Karlsruhe Institute of Technology

Institute of Mechanics

On an efficient implementation of 'Solid-Shell' finite elements with quadratic shape functions for explicit time integration

October 2009

Steffen Mattern Karl Schweizerhof

Institute of Mechanics
Kaiserstr. 12
D-76131 Karlsruhe
Tel.: +49 (0) 721/ 608-2071
Fax: +49 (0) 721/608-7990
E-Mail: info@ifm.kit.edu
www.ifm.uni-karlsruhe.de

On an efficient implementation of 'Solid-Shell' finite elements with quadratic shape functions for explicit time integration

Steffen Mattern Karl Schweizerhof

Abstract

The main application of explicit time integration algorithms for finite element simulations – as they are characterized by rather small time steps – are highly dynamic problems. An implementation of the element routines with a specific view on efficiency of the algorithms is essential, while the internal force vector has to be computed extremely often, compared to implicit algorithms. In this contribution, a specific implementation concept is presented, using automatic generation and optimization of code for a so-called 'Solid-Shell' formulation with quadratic shape functions in membrane direction and linear approximation in thickness direction. The efficiency of the numerical algorithms is shown and compared to otherwise programmed and compiled routines.

1 Introduction

In order to realize a continuum-like modeling of a shell structure needed to capture 3D effects as in laminated shells, the so-called 'Solid-Shell' element class, presented e.g. in [5], with linear interpolation of geometry and displacements in thickness as well as in shell surface direction is a suitable alternative to purely 3D analysis. As the formulation allows independent interpolation for the in-plane and the out-of-plane direction, a separate higher order interpolation only in the shell-surface plane as often needed for curved shells is possible.

The most widely-used explicit time integration method is the central difference scheme, often also called Verlet algorithm ([13]). For lumped mass matrices the costly solution of linear equations on global level is not necessary; further the usage of diagonalized mass matrices solely requires vector operations, which leads to low computational cost per time step. Unfortunately, due to the so-called Courant criterion ([3]), the time step size is limited to a critical value, which makes the central difference method mostly attractive for highly dynamic problems like impact, strong nonlinearities and short duration transient analyses, where small time steps are required anyway.

In this contribution, a 18-node shell element with bi-quadratic Lagrangian shape functions as presented in [4] is proposed for explicit time integration. This requires a special focus on efficient implementation, as most operations on element level have to be performed for every time step. To achieve this, the numerically '*costly*' parts of the element routine were implemented using the automatic code generation tool AceGen, q.v. [9, 7]. The benefits of this approach are shown on the basis of numerical examples, where generated and optimized code is compared to manually programmed code, regarding computational cost.

2 Explicit Time Integration

For numerical time integration, the well-known central difference method is used and implemented as proposed in [2] or originally for molecular dynamics by [13]. Here the governing equations are shown briefly. For the current time step n, the accelerations are computed as

$$\ddot{\mathbf{d}}^n = \mathbf{M}^{-1} \left(\mathbf{f}^n - \mathbf{C}\, \dot{\mathbf{d}}^{n-1/2} \right), \tag{1}$$

with the diagonalized system mass matrix $\mathbf{M}$, the system load vector $\mathbf{f}^n$, the system damping matrix $\mathbf{C}$ and the velocities $\dot{\mathbf{d}}^{n-1/2}$ at time step $n - 1/2$. The velocity between two time steps is updated by

$$\dot{\mathbf{d}}^{n+1/2} = \dot{\mathbf{d}}^{n-1/2} + \overline{\Delta t^n}\, \ddot{\mathbf{d}}^n \tag{2}$$

with $\overline{\Delta t^n} = \frac{(\Delta t^n + \Delta t^{n-1})}{2}$, which leads to the displacements

$$\mathbf{d}^{n+1} = \mathbf{d}^n + \Delta t^n\, \dot{\mathbf{d}}^{n+1/2}, \tag{3}$$

with the current time step size $\Delta t^n = t^{n+1} - t^n$. The time step size is limited by the COURANT-criterion by

$$\Delta t \leq \alpha\, \Delta t_{crit} = \alpha\, \frac{2}{\omega_{max}} \approx \alpha \left(\min_e \frac{l_e}{c_e} \right), \tag{4}$$

where ω_{max} is the largest eigenfrequency, l_e represents a characteristic element length and c_e the wave propagation velocity. The COURANT-criterion is based on linear problems, so in order to consider non-linearities, the factor $\alpha < 1$ is introduced.

The implementation of the central difference method requires no solution of linear equations, only vector operations are performed on global level if diagonal mass matrices are used. This leads to very little CPU-time requirements per time step, compared to implicit methods. The limitation of the time step by Equation 4 makes this method especially appropriate for highly dynamic applications such as crash or impact. For long-term dynamic problems, a very large number of time steps is required, which may lead to a long simulation time, however it is a purely time marching scheme. Efficiency depends mainly on the evaluation of the internal forces, the main topic of the following sections.

3 The 'Solid-Shell' Concept

In this section, a very short introduction into the 'Solid-Shell' concept is given. For more detailed information, it is referred to the comprehensive literature, e.g. [11, 5]. The 'Solid-Shell' concept provides a shell formulation with displacement degrees of freedom only. Under the assumption of the degenerated shell concept that the normals to the mid-surface remain straight, following the notation given in Figure 1, the initial geometry is given by

$$\mathbf{X}(\xi, \eta, \zeta) = \frac{1}{2} \left((1 + \zeta)\, \mathbf{X}_u(\xi, \eta) + (1 - \zeta)\, \mathbf{X}_l(\xi, \eta) \right), \tag{5}$$

Linear interpolation of the displacements of the upper and the lower surface leads to

$$\mathbf{u}(\xi, \eta, \zeta) = \frac{1}{2} \left((1 + \zeta)\, \mathbf{u}_u(\xi, \eta) + (1 - \zeta)\, \mathbf{u}_l(\xi, \eta) \right). \tag{6}$$

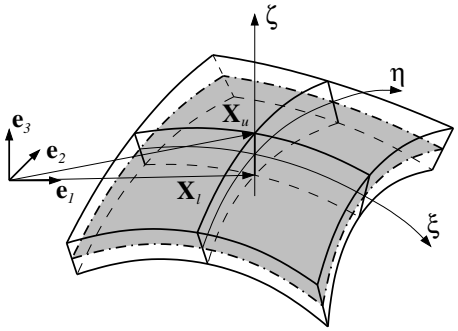


Figure 1: geometry of a solid shell

In this contribution, quadratic, isoparametric 'Solid-Shell' elements are used with bi-quadratic interpolation in membrane and linear interpolation in thickness direction as presented in [4]. For the discretization of the initial and geometry, this leads to

$$\mathbf{X}^{el}(\xi,\eta,\zeta) = \sum_{i=1}^{9} \left(\frac{1}{2} N_i(\xi,\eta)\, \mathbf{\Theta}(\zeta)\, \mathbf{X}_i \right), \tag{7}$$

with the vector of upper and lower nodal displacements $\mathbf{X}_i = \left[\begin{array}{cc} \mathbf{X}_{iu} & \mathbf{X}_{il} \end{array} \right]^T$, the linear thickness interpolation matrix

$$\mathbf{\Theta}(\zeta) = \left[\begin{array}{cccccc} 1+\zeta & 0 & 0 & 1-\zeta & 0 & 0 \\ 0 & 1+\zeta & 0 & 0 & 1-\zeta & 0 \\ 0 & 0 & 1+\zeta & 0 & 0 & 1-\zeta \end{array} \right] \tag{8}$$

and the bi-quadratic Lagrangian shape functions.

As mentioned in section 2, an efficient usage of the central difference method implies diagonalized mass matrices. The entries of the consistent element mass matrix $\mathbf{M}^{el}$

$$M_{ij}^{el} = \int_V \rho\, \frac{1}{4}\, (1+\zeta\,\zeta_i)\, N_i(\xi,\eta)\, (1+\zeta\,\zeta_j)\, N_j(\xi,\eta)\, dV \tag{9}$$

with $N_i(\xi,\eta)$ for in-plane interpolation, are therefore summed up row by row, in order to achieve the diagonalized form

$$M_{ij}^{el,d} = \left\{ \begin{array}{cc} \sum_{k=1}^{ndof} M_{ik}^{el} & i = j \\ 0 & i \neq j \end{array} \right. \tag{10}$$

Other methods to achieve diagonalized mass matrices are described e.g. in [6] and will be discussed for different numerical examples.

A very important issue concerning the implementation of finite elements is the activation of artificial stresses for different loading situations, the so-called '*locking*'. Though not as distinctive as in elements with linear displacement interpolation, the effects can be reduced – or even canceled – by several corrections within the element formulation. Proposals for locking-free 'Solid-Shell' elements, can be found besides the cited literature also in [1, 12]. The numerical examples in section 5 are performed, however, for simplicity reasons with a fully integrated element without any modifications against locking.

4 Efficient Implementation of the Internal Force Vector

4.1 Operations on Element Level

As the time integration scheme is characterized by rather small time steps, the right-hand side vector $\mathbf{f} = \mathbf{f}^{ext} - \mathbf{f}^{int}$ from Equation 1 containing the external $\mathbf{f}^{ext}$ and the internal forces $\mathbf{f}^{int}$ has to be formed very often. The internal forces are integrated on element level and assembled to the global force vector, which leads to

$$\mathbf{f} = \mathbf{f}^{ext} - \int_V \left(\mathbf{B}^T \sigma\right) \det \mathbf{J} \; dV \tag{11}$$

where $\mathbf{B}$ represents the derivation of the strains with respect to the nodal displacements and σ contains the stresses. The integration is performed numerically e.g. by a GAUSS-LEGENDRE-rule, which leads to 18 operations $\mathbf{B}^T \sigma$ for each element with a '*full*' 3 x 3 x 2-point integration.

In test examples with the own code FEAP-MEKA, proportions of more than 90 % of the total CPU-time for a simulation could be measured for the above operations on element level.

4.2 '*Automatic*' Code Generation

In order to reach an efficient and comfortable implementation of the subroutines on element level, the improved – so-called '*automatic*' – code generation and optimization tool ACEGEN, a plug-in for the computer algebra software MATHEMATICA is used. The program is developed by the group of KORELC, see [7, 10, 8].

For the numerical integration of Equation 11, the operation

$$\mathbf{f}^{int}_{GP} = \mathbf{B}^T(\xi_i, \eta_j, \zeta_k)\, \sigma(\xi_i, \eta_j, \zeta_k) \det \mathbf{J}(\xi_i, \eta_j, \zeta_k) \tag{12}$$

has to be evaluated at each GAUSS-point, multiplied with the weights and then summed up. ACEGEN allows the usage of MATHEMATICAs symbolic capabilities, so implemented functions can be used to perform matrix operations or differentiations. This increases the convenience of programming and decreases the number of programming errors considerably. Also the computational speed is increased by using automatic code generation, which is shown in the following section. For implementation, the following steps – together with the used commands – have to be carried out:

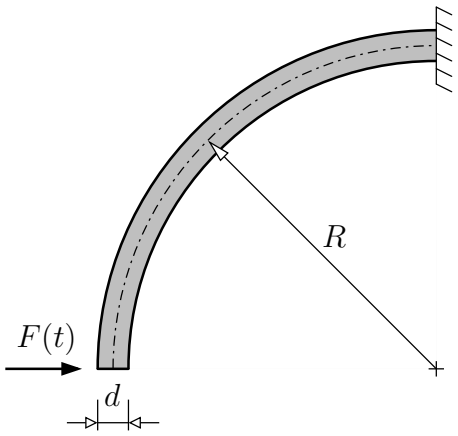
Initialization: The subroutine as well as the input (geometry, current displacements, coordinates and weight of the current GAUSS-point and material parameters) and output variables (internal force vector, evaluated at the current integration point) are defined. The used ACEGEN commands are `SMSInitialize` and `SMSModule`.

Element matrices: After the import of the element data, the necessary matrices – Jacobian, convective base vectors, elasticity tensor, etc. – can be evaluated by using MATHEMATICA's symbolic capabilities as the tensor multiplication. Differentiation with respect to variables or tensors is also possible (`SMSD`), which is used i.e. to evaluate the $\mathbf{B}$-Matrix ($\varepsilon_{,\mathbf{d}_e}$), defined in Equation 11.

Export internal force vector: The internal force vector at the current integration point – as given in Equation 12 – has to be exported, to be available outside the subroutine. The command `SMSExport` is used with the option `"AddIn"=True` in order to automatically sum the results for all GAUSS-points to the global memory field.

```
[...]
v(1061)=v(1624)*v(616)
v(1062)=v(1625)*v(616)
v(1063)=v(1626)*v(616)
v(1064)=v(1624)*v(572)+v(1621)*v(599)
v(1065)=v(1625)*v(572)+v(1622)*v(599)
v(1066)=v(1626)*v(572)+v(1623)*v(599)
[...]
```

Figure 2: exemplary section of automatically generated and optimized FORTRAN code



- properties

$$\begin{aligned} E &= 1.0 \cdot 10^{10} \\ \nu &= 0.0 \\ \rho &= 10,000 \\ R &= 10.0 \\ d &= 0.1 \\ F_0 &= 1.2 \cdot 10^6 \end{aligned}$$

- load curve

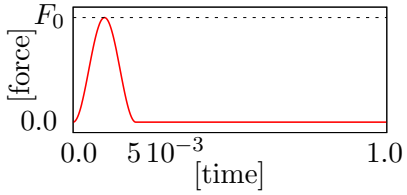


Figure 3: example – bended curved beam

Code generation: In the last step, FORTRAN-code is generated and automatically optimized, using the command `SMSWrite`. The language (FORTRAN, C, etc.) and the level of optimization depend on options, given in `SMSInitialize`. Figure 2 shows a portion of this automatically generated FORTRAN-code.

The only disadvantage of this procedure is, that the automatically generated and optimized code is not longer readable and any change in the program requires a complete re-generation of the subroutine with ACEGEN

5 Numerical Example

The numerical example – a bended curved beam, depicted in Figure 3 – is chosen to compare the numerical effort of the manually programmed element subroutine with the automatically generated code. The results in this example are only checked concerning plausibility, no further investigations concerning convergence and locking are performed here. Geometry and material parameters are given in Figure 3, the used material law is based on linear elasticity.

The simulation of the problem was carried out with three different implementation types:

- '*regular*' – manually programmed routines without adaption of matrix multiplications

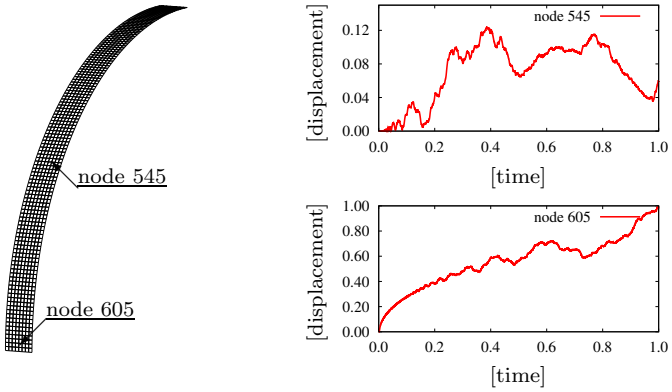


Figure 4: discretization and results at two different nodes

	regular	man. opt.	ACEGEN
time steps	13,470	13,470	13,470
$\varnothing\ \Delta t$	$7.424 \cdot 10^{-5}$	$7.424 \cdot 10^{-5}$	$7.424 \cdot 10^{-5}$
wall time	12 h 49 min 11 s	6 h 10 min 55 s	10 min 19 s

Table 1: statistics – w/o optimization by compiler

	regular	man. opt.	ACEGEN
time steps	13,470	13,470	13,470
$\varnothing\ \Delta t$	$7.424 \cdot 10^{-5}$	$7.424 \cdot 10^{-5}$	$7.424 \cdot 10^{-5}$
wall time	2 h 03 min 57 s	2 h 11 min 33 s	10 min 36 s

Table 2: statistics – highest optimization by compiler

- '*manually optimized*' – manually programmed routines avoiding unnecessary operations
- '*automatically optimized*' – generation and optimization of routines with ACEGEN

All three implementations lead to the same results, depicted in Figure 4 with the chosen FE-mesh. First, the code was compiled without any optimization flags, provided by the compiler. As given in Table 5, the simulation time can be reduced by 50 % by manual optimization; the automatic generated code only requires 2.8 % of the time, compared to the regular implementation. The highest optimization level of the compiler cancels the effect of manual optimization completely, see Table 1. Still the generated code is much faster, it requires only 8.55 % of the time compared to the manually programmed but compiler optimized code. Optimization by the compiler has no effect on the simulation time with the ACEGEN generated code.

6 Conclusions and Outlook

In order to save computational cost within an explicit time integration algorithm, especially the integration of the nodal force vector on element level has to be implemented with focus on efficiency. A manual implementation of the routines with minimization of operations can be cumbersome and error-prone. Efficient programming,

together with a minimization of mistakes during programming can be achieved, using an automatic code generator like ACEGEN. In the presented contribution, the implementation was carried out for a 'Solid-Shell' finite element with quadratic in-plane interpolation, which lead to a considerable reduction of CPU-time without changes in the element formulation. Further implementations of improved 'Solid-Shell' and so-called degenerated shell elements may lead to similar improvements, concerning efficiency, which will be a major part of the future work in the project.

References

[1] R.J. Alves de Sousa, R.P.R. Cardoso, R.A. Fontes Valente, J.W. Yoon, R.M. Natal Jorge, and J.J. Gracio. A new one-point quadrature enhanced assumed strain (EAS) solid-shell element with multiple integration points along thickness: Part I – geometrically linear applications. *Int. J. Num. Meth. Eng.*, 62:952–977, 2005.

[2] T. Belytschko, W.K. Liu, and B. Moran. *Nonlinear finite elements for continua and structures*. Wiley, 2004.

[3] R. Courant, K.O. Friedrichs, and H. Lewy. Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen*, 100:32–74, 1928.

[4] R. Hauptmann, S. Doll, M. Harnau, and K. Schweizerhof. 'solid-shell' elements with linear and quadratic shape functions at large deformations with nearly incompressible materials. *Computers & Structures*, 79(18):1671–1685, 2001.

[5] R. Hauptmann and K. Schweizerhof. A systematic development of 'solid-shell' element formulations for linear and non-linear analyses employing only displacement degrees of freedom. *Int. J. Num. Meth. Eng.*, 42(1):49–69, 1998.

[6] Thomas J. R. Hughes. *The finite element method*. Dover Publ., dover ed., 1. publ. edition, 2000.

[7] J. Korelc. Automatic generation of finite-element code by simultaneous optimization of expressions. *Theoretical Computer Science*, 187(1-2):231–248, 1997.

[8] J. Korelc. Multi-language and multi-environment generation of nonlinear finite element codes. *Engineering with Computers*, 18(4):312–327, 2002.

[9] J. Korelc. http://www.fgg.uni-lj.si/symech/, 2008.

[10] J. Korelc and P. Wriggers. Computer algebra and automatic differentiation in derivation of finite element code. *ZAMM*, 79:811–812, 1999.

[11] H. Parisch. A continuum-based shell theory for non-linear applications. *Int. J. Num. Meth. Eng.*, 38:1855–1883, 1995.

[12] S. Reese. A large deformation solid-shell concept based on reduced integration with hourglass stabilization. *Int. J. Num. Meth. Eng.*, 69(8):1671–1716, 2007.

[13] L. Verlet. "Experiments" on classical fluids I. Thermomechanical properties of Lennard-Jones molecules. *Physical Review*, 159:98–103, 1967.