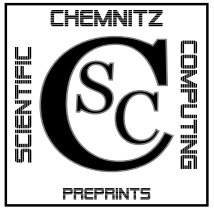
TECHNISCHE UNIVERSITÄT CHEMNITZ

Janine Glänzel Roman Unger

High Quality FEM-Postprocessing and Visualization Using a Gnuplot Based Toolchain

CSC/14-03



Chemnitz Scientific Computing Preprints

Chemnitz Scientific Computing Preprints – ISSN 1864-0087

Impressum:

Chemnitz Scientific Computing Preprints — ISSN 1864-0087
(1995–2005: Preprintreihe des Chemnitzer SFB393)

Herausgeber:
Professuren für
Numerische und Angewandte Mathematik
an der Fakultät für Mathematik
der Technischen Universität Chemnitz

Postanschrift:
TU Chemnitz, Fakultät für Mathematik
09107 Chemnitz
Sitz:
Reichenhainer Str. 41, 09126 Chemnitz

http://www.tu-chemnitz.de/mathematik/csc/

References

[BMP03] S. Beuchler, A. Meyer, and M. Pester. *SPC-PM3AdH v1.0 - Programmer's Manual*. Preprint SFB393 01-08 TU Chemnitz, 2003.

[Glä09] J. Glänzel. *Kurzvorstellung der 3D-FEM Software SPC-PM3AdH-XX*. Preprint CSC/09-03 TU Chemnitz, 2009.

[Glä14] J. Glänzel. *Korrektur thermoelastischer Verformungen durch den Einsatz der adaptiven FEM*. Eingereichte Dissertationsschrift, TU Chemnitz, 2014.

[Jan10] Philipp K Janert. *Gnuplot in action: understanding data with graphs*. Manning, Greenwich, CT, 2010.

[Mec04] Robert Mecklenburg. *Managing Projects with GNU Make*. O'Reilly Media, Sebastopol, 2004.

[Mey01] A. Meyer. *Programmer's Manual for Adaptive Finite Element Code SPC-PM 2Ad* . Preprint SFB393 01-18 TU Chemnitz, 2001.

[Mey14a] A. Meyer. *Programmbeschreibung SPC-PM3-AdH-XX Teil 1*. Preprint CSC/14-01 TU Chemnitz, 2014.

[Mey14b] A. Meyer. *Programmbeschreibung SPC-PM3-AdH-XX Teil 2*. Preprint CSC/14-02 TU Chemnitz, 2014.

TECHNISCHE UNIVERSITÄT CHEMNITZ

Chemnitz Scientific Computing Preprints

Janine Glänzel Roman Unger

High Quality FEM-Postprocessing and Visualization Using a Gnuplot Based Toolchain

CSC/14-03

Abstract

In the paper a toolchain for postprocessing and visualization of simulation datas, arising in finite element computations is described. This toolchain is based on gnuplot and free software. It is developed on the simulation problem of thermoelasticity, but it is not restricted on this problem class and adaptable on all other simulation problems, dealing with scalar- and vector fields. The most involved programs are gnuplot, perl, make and ffmpeg.

CSC/14-03 ISSN 1864-0087 July 2014

Contents

1. Introduction **1**

2. Simulation problem and emerging datas **1**

3. Gnuplot basics **3**
3.1. Temporal envelopment plots for single nodes (V1) 5
3.2. Usage possibilities for the epslatex-terminal 5
3.3. Automatism with Makefile and DoPic.pl 6

4. Mesh plotting (V2) **7**

5. Temperature field plotting (V3) **9**

6. Video creation (V4) **12**

7. Colormaps **12**
7.1. Colormap pictures and short description 14

8. Summary and outlook **16**

9. Acknowledgement **17**

A. Script-Listings **18**
A.1. Perlscript: DoPic.pl 18
A.2. Makefile 20

B. Colormap Sourcecodes **22**
B.1. Sourcecode: colormap_tg 22
B.2. Sourcecode: colormap_01 23
B.3. Sourcecode: colormap_99 24

Author's addresses:

Janine Glänzel
Fraunhofer-Institut für Werkzeugmaschinen und Umformtechnik IWU
Abteilung 120, Reichenhainer Straße 88, 09126 Chemnitz, Germany
janine.glaenzel@iwu.fraunhofer.de
http://www.tu-chemnitz.de/~glj

Roman Unger
TU Chemnitz, Fakultät für Mathematik
09107 Chemnitz, Germany
roman.unger@mathematik.tu-chemnitz.de
http://www.tu-chemnitz.de/~uro


```
            return ($red,$green,$blue);}

  if  ( $u < 0.2 )
      {
      $red    = 1.0-5*$u;
      $green  = 0;
      $blue   = 1;
      return ($red,$green,$blue);
      }

  if  ( $u < 0.4 )
      {
      $red   = 0;
      $green = 5*($u-0.2);
      $blue  = -5.0 * ($u-0.2) + 1.0;
      return ($red,$green,$blue);
      }

  if  ( $u < 0.6  )
      {
      $red = 5.0 * ($u-0.4);
      $green = 1;
      $blue  = 0;
      return ($red,$green,$blue);
      }

  if  ( $u < 0.8  )
      {
      $red = 1;
      $green = -5.0 * ($u-0.6) + 1.0;
      $blue  = 0;
      return ($red,$green,$blue);
      }

  # case u \in [ 0.8 , 1 ]
  $red   = 1;
  $green = 5.0 * ($u-0.8);
  $blue  = 5.0 * ($u-0.8);
  return ($red,$green,$blue);
}
```

```
      {
      $red    = 0;
      $green  = 4.0*$u;
      $blue   = 1;
      return ($red,$green,$blue);
      }

  if  ( $u < 0.5 )
      {
      $red   = 0;
      $green = 1;
      $blue  = -4.0 * ($u-0.25) + 1.0;
      return ($red,$green,$blue);
      }

  if  ( $u < 0.75  )
      {
      $red = 4.0 * ($u-0.5);
      $green = 1;
      $blue  = 0;
      return ($red,$green,$blue);
      }

  # case u \in [ 0.75 , 1 ]
  $red   = 1;
  $green = -4.0 * ($u-0.75) + 1;
  $blue  = 0;
  return ($red,$green,$blue);
}
```

B.3. Sourcecode: colormap_99

```
sub colmap_99
{
  my $u=shift;
  my $red;
  my $green;
  my $blue;

  if($u<0){ $red=1;$green=0;$blue=1;
            return ($red,$green,$blue);}
  if($u>1){ $red=1;$green=1;$blue=1;
```


1. Introduction

The numerical simulation of time dependent thermoelastical problems with adaptive finite-element methods leads to a large amount of simulation data. The postprocessing and visualization of this datas with the most tools, using graphical user interfaces is a job, consuming a lot of time. Especially for series of simulations an automated, script-based toolchain leads to short working times and moreover it assures repeatable results.

In the developed toolchain the following programs are involved:

gnuplot A free plot program, available for all platforms (Linux, Mac, Windows) to create the final plots and pictures. (see: `www.gnuplot.org`)

perl A scripting language, especially used for file handling and metascript generation. (see: `www.perl.org`)

make The classical make for resolving file dependencies. The normal usage of make is source code compiling, but it is useful for all purposes to handling, generating and updating files, depending on other files by evaluation of the file time stamps. (see: `www.gnu.org/software/make`)

ffmpeg A cross-platform program to convert, record and stream video (and audio too). It is used to visualize time dependent fem-solutions as video. (see: `www.ffmpeg.org`)

All of these programs are free available for the most computing platforms, the description here is done for a LINUX-platform, but is adaptable for Windows and Mac too.

2. Simulation problem and emerging datas

In this section a short description of the thermoelastic problem, exemplary used for the description of the postprocessing steps, is given, however the usefulness of the developed toolchain is not restricted to the class of thermoelasticity, it can be easily used or adapted for other simulation problems, for example for flow problems.

Inside the thermoelastic problem a temperature field $T(x,t)$ and a coupled displacement field $u(x,t)$ are searched such that the instationary heat equation for

all $t \in [0, t_{end}]$

$$\begin{aligned} c\varrho\frac{\partial T(x,t)}{\partial t} &= \nabla \cdot (\kappa \nabla\ T(x,t)) + f_T(x,t) + \bar{\gamma}(T - T_A) \qquad \forall x \in \Omega \\ T(x,0) &= T_0(x) \qquad \forall x \in \Omega \\ T(x,t) &= g_{T_D}(x,t) \qquad \forall x \in \Gamma_D \\ -\vec{n}^T\ (\kappa \nabla\ T(x,t)) &= g_{T_N}(x,t) \qquad \forall x \in \Gamma_N \\ -\vec{n}^T\ (\kappa \nabla\ T(x,t)) &= \gamma(T - T_a) \qquad \forall x \in \Gamma_R \end{aligned}$$

together with the coupled Lame-equation, describing the resulting displacement field,

$$\begin{aligned} -\mu\boldsymbol{\Delta}u(x,t) - (\lambda+\mu)\,\nabla\,\mathbf{div}\ u(x,t) &= \\ f(x,t) - (2\mu &+ 3\lambda)\alpha\nabla\ T(x,t) \quad \forall x \in \Omega \\ u(x,t) &= g_D(x) \qquad \forall x \in \Gamma_D \\ \sigma(u(x,t))\,\vec{n} &= g_N(x) \qquad \forall x \in \Gamma_N \end{aligned}$$

is fulfilled togehter with given boundary conditions. For a detailed description of the coupled thermoelastic problem and all terms see [Glä14].

Because the given problem is time dependent, the finite element simulation is done by a time stepping algorithm. During the finite element simulation of this thermoelastic problem vectors $\underline{T} \in \mathbb{R}^{N_T}$ and $\underline{u} \in \mathbb{R}^{N_u}$ are generated, holding the node values for the temperature field and displacement field. Together with the finite-element-mesh this are the source datas for visualization.

The main applications of visualization are:

(V1) Visualize the temporal development of a single node value at fixed position x_0 for example for the temperature value $T(x_0, t)$ or a displacement component $u_i(x_0, t)$ for $t \in [0, t_{end}]$.

(V2) Visualize a displaced finite-element-mesh at a fixed time value t_0 as a 2d-plot, e.g. $X + u(x, t_0)$.

(V3) Visualize a full scalar field like the temperature field, components of the displacement field or the norm of the displacement field at a fixed time value t_0 as a 2d-plot, f.e. $T(x, t_0)$.

(V4) Especially for points (V2) and (V3) sometimes the full time development is of interest, which can be done by generating appropriate image sequences and converting this sequences to a video with the help of a video compression tool like ffmpeg.

In the next sections the used tools for this applications will described.


```
    $blue  = 0;
    return ($red,$green,$blue);
    }

  if  ( $u < 0.75  )
    {
    $red = 1;
    $green = -4 * ($u-0.5) + 1.0;
    $blue  = 0;
    return ($red,$green,$blue);
    }

  if  ( $u < 0.9  )
    {
    $red = 1;
    $green = 0;
    $blue  = 6.66 * ($u-0.75);
    return ($red,$green,$blue);
    }

  # case u \in [ 0.9 , 1 ]
  $red   = 1;
  $green = 10.0 * ($u-0.9);
  $blue  = 1;
  return ($red,$green,$blue);
}
```

B.2. Sourcecode: colormap_01

```
sub colmap_01
{
  my $u=shift;
  my $red;
  my $green;
  my $blue;

  if($u<0){ $red=0;$green=0;$blue=1;
            return ($red,$green,$blue);}
  if($u>1){ $red=1;$green=0;$blue=0;
            return ($red,$green,$blue);}

  if  ( $u < 0.25 )
```

B. Colormap Sourcecodes

In the following subsections the color mapping functions, corresponding with the colormaps, shown in figures 6a, 7a, 8a are given in perl-syntax. The usage of this functions is calling `(my $red, my $green, my $blue)=colmap_XX($u)`.

B.1. Sourcecode: colormap_tg

```
sub colmap_tg
{
  my $u=shift;
  my $red;
  my $green;
  my $blue;

  if($u<0){ $red=0;$green=0;$blue=0;
            return ($red,$green,$blue);}
  if($u>1){ $red=1;$green=1;$blue=1;
            return ($red,$green,$blue);}


  if  ( $u < 0.2 )
      {
      $red    = 0;
      $green  = 0;
      $blue   = 5*$u;
      return ($red,$green,$blue);
      }

  if  ( $u < 0.4 )
      {
      $red   = 0;
      $green = 5*($u-0.2);
      $blue  = -5.0 * ($u-0.2) + 1.0;
      return ($red,$green,$blue);
      }

  if  ( $u < 0.5 )
      {
      $red   = 10 * ($u-0.4);
      $green = 1;
```


3. Gnuplot basics

In preparation for the special applications, given in (V1) ⋯ (V4) some basic informations about the used special features of **gnuplot** will given here. For the elementary steps in **gnuplot** exist a lot of literature and scripts in the web and books too, for example [Jan10].

The most simplistic case of usage for **gnuplot** is having a two column datafile `x.dat` like

```
0.00000     0.00000
0.10000     0.09983
0.20000     0.19867
...
```

which can be plotted inside **gnuplot** by calling

```
plot "x.dat"
```

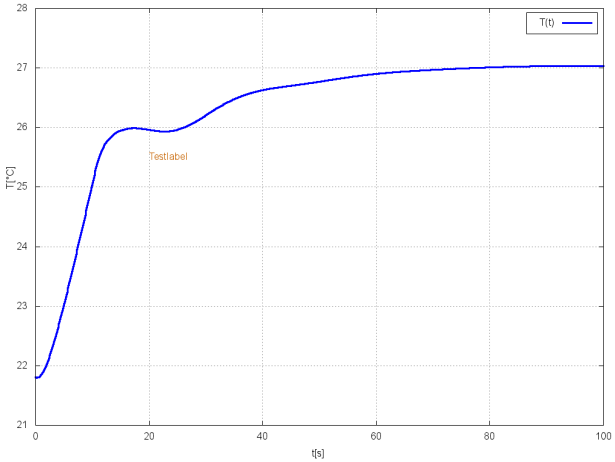
To use the plot in other documents exist in **gnuplot** a possibility to create a file in a choosen graphic format, therefore the so called 'terminal' variable is useful. In the example from above the plot will create as a picture in png-format (png: portable network graphic) by extend the plot call to

```
set terminal png
set output "x.png"
plot "x.dat"
```

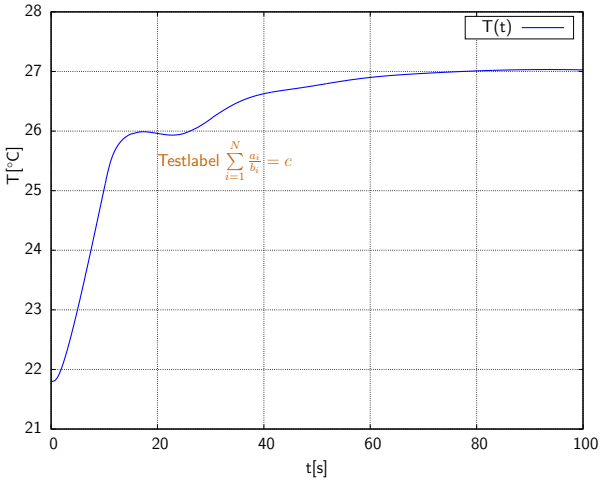
Here we start with some special features, like the used terminal. The terminal is the choosen output driver of **gnuplot**, it is possible to generate pixel based pictures like png or vector based pictures like eps or pdf, but one special point for the later usage of these generated pictures is the choosen font for titles, labels and so on. Using a complete by **gnuplot** generated picture in LaTeX is possible with a simple `\includegraphics` but the fonts will taken as they are and so they differ from the used font in the main document in most cases.

To overcome this disadvantage it is possible to separate the plot from the inscription by using special terminals in **gnuplot** like the `epslatex` - terminal. For a first impression in the figures 1a and 1b, the same plot is done with the terminal png (fig. 1a) and with the terminal epslatex (fig. 1b)

It is evident, that the quality of the second variant (fig. 1b) is much better, especially the font style and font size are the same like in the whole LaTeX-document, please note that the font in the plot is choosen without serifs. Especially the test label in figure 1b shows that the whole power of the math mode in LaTeX can be used inside **gnuplot** with appropriating terminals like epslatex.



(a) Example picture for terminal png



(b) Example picture for terminal epslatex

Figure 1: Comparison of different terminals


```
ALLPLOTS  = gnutest1.pdf
ALLPLOTS += colmapshow_cm99.pdf
ALLPLOTS += colmapshow_cm01.pdf
ALLPLOTS += colmapshow_cmtg.pdf

# The main target
all: $(ALLPLOTS)

# and the build rule
%.pdf : %.txt
       perl DoPic.pl  $<



.PHONY : clean realclean

ALLDVI =$(ALLPLOTS:.pdf=.dvi)
ALLEPS =$(ALLPLOTS:.pdf=.eps)
ALLTEX =$(ALLPLOTS:.pdf=.tex)
ALLTEXI=$(ALLPLOTS:.pdf=-input.tex)
ALLEPSI=$(ALLPLOTS:.pdf=-input.eps)
ALLPDFI=$(ALLPLOTS:.pdf=-input.pdf)
ALLPS  =$(ALLPLOTS:.pdf=.ps)
ALLLOG =$(ALLPLOTS:.pdf=.log)
ALLAUX =$(ALLPLOTS:.pdf=.aux)


clean :
rm -fv $(ALLDVI)  $(ALLEPS) $(ALLTEX) $(ALLTEXI)
rm -fv $(ALLEPSI) $(ALLPS)  $(ALLLOG) $(ALLAUX) $(ALLPDFI)
rm -fv $(ALLPLOTS)

realclean : clean
```

A.2. Makefile

```
#################################################################
#
#   Makefile to create all plots
#
#   Janine Glänzel, Roman Unger ... 06/2014
#
#################################################################
#
#   Copyright (C) 2014  Janine Glänzel, Roman Unger
#
#   janine.glaenzel@mathematik.tu-chemnitz.de
#   roman.unger@mathematik.tu-chemnitz.de
#
#   This program is free software: you can redistribute it
#   and/or modify it under the terms of the GNU General Public
#   License as published by the Free Software Foundation,
#   either version 3 of the License, or (at your option) any
#   later version.
#
#   This program is distributed in the hope that it will be
#   useful, but WITHOUT ANY WARRANTY; without even the implied
#   warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
#   PURPOSE.
#   See the GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public
#   License along with this program.
#   If not, see <http://www.gnu.org/licenses/>.
#
#################################################################


# The target files are PDFs.
#
# To get foo.pdf create a gnuplot file foo.txt
# and set in this gnuplotfile the output as
# set output 'foo-input.tex'
```


3.1. Temporal envelopment plots for single nodes (V1)

Plotting the temporal envelopment of single node values is one of the simplistic usages of **gnuplot**, the only thing to do is generating a 2- or more column datafile `gnu.dat` with the corresponding time in the first column and scalar plot values in column 2 (or more columns for other values). Then inside the **gnuplot** script this values can be visualized by calling

```
plot "gnu.dat"
```

which is the minimalistic variant. All other possibilities for tuning the picture can be used too. For getting qualitative good pictures the used terminal has an important influence, which will decribed in the next section in more detail.

3.2. Usage possibilities for the epslatex-terminal

The usage of the epslatex terminal in **gnuplot** is done like with all other file orientated terminals, at first the terminal must set with the `set terminal` command, then a output file name must be given, for example with

```
set terminal epslatex size 15cm,12cm color colortext dashed
set output 'gnutest1-input.tex'
```

This terminal definition together with all other plot commands is placed in a textfile `gnutest1.txt`.

Then a **gnuplot** call will produce 2 files, one file `gnutest1-input.tex` holding all the text, labels, etc. and the other file `gnutest1-input.eps` with the graphical output. There are now two possibilities to use this in the LaTeX document.

The first one is to include `test1-input.tex` with a simple

```
\input{gnutest1-input.tex}
```

The advantage of this variant is their simplicity, there is also no matter for using pdflatex, the epsfile `gnutest1-input.eps` must only converted to pdf with `epstopdf gnutest1-input.eps`.

The disadvantage is, that a scaling during the input is not possible and a scaling inside the terminal command collides in some cases with the font sizes. So sometimes a second variant with a separate LaTeX run, to produce a complete pdf-file and use this in the main document is the better one.

Therefore the following steps are needed:

1. Create `gnutest1.txt` and run gnuplot with this inputfile
2. Create a minimalistic LaTeX file, called `gnutest1.tex` with the lines

```
\documentclass[12pt]{article}
\usepackage{german,indentfirst,graphicx,color}
\usepackage{SIunits}
\usepackage{amsmath}
\pagestyle{empty}
\begin{document}
\begin{center}
  {   \sffamily   \input{gnutest1-input.tex}  }
\end{center}
\end{document}
```

3. Run LaTeX on this file with `latex gnutest1.tex`.
4. Convert the produced dvi-file to a postscript file `gnutest1.ps` with matching boundaries (option -E) by `dvips -E gnutest1.dvi`
5. Create `gnutest1.pdf` with `epstopdf gnutest1.ps`

Now one can use this in the main document with
`\includegraphics[width=XXcm]{gnutest1.pdf}`
to include the picture.

The advantage of this variant is an easy scaling after including, the disadvantage is that the given 5 steps must done for every plot. To overcome this disadvantage in the next section an automatism for this steps will shown.

3.3. Automatism with Makefile and DoPic.pl

To overcome the disadvantage of to much program calls for every plot at first a perlscript `DoPic.pl` is needed, which does the steps 1 to 5 from page 6 by a simple call: `perl DoPic.pl gnutest1.txt`. The script is not very difficult, it only has to manage some filenames and calls the matching programs. Sourcecode of the script is given in the appendix on page 18.

Please note that for correct work the following naming convention is mandatory: If the **gnuplot** inputfile is named `foo.txt` the outputname for the epslatex terminal inside this file **must** be foo-input.tex.

In principle all plots can handled only with this script `DoPic.pl` but sometimes it is nasty to run this by hand if the number of plots increases and some of the plots are changed because of some reason. So it is obvious to couple the script


```
if (@ARGV != 1) { die "usage: $0 gnuplotfile.txt\n";}

my $f=$ARGV[0];

my $stem=$f;
$stem =~ s/\.txt$//;
my $res;

# STEP 1 gnuplot, it creates stem-input.tex and stem-input.eps
$res = system("gnuplot $stem.txt");
die "gnuplot error res=$res\n" until ($res==0);

# STEP 2 create the latexfile stem.tex
open (TEX,">$stem.tex") || die "Cant write $stem.tex\n";
print TEX "\\documentclass[12pt]{article}\n";
print TEX "\\usepackage{german,indentfirst,graphicx,color}\n";
print TEX "\\usepackage{SIunits}\n";
print TEX "\\usepackage{amsmath}\n";
#print TEX "\\usepackage{amssymb}\n";
print TEX "\\pagestyle{empty}\n";
print TEX "\\begin{document}\n";
print TEX "\\begin{center}  {   \\sffamily   ";
print TEX "\\input{$stem-input.tex}  } \\end{center}\n";
print TEX "\\end{document}\n";
close TEX;

# STEP 3, run latex to get stem.dvi
$res = system("latex $stem.tex");
die "latex error res=$res\n" until ($res==0);

# STEP 4, run dvips -E to get stem.ps
$res = system("dvips -E $stem.dvi");
die "dvips error res=$res\n" until ($res==0);

# STEP 5, run epstopdf to get stem.pdf
$res = system("epstopdf $stem.ps");
die "epstopdf error res=$res\n" until ($res==0);

# STEP 6, run epstopdf on stem-input.eps  to get stem-input.pdf
# which allows direct including of the stem-input.tex
$res = system("epstopdf $stem-input.eps");
die "epstopdf error res=$res\n" until ($res==0);
```

A. Script-Listings

A.1. Perlscript: DoPic.pl

```
#!/usr/bin/perl
#
# Create foo.pdf togehter with many intermediate files
# from gnuplot file foo.txt
#
# Please use the naming convention, that in a
# gnuplot file foo.txt  the output is set to:
# set output 'foo-input.tex'
#
# Janine Glänzel, Roman Unger ... 06/2014
#
################################################################
#
#   Copyright (C) 2014  Janine Glänzel, Roman Unger
#
#   janine.glaenzel@mathematik.tu-chemnitz.de
#   roman.unger@mathematik.tu-chemnitz.de
#
#   This program is free software: you can redistribute it
#   and/or modify it under the terms of the GNU General Public
#   License as published by the Free Software Foundation,
#   either version 3 of the License, or (at your option) any
#   later version.
#
#   This program is distributed in the hope that it will be
#   useful, but WITHOUT ANY WARRANTY; without even the implied
#   warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
#   PURPOSE.
#   See the GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public
#   License along with this program.
#   If not, see <http://www.gnu.org/licenses/>.
#
################################################################

use strict;
```


with a makefile to use make to recreacte all updated plots. For the principal usage of `make` see e.g. [Mec04] or other free available documentations. The only speciality in the Makefile, given in the appendix on page 20, is the build rule for the pdf-file, depending on the corresponding textfile for **gnuplot** by:

```
%.pdf : %.txt
	perl DoPic.pl  $<
```

which is reponsible for calling the perlscript `DoPic.pl` for every picture, which needs an update. The used pictures are given inside the Makefile by define and expand the variable `ALLPLOTS` e.g.

```
ALLPLOTS  = gnutest1.pdf
ALLPLOTS += colmapshow_cm01.pdf
ALLPLOTS += colmapshow_cmtg.pdf
ALLPLOTS += gnu1_epslatex.pdf
 ...
```

Placing `Makefile`, `DoPic.pl` and all the textfiles for **gnuplot** together with all datas in a subdir, called `gnuplots` only a simple `make` in this directory is needed to (re-)create all the plots. To use it in the main LaTeX file don't forget to expand the graphicspath by addinge the line

```
\graphicspath{{./gnuplots/}}
```

at the beginning of the main document. As an extension inside the makefile a section called 'clean' is defined, by calling `make clean` or `make realclean` all outputfiles will deleted.

Some example scripts and data files for this section and later sections too are given at the authors webpages:
`http://www.tu-chemnitz.de/~glj/publications/CSC14-03`
`http://www.tu-chemnitz.de/~uro/publications/CSC14-03`

4. Mesh plotting (V2)

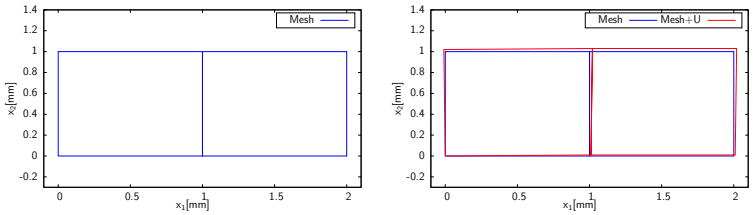
To plot a finite element mesh with gnuplot, a datafile must created either inside the FE-program or out of output datas with the help of a script in a postprocessing step. However the structure of the datafile for **gnuplot** must hold the structure like in the following simple example.

For every (quadrangular) finite element the node coordinates (x_1, x_2) and displacement values (u_1, u_2) are given for all 4 nodes and node 1 again, followed by an empty line. As an example datafile let's create a file `mp1.txt` with the following content.

```
#  x1    x2       u_1      u_2
  0.0   0.0       0.0      0.0
  1.0   0.0       0.001    0.001
  1.0   1.0       0.002    0.003
  0.0   1.0      -0.001    0.002
  0.0   0.0       0.0      0.0

  1.0   0.0       0.001    0.001
  2.0   0.0       0.001    0.001
  2.0   1.0       0.002    0.003
  1.0   1.0       0.002    0.003
  1.0   0.0       0.001    0.001
```

Here two elements with displacement values are given. This are a unit square and a unit square moved by one unit in x_1 direction. Columnwise the values for the node coordinates (x_1, x_2) and displacement values (u_1, u_2) are given. The corresponding plots for the mesh and the displaced mesh are shown in fig. 2.



(a) mesh only (b) mesh and displacement

Figure 2: Simple meshplot example

To create a mesh plot like in fig. 2a the **gnuplot**-command is

```
plot "mp1.txt" title "Mesh"  with lines linetype 1 \
     linecolor rgb "#0000FF"  linewidth 3
```

where `mp1.txt` is the datafile, described above, holding the element values.

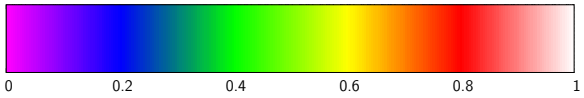
It is also possible to add the displacement values to the node values for creating plots like in fig. 2b. In most practical cases the displacement values are so small, that in a plot of $x+u$ one can not distinguish between the mesh and the displaced mesh. In such cases a it is useful to exaggerate the displacement values. All this can be done inside the **gnuplot**-command, without changing the datafile. As an example see the command which produced the plot of $x + 10u$, shown in fig. 2b.

short working times. Especially for video creation, where a lot of single pictures (approx. 1500 frames for 1 minute video) are needed, it makes it possible to create this datas. For full 3-dimensional problems a script based visualization with the help of a raytracer (e.g. PovRay) will shown in a future work. Example scripts and data files are given at the authors webpages:
`http://www.tu-chemnitz.de/~glj/publications/CSC14-03`
`http://www.tu-chemnitz.de/~uro/publications/CSC14-03`

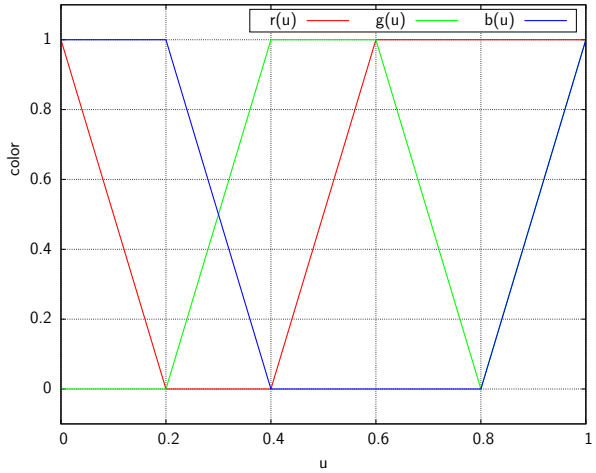
9. Acknowledgement

This research is supported by the Deutsche Forschungsgemeinschaft (DFG) in context of the Collaborative Research Centre/Transregio 96 thermo-energetic design of machine tools.

The colormap **colormap_99** (shown in figure 8 with sourcecode on page 24) is used inside the adaptive FEM-programs 'SPC-PM2Ad' (see [Mey01]) or 'SPC-PM3AdH-XX' (see [Glä09], [Mey14a], [Mey14b], [BMP03]) when choosing '99 cols'. Figure 8a is an example for the color mapping and in figure 8b the color mapping functions (1),(2),(3) for the red, green and blue color channel are shown.



(a) Example for colormap_99



(b) Color mapping functions

Figure 8: Colormap '99 cols' out of SPC-PM2Ad: colormap_99

8. Summary and outlook

In this paper a toolchain for producing plotfiles or video sequences in a pleasant quality was developed. Especially for 2-dimensional problems, arising from the finite element code 'SPC-PM2Ad' (see [Mey01]) or surface- and slice plots in the 3-dimensional finite element code 'SPC-PM3AdH-XX' (see [Glä09], [Mey14a], [Mey14b], [BMP03]) this toolchain leads to good and reproducible pictures in


```
plot "mp1.txt" title "Mesh"  with lines linetype 1 \
          linecolor rgb "#0000FF"  linewidth 3 ,\
     "mp1.txt" using ($1+10*$3):($2+10*$4) \
          title "Mesh+U"  with lines linetype 1 \
          linecolor rgb "#FF0000"  linewidth 3
```

Note the `using ($1+10*$3):($2+10*$4)` part of the plot command, it calls **gnuplot** to use column 1 plus 10 times column 3 to plot it against column 2 plus 10 times column 4, which leads to an exaggerated displacement plot $x + 10u$.

For an example with a 'real' finite element mesh see fig. 3 where an example plot for a thermoelastic problem is shown [Glä14]. The plot consists of approximatly 1000 finite elements.

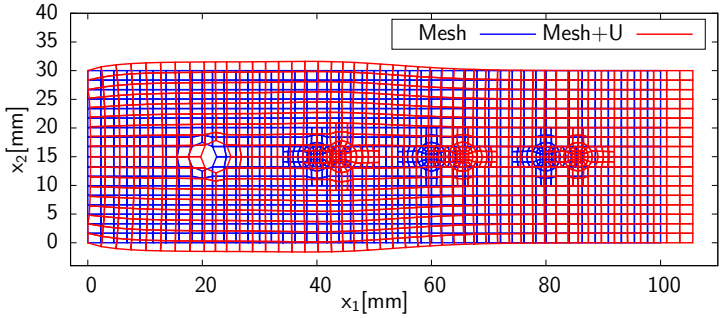


Figure 3: Example picture for mesh plot [Glä14]

5. Temperature field plotting (V3)

Plotting two dimensional colorflows with **gnuplot** is possible by using **gnuplot**'s splot together with pm3d and the palette definition. However, it is sometimes very complicated to achieve the needed results on this way.

A more flexible way to achieve this destination is the usage of simple plotting colored polygons and compute the colors before plotting. Plotting colored polygons can be done with inline data plot, for understanding again a simple example with 3 colored quadrangles.

The first step is to compute inside the finite element program or by script out of some datafiles from the output of the FE-program a color value for every finite element.

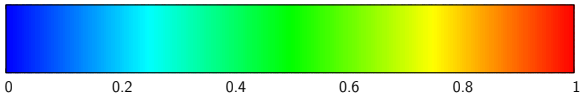
Let's say we computed the color red (#FF0000) for element 1, the color green (#00FF00) for the second and the color blue (#0000FF) for the third element. All elements must be plotted with a single plot call, coordinates are given inline in the plot command, closed by the letter 'e'. For these 3 elements the plot command is:

```
plot [-0.2:3.2] [-0.1:1.1] \
 "-" title "" with filledcurve lc rgb "#ff0000" \
              fillstyle transparent solid 1.000000 ,\
 "-" title "" with filledcurve lc rgb "#00ff00" \
              fillstyle transparent solid 1.000000 ,\
 "-" title "" with filledcurve lc rgb "#0000ff" \
              fillstyle transparent solid 1.000000 ;
  0.0   0.0
  1.0   0.0
  1.0   1.0
  0.0   1.0
  0.0   0.0
e
  1.0   0.0
  2.0   0.0
  2.0   1.0
  1.0   1.0
  1.0   0.0
e
  2.0   0.0
  3.0   0.0
  3.0   1.0
  2.0   1.0
  2.0   0.0
e
```

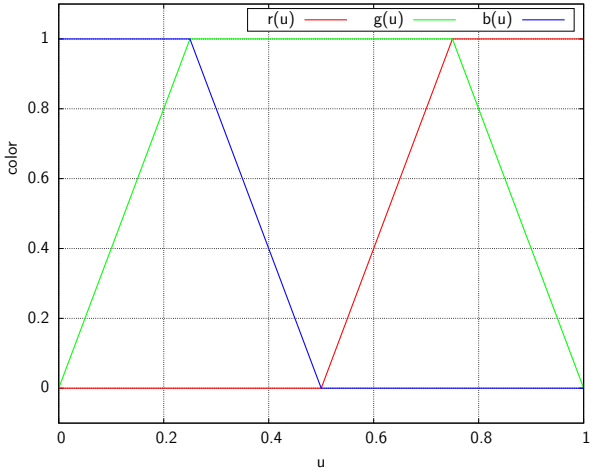
Please note that the plot lines separated by a comma but the last line is closed with a semicolon. Also the number of plot lines and the number of coordinate blocks, closed by letter 'e' must match exactly. The result of given plot command is shown in figure 4.

For practical relevant cases the used colorflow depends on the underlying datas, some basics and examples for colormaps are given in section 7 togehter with the source code of the color mapping functions in appendix B. In figure 5 an example for a temperature field plot, corresponding with the mesh and displacement of figure 3 is shown. Here the 'multiplot' functionality of **gnuplot** is used to plot the temperature field togehter with the color bar into a single plot. The plot

The colormap **colormap_01** (shown in figure 7 with sourcecode on page 23) is a simple rainbow color flow used in many visualization tools like matlab, octave etc. Figure 7a is an example for the color mapping and in figure 7b the color mapping functions (1),(2),(3) for the red, green and blue color channel are shown.



(a) Example for colormap_tg

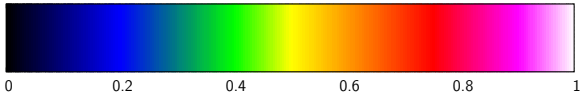


(b) Color mapping functions

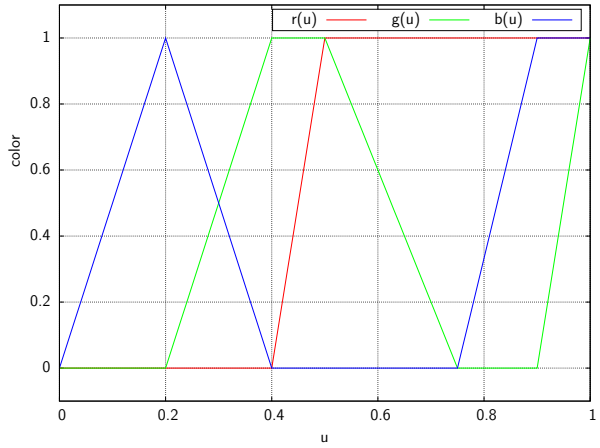
Figure 7: Simple rainbow colormap: colormap_01

7.1. Colormap pictures and short description

The colormap **colormap_tg** (shown in figure 6 with sourcecode on page 22) is used inside a commercial visualization program, coming together with a thermographic camera. Figure 6a is an example for the color mapping and in figure 6b the color mapping functions (1),(2),(3) for the red, green and blue color channel are shown.



(a) Example for colormap_tg



(b) Color mapping functions

Figure 6: Colormap for a thermographic camera: colormap_tg

consist of 4310 quadrangular elements, so the **gnuplot**-script with all this lines and coordinate values must generated from the output data of the finite element program with a script, here this is done with a perl-script.

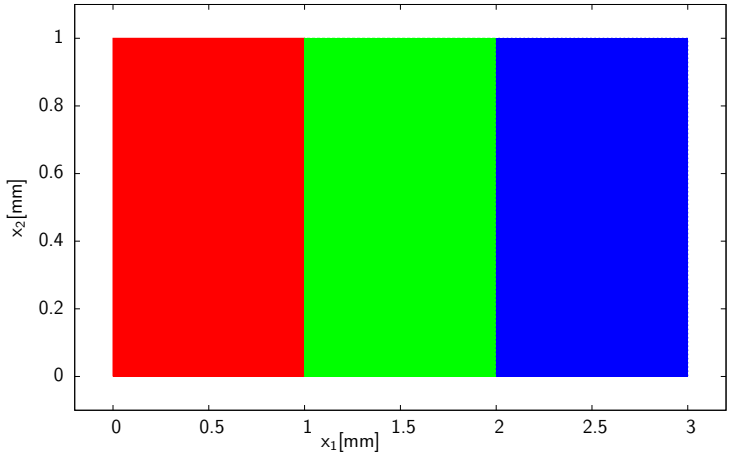


Figure 4: Simple example for temperature plot

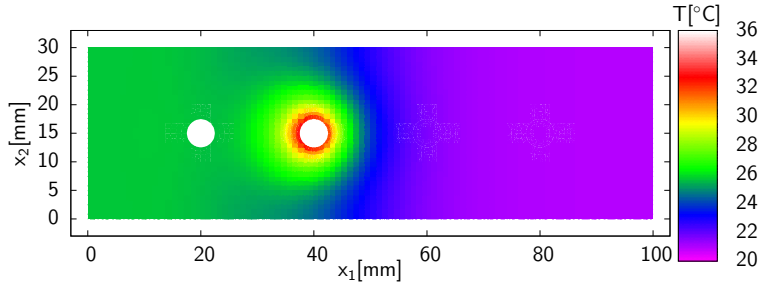


Figure 5: Example picture for temperature plot [Glä14]

6. Video creation (V4)

Especially for time dependent datas often a visualization with a video is one of the most impressive variants. In preparation of a video all the single frames must created. The so called 'frame rate' of a video is the number of pictures per second, values of 25 frames per second are normal. All this frames must generated with a script (e.g. perl) as picture files, for example in portable network graphics (png) format and named in a convention like `frame-012345.png`.

Then this frames can converted to a mp4-video with the help of ffmpeg by calling

```
ffmpeg -r 25 -f image2 -i upng/frame-%06d.png \
       -vcodec libx264 -crf 15  test.mp4
```

which creates a mp4 (h.264) format video which should play on the most platforms.

The given options to ffmpeg are in detail:

-r 25 frame rate 25 frames per second

-f image2 input data is a image sequence

-i upng/frame-%06d.png input files are in subdir `upng` and named by `frame-` followed by a six digit number with leading zeros + extension `.png`

-vcodec libx264 video coding is done with libx264, which produces a video in h.264 (known as mp4) format

-crf 15 a quality switch, small values means better quality, usual: 15-20

There are many more options and possible video formats which are useable with ffmpeg, for more details see `www.ffmpeg.org`.

7. Colormaps

In this section some examples of colormaps, used for the temperature plots are shown. The corresponding source code can be found in appendix B. In all cases a colormap is a function $c : \mathbb{R} \to [0,1]^3$, mapping a given value T to an RGB-triplet $[r(T), g(T), b(T)]^T$ with the one dimensional color mapping functions r,g,b

$$r : \mathbb{R} \to [0,1] \quad \text{mapping for color red} \tag{1}$$
$$g : \mathbb{R} \to [0,1] \quad \text{mapping for color green} \tag{2}$$
$$b : \mathbb{R} \to [0,1] \quad \text{mapping for color blue.} \tag{3}$$

However, for using the whole color range, the first step for arbitrary values $T \in \mathbb{R}$ is mapping these values to the interval $[0,1]$ which can be done after computing the boundary values T_{min} and T_{max} by an affin linear transformation

$$\tilde{T} : [T_{min}, T_{max}] \to [0,1]$$

defined by

$$\tilde{T}(T) := \frac{T - T_{min}}{T_{max} - T_{min}}.$$

Then the correponding color can be computed by one of the color mapping functions, given in the next subsection with an example color bar. The source code can be found in appendix B, the usage of this functions is calling
`(my $red, my $green, my $blue)=colmap_XX($T)` .