Simulation of Bevel Gear Cutting Processes

Windows High Performance Computing
24./25. April 2007

Dipl.-Ing. Uwe A. Rütjes
Outline

- Introduction
  - Definition of Bevel Gears
  - Manufacturing of Bevel Gears
  - Reasons for Simulating the Manufacturing Process
  - Goals of the Simulation

- Simulation System
  - Build Up
  - Implementation

- Debugging the Application
Definition of Bevel Gears

- **Geometry**
  - The body is cone-shaped
  - The teeth height vary
  - The teeth can be epicyclic

- **Application**
  - Bevel gears transmit power over intersecting or skewed axis
  - Main area of application is the differential gear
Manufacturing of Bevel Gears

- The bevel gear is mating with a virtual generating gear.
- The cutting blades imitate the teeth of the virtual generating gear.
- The virtual generating gear rotates about its axis (the cradle).
- The cutter head also has to apply the movements.
- Very complex movements.
- Hard to predict the resulting conditions on the cutting blades.
Reasons for Simulating the Manufacturing Process

Planning a Manufacturing Process

- **Goal:**
  - Decrease of costs by planning the manufacturing process
  - Increase of tool life
  - Increase of productivity

- **Milling of bevel gears:**
  - Reduction of main- and auxiliary process time
  - Dry manufacturing process

- **Difficulties with a dry manufacturing process:**
  - an insufficient design of the manufacturing process results in an extreme variation of the tool life and unpredictable tool wear
  - a careful design of the manufacturing process is necessary
Goals of the Manufacturing Simulation

Avoidance of Cost-consuming Trials by Simulations

- Simulation
  - Calculation of the real tooth flank topography
  - Analysis of the manufacturing process by determining chip creation parameters
  - Arithmetical determination of the load of the cutting edge of the workpiece

- Advantages of Simulations
  - Omitting cost-intensive trials
  - Early detection of unfavourable manufacturing parameters
  - Optimisation of machine kinematics and the tool design
Outine

- Introduction
  - Definition of Bevel Gears
  - Manufacturing of Bevel Gears
  - Reasons for Simulating the Manufacturing Process
  - Goals of the Simulation

- Simulation System
  - Build Up
  - Implementation

- Debugging the Application
Construction of the Simulation System – Classification of the Modules

- The objectives for the Simulation Systems are:
  - Easy implementation
  - Easy maintenance
  - Easily expandable
  - Components are easily substituted

- For this reason a modular design was chosen

- Complete new design and implementation in Fortran95

### Linear Algebra
- Vector and matrix calculations; solutions of linear equation systems

### Geometric Objects
- **Lines**
  - Line types; calculates points, tangents, normals

- **Surfaces**
  - Surfaces types; calculates points, normals, tangents

### High Level Objects
- **Kinematics**
  - Computes different transformations depended on the time

- **Tool**
  - Tool types and geometries of the tools and cutting edges

- **Workpiece**
  - Geometries and functionalities of the work pieces

### Process event
- Simulation framework simulation flow for all implemented processes
Construction of the Simulation System – Implementation

- The modules were implemented as Fortran MODULES

- The single objects are implemented as TYPE's with public accessible routines

```fortran
MODULE Vektor
  !
  IMPLICIT NONE
  !
  TYPE, PUBLIC :: Vektor4
  PRIVATE
     REAL (KIND=real_normal), DIMENSION(4) :: vektor
  END TYPE Vektor4
  !
  INTERFACE SetupVektor4
    MODULE PROCEDURE ...
  END INTERFACE SetupVektor4
  !
  INTERFACE Operator (.cross.)
    MODULE PROCEDURE cross4
  END INTERFACE Operator (.cross.)
  !
  INTERFACE ASSIGNMENT (=)
    MODULE PROCEDURE assign4
  END INTERFACE ASSIGNMENT (=)
CONTAINS
  ...
END MODULE Vektor
```
Construction of the Simulation System – Implemented Modules

- **Workpiece**
  - free geometry definition
  - meshing surface

- **Tool**
  - definition of the cutting blade
  - free positioning of the blade

- **Kinematics**
  - arbitrary transformations
  - provided as spline, polynominal or function

- **Penetration**
  - envelope of tool calculated from blade geometry and kinematics
  - meshing of the envelope
Outline

- Introduction
  - Definition of Bevel Gears
  - Manufacturing of Bevel Gears
  - Reasons for Simulating the Manufacturing Process
  - Goals of the Simulation

- Simulation System
  - Build Up
  - Implementation

- Debugging the Application
Debugging the Application – 1

- **Debugging with PARAMETERS and Global Variables**

- **Un-initialised Variables**
  - Global Variables
  - Arrays Dimension(1)
  - Cannot View Register Variable

- **Debugger is not stopping**

- **OpenMP and Pointer**

- **Stack Overflow**

---

Parameters are not Shown in the Debugger

```
REAL(KIND=real_normal) :: delta, umin, t, gesamtleenge
INTEGER :: SchneidenPktc, anzahl, counter, i, j, ebeneMin, ebeneMax, pktMin, pktMax
LOGICAL :: fehler

REAL(KIND=real_normal), PARAMETER :: nullpunktfeur = 0.5
INTEGER :: numEbenen
INTEGER, PARAMETER :: numEdMax = 155

! Anzahl der Punkte festlegen
numEbenen = MIN(numEdMax, INT(((ende-start)/0.013)+97))
numEbenen = MIN(numEdMax, INT(((ende-start)/0.013)+85))
numEbenen = MAX(numEbenen, 4) ! Aber mindestens 4 Ebenen
CALL getMesserkopfMess (vgl, messer, messerkopf)

! Anzahl der Punkte im Schneidenprofil festlegen
```

---

<table>
<thead>
<tr>
<th>Überwachen</th>
<th>Name</th>
<th>Wert</th>
<th>Typ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto</td>
<td>nullpunktfeur</td>
<td>Undefined variable nullpunktfeur</td>
<td></td>
</tr>
<tr>
<td>Auto</td>
<td>numEdMax</td>
<td>Undefined variable numEdMax</td>
<td></td>
</tr>
</tbody>
</table>

© WZL/Fraunhofer IPT
Debugging the Application – 2

- Debugging with PARAMETERS and Global Variables

- Un-initialised Variables
  - Global Variables
  - Arrays Dimension(1)
  - Cannot View Register Variable

- Debugger is not stopping

- OpenMP and Pointer

- Stack Overflow

Un-initialised Variables

```plaintext
selector(1:NUMBER_OF_FLANKS(THIS_CHAPTER)) = &
flank_selector(1:NUMBER_OF_FLANKS(THIS_CHAPTER))

INTEGER, DIMENSION(1) :: lcc, lcc2
if
lcc = 0
lcc2 = 0
```
Debugging the Application – 3

- Debugging with PARAMETERS and Global Variables
- Un-initialised Variables
  - Global Variables
  - Arrays Dimension(1)
  - Cannot View Register Variable
- Debugger is not stopping
- OpenMP and Pointer
- Stack Overflow

Debugger is not Stopping Exactly

Problem with viewing variables after running for some time
  - The shown variable values are not the real value for this step
  - Especially during DO-Loops the Debugger is not stopping in time
  - This leads to wrong variables
  - After some time (5-10 min) the variables have changed

Sorry, no example
Debugging the Application – 4

- Debugging with PARAMETERS and Global Variables

- Un-initialised Variables
  - Global Variables
  - Arrays Dimension(1)
  - Cannot View Register Variable

- Debugger is not stopping

- OpenMP and Pointer

- Stack Overflow

---

**Behaviour of Pointers**

The behaviour of pointers is changing for different link – libraries
- At first the program was developed with Debug Single-threaded libraries
- Pointers were not NULLIFY’ed
- After changing to openMP a different behaviour was encountered
- A not NULLIFY’ed pointer passed the ASSOCIATED() query, but when accessing the array/structure an exception occurred

This is in general no problem. But sudden behaviour changes complicate the implementation
Debugging the Application – 5

- Debugging with PARAMETERS and Global Variables
- Un-initialised Variables
  - Global Variables
  - Arrays Dimension(1)
  - Cannot View Register Variable
- Debugger is not stopping
- OpenMP and Pointer
- Stack Overflow

### Local Arrays on Stack

**SUBROUTINE** swapTriPackParameters (s, t, p, n, number, row, column)
**IMPLICIT** NONE

! REAL(KIND=real_normal), **DIMENSION**(number) :: s, t
**TYPE**(Vektor4), **DIMENSION**(number) :: p, n
**INTEGER** :: number, row, column

! REAL(KIND=real_normal), **DIMENSION**(row,column) :: snew, tnew, stemp, ttemp
! **TYPE**(Vektor4), **DIMENSION**(row,column) :: pnew, nnew, ptemp, ntemp
REAL(KIND=real_normal), **DIMENSION**(::,:), **ALLOCATABLE** :: snew, tnew, & stemp, ttemp
**TYPE**(Vektor4), **DIMENSION**(::,:), **ALLOCATABLE** :: pnew, nnew, ptemp, ntemp
**INTEGER** :: i, start, ende
**LOGICAL** :: changeS, changeT

Some local variables are created on the stack and cause exceptions
- After reading ‘Passing Arrays efficiently’ the explicit bounds to the array where given
- But nonetheless an stack overflow occurred, without showing the reason
- The reason have not been passed arrays, but the local arrays
- After changing to **ALLOCATABLE** the stack overflow was gone
- This was expected

Solving this did cost a lot of time, because no hint for the reason was given. Additionally this error occurs after a long simulation time.