ElmerPost

Post-processing your results

Thomas Zwinger

thomas.zwinger[@]csc.fi

Computational Environment & Application
CSC–Scientific Computing Ltd.
The Finnish IT center for science
Espoo, Finland
Contents

ElmerPost Program
  The Interface
  The Interface - Menus
  How to Read Files
  Command Window

Output for Other Post-processors
ASCII Based Output
ElmerPost Program

written by J. Ruokolainen
ElmerPost Program

- written by J. Ruokolainen
- utilizes OpenGL (e.g. Mesa) and TCL/TK
ElmerPost Program

- written by J. Ruokolainen
- utilizes OpenGL (e.g. Mesa) and TCL/TK
- nice add-ons as modules (JPG and MPEG output)
ElmerPost Program

- written by J. Ruokolainen
- utilizes OpenGL (e.g. Mesa) and TCL/TK
- nice add-ons as modules (JPG and MPEG output)
- fast and easy way to post-process
ElmerPost Program

- written by J. Ruokolainen
- utilizes OpenGL (e.g. Mesa) and TCL/TK
- nice add-ons as modules (JPG and MPEG output)
- fast and easy way to post-process
- launch simply with `ElmerPost "winsize 1000 800"`
The Interface

1. Read result
2. Mesh display
3. Iso-contours
4. Iso-surfaces
5. Vector-field
6. Particles
7. Color-bar
8. Refresh
9. View settings
10. Output
11. Command
The Interface - Menus

File
Open   same as button 1 on the previous slide
Save Image   save ppm or PostScript (not too large!!) image
Load Sicopolis load result file from SICOPOLIS (R. Greve’s SIA code)
Quit   exit ElmerPost

Display buttons 2 - 7 from previous slide in a menu

Help   short help screen (quite useful)

Modules if installed: jpeg and mpeg-export
The Interface - Menus

File
Open: same as button 1 on the previous slide
Save Image: save ppm or PostScript (not too large!!) image
Load Sicopolis: load result file from SICOPOLIS (R. Greve’s SIA code)
Quit: exit ElmerPost

Display
buttons 2 - 7 from previous slide in a menu

Help
short help screen (quite useful)

Modules
if installed: jpeg and mpeg-export

Edit
Colormap: change the colormap
Grouping: toggle entities for display/hide
Material: define some optical effects, like opacity
Background: background color
Timestep Control: displays a series of time steps/iteration steps
Math Module Window: MATC command interface
Camera Settings: change your point of view (which is sometimes necessary)
Object Clip Planes: how far your model can reach
How to Read Files

- **Steady simulation**: by default contains results of all iteration steps.
How to Read Files

- **Steady simulation**: by default contains results of all iteration steps

  ⇒ the last one contains the correct result
How to Read Files

- **Steady simulation**: by default contains results of all iteration steps
  - the last one contains the correct result
  - use the highest possible value (obtained by pressing **All**) in both fields (**First** and **Last**)

![Screenshot of a simulation interface](image)

Zwinger – p.6/9
How to Read Files

- **Steady simulation**: by default contains results of all iteration steps
  - the last one contains the correct result
  - use the highest possible value (obtained by pressing **All**) in both fields (**First** and **Last**)

- **Transient simulation**: load all (by pressing **All + Read File**)

![Diagram of read file options](image-url)
How to Read Files

- **Steady simulation**: by default contains results of all iteration steps
  - the last one contains the correct result
  - use the highest possible value (obtained by pressing **All**) in both fields (**First** and **Last**)
- **Transient simulation**: load all (by pressing **All** + **Read File**)
  - different time steps can be accessed using the **Timestep Control** entry in menu **Edit**
either for geometry manipulation:

- `translate 1 0 1`
- `rotate 90 0 0`
- `scale 1 10 1`
Command Window

- either for geometry manipulation:
  
  ```
  translate 1 0 1
  rotate 90 0 0
  scale 1 10 1
  ```

- or for MATC expressions:

  ```
  math on = nodes
  math nodes(2,time(0)) = 100*(on(2,time(0)) + Mesh.Update(2,time(0)))
  ```

Command Window

- either for geometry manipulation:

  
  \[
  \begin{align*}
  &\text{translate } 1 \ 0 \ 1 \\
  &\text{rotate } 90 \ 0 \ 0 \\
  &\text{scale } 1 \ 10 \ 1 \\
  \end{align*}
  \]

- or for MATC expressions:

  \[
  \begin{align*}
  &\text{math on } = \text{ nodes} \\
  &\text{math nodes}(2,\text{time}(0)) = 100 \times (\text{on}(2,\text{time}(0)) + \text{Mesh.Update}(2,\text{time}(0)))
  \end{align*}
  \]

- similar window also in **Timestep Control**: the current time level is addressed by \text{time}($t$)
Output for Other Post-processors

ResultOutputSolve offers during runtime:

Output Format =

- GID
- Gmsh
- Vtk
- VTK legacy
- Dx Format
- Open DX
- vtu
- ParaView
Output for Other Post-processors

• **ResultOutputSolve** offers during runtime:
  - **Output Format** =
    - GID
    - Gmsh
    - Vtk
    - VTK legacy
    - Dx Format
    - Open DX
    - vtu
    - ParaView

Solver 1
- **Equation** = "ResultOutput"
- **Procedure** = "ResultOutputSolve" "ResultOutputSolver"
- **Output File Name** = "test"
- **Output Format** = string "vtu"
- **Scalar Field 1** = String "Temperature"
- **Vector Field 1** = String "Velocity"

End
Output for Other Post-processors

ResultOutputSolve offers during runtime:

- GID
- Gmsh
- Vtk
- VTK legacy
- Dx Format
- Open DX
- vtu
- ParaView

Solver 1

- Equation = "ResultOutput"
- Procedure = "ResultOutputSolve" "ResultOutputSolver"
- Output File Name = "test"
- Output Format = string "vtu"
- Scalar Field 1 = String "Temperature"
- Vector Field 1 = String "Velocity"

End
ASCII Based Output

ResultOutputSolve offers during runtime:

```
Procedure = File "SaveData" ...
... SaveScalars  bulk output
... SaveLine     line output
... SaveMaterials material parameters added as variables
... SaveBoundaryValues boundary parameters added as variable
```