv^{(j)}(\tilde{x})}{dy} = \frac{\frac{dw^{(j)}}{dy}\cdot\sum_{l=1}^{N} w^{(l)}(\tilde{x}) - w^{(j)}(\tilde{x})\cdot\sum_{l=1}^{N}\frac{dw^{(l)}}{dy}}{\left\{\sum_{l=1}^{N} w^{(l)}(\tilde{x})\right\}^2} \qquad (2.12)
$$

Other components of basis are

$$
u^{(2)}(\tilde{x};\tilde{x}_i) = x_i - \sum_{j=1}^{N} x_j\cdot v^{(j)}(\tilde{x}) \qquad (2.13)
$$

Its derivative can be estimated like this

$$
\frac{du^{(2)}}{dx} = -\sum_{j=1}^{N} x_j\cdot\frac{dv^{(j)}}{dx} \qquad (2.14)
$$

$$\frac{du^{(2)}}{dy} = -\sum_{j=1}^{N} x_j \cdot \frac{dv^{(j)}}{dy} \tag{2.15}$$

$$u^{(3)}(\tilde{x};\tilde{x}_i) = y_i - \sum_{j=1}^{N} y_j \cdot v^{(j)}(\tilde{x}) \tag{2.16}$$

$$\frac{du^{(3)}}{dx} = -\sum_{j=1}^{N} y_j \cdot \frac{dv^{(j)}}{dx} \tag{2.17}$$

$$\frac{du^{(3)}}{dy} = -\sum_{j=1}^{N} y_j \cdot \frac{dv^{(j)}}{dy} \tag{2.18}$$

With basis vector and weight functions, $2 \times N$ matrix B is obtained.

$$\underline{\underline{B}} = \underline{\underline{U}} \cdot \underline{\underline{W}} = \begin{bmatrix} w^{(1)}(\tilde{x})u^{(2)}(\tilde{x};\tilde{x}_1) & \cdots & w^{(N)}(\tilde{x})u^{(2)}(\tilde{x};\tilde{x}_N) \\ w^{(1)}(\tilde{x})u^{(3)}(\tilde{x};\tilde{x}_1) & \cdots & w^{(N)}(\tilde{x})u^{(3)}(\tilde{x};\tilde{x}_N) \end{bmatrix} \tag{2.19}$$

$$\frac{d}{dx}\left(\underline{\underline{B}}\right) = \begin{bmatrix} \frac{dw^{(1)}}{dx}(\tilde{x})u^{(2)} + w^{(1)}\frac{du^{(2)}}{dx} & \cdots & \cdots & \frac{dw^{(N)}}{dx}(\tilde{x})u^{(2)} + w^{(N)}\frac{du^{(2)}}{dx} \\ \frac{dw^{(1)}}{dx}(\tilde{x})u^{(3)} + w^{(1)}\frac{du^{(3)}}{dx} & \cdots & \cdots & \frac{dw^{(N)}}{dx}(\tilde{x})u^{(3)} + w^{(N)}\frac{du^{(3)}}{dx} \end{bmatrix} \tag{2.20}$$

$$\frac{d}{dy}\left(\underline{\underline{B}}\right) = \begin{bmatrix} \frac{dw^{(1)}}{dy}(\tilde{x})u^{(2)} + w^{(1)}\frac{du^{(2)}}{dy} & \cdots & \cdots & \frac{dw^{(N)}}{dy}(\tilde{x})u^{(2)} + w^{(N)}\frac{du^{(2)}}{dy} \\ \frac{dw^{(1)}}{dy}(\tilde{x})u^{(3)} + w^{(1)}\frac{du^{(3)}}{dy} & \cdots & \cdots & \frac{dw^{(N)}}{dy}(\tilde{x})u^{(3)} + w^{(N)}\frac{du^{(3)}}{dy} \end{bmatrix} \tag{2.21}$$

And $N \times N$ matrix I is a unit matrix which diagonal term is 1 and the others are all zero.

$$\underline{\underline{I}} = \begin{bmatrix} 1 & & \underline{0} \\ & \ddots & \\ \underline{0} & & 1 \end{bmatrix} \tag{2.22}$$

$N \times N$ matrix V is composed of vector $\tilde{v}$.

$$\underline{\underline{V}} = \begin{bmatrix} v^{(1)}(\tilde{x}) & v^{(2)}(\tilde{x}) & \cdots & v^{(N)}(\tilde{x}) \\ \vdots & \vdots & \ddots & \vdots \\ v^{(1)}(\tilde{x}) & v^{(2)}(\tilde{x}) & \cdots & v^{(N)}(\tilde{x}) \end{bmatrix} \tag{2.23}$$

In original equations, 2×2 matrix A is somewhat difficult to obtain. In this report, I write it simply.

$$\underline{\underline{A}} = A_{ij} = \sum_{k=1}^{N} a_{ij}^{(k)}(\tilde{x}) \qquad i, j = 1, 2 \tag{2.24}$$

$$a_{ij}^{(k)}(\tilde{x}) = \begin{cases} 0 & , \tilde{x} = \tilde{x}_k \\ u^{(i+1)}(\tilde{x}; \tilde{x}_k) \cdot w^{(k)}(\tilde{x}) \cdot u^{(j+1)}(\tilde{x}; \tilde{x}_k) & , \tilde{x} \neq \tilde{x}_k \end{cases} \tag{2.25}$$

and

$$\frac{d}{dx}\left\{\underline{\underline{A}}\right\} = \begin{bmatrix} \frac{dA_{11}}{dx} & \frac{dA_{12}}{dx} \\ \frac{dA_{21}}{dx} & \frac{dA_{22}}{dx} \end{bmatrix} \tag{2.26}$$

$$\frac{d}{dy}\left\{\underline{\underline{A}}\right\} = \begin{bmatrix} \frac{dA_{11}}{dy} & \frac{dA_{12}}{dy} \\ \frac{dA_{21}}{dy} & \frac{dA_{22}}{dy} \end{bmatrix} \tag{2.27}$$

$$\frac{da_{ij}^{(k)}}{dx} = \begin{cases} 0 & , \tilde{x} = \tilde{x}_k \\ \begin{array}{l} \frac{du^{(i+1)}}{dx} w^{(k)} u^{(j+1)} \\ + u^{(i+1)} \frac{dw^{(k)}}{dx} u^{(j+1)} \\ + u^{(i+1)} w^{(k)} \frac{du^{(j+1)}}{dx} \end{array} & , \tilde{x} \neq \tilde{x}_k \end{cases} \tag{2.28}$$

Vector $\tilde{g}$ can be obtained through $u^{(2)}, u^{(3)}$ which are calculated before.

$$\tilde{g} = \begin{Bmatrix} g^{(2)} \\ g^{(3)} \end{Bmatrix} = \begin{Bmatrix} p^{(2)}(\tilde{x}) - \sum_{j=1}^{N} p^{(2)}(\tilde{x}_j) \cdot v^{(j)}(\tilde{x}) \\ p^{(3)}(\tilde{x}) - \sum_{j=1}^{N} p^{(3)}(\tilde{x}_j) \cdot v^{(j)}(\tilde{x}) \end{Bmatrix} \tag{2.29}$$

$$\frac{d\tilde{g}}{dx} = \frac{d}{dx}\left\{ \begin{array}{c} g^{(2)} \\ g^{(3)} \end{array} \right\} = \left\{ \begin{array}{c} 1 - \sum_{j=1}^{N} p^{(2)}(\tilde{x}_j) \cdot \frac{dv^{(j)}(\tilde{x})}{dx} \\ 0 - \sum_{j=1}^{N} p^{(3)}(\tilde{x}_j) \cdot \frac{dv^{(j)}(\tilde{x})}{dx} \end{array} \right\} \tag{2.30}$$

$$\frac{d\tilde{g}}{dy} = \frac{d}{dy}\left\{ \begin{array}{c} g^{(2)} \\ g^{(3)} \end{array} \right\} = \left\{ \begin{array}{c} 0 - \sum_{j=1}^{N} p^{(2)}(\tilde{x}_j) \cdot \frac{dv^{(j)}(\tilde{x})}{dy} \\ 1 - \sum_{j=1}^{N} p^{(3)}(\tilde{x}_j) \cdot \frac{dv^{(j)}(\tilde{x})}{dy} \end{array} \right\} \tag{2.31}$$

Finally, we come to get shape functions $\tilde{N}$.

$$u^h = \tilde{N}^T \tilde{u} \tag{2.32}$$

$$\tilde{N} = \tilde{v} + \left\{ \underline{\underline{I}} - \underline{\underline{V}} \right\}^T \underline{\underline{B}}^T \underline{\underline{A}}^{-1} \tilde{g} \tag{2.33}$$

Derivatives are,

$$\begin{aligned} \frac{d\tilde{N}}{dx} &= \frac{d\tilde{v}}{dx} - \left\{ \frac{d\underline{\underline{V}}}{dx} \right\}^T \underline{\underline{B}}^T \underline{\underline{A}}^{-1} \tilde{g} + \left\{ \underline{\underline{I}} - \underline{\underline{V}} \right\}^T \\ &\quad \cdot \left\{ \left(\frac{d\underline{\underline{B}}}{dx} \right)^T \underline{\underline{A}}^{-1} \tilde{g} - \underline{\underline{B}}^T \underline{\underline{A}}^{-1} \frac{d\underline{\underline{A}}}{dx} \underline{\underline{A}}^{-1} \tilde{g} + \underline{\underline{B}}^T \underline{\underline{A}}^{-1} \frac{d\tilde{g}}{dx} \right\} \end{aligned} \tag{2.34}$$

$$\begin{aligned} \frac{d\tilde{N}}{dy} &= \frac{d\tilde{v}}{dy} - \left\{ \frac{d\underline{\underline{V}}}{dy} \right\}^T \underline{\underline{B}}^T \underline{\underline{A}}^{-1} \tilde{g} + \left\{ \underline{\underline{I}} - \underline{\underline{V}} \right\}^T \\ &\quad \cdot \left\{ \left(\frac{d\underline{\underline{B}}}{dy} \right)^T \underline{\underline{A}}^{-1} \tilde{g} - \underline{\underline{B}}^T \underline{\underline{A}}^{-1} \frac{d\underline{\underline{A}}}{dy} \underline{\underline{A}}^{-1} \tilde{g} + \underline{\underline{B}}^T \underline{\underline{A}}^{-1} \frac{d\tilde{g}}{dy} \right\} \end{aligned} \tag{2.35}$$

If number of total node is large, cpu time costs much. So it is better to memorize nodes in DOI per each node.

Chapter 3

Nodal point intergration

In most meshfree methods, integration cell methods are used which look like Gauss quadrature in plain FEM. Also support wise methods like LBIE and MLPG are used. These methods require integration points different than nodal points. In JuliaS2B code, nodal point integration scheme is used which does'nt require any other integration points. In fact, there was a trial in early time to use nodal integration scheme like SPH[9]. But that scheme was in effect only for rectangular distribution of nodes, and additional problems were continued.

The bucket-cell integration scheme, which I call, is basically based on NBNM which is suggested by Nagashima[10]. Nagashima used buckets which lie on domain regardless of nodal positions in order to evalute weight per node as fig. 3.1. In JuliaS2B, bucket shape is different. It lookes like the element of plain FEM. This is seen in fig. 3.2. For one bucket, 4-nodes are used. Weight per node in bucket is calculated as below. And by looping all buckets total weight per node is obtained. So this scheme requires cpu time less than other meshless-integration schemes. But accuracy is still not proved. This will be future work.

Distance of node i from center of bucket is

$$w(i) = \|\tilde{x}_c - \tilde{x}(i)\| \tag{3.1}$$

where $\tilde{x}_c$ is center of bucket, $\tilde{x}(i)$ is nodal position of which comprises bucket-cell. Sub-weight for node k to bucket-cell i is

$$a(i,k) = w(k) \cdot A / \sum_{j=1}^{N} w(j) \tag{3.2}$$

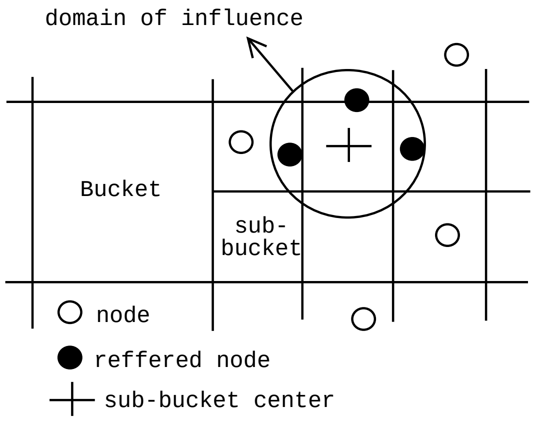

Figure 3.1: Nodal weights in NBNM by Nagashima san

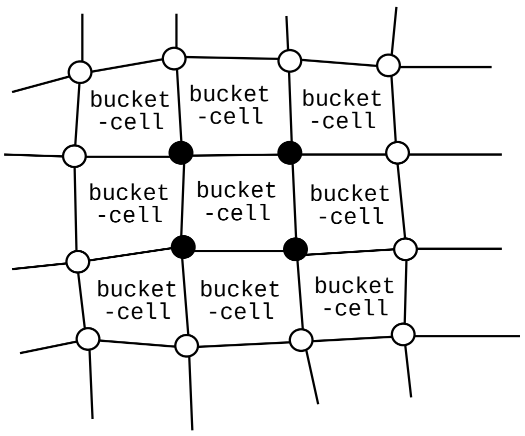

Figure 3.2: Bucket-cell integration scheme

where A is an area of bucket-cell and N is the number of nodes which comprise corresponding bucket, generally just 4. Total weight of node k is

$$W(k) = \sum_{i=1}^{M} a(i,k) \tag{3.3}$$

where M is the number of total buckets.

With these weights, we can estimate mass per node easily.

$$M_I = \int_{\Omega} \rho \phi_I d\Omega = \sum_{i}^{N} \rho \cdot \phi_I \cdot W \tag{3.4}$$

Internal force too.

Chapter 4

Explicit time integration

Like many hydrocode, JuliaS2B uses explicit time integration method, or predict-correct one. When loop starts, initial time increment is estimated. Then velocity and displacement are calculated(prediction). With obtained velocity and displacement, internal force per node is obtained which is used in calculating acceleration. After estimating accereration, velocity and displacement terms are recalculated(correction). And loop again. Examples are following. This method is also called as central difference method [11]. Overall step is like fig. 4.1.

Unlike implicit time integration, explicit one is conditionally stable. If time increment is bigger than criterion, results will diverge or process will crash. This means that stress wave propagation in domain cannot be discretized arbitrarily and is characterized by interval of integration points and stress wave velocity. Generally,

$$\Delta t \leq \frac{L_e}{c_d} \tag{4.1}$$

L_e is characteristic element dimension and c_d is current effective dilatational wave speed. In 1-dimensional problem, L_e is length of element and c_d becomes longitudinal wave speed like this.

$$c_d = \sqrt{\frac{E}{\rho}} \tag{4.2}$$

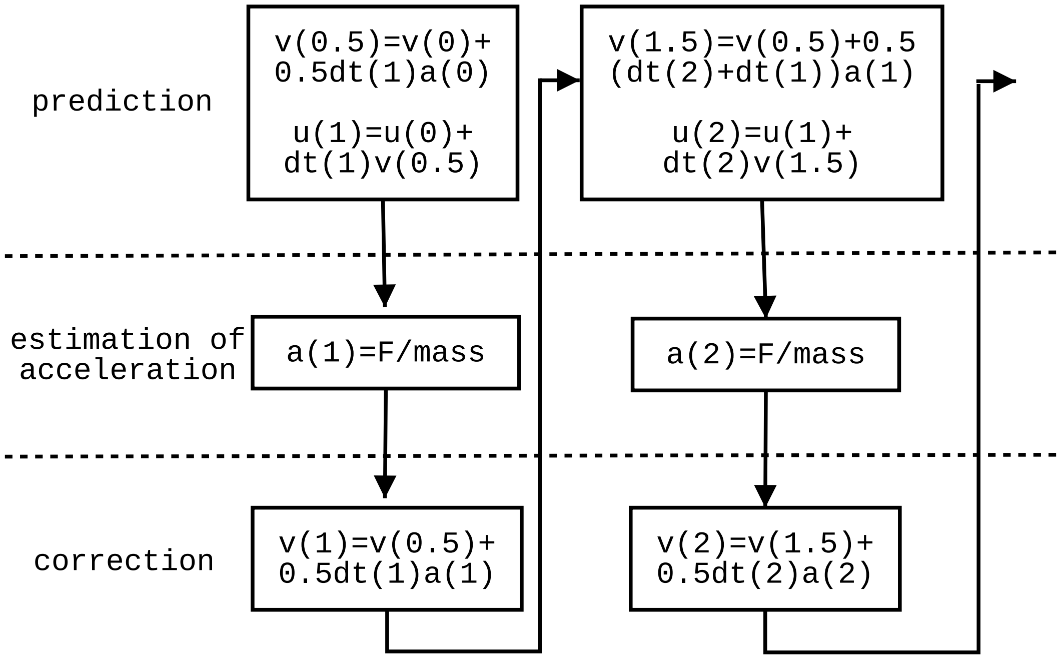
prediction
v(0.5)=v(0)+ 0.5dt(1)a(0)
u(1)=u(0)+ dt(1)v(0.5)
v(1.5)=v(0.5)+0.5 (dt(2)+dt(1))a(1)
u(2)=u(1)+ dt(2)v(1.5)
estimation of acceleration
a(1)=F/mass
a(2)=F/mass
correction
v(1)=v(0.5)+ 0.5dt(1)a(1)
v(2)=v(1.5)+ 0.5dt(2)a(2)

Figure 4.1: Step flow for central difference method

Chapter 5

Contact Treatment

In plain FEM, many contact treatments have been developed. Generally used schemes are master-slave surface contact, master surface-slave node contact, and self-surface contact algorithms. Basically these methods use faces of elements for checking contact. Unlike plain FEM, meshfree methods does'nt have any face for nodes. So there is need to establish line or surface among nodes to detect contact. Then method which is similar to master surface-slave node contact algorithm of plain FEM will be adapted.

I named Triangular Contact Check algorithm and in this TCC algorithm, contact treatment is composed of two steps. First, check for multi-contact cases for all contact sets. Then for decided pairs contact force estimation and put-back job are done. Before contact treatment, contact surfaces must be defined. Surface sets are composed of nodes on boundaries of objets. For 2-dimensional case, these surface sets are composed of primary master node and secondary master node. Like fig. 5.1, a surface is composed of node n1 and n2. Next surface is composed of n2 and n3. And so on. This counter clock-wise direction is important in defining master surface set. If direction is clock wise, normal direction of surface will be opposite. This setting will be done in parsor or pre-defined as input data. All are on code-editor's choice. There's no need to define surface sets for slave nodes which are on edges of slave objects. With these master surface sets, let's look at first step of TCC.

For a slave node, all master surface sets are checked. That is, it is checked whether slave node is located on searching area of master surface. This searching are is built between primary master node and secondary master node. To prevent some error or mischeck, searching area must be

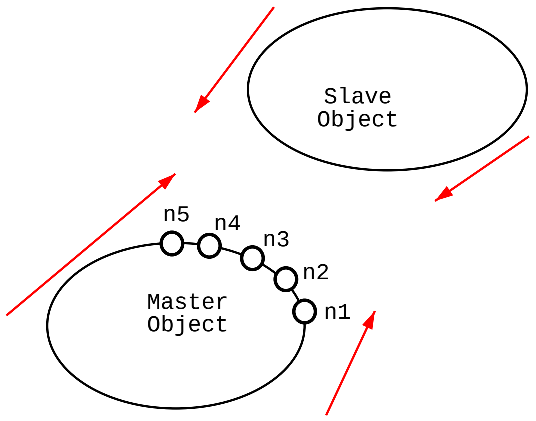

Figure 5.1: Declaration order of contact nodes

somewhat larger than intersection of two master nodes. Bias 'b' is placed on nodal positions and this make larger searching area like fig. 5.2. Generally, bias is a quarter of length of master surface. If slave node is located on searching area, area of triangle which is composed of primary/secondary master node and slave node is estimated. If this area is negative, slave node is outside of master surface as shown in fig. 5.3. But for positive, slave node penetrates master surface and then flag of that slave node is saved as '1'. Also 'NumberOfPair' is added by 1 and NPair is saved as related master surface set number. Why so complex? Because a single slave node can be located in multi-master surface sets like fig. 5.4. After whole loop, 'NumberOFPair' can be larger than '1'.

Next, let's look at second step. Again for a slave node, there can be a contact pair or contact pairs. For each pair, a triangle can be built. Angles of each master node to slave node can be estimated as 'p' and 'q' like fig. 5.5. A master surface which has minimum differece of 'p' and 'q' among all pairs is decided as a final contact pair. With this contact pair, contact treatment is done.

From now on, contact force will be estimated. With front/rear node positions, normal/tangent directions of surface can obtained. Also length of surface. At previous step, triangular area is estimated. So DOP(Depth Of Penetration) of slave node can be estimated with area/length of surface like fig. 5.6.

For the slave node contact force is estimated with DOP and Mass of

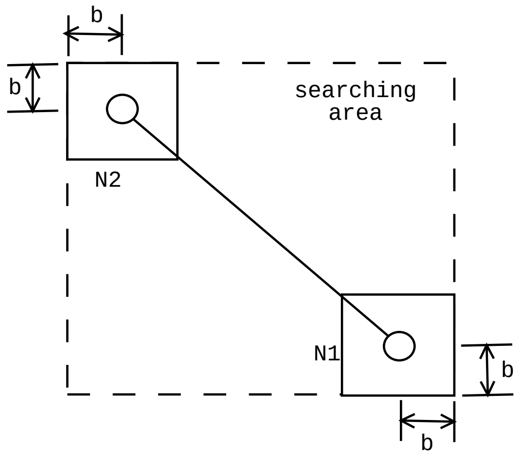

Figure 5.2: Size of search area

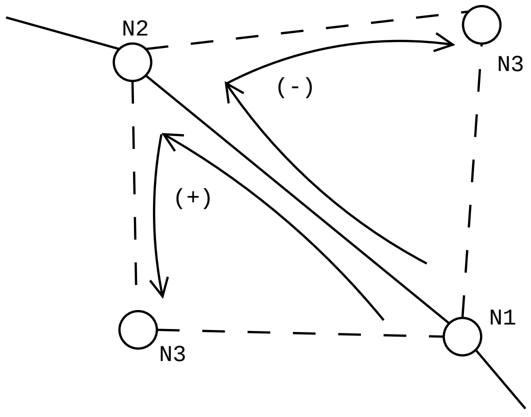

Figure 5.3: Checking penetration or not

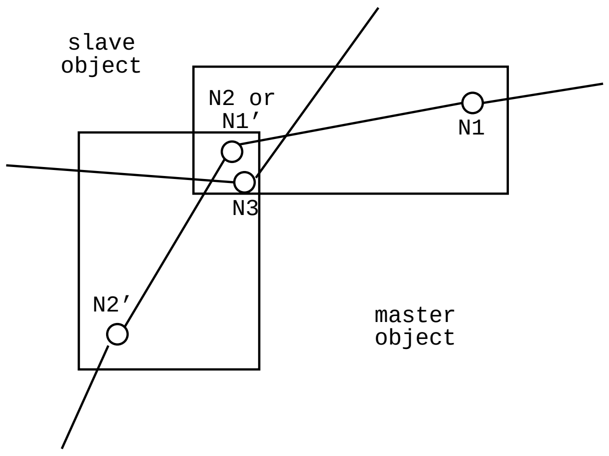

Figure 5.4: Cases that two master surfaces have same slave node

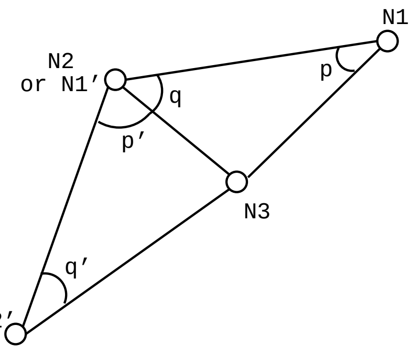

Figure 5.5: Find which set shows minimum difference

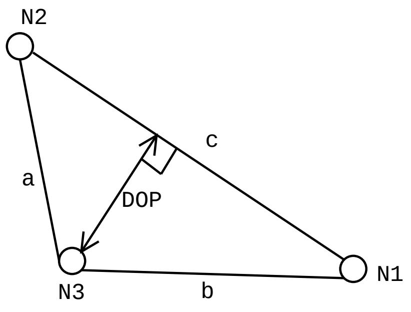

Figure 5.6: Definition of Depth Of Penetration

slave node. That is,

$$F_x^{cont}(N3) = DOP_x \cdot M(N3)/\Delta T^2 \tag{5.1}$$

$$F_y^{cont}(N3) = DOP_y \cdot M(N3)/\Delta T^2 \tag{5.2}$$

ΔT means explicit time step increment at current step. With x- or y-component of DOP coordinate of slave node is reassigned and slave node will be positioned on master surface.For Then slave contact force is redistributed on contacting master nodes according to geometrical property. That is, if position of put-backed slave node is close to a master node, this master node will have much portion of contact force. Distribution of master contact forces is just an interpolation.

Chapter 6

Constitutive Model

In meshfree methods, shape function estimation costs much calculation time. So total largrangian method which refers initial configuration is preferred than updated lagrangian one. So weak form equations must be rewritten with refer to initial configuration. In non-linear FEM, many methods have been revised to relate initial state variables with current state ones. Also objectivity concept is used to verify configuration changes. For the standard equation of motion,

$$\underline{\underline{M}}\ddot{\tilde{u}} + \tilde{f}^{int} = \tilde{f}^{ext} \tag{6.1}$$

In total lagrangian method, each term can be written as

$$\tilde{f}_I^{int} = \int_{\Omega_x} P_{Ji} \frac{\partial N_I}{\partial X_J} \tilde{e}_i d\Omega \tag{6.2}$$

$$\tilde{f}_I^{ext} = \int_{\Gamma_X^T} T_i^0(X,t) N_I(X) \tilde{e}_i d\Gamma + \int_{\Omega_x} B_i(X,t) N_I(X) \tilde{e}_i d\Omega \tag{6.3}$$

While $\underline{\underline{M}}$ is mass matrix, $\tilde{u}$ is displacement of node and P is the first Piola-Kichhoff stress. So constitutive models which connect displacement and velocity of nodes stress terms are needed. As for conventianal FEM, there are so many constitutive models upon various material behaviors. Here, moony-rivln type is used on hyperelasticity and tangent stiffness methods on visco-plasticity. All constitutive models are treated as simple as possible. Additionally, Julia2D uses singular shape functions. So there is no need to update essential boundary conditions at every time step like conventional meshfree methods.

6.1 Hyperelasticity

Like rubber, material which shows large ratation and deformation in elastic range is considered as hyperelastic material. Estimation procidure is followed. With predicted displacements deformation gradients are estimated.

$$\tilde{F} = \frac{\partial \tilde{x}}{\partial \tilde{X}} = \tilde{1} + \frac{\partial \tilde{u}}{\partial \tilde{X}} \tag{6.4}$$

With deformation gradient, right Cauchy tensor is obtained.

$$\tilde{C} = \tilde{F}^T \cdot \tilde{F} \tag{6.5}$$

And invariant trio are estimated as following.

$$I_1 = \mathrm{tr}(\tilde{C}) \tag{6.6}$$

$$I_2 = \frac{1}{2}\left[\left(\mathrm{tr}(\tilde{C})\right)^2 - \mathrm{tr}(\tilde{C}^2)\right] \tag{6.7}$$

$$I_3 = \det(\tilde{C}) \tag{6.8}$$

Now, 2nd Piola-Kirchhoff stress tensor is right before estimation.

$$\tilde{S} = 2\left[(C_1 + C_2 I_1)\tilde{1} - C_2\tilde{C} - \left(C_1 I_3^{1/3} + 2C_2 I_3^{2/3} - \lambda \ln I_3\right)\tilde{C}^{-1}\right] \tag{6.9}$$

1st Piola-Kirchhof stress tensor is estimated through simple transformation rule.

$$\tilde{P} = \tilde{S} \cdot \tilde{F}^T \tag{6.10}$$

Bibliography

[1] T. Belytschko, Y. Y. Lu and L. Gu, "Element Free Galerkin Methods," *International Journal for Numerical Methods in Engineering*, **37**, pp. 229-256(1994)

[2] W. K. Liu, S. Jun and Y. F. Zhang, "Reproducing Kernel Particle Methods," *International Journal for Numerical Methods in Fluids*, **20**, pp. 1081-1106(1995)

[3] K. T. Danielson, S. Hao, W. K. Liu, R. Aziz Uras and S. Li, "Parallel computation of meshless methods for explicit dynamic analys," *International Journal for Numerical Methods in Engineering*, **47**, pp. 1323-1341(2000)

[4] P. Lancaster, K. Salkauskas, "Surfaces generated by moving least squares method," *Mathematics of Computation*, **37**, pp. 141-158(1981)

[5] T. Belytschko, P. Krysl and Y. Krongauz, "A three-dimensional explicit Element-Free Galerkin method," *International Journal for Numerical Methods in Fluids*, **24**, pp. 1253-1270(1997)

[6] S. Jun, W. K. Liu and T. Belytschko, "Explicit Reproducing Kernel Particle Methods for large deformation problems," *International Journal for Numerical Methods in Engineering*, **41**, pp. 137-166(1998)

[7] F. Günther, W. K. Liu, D. Diachin and M. A. Christon, "Muti-scale meshfree parallel computations for viscous compressible flows," *Computer Methods in Applied Mechanics and Engineering*, **190**, pp. 279-303(2000)

[8] I. Kaljevic and S. Saigal, "An improved Element Free Galerkin formulation," *International Journal for Numerical Methods in Engineering*, **40**, pp. 2953-2974(1997)

[9] S. Beissel and T. Belytschko, “Nodal integration of the Element-Free Galerkin method,” *Computer Methods in Applied Mechanics and Engineering*, **139**, pp. 49-74(1996)

[10] T. Nagashima, “Node-by-node meshless approach and its applications to structural analyses,” *International Journal for Numerical Methods in Engineering*, **46**, pp. 341-385(1999)

[11] Hibbitt and Karlsson and Sorenson Inc., ABAQUS Theory Manual, V5.5(1995)