An alternative formulation of the FE method for arbitrary discrete/continuous models

Wlodek Abramowicz, Impact Design Europe
Ul. 3 MAJA 18 •
MICHALOWICE 05-816 • POLAND
E-mail: wa@impactdesign.pl

Summary

The paper describes new Object Oriented formulation of the FE algorithm that encompasses traditional FE elements, Super Elements, experimental data, codes of practice and rigid body mechanics in a single calculation environment. The new formulation is implemented in software for dynamic crash simulation of an arbitrary 3D frame structure discretized into Superbeam Elements and subjected to large dynamic crash loadings. The paper presents basics of the general algorithm and element formulation. The theoretical part is followed by two examples of application of the new code to real word structures.

Introduction

The crashworthiness community entered the third millennium with well-established tools for numerical simulation of crash events at various levels of modeling precision. The fully-grown FE codes for dynamic crash simulation provide for an excellent tool for detailed dynamic analysis of crash events. On the other hand simulation techniques based on analytical and semi-analytical solutions, macro element approach as well as on detailed experimental databases provide for a rapid simulation support at early design/analysis level when building of a detailed FE mesh is not feasible. An apparent advantage of the latter tools is the easy of modeling and negligible processing time; the major drawback is the limited area of application and dedicated modeling, processing and post processing techniques.

The major problem of existing simulation environments is the lack of compatibility between simulations tools used at various levels of the design/calculation loop. The difficulty of combining various discretization/solution techniques into single software is not only a technical problem of different input/output organization. The primary problem lies deep in the generic formulation of the FE and semi-analytical or macro element methods. The calculation algorithms of these methods are frequently inherently different. Therefore, it is very difficult, if not impossible, to implement both approaches in a traditional FE program based almost exclusively on the matrix technique.

This paper presents an alternative Object Oriented Formulation (OOF) for arbitrary discrete/continuous models of mechanical systems. The OOF strictly separates the level of an individual element from the global dynamic equilibrium problem of the entire model (resulting from the space semi-discretization) and time integration schemes (time semi-discretization). The OOF makes it possible to implement various ‘elements’ in a single computer code (e.g. macro elements, traditional FE elements, rigid bodies, code of practice, experimental characteristics etc.).

Preliminaries

Most of consistent formulations of the FE methods in nonlinear structural mechanics are based on the principle of virtual work (or virtual velocities)[1] [1]-[4].

Eq. 1 $$\delta W(\delta u, u) \equiv \delta W^{\text{int}} - \delta W^{ext} + \delta W^{kin} = 0,$$

where arguments δu *and* u correspond, respectively, to the kinematically admissible test and trial functions. The consistent procedure of FE space semi-discretization, based on Eq.1, reduces the problem of dynamic equilibrium of continuum to the problem of dynamic equilibrium of a discrete system, which is then solved by using one of the numerous time-stepping routines. At this level of generality momentum equations, resulting from Eq. 1, can be categorized into two groups that differ in the representation of internal nodal forces and are referred to as 'stiffness' and 'force' method, respectively[2].

Eq. 2 $$M\,\ddot{u} + K\,u = f^{ext} \qquad \text{(a)}$$

$$M\,\ddot{u} = f^{ext} - f^{\text{int}} \qquad \text{(b)}$$

In Eq. 2 M, K *and* u denote, respectively, the mass and stiffness matrices, while u is the displacement vector. The internal and external nodal forces are denoted by f^{in} *and* f^{ext}, respectively. For the sake of conciseness dumping forces are not listed explicitly in neither of momentum equations. In Eq. 2a the internal forces vary linearly with the displacement vector at each discrete time interval and/or iteration step. Likewise the stiffness matrix K (in general non linear) is kept constant at each iteration step.

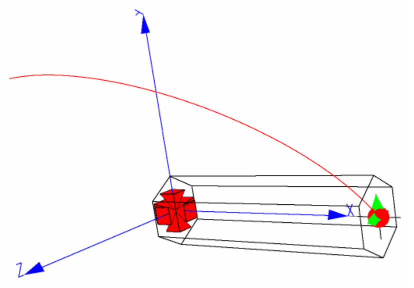


Figure 1 Illustrative example: three-dimensional rod, initially along x axis, is fixed at the start node and subjected to kinematic loading at the end node. The space trajectory of the loaded end corresponds to finite straining and finite rotations of the rod.

The formulation involving explicit definition of the stiffness matrix was a milestone in the development of modern FE codes for structural mechanics problems. Virtually all quasi-static and dynamic formulations of FEM for small and moderate deformations are based on this approach. The 'stiffness method', however, is not efficient in the simulation of crash events, which involve both finite deformations as well as finite rotations of contributing elements. The major problem lies in the complexity, ill conditioning and singularities of the stiffness matrix that are ever

[1] The principle of stationary potential energy is a special case of the principle of virtual work applicable to potential systems.

[2] The following nomenclature is adopted: $\mathrm{r}_{\text{index}}^{\text{tag}}$ the superscript is a 'tag' that identifies property of a given symbol (e.g. external, internal). The superscript stands for an index, which identifies independent variable.

present in the case of large rotations and strain-softening response of contributing elements, [6].
The complexity of the stiffness matrix method is illustrated on the elementary example of a simple elastic rod element pin-ended at one node and subjected to arbitrary kinematic loading at the other node, Figure 1. In the standard FE formulation the 2D element stiffness matrix, $K(t_n)$, of elastic rod at nth time step is given as (see e.g.[3]):

Eq. 3
$$K(t_n) = \underbrace{\frac{AE}{L}\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}}_{[K]} + \underbrace{\frac{AE}{L^2}\begin{bmatrix}3u_x & u_y\\ u_y & u_x\end{bmatrix}}_{[N_1]} + \underbrace{\frac{AE}{2L^3}\begin{bmatrix}3u_x^{\ 2}+u_y^{\ 2} & 2u_xu_y\\ 2u_xu_y & u_x^{\ 2}+3u_y^{\ 2}\end{bmatrix}}_{[N_2]}$$

where A is the cross-sectional area of the rod, L is initial length, $u_x\ and\ u_y$ are components of the displacement vector, u, in the total Lagrange description and E is the Young modulus. Commonly the three matrices in Eq. 3 are referred to as infinitesimal displacements stiffness matrix, K, and linear and quadratic contribution matrices, $[N_1]\ and\ [N_2]$, respectively. The stiffness matrix in Eq. 3 is worked out for moderate nominal strains, $\varepsilon_x\ (\varepsilon_x^{\ 2} << 1)$ and arbitrary rotations. This matrix can be further generalized to the case of large strains by using the basic relations between components of Green-Lagrange, e_x, and nominal strain, ε_x, tensors, [5].

Eq. 4
$$\varepsilon_x = \sqrt{1-2e_x} - 1 \quad and \quad \varepsilon_x = \frac{e_x}{\varepsilon_x + \frac{1}{2}\varepsilon_x^{\ 2}} \quad where \quad e_x = \frac{u_x}{L} + \frac{1}{2}\left(\frac{u_x^{\ 2}}{L^2} + \frac{u_y^{\ 2}}{L^2} + \frac{u_z^{\ 2}}{L^2}\right)$$

The resulting form of the stiffness matrix is then

Eq. 5
$$K(t_n) = \frac{1}{(1+\frac{1}{2}\varepsilon_x)\sqrt{1+2e_x}}\frac{AE}{L}\begin{bmatrix}1+\frac{3}{2}u_x+\frac{1}{2}u_x^{\ 2} & \frac{1}{2}u_y+\frac{1}{2}u_xu_y & \frac{1}{2}u_z+\frac{1}{2}u_xu_z\\ u_y+\frac{1}{2}u_xu_y & \frac{1}{2}u_y^{\ 2}+\frac{1}{2}u_z^{\ 2} & 0\\ u_z+\frac{1}{2}u_xu_z & 0 & \frac{1}{2}u_z^{\ 2}+\frac{1}{2}u_y^{\ 2}\end{bmatrix}$$

The matrix in Eq. 5 is singular, ill conditioned for large rotations and CPU time expensive. The complex form of the matrix results from the derivation technique involving directly constitutive relations of the rod material (in this case simple elastic material). The resulting definition of the stiffness matrix is also an example of a poor design of computer code based on the Object Oriented Programming paradigm as it involvers, at one programming level, all detailed information on the material, the mathematical model of the element and contribution of the element to the response of entire computational model. This subject will be further discussed in the next section.
The stiffness matrix in Eq. 5 is given for the total Lagrangian formulation. Significant simplification of the resulting expression is obtained for co-rotational formulation when the axial force in the rod, $P(t_n)$, is defined in the current rather then initial configuration. The result is, [5]:

Eq. 6
$$K(t_n) = \frac{P(t_n)}{L(t_n)}\begin{bmatrix}\frac{(L+u_x)}{u_x} & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

Although the programming structure of the matrix in Eq. 6 is much better it is still singular at $u_x = 0$ and requires separate routine for this particular configuration. The above example illustrates only a small fraction of the implementation problems originating from the stiffness matrix formulation. Therefore application of the 'stiffness method' to the crash simulation is marginal.

Most of the difficulties encountered in the 'stiffness matrix method' disappear when internal nodal forces are kept constant at each iteration step, Eq. 2b. In this case local and global stiffness matrices are obsolete. Furthermore, in the case of diagonal form of the mass matrix, M, in Eq. 2b the governing momentum equations are decoupled and solution to the problem is obtained without solving any algebraic equations[3]. This simple and at the same time ingenious idea is due to Belytschko who formulated the corresponding theory in early 1970s. Belytschko's method is now referred to as the element-by-element technique. This technique linked with the central difference method forms the basis of most commercially successful codes for crash simulation.
In FE codes solution to the momentum equations, Eq. 2b, is build and solved following the standard discretize/approximate/iterate paradigm. This method allows for effective implementation of standardized solution routines involving large number of finite elements of the same class, e.g. 10^5 elements. At the same time the standardization of the matrix method makes it difficult to implement solutions to more complex elements that require dedicated subroutines or subprograms to compute internal nodal forces at every iteration step (e.g. Super Elements). Furthermore, the global formulation given by Eq. 2b is still not convenient for OOP implementation. Generalization of the element-by–element technique to the element-by–element/node-by-node method is briefly outlined in the next section.

The node-by-node/element-by-element method

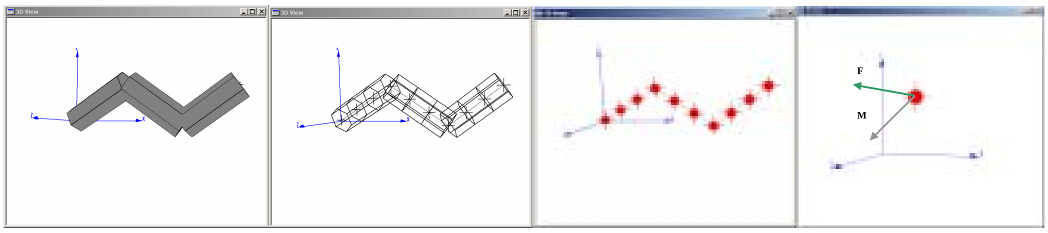
Object Oriented Programming technique is a relatively new approach to the implementation of complex models. Despite of the widespread of OO programming techniques in the computer information industry they have received relatively little attention in implementation of algorithm relevant to structural mechanic. Fundamental attribute of the OOP is the possibility of implementation of ideas or concepts (notions) in separate programming entities referred to as objects[4]. Generally, an object embodies both specific data and the functions that manipulate it, [7]. The above general definition of an object is rather vague, however, in the area of structural engineering definition of objects such as 'vector', 'tensor', 'coordinate system' or 'node' is intuitive and quite obvious to an engineer. In the OOF it is important to realize *what a given method does* (or how it works in general terms). Once this objective is achieved the precise definition of *how it does it* at various levels of generality (referred to as levels of abstraction in OOP jargon) is a relatively easy task. Accordingly the exposition of the present formulation starts from the most general ideas (high level of abstraction) and concludes at precise definition of most important objects. The exposition of the method is limited to the 'force method', Eq. 2b, and

[3] Conditional stability of the discussed explicit integration method constitutes a severe limitation, which can be devastating to the simulation results in the case of time increments larger then the critical time step.
[4] In OOP stringent distinction is made between 'class' and 'object' concepts. Class is a segment of code that defines properties and behaviors of objects. Instant(s) of a class created in computer memory is referred to as object(s). This distinction is immaterial for the present descriptive presentation of the method.

explicit integration scheme based on the central difference algorithm. The generalization to the case of 'stiffness method' and other time stepping routines is a subject of forthcoming publications.

Basic paradigm of the FE method

The leading idea of the FE formulation is the partition of a structure (or solid) into 'elements' of finite size; as shown schematically in Figure 2a and b.



a). b). c). d).

Figure 2 Schematic illustration of the basic paradigm of the Finite Element Method.

The dynamic response of each Element is then 'granulated' at discrete points referred to as 'nodes', Figure 2c. The forces defined at the element's nodes must guarantee the internal equilibrium of an element. The inertia properties of each element are represented by 'lumped' masses or continuous inertia (inertia tensor) distributed at element's nodes. At this point the dynamic equilibrium (discrete momentum equations) of the discretized model is represented by a set of nodes subjected to the action of a vector sum of forces exerted by elements connected to a given node and discrete external forces applied to the node itself, Figure 2d. In the explicit scheme both forces exerted at a node as well as inertia properties are kept constant at each time step. Consequently, at all iteration steps the problem is reduced to the dynamics of a set of uncoupled material points (rigid bodies). Integration of corresponding momentum equations is done in a node-by-node manner and renders incremental displacements and velocities of each node. These kinematic quantities are used, in turn, to update the nodal forces and inertia properties of corresponding elements. The update procedure is done element-by-element and involves elements only. The basic outcomes of the above general description of the FE method are:

1. Two basic types of objects: *Nodes* and *Elements* are used to define the FE model.
2. Two types of operations: time integration and update of elements is performed in turn on separate sets of nodes and elements, respectively, at each iteration step.
3. The exchange of information between nodes and elements is done following each of the above operations.

The second conclusion is especially important for the further development of the present method. It indicates that operations on the node set (incremental solution to momentum equations) can be effectively separated from the operations on the set of elements (elements update). In other words the global equilibrium algorithm can be implemented without direct reference to the specific type of an element. This in turn guarantees an open architecture of the code: the developer or the user of the code can define various elements without a direct interaction with the global equilibrium algorithm and adopted incremental solution procedure.

Further on the first conclusion shows that the whole algorithm can be implemented without 'global matrix' of any kind. In fact there is no need to define any matrix in the present formulation. Finally, a number of time integration routines can be defined on the set of nodes without direct manipulation of the node's code.

The node and the interface objects

The broad notion of the node object introduced in the previous section is not precise enough for detailed description in terms of computational mechanics. Closer examination of the functionality of a node shows that 'nodes' defined on elements and 'nodes' at which various elements meet together constitute two different objects. This is explained on the example of two beam elements meeting at a single node, Figure 3.

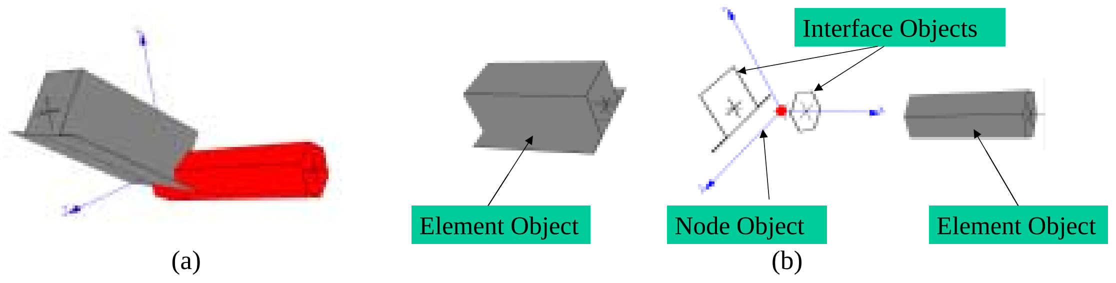


Figure 3 Connection of two beam elements at a single node (a). The concept of the interface objects (b).

Commonly the cross-sectional forces in a beam are given at the centroid. Resulting forces and moments are defined by summing contributions parallel and perpendicular to the cross-sectional plane (e.g. axial and shear forces or bending and torsion moments). The object capable of handling this functionality must have an orientation in space. Such an object is referred to as 'the interface' or 'the interface object'. Orientation of the interface is defined by the triplet of unit base vectors $\{\vec{\alpha},\vec{\beta},\vec{\gamma}\}$, two perpendicular unit base vectors $\vec{\alpha}$ *and* $\vec{\beta}$ define the cross-sectional plane while the third vector $\vec{\gamma}=\vec{\alpha}\times\vec{\beta}$ defines normal to the plane. Three coordinates of each base vector define position of that vector in the node coordinate system. The orientation object implements the idea of the orthogonal Cartesian reference system in the corotational formulation of classical continuum mechanics.

Similarly like in the case of interface spatial position and orientation of a node is defined by a separate orientation object, $\{\vec{i},\vec{j},\vec{k}\}$, where coordinates of consecutive unit base vectors are given in the global (reference) coordinate system. Interfaces of all elements meeting at a given node are rigidly attached to the node throughout the entire calculation process so that motion of the node uniquely defines corresponding motions of all interfaces connected to that node.

Transformation of vectors between node and interface

Transformation of vectors defined in the node or interface frames is given by simple multiplication of vectors by scalars or projections of a vector onto the base vectors. The resulting operations are analogous to standard transformations defined by means of second order tensors. For example if $v[v_1,v_2,v_3]$ is the representation of a vector in the interface frame its representation $\tilde{v}$ in the node frame is defined by the *scatter* function:

Eq. 7
$$\tilde{v} = :: scater(v, node) = v_1\vec{\alpha}+v_2\vec{\beta}+v_3\vec{\gamma} = \\ (v_1\alpha_1+v_2\beta_1+v_3\gamma_1)\vec{i}+(v_1\alpha_2+v_2\beta_2+v_3\gamma_2)\vec{j}+(v_1\alpha_3+v_2\beta_3+v_3\gamma_3)\vec{k}$$

The ‘scope resolution operator’, ::, in front of the function name emphasizes that syntax of the function call in Eq. 7 is exactly the same as in an actual source code (in C++ programming language). The function *scatter* takes two arguments: the vector and pointer to the orientation object to which the vector must be transformed. This simple example illustrates how easy functions defined on various objects are constructed in OO languages. On the contrary if vector $\tilde{w}[\tilde{w}_1, \tilde{w}_2, \tilde{w}_3]$ is defined in the node frame the representation, w, in the interface frame is given by the *gather* function:

Eq. 8 $$w \equiv :: gather(\tilde{w}, \text{interface}) = (\tilde{w} \cdot \vec{i})\vec{i} + (\tilde{w} \cdot \vec{j})\vec{j} + (\tilde{w} \cdot \vec{k})\vec{k}$$

Momentum equations of a node object

The reduction of interface(s) forces and moments to the node frame is done in a usual way and results in the well-known equations of momentum and moment of momentum [8] [9]:

Eq. 9 $$\begin{aligned} \dot{K}^n &= M^n - m\rho_c \times \dot{v}^o \\ m\dot{v}^o &= F - m\dot{\omega}^n \times \rho_c - m\omega^n \times (\omega^n \times \rho_c) \end{aligned}.$$

In Eq. 9 F *and* M are the resulting forces and moments exerted on a node by connected elements and applied loadings, m is the total resulting mass, ω is the instantaneous angular velocity of the node, K stands for the moment of momentum and ρ_c is the position vector of the center of gravity of the node. The superscript ‘o’ indicates origin of the node frame while superscript ‘n’ indicates that given quantity is defined in the node coordinate system. If origin of the node coincides with the center of gravity Eq. 9 are reduced to the familiar Newton-Euler form

Eq. 10 $$\begin{aligned} m\dot{v}_c &= F \\ J^n\dot{\omega}^n &= M^n - \omega^n \times J^n\omega^n \end{aligned}$$

where J^n is the tensor of inertia and $\dot{v}_c$ is the acceleration of the center of gravity of a node in the global frame. It should be emphasized that in the Newton-Euler equations dynamics of translatory motion is given in global frame while the rotary motion is described in node (local) frame. In addition ω is a pseudo coordinate and in general its integral does not have physical meaning.

Rotations and translations of the node object

Calculation of finite rotations constitutes perhaps the major difficulty in implementation of codes for large rotations/large strains models. The difficulty stems form the fact that finite rotation around a stationary spatial axis is inherently a four-parameter object (quaternion), which in the classic matrix formulation is represented by a three-parameter set like Euler angles or parameters, Bryant/Cardan angles etc. Such a three-parameter representation has always singularities at discrete points that lead to computational problems. On the other hand, velocities of rotation or incremental rotations are true vector quantities that fit well into a three-parameter matrix algorithm. The problem is easily solved in the OOF, which takes advantage of a strict separation of algorithms for node’s incremental mechanics, described by vector quantities, and spatial update of node’s configuration based on the exact Euler vector formula for finite rotations. The details of the update procedure are as follows: the motion of a node is computed incrementally. After each iteration step the following quantities are given: position of the node origin, r_{n-1}, at time step $n-1$ the corresponding increment of the position vector, Δr, and an average angular velocity of node ω_n at a given time increment. Computation of the position vector is trivial

Eq. 11 $$r_n = r_{n-1} + \Delta r\,.$$

Update of the orientation is done as follows: the angular velocity vector ω_n transformed to the global frame defines the (unit) vector of an instantaneous axis of rotation, a, of the node. The incremental rotation angle around this axis is $\Delta\theta_n = \omega_n\ \Delta t$. So that the incremental rotations of node base vectors $\{\vec{i},\vec{j},\vec{k}\}$ are given as

Eq. 12
$$\begin{aligned}
&v =:: scatter(\omega_n,\ global)\\
&a = \frac{v}{|v|}\\
&\vec{i}_n = \vec{i}_{n-1} + (a\times\vec{i}_{n-1})\sin(\Delta\theta_n) + 2[a\times(a\times\vec{i}_{n-1})][1-\cos(\Delta\theta_n)]\\
&\vec{j}_n = \vec{j}_{n-1} + (a\times\vec{j}_{n-1})\sin(\Delta\theta_n) + 2[a\times(a\times\vec{j}_{n-1})][1-\cos(\Delta\theta_n)]\\
&\vec{k}_n = \vec{i}_n \times \vec{j}_n
\end{aligned}$$

where v is an auxiliary vector and a is an unit vector along instantaneous axis of rotation. It is interesting to note that in a properly designed OO code the implementation of the Euler formula in Eq. 12 requires virtually a single line of code!

The Element object

The general term 'element object' refers to any finite part of a solid or structure. It is assumed that the response of an element is uniquely defined by the kinematics of interfaces defined on that element. For example, in the case of a beam element in Figure 4 the dynamic response of the 'element' is uniquely defined by the kinematics of two interfaces regardless of the employed beam theory.

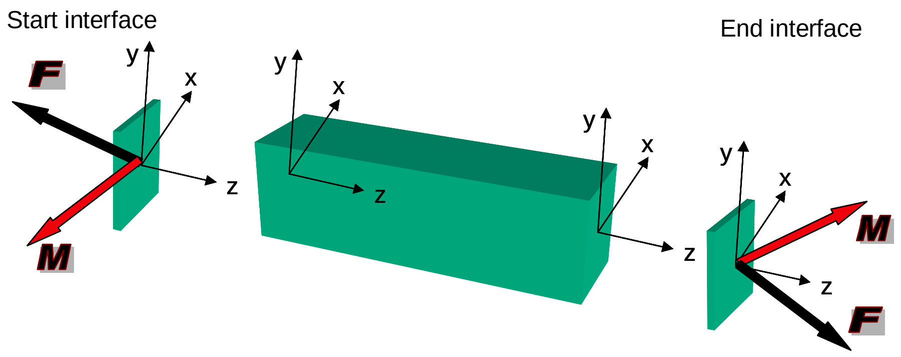


Figure 4 The internal mechanics of a beam element is uniquely defined by the kinematics of two interfaces.

In the present formulation elements are divided into two basic groups: Finite Elements and Macro Elements (ME). The division is functional rather then formal. The name Finite Element applies to all classic elements that are defined and solved by using standard discretize/approximate FEM paradigm. In the present formulation these elements can be implemented directly as 'encapsulated objects' with only minor re-coding of standard FE algorithm. All remaining elements are referred to as Macro

Elements and can be defined by an algorithm, expression, separate program and/or any rule that can be cast into the form of a computer code.

Multi-model representation of the Element

Introduction of the interface object allows for an effective separation of the ‘Element Level’ and ‘Node Level’ in the formulation and implementation of the discrete modeling method. This in turn paves the way to the concept of multi-model representation of an element in the simulation model. The leading idea of the multi-model element concept is illustrated in Figure 5 on the example of a typical thin-walled prismatic member of a car body.

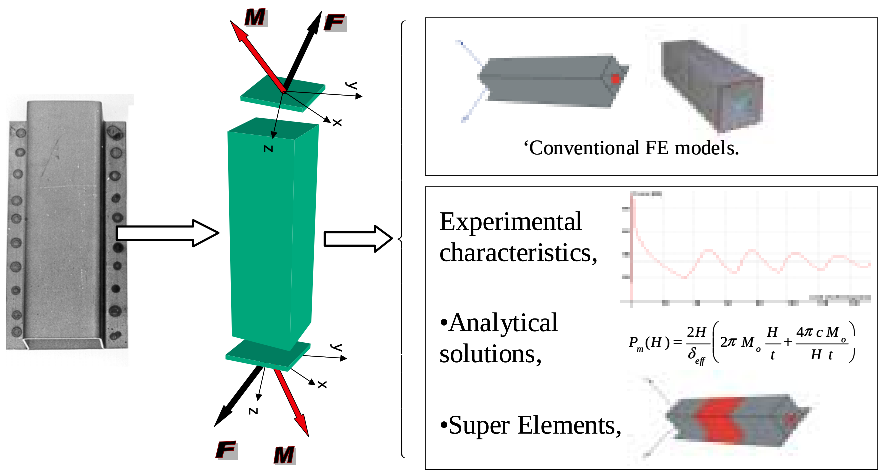

M
F
y
x
z
F
M
‘Conventional FE models.
Experimental characteristics,
•Analytical solutions,
$P_m(H) = \frac{2H}{\delta_{eff}}\left(2\pi\, M_o \frac{H}{t} + \frac{4\pi\, c\, M_o}{H\ t}\right)$
•Super Elements,

Figure 5 Schematic illustration of the multi-model representation of an element. Since elements are effectively separated form node objects implementation of ‘multi-model element’ is done in a natural way by using polymorphism, an elementary mechanisms of the OOP. In the present example the prismatic section could be discretized into a single elastic beam, Superbeam Elements or standard quadrilateral elements. Other discretization methods could involve experimental databases, analytical solutions, codes of practice etc.

Depending on the type of the problem and ‘resolution’ of the simulation model the beam element corresponds to elastic beam(s), Superbeam Element(s), experimental stiffness and/or crushing characteristic or a detailed FE shell model. Since all these beam elements ‘communicate’ with the corresponding nodes via interface objects particulars of the element implementation are not ‘visible’ at the node level and consequently do not affect neither the assembling routine of the entire model nor the selected time stepping routine.
For the first time the multi-model representation of Macro Element was successfully implemented in the software DAMAGE, [11], for simulation of ship collisions and grounding. In DAMAGE all typical variations of structural component(s) (or substructures) can be defined simultaneously in the model database and activated/deactivated by the user by a single click of the mouse. Such an organization of the software allows for an easy verification of various design variants as well as for fast assembling of computational models composed of typical components.

The iterator object

The introduction of the interface object effectively splits the solution to the discrete model into operations on two separate sets: the set of ‘elements’ and the set of ‘nodes’, Figure 6.

The global equilibrium of the system is governed by the equilibrium of nodes. Consequently the time stepping routines derived from an abstract class 'Iterators' are defined on the set of nodes of a given model. The iterator class does not have any knowledge as to the number or type of elements connected to contributing nodes. The second set is the set of elements derived form a single abstract class 'Element'. The operations defined on the set of elements involve constitutive update, contact-impact algorithms etc. Similarly like in the case of nodes the element objects do not have any knowledge on the global properties of the model and communicate only with nodes via interface objects.
The general concept of the code based on the present approach is shown in Figure 6. At the time step *i* the kinematic variables, calculated at the preceding step, are defined for each node of the system and transformed to all interfaces attached to that element.

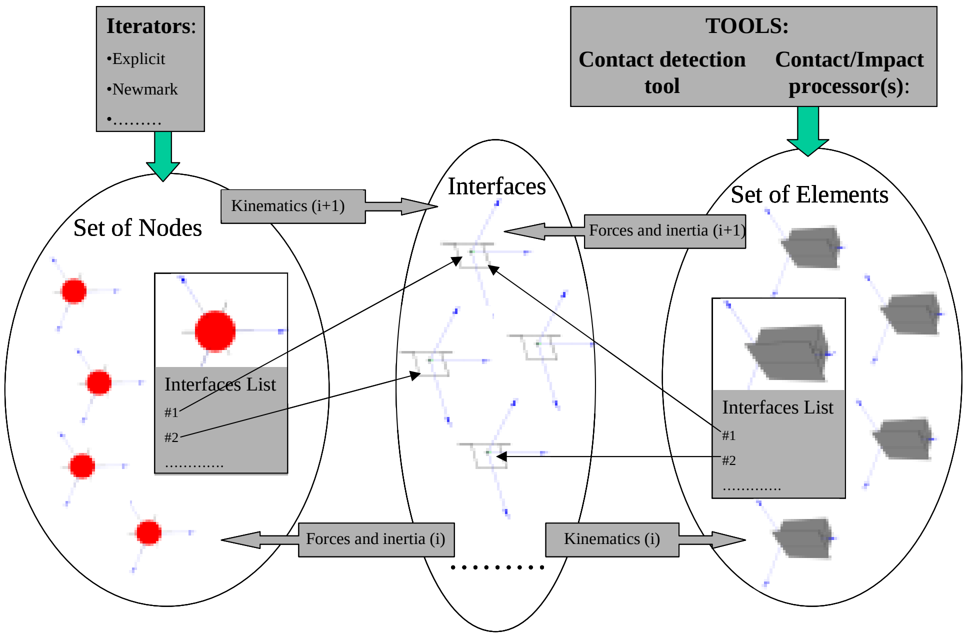


Figure 6 Schematic representation of the iteration process based on the present node-by-node element-by-element algorithm present implementation of FEM based on three basic objects: node, interface and element.

This forms the basis for the constitutive update routine in which kinematic data are read-in by each contributing element. On that basis the resulting internal forces and inertia properties are defined on all interfaces attached to given element. On the *i+1* step inertia characteristics, forces and moments are determined in each node and the next iteration step is performed.
The detailed flowchart for the explicit time integration scheme is given in Table 1 and Table 2. The flowchart follows the standard explicit procedure, see e.g. [1] and [2]. The only difference is that calculation routines are rigorously split into operations on the set of nodes (global equilibrium) and the set of elements (constitutive update).

Table 1 Initialization of the explicit t time stepping routine

Elements set	***Nodes set***
Calculate initial inertia properties m^e and J^e and initialize interfaces.	Set initial velocities $\dot{r}^0 and\ \omega^0$ and initialize inertia objects $m\ and\ J$.

Table 2 Main iteration loop of the explicit time stepping routine

Operations on nodes

1. Get forces and moments from interfaces and calculate reduced forces and moments F^n, M^n (*::scatter* function, Eq. 7).
2. Compute accelerations: $\ddot{r}^n = m^{-1}F^n$ and $\dot{\omega}^n = L(F^n, M^n)$, Eq. 10
3. Update nodal velocities: $\begin{cases} \dot{r}^{n+1/2} = 1/2\,\ddot{r}^0\,\Delta t + \dot{\mathrm{r}}^0 \\ \omega^{n+1/2} = 1/2\,\dot{\omega}^0 \Delta t + \omega^0 \end{cases}$ if n=0 and

   $\begin{cases} \dot{r}^{n+1/2} = \ddot{r}^n\,\Delta t + \dot{\mathrm{r}}^{\mathrm{n-1/2}} \\ \omega^{n+1/2} = \dot{\omega}^n \Delta t + \omega^{n-1/2} \end{cases}$ when n >0
4. Enforce essential boundary conditions for predefined set of nodes: $\dot{r}_b^{n+1/2} = f_g(t^{n+1/2})$ *and* $\omega_b^{n+1/2} = f(t^{n+1/2})$
5. Update nodal displacements and orientation: Eq. 11and Eq. 12
6. Update kinematic variables at connected interfaces (*::gather* function, Eq. 8)
7. Write output data for nodes

Operations on elements

1. Get kinematic variables from interfaces connected to the element
2. Update the state of elements
3. Compute measures of deformation and resulting forces. Compute inertia properties (typically once per 10^2-10^3 iteration steps)
4. Set forces and inertia on connected interfaces.
5. Write output data for elements
6. Go to the next iteration step

Example of implementation

The Object Oriented Formulation introduced in this paper is implemented in the dedicated program for three-dimensional dynamic analysis of thin-walled space frames. The basic building block of the program is the Superbeam Element (SBE) developed by the present author on the basis of earlier research on the large plastic deformations of plastic shells and the concept of the Superfolding Element, [11]-[15]. The Superbeam Element concept is explained in Figure 7. The smallest size of the Superbeam is defined by the length $2H$ of the plastic folding wave in prismatic member. For convenience such an elementary Superbeam is referred to as a deformable cell. Longer Superbeam Elements are defined by connecting together two deformable cells by an elastic/plastic beam. In this implementation deformable cells can be viewed as two generalized plastic hinges at both ends of the elastic/plastic beam. The basic characteristics of the crushing response of each deformable cell under arbitrary combined loading of axial and shear forces and bending and twist moments is calculated at the pre-processing level by a separate program – Superbeam Constructor.

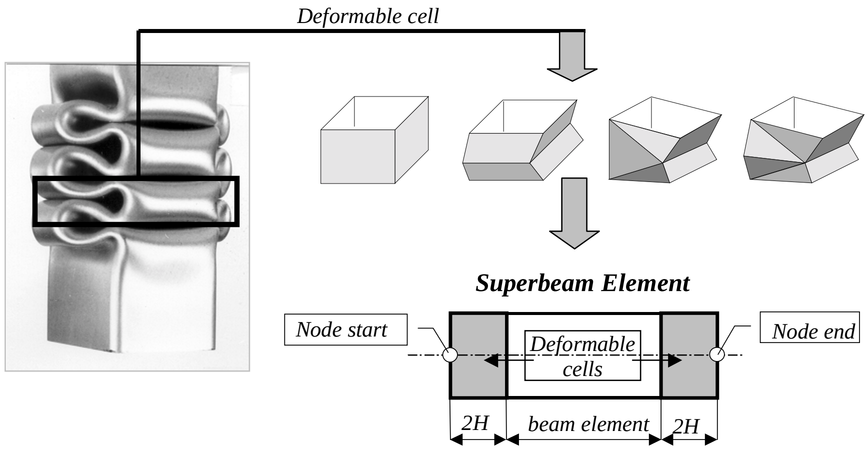


Figure 7 The Superbeam Element modeling concept. Crushing response of a single layer of folds is described by the deformable cell element of the height equal to the length of the plastic folding wave, *2H*. Two deformable cells separated by elastic/plastic core form the longer version of Superbeam element.

Introduction of the intermediate beam between two deformable cells to form a single general Superbeam Element adds the necessary modeling flexibility in the case of space frames made of long thin-walled members, Figure 8.

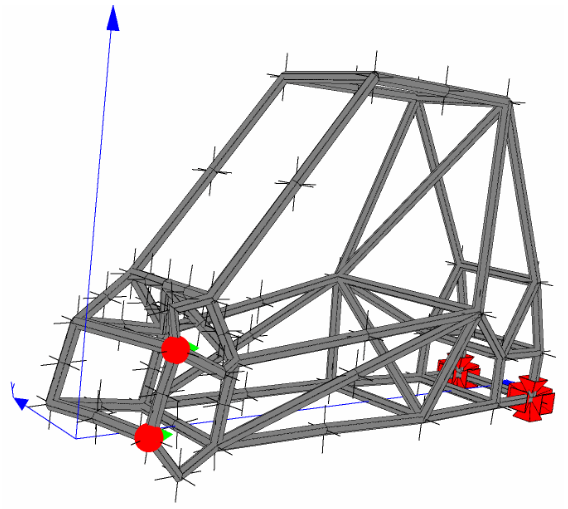


Figure 8 The 10 m/s offset crash response of a buggy frame discretized into 91 Superbeam Elements. CPU time: 6 seconds on 1.7 GHz laptop.

Performances of the new software are illustrated on two examples of real world thin-walled automotive components.

Quasi static crushing of 'S' frames

Three types of 'S' frames in Figure 9, with offset angles of 15, 30 and 45 degrees, respectively were tested quasi-statically and dynamically by Ohkami at all [16]. The quasi-static tests were performed with constant crush velocity of 10 – 50 mm/min with both sides of the specimens fixed for rotation. The crushing characteristics for 15 and 30 degrees frames are compared with the experimental data in Figure 10. It transpires form the above comparison that both pre- as well as post-collapse response of the frame is predicted with very good accuracy. The experimentally recorded peak forces are 27.7 [kN] and 18.1 [kN] for 15 and 30 degrees frames respectively. These should be contrasted with calculated forces of 27.5 [kN] and 19.1 [kN], respectively.

STATIC

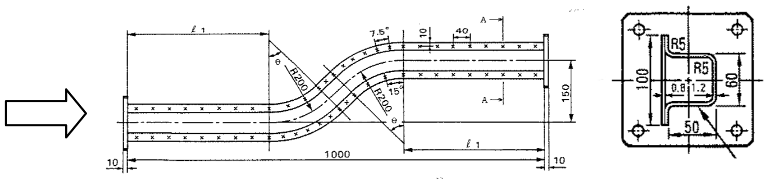
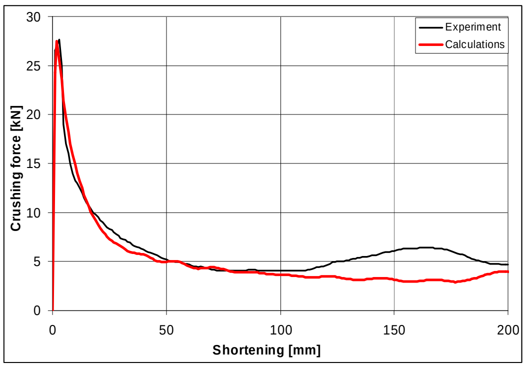
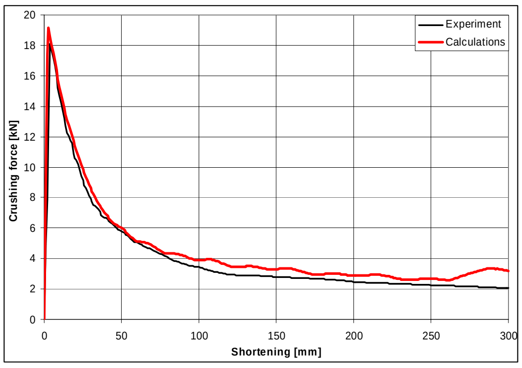


Figure 9 Experimental setup of Ohkami experiment.

In both cases the 'S' frames were discretized into only 3 Superbeam elements. Consequently the CPU time for the whole quasi-static simulation takes only few dozen of seconds on an average PC despite the fact that quasi-static response of both frames is calculated here by using the conditionally stable explicit routine that requires very small time step.



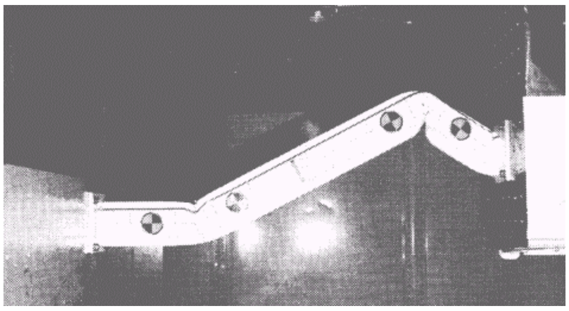
2 10^5 steps CPU – 17 seconds



3 10^5 steps CPU – 25 seconds

Figure 10 Experimental and numerical crushing characteristics of 'S' frames with 15 and 30 degrees offset angle.

The experimentally recorded and calculated deformed shapes of both frames at 100 [mm] crushing are compared in Figure 11 showing again very good agreement.



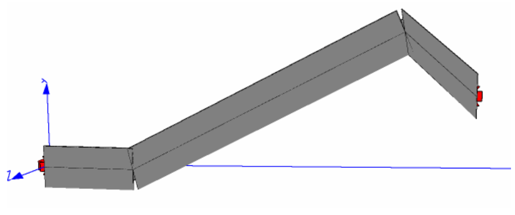


Figure 11 Undeformed and deformed shapes of the 'S' frame predicted by the FE and Superbeam Element models. Deformation corresponds to 100 millimeters of crushing distance.

A specific feature of the Superbeam Element is a very weak influence of the discretization density onto the simulation results. Illustration of this statement is given in Figure 12. Two graphs in this figure correspond to the crushing response of S frame discretized into 3 and 16 Superbeam Elements, respectively. The later model corresponds to the densest discretization permitted by the Superbeam method where each element has the length of a representative plastic fold. It is evident from the graphs in Figure 12 that radical change in the number of discretizing elements does not affect crushing response of the model. This result is explained by the fact that the cross-section collapses at the same locations in both models so that introduction of additional elements did not increase number of plastic hinges in the model.

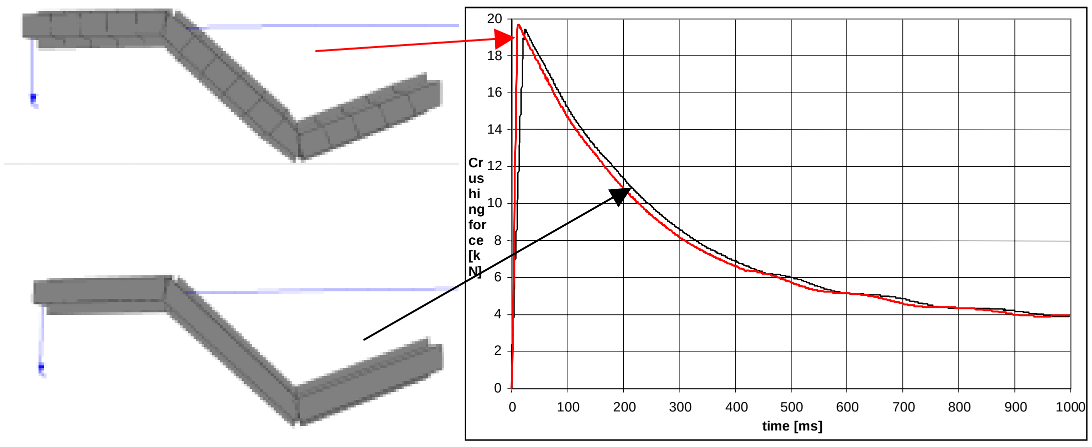


Figure 12 Crushing response of S frame discretized into 3 and 16 Superbeams. The density of discretization does not affect the accuracy of calculations.

Dynamic response of rear end frame

The results on dynamic and quasi-static crushing of a rear frame of a passenger car are reported by Takada and Abramowicz [17]. The corresponding FE and Superbeam computational models are shown in Figure 13. The FE model is made of over 5000 shell elements in contrast to the Superbeam model that contains only 6 elements. The frame is fixed at the wider ends and subjected to low rate kinematic loading of 1.2 [m/s] at the opposite end.

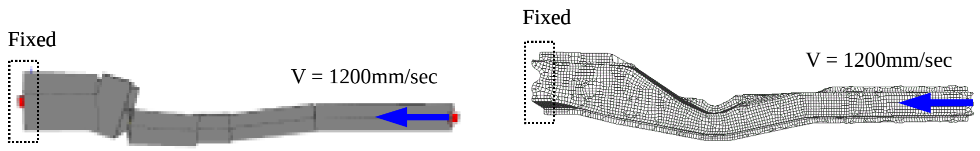


Figure 13 Discretization of a rear frame component into Superbeam and classic FE elements. The Superbeam model consists of 6 elements while FE model takes over 5000 shell elements.

In engineering analysis of frames and longitudinals of a car body an important design feature is the location of local collapse spots where plastic hinges are created. Frames and longitudinals are designed to bend about these hinges in a way that prevents penetration of deformed segments into the passenger compartment.

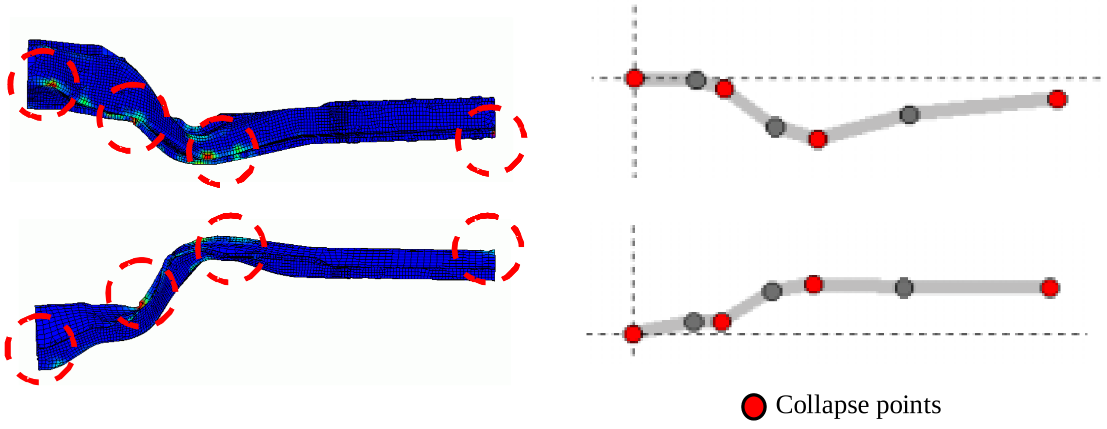


Figure 14 Top and side view of the location of potential plastic hinges in FE and Superbeam models.

Results of such an analysis is shown in Figure 14 for both FE and Superbeam models. It is seen that the Superbeam model reproduces location of plastic collapse of sections with a remarkable accuracy. It should also be noted that FE analysis took over 22 hours on the CRAY T90 supercomputer. This should be contrasted with the Superbeam analysis that took 68 seconds on a 0.5 GHz laptop.

Conclusions

This paper gives a concise description of the object-oriented formulation and implementation of the dynamic algorithm for arbitrary mechanical system that can be modeled by means of the ‘node’/‘element’ paradigm. The major benefit of the new formulation is the possibility of integration in a single computer program elements formulated at various levels of modeling accuracy and methodology of formulation. The new formulation and programming concept is validated for Superbeam Elements that model large elastic plastic deformations of thin walled prismatic beams. The successful implementation of this particular macro-element justifies further development of the program to include other types of macro-elements as well as classic FE elements.

References

[1] Belytschko T Liu W K and Moran B (2000) *Nonlinear Finite Elements for Continua and Structures* John Wiley.

[2] Belytshko T and Hughes TJR (1983) *Computational Methods for Transient Analysis* North-Holland, Amsterdam.

[3] Cook RD Malkus DS and Plesha ME (1989) *Concepts and Applications of Finite Element Analysis*, John Wiley.

[4] Zienkiewicz OE and Taylor RL (2000) *The Finite Element Method (Vol1. The Basis)* Butterworth-Heinemann.

[5] Abramowicz W Limitations of the matrix method in the FEM formulation for large deformations and strains (2001) Impact Design Europe internal report 1/2001.

[6] Abramowicz W Object Oriented, Reduced Operation Set Formulation of the Multi-Body Macro and Finite Element method (2001) Impact Design Europe internal report 2/2001.

[7] MSDN (The Microsoft Developer Network), © *1991-2000 Microsoft Corporation*, http://msdn.microsoft.com/.

[8] Rosenberg M R (1977) *Analytical Dynamics of Discrete Systems* Plenum Press.

[9] Blajer W (1998) *Methods of Multibody Dynamics*, Monographs of Radom Technical University (in polish)

[10] Abramowicz W and Sinmao M (1999) *User's Manual and Modeling Guide for the Program DAMAGE, v. 4.0*, Joint MIT-Industry Program on Tanker Safety, report No. 66, Department of Ocean Engineering, MIT Cambridge MA.

[11] Abramowicz W. (2001). *Macro element method on crashworthiness of Vehicles* in Crashworthiness – energy management and occupant protection (Ambrosio J. Editor) SpringerWienNewYork.

[12] Abramowicz W. (1996). *Extremal Paths in Progressive Plasticity*, Int. J. Impact Engng, **18:7-8**, 753-764.

[13] Abramowicz W. and Wierzbicki T. (1989). *Axial crushing of multi-corner sheet metal columns* J. App. Mech., **56**:**1**, 113-120.

[14] Wierzbicki T.and Abramowicz, W. (1987 **-** 1989). The Manual of Crashworthiness Engineering Vol. I **-** IV, Center for Transportation Studies, Massachusetts Institute of Technology.

[15] *Crash Cad Manuals - Superbeam Constructor*, (1998) Impact Design Boston MA.

[16] Ohkami, Y., et al Collapse of Thin Walled Curved Beam with Closed Hat Section – Part 1: Study on Collapse Characteristic, SAE 900560.

[17].. Takada K., Abramowicz W., *Novel Formulation of the 3D Large Deformation Beam Element for Dynamic Crash Analysis,* (in Japanese), JSAE papers 2002 – in print.