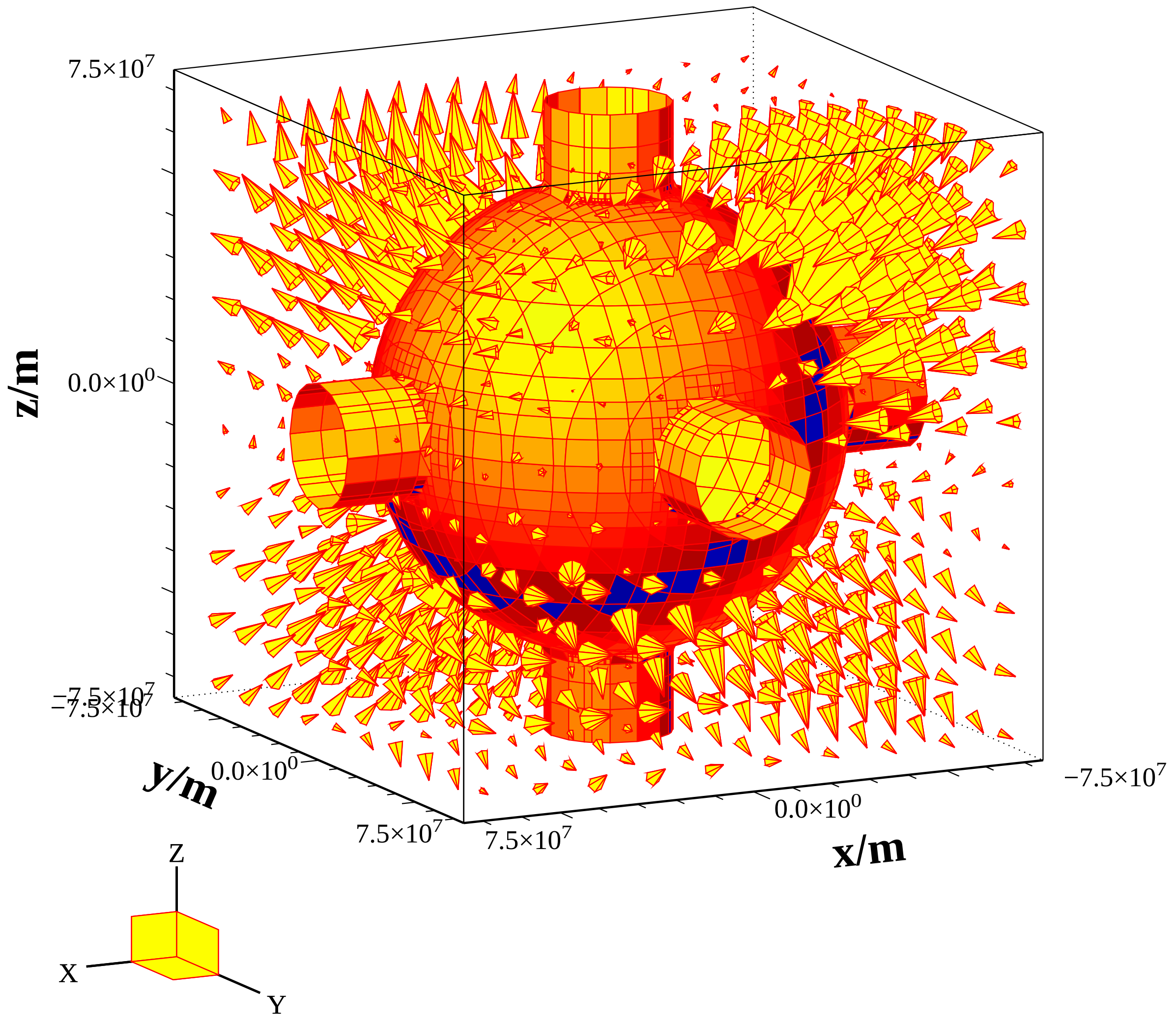
GdfidL v1: Syntax and Semantics

Warner Bruns



October 20, 1999

[1]GdfidL is the abbreviation for "Gitter drüber, fertig ist die Laube"

Contents

List of Figures

0.1 Introduction

Features:

- The geometry is approximated by generalized diagonal fillings. This reduces the approximation error typically by a factor of ten.
- Periodic boundary conditions for all three cartesian directions can be specified simultaneously for eigenvalue computations.
- The time domain computation uses "Perfectly Matched Layers" for it's absorbing boundary conditions.
- The programs can be run interactively. All commands are explained when you issue the special command **help**.
- The solver and the postprocessor have a built-in macro processor.
  - You can define symbols storing numerical values or character values.
  - Arithmetic expression evaluator.
  - You can groups of commands as **macro**'s. The **macro**'s can have an unlimited amount of parameters.
  - 'do-loop's are implemented.
  - 'if' 'endif'
- Almost all commands may be abbreviated as long the abbreviation is unique in context.

GdfidL consists of 3 separate programs that work together. These programs are:

- **gd1** & **single.gd1**: These programs read the description of the problem and compute resonant fields or time dependent fields. **gd1** computes in double precision, while **single.gd1** computes in single precision. **single.gd1** needs somewhat less memory and often less cpu-time.
- **gd1.pp** : This is the postprocessor. It displays the fields, computes integrals over the fields to compute quality factors and the like. It also computes scattering parameters and wakepotentials from data that have been computed by **gd1** or **single.gd1**.
- **mymtv2**: This program displays the plots on a X11 terminal and produces PostScript files

mymtv2 was not written by me, but is a modified version of the program *plotmtv* written by Kenny Toh. The original sources for plotmtv can be found on the internet: plotmtv can be found at ftp.x.org: /contrib/applications/Plotmtv.1.3.2.tar.Z

Chapter 1

gd1

1.1 Typical usage of gd1

gd1 reads its information from the standard input unit. You will normally use two or more **xterm**'s to operate **gd1**. In one of the **xterm**'s you edit an inputfile that describes your geometry, in the other **xterm** you iteratively start **gd1** and try out how **gd1** reacts on your input.

There is no standalone meshgenerator. **gd1** has its meshgenerator built in.

gd1 reads the information about the geometry that you are interested in from stdin or from a file that you specify via **include(filename)**. **gd1** generates the mesh and computes the resonant fields or time dependent fields, depending on the input you give him. The results are written to a database that can be read by **gd1.pp**.

1.2 Input for gd1

gd1's input is pure text. You give **gd1** the information about your problem in distinct sections.

The required information that you have to give **gd1** is

- The name of the resultfile. (-general)
- The borders of the computational volume. (-mesh)
- The wanted mesh-spacing (-mesh).
- The boundary conditions at the borders of the computational volume. (-mesh) (-fdtd/-ports)
- The description of the geometry. (-brick, -gccylinder, -ggcylinder, -gbor, -stlfile)
-
  - For an eigenvalue computation:
    * The wanted number of frequencies. (-eigenvalues)

    * An estimation of the highest frequency. (-eigenvalues)
  - For a time domain computation:
    * The location of ports. (-fdtd/-ports)
    * The excitation. (-fdtd/-pexcitation)
    * The minimal time to be simulated. (-fdtd/-time)
    * The maximal time to be simulated. (-fdtd/-time)

The symbols in brackets above indicate in which section of **gd1** you specify the required parameters.

1.3 Behind the scenes: When the mesh is generated

When you define a geometric primitive, the only thing that happens immediately is that the information about the geometric primitive is stored in an internal database. When you say **doit** in the section **-eigenvalues** or in the section **-fdtd**, the mesh is generated on the fly. When you display the mesh-filling via the section **-volumeplot**, the mesh is also generated on the fly.

1.4 General sections

These are the sections where you specify parameters that are required for every field computation. In the section **-general**, you specify the name of the file where the results shall be written to. In the section **-mesh**, you have to specify the borderplanes of the computational volume, and the default mesh spacing. You also the boundary conditions at the borderplanes here. In the section **-material**, you specify the electric properties of the materials.

1.4.1 Entry Section

```
##############################################################################
# Flags: nomenu, noprompt, nomessage,                                        #
##############################################################################
# gdfidl (V 1.0) ( Sun Oct 17 17:54:54 GMT 1999, compiled on host: wb003 )   #
##############################################################################
# -general  -- Output file, annotations.                                     #
# -mesh     -- Bounding box, spacing, fixed meshplanes, boundary conditions. #
# -material -- Material properties.                                          #
# ***** Geometric primitives ****                                            #
# -brick      -- Simple rectangular brick.                                   #
# -gccylinder -- Circular cylinder in general direction.                     #
# -ggcylinder -- General cylinder in general direction.                      #
# -gbor       -- Body of revolution in general direction.                    #
# -stlfile    -- CAD import via STL-file.                                    #
# ***** Solver Sections ****                                                 #
#  -eigenvalues -- Resonant fields                                           #
#  -fdtd        -- Time dependent fields                                     #
# *******                                                                    #
#  -volumeplot  -- displays mesh filling.                                    #
# ***** Miscalleneous ******                                                 #
#  -debug       -- Specify debug levels                                      #
#                                                                            #
##############################################################################
# ?, end, help                                                               #
##############################################################################
```

This is the section you are in when you start **gd1**. The menu shows all sections currently accessible. The section `-fdtd` has some subsections. You enter a section by specifying its name.

Example
To enter the section `-general`, you say:

```
-general
```

1.4.2 -general : Annotations, filenames

```
################################################################################
# Flags: nomenu, noprompt, nomessage,                                          #
################################################################################
# section: -general                                                            #
################################################################################
# outfile    = /tmp/--username--/--SomeDirectory--/Results                     #
# scratchbase= /tmp/bruw1931/garbage/gdfidl-scratch-pid=10923-                 #
# text( 1)= ' '                                                                #
# text( 2)= ' '                                                                #
# text( 3)= ' '                                                                #
# text( 4)= ' '                                                                #
# text( 5)= ' '                                                                #
# text( 6)= ' '                                                                #
# text( 7)= ' '                                                                #
# text( 8)= ' '                                                                #
# text( 9)= ' '                                                                #
# text(10)= ' '                                                                #
# text(11)= ' '                                                                #
# text(12)= ' '                                                                #
# text(13)= ' '                                                                #
# text(14)= ' '                                                                #
# text(15)= ' '                                                                #
# text(16)= ' '                                                                #
# text(17)= ' '                                                                #
# text(18)= ' '                                                                #
# text(19)= ' '                                                                #
# text(20)= ' '                                                                #
################################################################################
# ?, return, help                                                              #
################################################################################
```

- **outfile** The name of the file[1] where the solutions shall be stored in. This file may exist. If it exists, then it will be overwritten.

- **scratchbase** The base name of scratchfiles that **gd1** needs for its operation. If you have an environment variable `TMPDIR` set, SOLVER will as default value set `scratchbase` to the string

```
    $TMPDIR/gdfidl-scratch-pid=XXXXX-
```

[1]This will actually be a directory. The reason is: For current (1999) workstations, it is quite easy to generate data in excess of 2GBytes. But many current filesystems cannot handle files larger than 2GBytes. So we choose to implement the computed data as a hierarchy of files. A Directory is just a hierarchy of files, so we just used that.

Here `$TMPDIR` is the value of the environment variable `TMPDIR`, and `XXXXX` is the number of the process-id of **gd1**.

- **text(*)=** This annotation text is plotted together with the field plots. This is especially useful for documenting the geometry parameters of a calculation.
  syntax:
  ```
   text()= ANY STRING, NOT NECESSARILY QUOTED
  ```
  or
  ```
   text(NUMBER)= ANY STRING, NOT NECESSARILY QUOTED
  ```

  - `ANY STRING, NOT NECESSARILY QUOTED`:
    The string to be included in the plots,
  - `NUMBER`:
    Optionally, the line number, where the text should be plotted.

  In the first case, without `NUMBER`, the string following `text()=` is placed in the next free line. In the case with `NUMBER`, it is guaranteed, that the string is placed in the `NUMBER`.st line. You can specify up to 20 annotation strings, the maximum length of each annotation string is is 80 characters.

Example
The following specifies that the results of the current computation shall be stored in the database with name '/tmp/bruw1931/garbage/resultdirectory'. The names of scratchfiles that **gd1** generates shall start with '/tmp/bruw1931/garbage/delete-me-'. Together with the plots that **gd1.pp** will produce, the string 'strange what one can do with this beast' shall appear.

```
 -general
     outfile= /tmp/bruw1931/garbage/resultdirectory
     scratchbase= /tmp/bruw1931/garbage/delete-me-

     text(1)= strange what one can do with this beast
```

1.4.3 -mesh : outer boundary conditions, spacings

gd1 needs to know

- what the boundaries of the computational volume shall be,
- what thee default spacing between mesh-planes shall be,
- what the boundary conditions at the outer planes of the compuational volume shall be.

You specify these values in this section.

```
################################################################################
# Flags: nomenu, noprompt, nomessage,                                          #
################################################################################
# section -mesh                                                                #
################################################################################
## Bounding box:                                                               #
#   pxlow = undefined      , pylow = undefined      , pzlow = undefined      #
#   pxhigh= undefined      , pyhigh= undefined      , pzhigh= undefined      #
#  volume= (undefined, undefined, undefined, undefined, undefined, undefined)#
## Boundary conditions:                                                        #
#   cxlow = electric, cylow = electric, czlow = electric                      #
#   cxhigh= electric, cyhigh= electric, czhigh= electric                      #
## periodic bc's for eigenvalues:                                              #
#   xperiodic= no , xphase= undefined                                          #
#   yperiodic= no , yphase= undefined                                          #
#   zperiodic= no , zphase= undefined                                          #
# # # # #                                                                      #
# spacing = undefined, minspacing= undefined                                   #
# graded  = no                                                                 #
# qfgraded= 1.20, dmaxgraded= undefined                                        #
## Commands: # # # # # # # # # #                                               #
# xfixed(N, X0, X1)                                                            #
# yfixed(N, Y0, Y1)                                                            #
# zfixed(N, Z0, Z1)                                                            #
################################################################################
# return, help                                                                 #
################################################################################
```

- **pxlow, pxhigh, pylow, pyhigh, pzlow, pzhigh, volume**
  (Plane at XLOW ...) These values are the coordinates of the planes that bound the computational volume.
  THESE PARAMETERS ARE MANDATORY.
  Alternatively to specifying via pxlow.., you can specify the volume of your computational volume as **volume= (XL, XH, YL, YH, ZL, ZH)**.
  gd1 ignores all parts of items that are specified outside of the box with these boundary planes.

- **cxlow, cxhigh, cylow, cyhigh, czlow, czhigh**
  (Condition at XLOW ...) These are the boundary conditions at the boundary planes of the computational volume. Possible values are `electric, magnetic`.

- **xperiodic, yperiodic, zperiodic**
  Possible values are `yes, no`. Toggles the application of periodic boundary conditions between the lower and upper boundary planes in x-, y- or z-direction. You can compute with more than one `?periodic= yes`. This is only implemented for eigenvalue computations. For time-domain computation, these parameters are ignored.

- **xphase, yphase, zphase**
  Specification of the phase (in degrees) to enforce between the lower and upper boundary planes, when **?periodic= yes**.

- **spacing**
  The average spacing, that **gd1** should use for discretizing the computational volume.
  THIS PARAMETER IS MANDATORY.
  Of course, specifying a small spacing will lead to a good discretization. Be aware however, that the memory requirement is proportional to $1/(spacing)^3$ and the CPU-time approximately to $1/(spacing)^{3.5}$.

- **minspacing**
  The smallest mesh-spacing allowed.
  gd1 tries to place meshplanes approximately homogeneously over the volume. It is guaranteed, that there are meshplanes at the boundary planes of `bricks`, and where you enforce them via `xfixed, yfixed, zfixed`, **If** the distance between such fixed meshplanes is more than `minspacing`. Otherwise, one of these fixed meshplanes is deleted, it is the meshplane with the higher coordinate value.

  If this parameter **minspacing** is not given, **gd1** uses the value of **spacing**/10 instead.

- **graded**
  Possible values are `yes, no`.
  If `graded= yes`, the space between fixed meshplanes will be filled with a graded mesh. The ratio of adjacent mesh-spacing will be approx. `qfgraded`, The largest mesh-spacing will be less or equal `dmaxgraded`.

- **qfgraded, dmaxgraded**
  These are the parameters that are mentioned above in the discussion of **graded**.

- **xfixed, yfixed, zfixed**
  gd1's mesh generator fills the computational volume in principle homogeneous (if `graded= no`). It is only guaranteed, that the borders of `brick`'s

lie on meshlines. Therefore, if you have a geometry, where geometric items other than bricks are in, you can improve the mesh by enforcing meshlines with these commands.

Syntax:

```
xfixed(NUMBER, LOW, HIGH)
yfixed(NUMBER, LOW, HIGH)
zfixed(NUMBER, LOW, HIGH)
```

- `NUMBER`:
  The number of meshlines to place in between `LOW`, `HIGH`. The total number of meshlines enforced by such a command is exactly `NUMBER`. In the case of `xfixed`, the meshlines are positioned at positions x_i:

$$x_i \quad = \quad LOW + (i-1)\frac{HIGH - LOW}{\max(1, NUMBER - 1)}; \quad i = 1, (1), NUMBER$$

- `LOW, HIGH`:
  The first and last coordinates of the meshlines to be enforced.

Note:
It is possible to specify `NUMBER` ≤ 0, in this case nothing happens.
It is possible to specify `NUMBER`=1, in this case a single meshplane at `LOW` is enforced.
It is not necessary, that `LOW` is really lower than `HIGH`.

Example

```
-mesh
    pxlow= 1e-2, pxhigh= 0
    pylow= 2e-2, pyhigh= 0
    pzlow= 3e-2, pzhigh= 0

    cxlow= ele, cxhigh= mag
    cylow= ele, cyhigh= mag

    zperiodic= yes, zphase= 120

    spacing= 1e-3
```

1.4.4 -material : electric / magnetic properties

```
###############################################################################
# Flags: nomenu, noprompt, nomessage,                                         #
###############################################################################
# section -material                                                           #
###############################################################################
# material=  3        # epsr    = undefined      # kappa    = undefined      #
# type= undefined     #    xepsr= undefined      #   xkappa = undefined      #
#                     #    yepsr= undefined      #   ykappa = undefined      #
#                     #    zepsr= undefined      #   zkappa = undefined      #
#                     # muer    = undefined      # mkappa   = undefined      #
#                     #    xmuer= undefined      #   xmkappa= undefined      #
#                     #    ymuer= undefined      #   ymkappa= undefined      #
#                     #    zmuer= undefined      #   zmkappa= undefined      #
###############################################################################
# return, help                                                                #
###############################################################################
```

- **material**
  The material index of the material whose parameters are to be changed. This number must be between 0 and 50 inclusive.

- **type**
  The type of the material. Possible values are "electric", "magnetic", "normal". An "electric" material is treated as perfect electric conducting for the field computation, a "magnetic" material is treated as perfect magnetic conducting in the field computation.
  - For eigenvalue computations, only the "epsr" and "muer" of a "normal" material are used for the field computation, ie. no losses are modeled for eigenvalue computations.
  - For time domain computations, "epsr", "muer", "kappa" and "mkappa" of a "type= normal" material are used for the field computation.
  - Both for time domain and for eigenvalue computations, materials with "type= electric" are considered perfectly conducting, and materials with "type= magnetic" are considered perfectly magnetic conducting. Any specified electric conductivities for materials with "type= electric" are used in the postprocessor, **gd1.pp**, to compute wall losses via a perturbation formula.

- **epsr**
  The relative permittivity of the material. If you specify eg. `epsr= 3`, all three epsr values xepsr, yepsr and zepsr are set to the value 3.

- **xepsr, yepsr, zepsr**
  The x-, y-, z-value of an anisotropic material. Only diagonal epsr matrices can be specified.

- **muer**
  The relative permeability of the material. If you specify eg. `muer= 4`, all three muer values xmuer, ymuer and zmuer are set to the value 4.

- **xmuer, ymuer, zmuer**
  The x-, y-, z-value of an anisotropic material. Only diagonal muer matrices can be specified.

- **kappa**
  The electric conductivity of the material in MHO/m (1/Ohm/m). If you specify eg. `kappa= 5`, all three kappa values xkappa, ykappa and zkappa are set to the value 5.

- **xkappa, ykappa, zkappa**
  The x-, y-, z-value of an anisotropic material. Only diagonal kappa matrices can be specified.

- **mkappa**
  The magnetic conductivity of the material in Ohm/m. If you specify eg. `mkappa= 6`, all three kappa values xmkappa, ymkappa and zmkappa are set to the value 6.

- **xmkappa, ymkappa, zmkappa**
  The x-, y-, z-value of an anisotropic material. Only diagonal mkappa matrices can be specified.

Note:
Three materials are predefined:

- Material '0' is a dielectric whose default values of epsr and muer are 1. This is vacuum. You can change the parameters epsr, muer, kappa and mkappa of the material '0', but you cannot change its `type= normal`.

- Material '1' is treated as a perfect electric material for the field computations. You can change its kappa values, but this only effects the wall-loss computations of **gd1.pp**. You cannot change its `type= electric`.

- Material '2' is treated as a perfect magnetic conducting material. You cannot change its `type= magnetic`.

Example
The following specifies that the material with number 3 shall be treated as a perfect magnetic conducting material, the material with number 4 is a lossy dielectric.

```
material= 3
   type= magnetic
material= 4
   type= normal, epsr= 3, kappa= 1, muer= 1
```

1.5 Geometric primitives

1.5.1 -brick: a rectangular brick

A `brick` is a rectangular box, with its edges parallel to the cartesian coordinate axes.

```
##############################################################################
# Flags: nomenu, noprompt, nomessage,                                        #
##############################################################################
# section -brick                                                             #
##############################################################################
# material=  1, sloppy= no                                                   #
# whichcells= all, taboo= none                                               #
# xlow = undefined     , ylow = undefined     , zlow = undefined             #
# xhigh= undefined     , yhigh= undefined     , zhigh= undefined             #
# # # # #                                                                    #
# volume= (undefined, undefined, undefined, undefined, undefined, undefined) #
##############################################################################
# doit, return, help                                                         #
##############################################################################
```

- **material**
  The material index that shall be assigned to the volume that makes up the brick.

- **sloppy**
  Possible values are `yes, no`. If no, meshplanes are enforced at the borderplanes of the brick.

- **whichcells**
  Possible values are `all`, or a material-index. If `whichcells= all`, all volume inside the brick is assigned the material-index, **provided** the former material is not `taboo`. If `whichcells` is a material-index, only the parts of the brick that are currently filled with the given index are assigned the new material-index.

- **taboo**
  Possible values are `none`, or a material-index. If `taboo= all`, all volume inside the brick is assigned the material-index. If `taboo` is a material-index, only the parts of the brick that are currently filled with another index than the given index are assigned the new material-index.

- **xlow, xhigh, ylow, yhigh, zlow, zhigh, volume**
  The coordinates of the bounding planes of the brick. Alternatively to specifying via xlow.., you can specify the volume of the brick as
  volume= (XL, XH, YL, YH, ZL, ZH).

- **doit**
  Discretizes a brick from the current data.

Example

```
-general
    outfile= /tmp/bruw1931/garbage/example
    scratch= /tmp/bruw1931/garbage/scratch

-mesh
    pxlow= 0, pxhigh= 1e-2
    pylow= 0, pyhigh= 2e-2
    pzlow= 0, pzhigh= 1e-2

    cxlow= ele, cxhigh= mag
    cylow= ele, cyhigh= mag
    czlow= ele, czhigh= mag

    spacing= 1e-3

-brick
    material= 1, sloppy= no
       xlow= 0.3e-2, xhigh= 0.8e-2
       ylow= 0.2e-2, yhigh= 1.3e-2
       zlow= 0.2e-2, zhigh= 0.6e-2
    doit

-volumeplot
    scale= 2.5
    doit
```

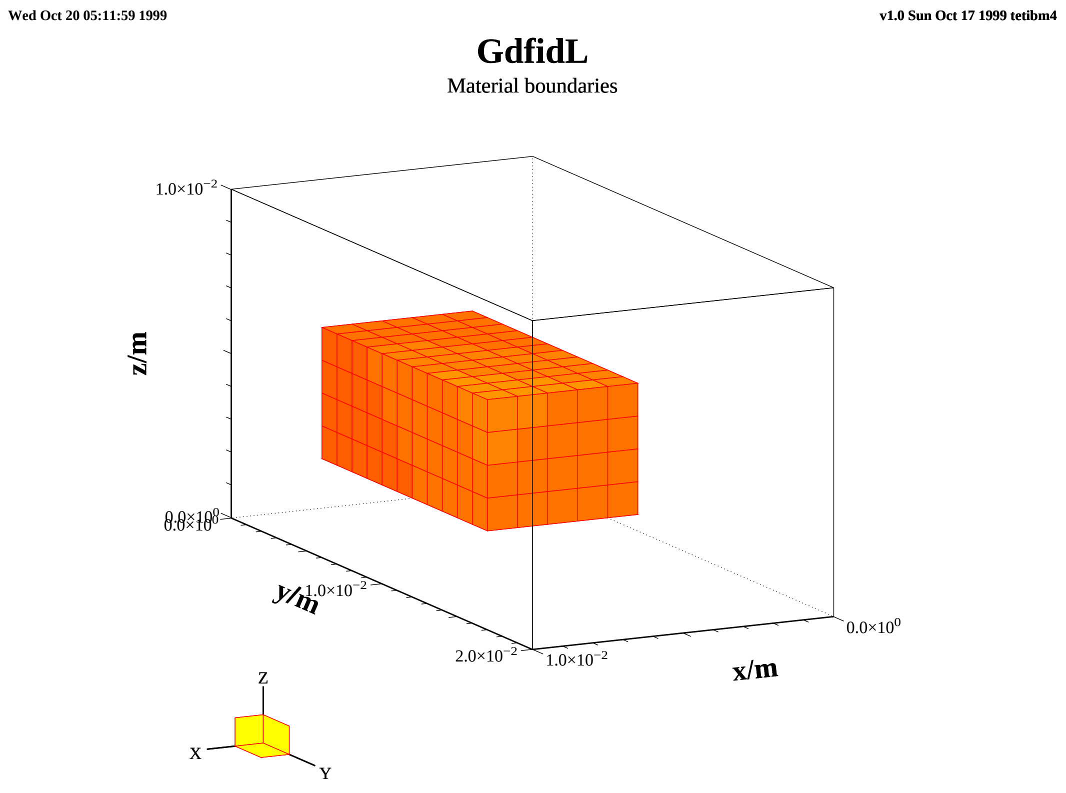


Figure 1.1: A simple brick

1.5.2 -gccylinder: A circular cylinder in general direction

A `gccylinder` is a circular cylinder, with its axis in an arbitrary direction.

```
###############################################################################
# Flags: nomenu, noprompt, nomessage,                                         #
###############################################################################
# section -gccylinder                                                         #
###############################################################################
# material  =  1                                                              #
# whichcells= all, taboo= none                                                #
# radius    = undefined                                                       #
# length    = undefined                                                       #
# origin    = ( undefined, undefined, undefined )                             #
# direction = ( undefined, undefined, undefined )                             #
###############################################################################
# doit, return, help                                                          #
###############################################################################
```

- **material**
  The material index that shall be assigned to the volume that makes up the `gccylinder`.

- **radius**
  The radius of the circular cylinder.

- **length**
  The length of the cylinder.

- **origin**
  The coordinates of the center of the foot-circle of the circular cylinder.

- **direction**
  The direction of the cylinder axis.

- **doit**
  Discretizes a circular cylinder from the current data.

Note: You can revert the direction of the cylinder by negating the **direction**, or (easier) by negating the **length**.

Note: if you want eg. a quarter of a circular cylinder, you have to use `-gbor`, see pages 24 ff.

Example

```
-general
    outfile= /tmp/bruw1931/garbage/example
    scratch= /tmp/bruw1931/garbage/scratch-

-mesh
    pxlow= 0, pxhigh= 1e-2
    pylow= 0, pyhigh= 2e-2
    pzlow= 0, pzhigh= 1.5e-2

    cxlow= ele, cxhigh= mag
    cylow= ele, cyhigh= mag
    czlow= ele, czhigh= mag

    spacing= 0.2e-3

-gccylinder
    material= 1, radius= 3e-3, length= 7e-3
    origin= (0.5e-2, 0.3e-2, 0.6e-2)
    direction= (-0.4, 1.5, 0.4)
    doit

-volumeplot
    scale= 2.5
    doit
```

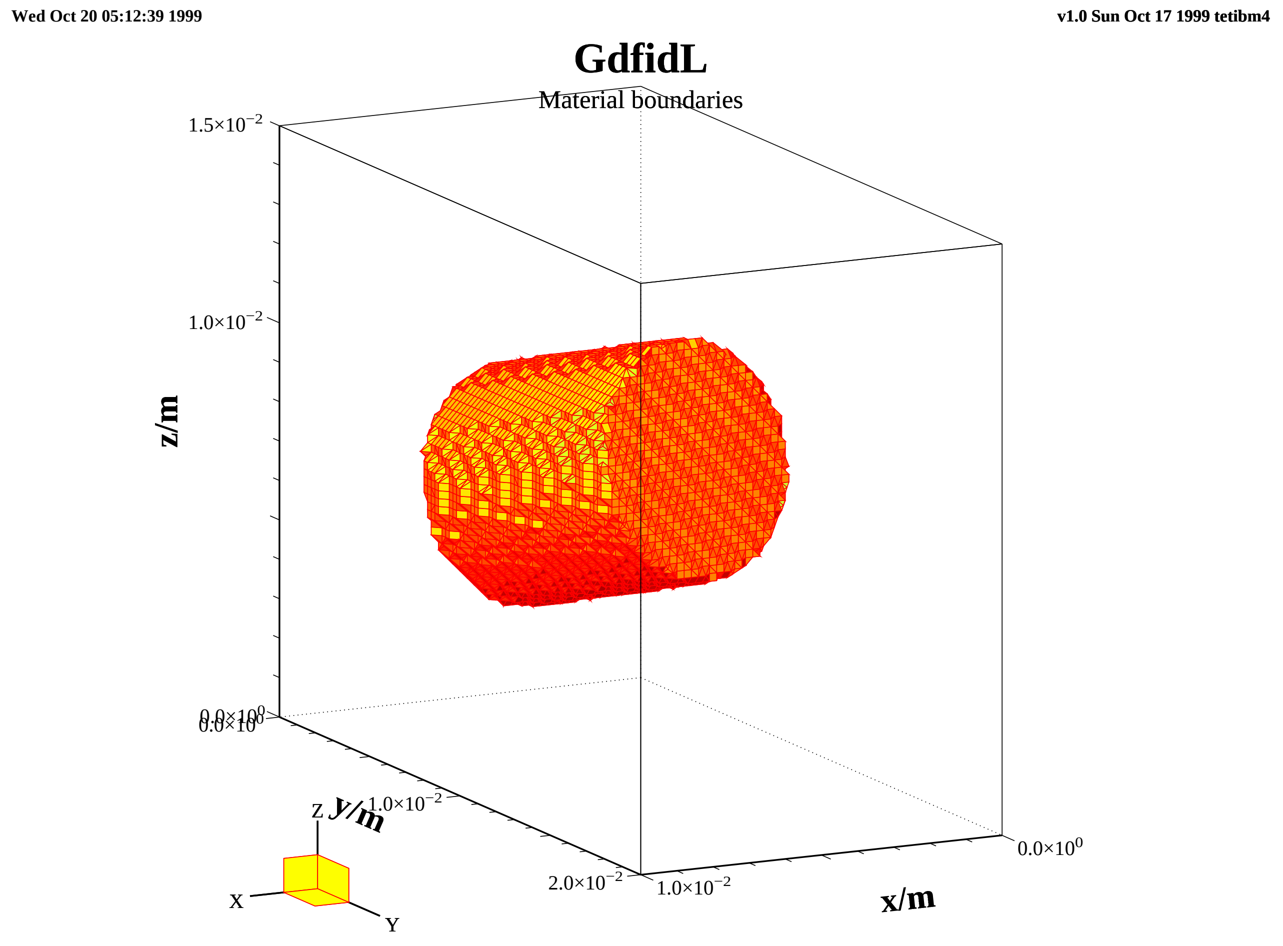


Figure 1.2: A simple gccylinder, with its axis directing towards (-0.4, 1.5, 0.4).

1.5.3 -ggcylinder: A general cylinder in general direction

A `ggcylinder` is a cylinder with a footprint described as a general polygon. This footprint is swept along an axis in a general direction, additionally, this footprint can shrink or expand along this axis, additionally, this footprint can be rotated along the axis, additionally, only parts of the `ggcylinder` that fulfill some additional condition will be filled.

```
################################################################################
# Flags: nomenu, noprompt, nomessage,                                          #
################################################################################
# section -ggcylinder                                                          #
################################################################################
# material  =  1                                                               #
# whichcells= all, taboo= none                                                 #
# originprime    = ( 0.0, 0.0, 0.0 )                                           #
# xprimedirection= ( 1.0, 0.0, 0.0 )                                           #
# yprimedirection= ( 0.0, 1.0, 0.0 )                                           #
# range     = ( undefined, undefined )                                         #
# pitch     = 0.0                                                              #
# expgrowth = 0.0                                                              #
# xlingrowth= 1.0                                                              #
# ylingrowth= 1.0                                                              #
# show      = off   (off | all | later | now)                                  #
# fixpoints = no           (yes|no)                                            #
# inside    = yes          (yes|no)                                            #
################################################################################
## syntax:                                                                     #
#  point= (Xi, Yi)                                                             #
#  arc, radius= RADIUS, type= [clockwise | counterclockwise]                   #
#       size= [small | large ]                                                 #
#       deltaphi= 5                                                            #
#  ellipse, center= (X0, Y0), size= [small | large ]                           #
#       deltaphi= 5                                                            #
################################################################################
# doit, return, help, list, reset, clear                                       #
################################################################################
```

- `material= MAT`:
  The material index that will be assigned to cells inside the volume.

- **whichcells**
  Possible values are `all`, or a material-index.
  If `whichcells= all`, all volume inside the `ggcylinder` is assigned the material-index, **provided** the former material is not `taboo`. If `whichcells` is a material-index, only the parts of the `ggcylinder` that are currently filled with the given index are assigned the new material-index.

- **taboo**
  Possible values are `none`, or a material-index.
  If `taboo= all`, all volume inside the `ggcylinder` is assigned the material-index. If `taboo` is a material-index, only the parts of the `ggcylinder` that are currently filled with another index than the given index are assigned the new material-index.

- `originprime`:
  The coordinates of the origin of the `ggcylinder` .

- `xprimedirection`:
  The direction of the x'-axis of the polygon that describes the footprint.

- `yprimedirection`:
  The direction of the y'-axis of the polygon that describes the footprint. The y'-direction will internally be enforced to be perpendicular to the x'-direction.

- `range`:
  Start and end values of the z'-coordinate of the cylinder in the coordinate system of `xprimedirection,yprimedirection,(xprime` $\times$ `yprime)`, relative to `originprime`.

- `pitch`:
  Phase in degrees/m. The footprint will be rotated along the axis.

- `expgrowth`
  Alpha of the exponential growth of the footprint along the axis in 1/m.

- `xlingrowth`:
  Factor of the linear growth of the x-coordinates of the footprint along the axis in 1/m.

- `ylingrowth`:
  Factor of the linear growth of the y-coordinates of the footprint along the axis in 1/m.

- `inside`:
  Flag, specifying whether cells inside of the `ggcylinder` should be assigned material index `MAT` , or whether the cells outside of it should be changed.

- `show`:
  Flag, specifying whether an outline of the specified `ggcylinder` should be displayed.
  If `show= off` , no outline will be displayed.
  If `show= later` , the outline will be shown later together with other specified `ggcylinder's` and `gbor's` .
  If `show= all` is present, the outlines of all specified `ggcylinder's` and `gbor's` so far will be displayed.

- `point= (XI,YI)`:
  `XI, YI` are the coordinates of the i.th point in the polygon that describes the footprint of the `ggcylinder` . There have to be minimum 3 points, or 2 points and an arc or 2 points and an ellipse.

- `arc`:
  (optional):
  Indication, that there should be a circular arc from the point that was specified before to the point that will be specified as next point.
  In order to determine the arc between two points, three parameters are necessary: The radius of the arc, whether the connection is clockwise or counterclockwise, and whether the connection should take the large path or the small path.

  - `radius= RADIUS`:
    The points are to be connected by an arc with radius= `RADIUS`
  - `size= [small | large]` (optional):
    Indication, whether the the connection between the two points should be on the smaller or the larger side of the arc.

  - `type= [clockwise | counterclockwise]`:
    Indication, whether the connection between the two points should proceed clockwise or counterclockwise along the arc.

- `clear`:
  Clears the current polygon path.

- `doit`:
  Discretizes a ggcylinder from the current data.

Example
The following decribes a cavity with rounded corners.

```
-general
   outfile= /tmp/bruw1931/garbage/example
   scratchbase= /tmp/bruw1931/garbage/scratch
   text()= Radius = 500.0e-03 mm

-mesh
   spacing= 100.0e-06
   graded= yes, dmaxgraded= 263.3e-06
   pxlow= -5e-03, pxhigh= 5e-03
   pylow= -4.34e-03, pyhigh= 0
   pzlow= -1.6665e-03, pzhigh=1.6665e-03
   cxlow= ele, cxhigh= mag
   cylow= ele, cyhigh= mag
   czlow= el, czhigh= el

-ggcylinder

   material= 1
   originprime= (0,0,0)
   xprimedirection= (1,0,0)
   yprimedirection= (0,0,1)
   range= (-4.2e-03, 4.2e-03)

   clear  # clear the polygon-list, if any
   point= (-3.3405e-03, -816.5e-06 )
      arc, radius= 500.0e-06, type= counterclockwise, size= small
   point= (-2.8405e-03, -1.3165e-03 )
   point= (2.8405e-03, -1.3165e-03 )
      arc, radius= 500.0e-06, type= counterclockwise, size= small
   point= (3.3405e-03, -816.5e-06 )
   point= (3.3405e-03, 816.5e-06 )
      arc, radius= 500.0e-06, type= counterclockwise, size= small
   point= (2.8405e-03, 1.3165e-03 )
   point= (-2.8405e-03, 1.3165e-03 )
      arc, radius= 500.0e-06, type= counterclockwise, size= small
   point= (-3.3405e-03, 816.5e-06 )

   fixpoints= yes # enshure mesh-planes at the points of the polygon
   doit

-volumeplot
   scale= 3
   doit
```

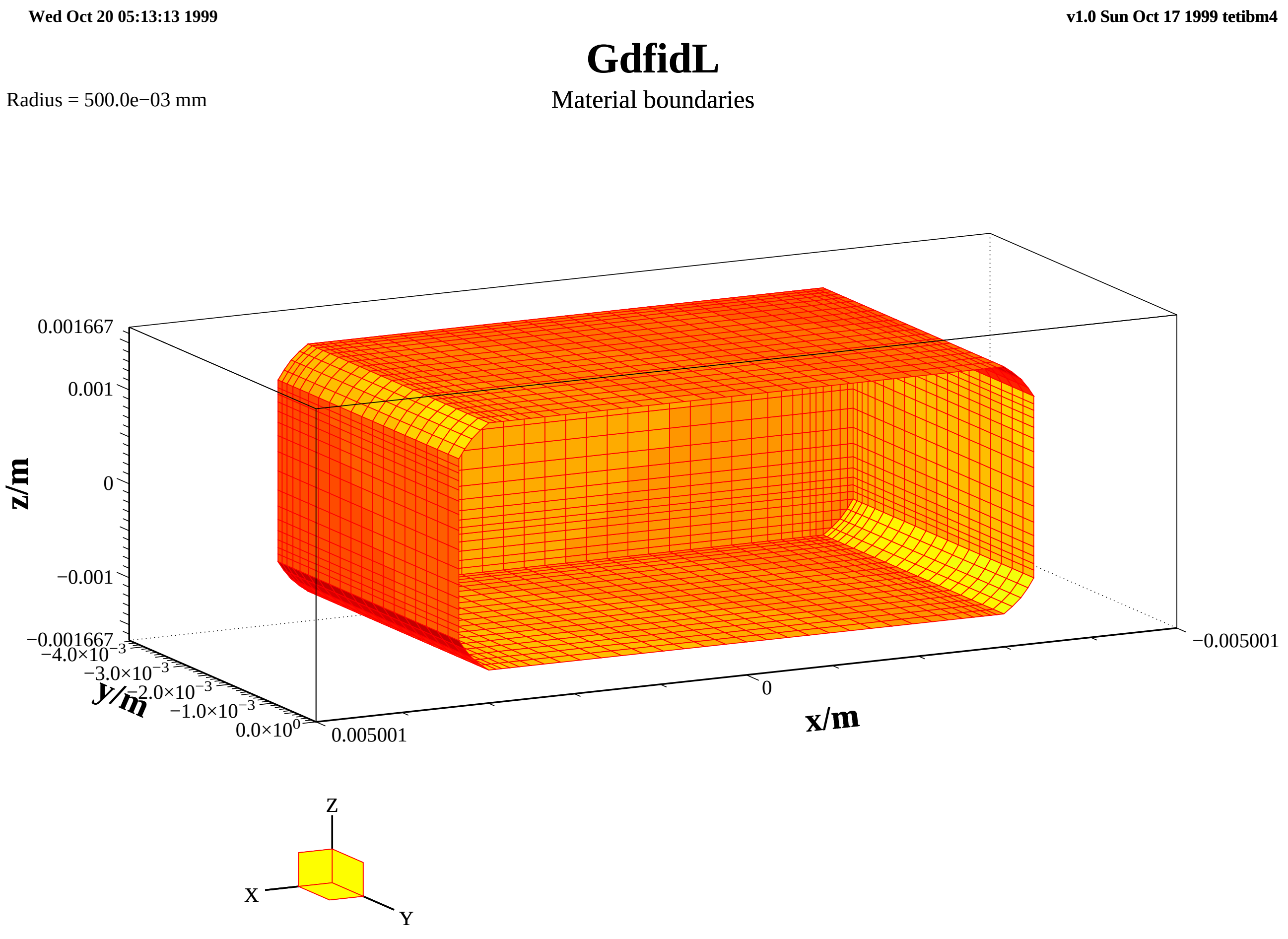


Figure 1.3: A simple ggcylinder

1.5.4 -gbor: A general body of revolution in general direction

A `gbor` is a body of revolution with a cross section described as a general polygon. This cross section is swept along an axis in a general direction, additionally, only cells that fulfill some additional condition will be filled.

```
################################################################################
# Flags: nomenu, noprompt, nomessage,                                          #
################################################################################
# section -gbor                                                                #
################################################################################
# material       =  1                                                          #
# whichcells     = all, taboo= none                                            #
# originprime    = ( 0.0, 0.0, 0.0 )                                           #
# zprimedirection= ( 0.0, 0.0, 1.0 )                                           #
# rprimedirection= ( 1.0, 0.0, 0.0 )                                           #
# range          = ( 0.0, 360.0 )                                              #
# show           = off                (off | all | later | now)                #
# inside    = yes                     (yes|no)                                 #
################################################################################
## syntax:                                                                     #
#  point= (Zi, Ri)                                                             #
#  arc, radius= RADIUS, type= [clockwise | counterclockwise]                   #
#       size= [small | large ]                                                 #
#       deltaphi= 5                                                            #
#  ellipse, center= (Z0, R0), size= [small | large ]                           #
#       deltaphi= 5                                                            #
################################################################################
# doit, return, help, list, reset, clear                                       #
################################################################################
```

- `material= MAT`:
  The material index that will be assigned to cells inside the volume.

- **whichcells**
  Possible values are `all`, or a material-index.
  If `whichcells= all`, all volume inside the `ggcylinder` is assigned the material-index, **provided** the former material is not `taboo`. If `whichcells` is a material-index, only the parts of the `ggcylinder` that are currently filled with the given index are assigned the new material-index.

- **taboo**
  Possible values are `none`, or a material-index.
  If `taboo= all`, all volume inside the `ggcylinder` is assigned the material-index. If `taboo` is a material-index, only the parts of the `ggcylinder` that

are currently filled with another index than the given index are assigned the new material-index.

- `originprime`:
  The coordinates of the origin of the `gbor`.

- `zprimedirection`:
  The direction of the z'-axis of the `gbor`.

- `rprimedirection`:
  The direction of the r'-vector of the polygon. The r'-direction will internally be enforced to be perpendicular to the z'-direction.

- `range`:
  Start and end values (in degrees)) of the phi-coordinate of the body of revolution.

- `inside`:
  Flag, specifying whether cells inside of the `gbor` should be assigned material index `MAT` , or whether the cells outside of it should be changed.

- `show`:
  Flag, specifying whether an outline of the specified `gbor` should be displayed.
  If `show= off` , no outline will be displayed.
  If `show= later` , the outline will be shown later together with other specified `ggcylinder's` and `gbor's` .
  If `show= all` is present, the outlines of all specified `ggcylinder's` and `gbor's` so far will be displayed.

- `point= (XI,YI)`:
  `XI, YI` are the coordinates of the i.th point in the polygon that describes the polygon of the `ggcylinder` . There have to be minimum 3 points, or 2 points and an arc or 2 points and an ellipse.

- `arc`:
  Indication, that there should be a circular arc from the point that was specified before to the point that will be specified as next point.
  In order to determine the arc between two points, three parameters are necessary: The radius of the arc, whether the connection is clockwise or counterclockwise, and whether the connection should take the large path or the small path.

  - `radius= RADIUS`:
    The points are to be connected by an arc with radius= `RADIUS`
  - `size= [small | large]` (optional):
    Indication, whether the the connection between the two points should be on the smaller or the larger side of the arc.

    – `type= [clockwise | counterclockwise]`:
      Indication, whether the connection between the two points should proceed clockwise or counterclockwise along the arc.

- `clear`:
  Clears the current polygon path.
- `doit`:
  Discretizes a ggcylinder from the current data.

Example

```
-general
   outfile= /tmp/bruw1931/garbage/example
   scratchbase= /tmp/bruw1931/scratch

-mesh
   pxlow= -5e-2, pxhigh= 12e-2
   pylow= -18e-2, pyhigh= 3e-2
   pzlow= -6e-2, pzhigh= 6e-2

   spacing= 0.2e-2

define(PlungerInnerRadius, eval(100e-3/2) )
define(PlungerCurvature, 16e-3)
define(PlungerAngle, eval(-67.5*@pi/180))

-gbor
    material= 1
    originprime= (0,0,0)
    zprimedirection= (eval(-cos(PlungerAngle)),\
                      eval(-sin(PlungerAngle)),\
                      0)
    rprimedirection= (0,0,1)
    range= (0,360)

    clear
       # point= (z,r)
    point= (0,0)
    point= (0, eval(PlungerInnerRadius-PlungerCurvature) )
       arc, radius= PlungerCurvature, size= small, type= counterclockwise
    point= (eval(-PlungerCurvature), PlungerInnerRadius)
    point= (-170e-3, PlungerInnerRadius)
    point= (-170e-3, 0 )
```

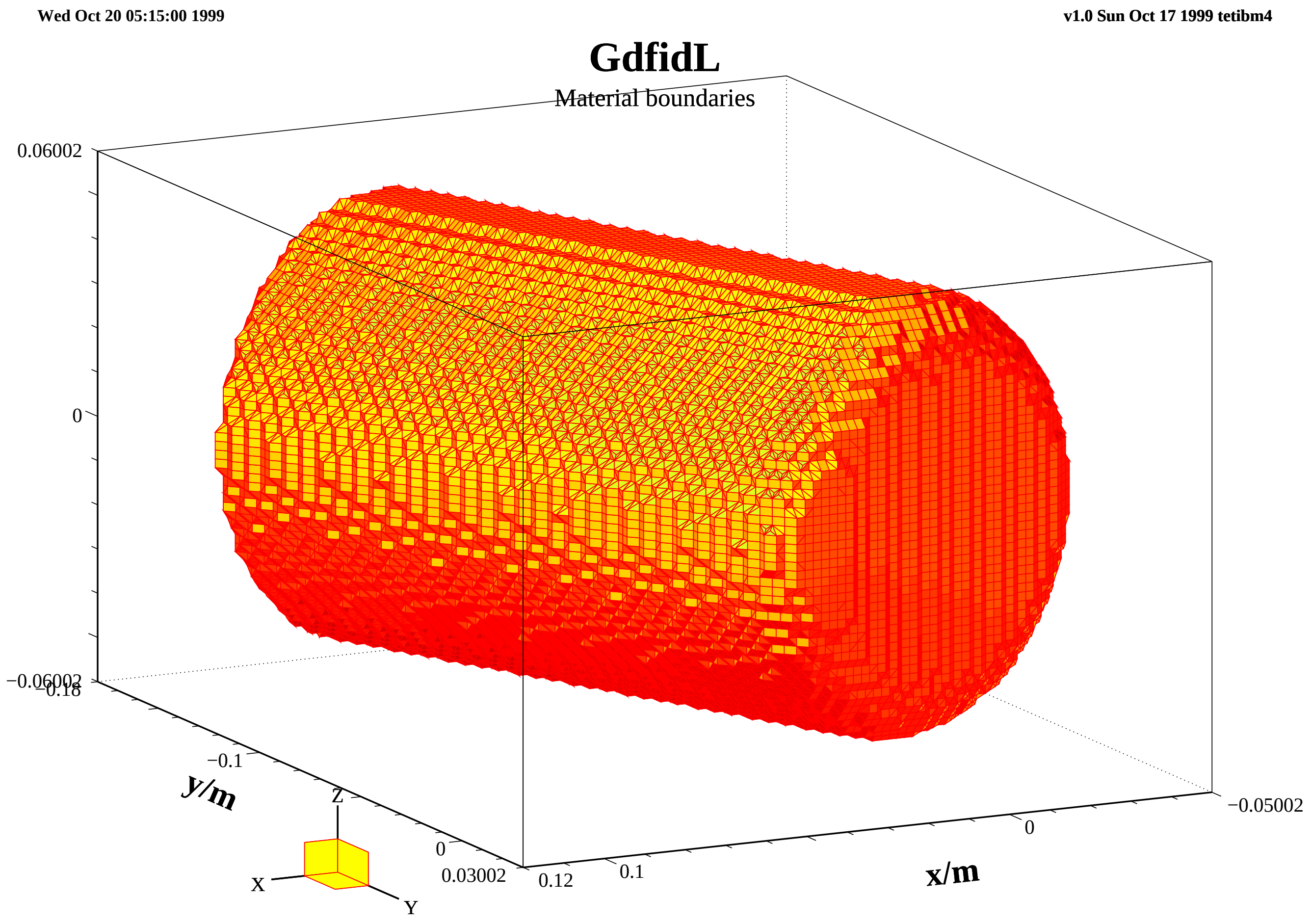


Figure 1.4: A simple gbor.

```
    doit

 -volumeplot
    scale= 3
    doit
```

Example:

```
define(INF,10000)       # some big number

define(MAXCELLS,1.e+5)

define(XLOW,0) define(XHIGH,5.0e-2)
define(YLOW,0) define(YHIGH,6.0e-2)
define(ZLOW,-1.1e-2) define(ZHIGH,0)

define(STPSZE,eval( ((XHIGH-XLOW)*(YHIGH-YLOW)*(ZHIGH-ZLOW)/MAXCELLS)**(1/3) ))

 -mesh
    volume= (XLOW, XHIGH, \
             YLOW, YHIGH, \
             ZLOW, ZHIGH )

    spacing= STPSZE

 -general
    outfile= /tmp/bruw1931/garbage/example
    scratchbase= /tmp/bruw1931/garbage/scratch
 define(R,1.0e-2)
    text()= Intersection of two circular cylinders with radius R
    text()= generated as general cylinders
    text()= where "taboo" is specified
    text()= stpsze= STPSZE, maxcells= MAXCELLS


#
# Fill everything with metal
#
 -brick
    material= 1
    volume= ( -INF,INF, -INF,INF, -INF,INF)
    doit

#
# First step,
# fill cells above the diagonal with material 3
# these cells will not be filled by the first circular cylinder,
# since i will specify "taboo= 3"
#
#
 -ggcylinder
    material= 3,
    origin= (0,0,0), xprime= (1,0,0), yprime= (0,1,0),
    range= (ZLOW,ZHIGH),

    clear
    point= (XLOW,YLOW), point= (XLOW,YHIGH),
    point= (eval(XLOW+(YHIGH-YLOW)),YLOW)
    doit

#
# Second step:
# fill a circular cylinder in x-direction,
# but NOT cells with material index 3 (taboo=3)
#
 -ggcylinder
    material= 0, taboo= 3,
    xprime= (0,1,0), yprime= (0,0,1),    # so the axis will be in +x
    origin= (0,4.e-2,0),                # shift of origin
    range= (XLOW,XHIGH),

    clear
       point= (-R,0),
         arc, radius= R, type= counterclockwise,
       point= (0,-R),
         arc, radius= R,
       point= (R,0),
    doit

#
# Third step:
# fill a circular cylinder in (approx) y-direction,
# but ONLY cells with material index 3 (whichcells=3),
#
 -ggcylinder
    material= 0, whichcells= 3, taboo= none
    xprime= (1,0,0), yprime= (0,0,-1),   # so the axis will be in +y
    origin= (2.e-2,0,0),                # shift of origin
    range= (YLOW,YHIGH),

    clear
       point= (-R,0),
          arc, radius= R, type= clockwise,
       point= (0,R),
          arc, radius= R,
       point= (R,0),
    doit

 -volumeplot
    eyeposition= ( 1, 2, 1 )
    scale= 3
    doit
```

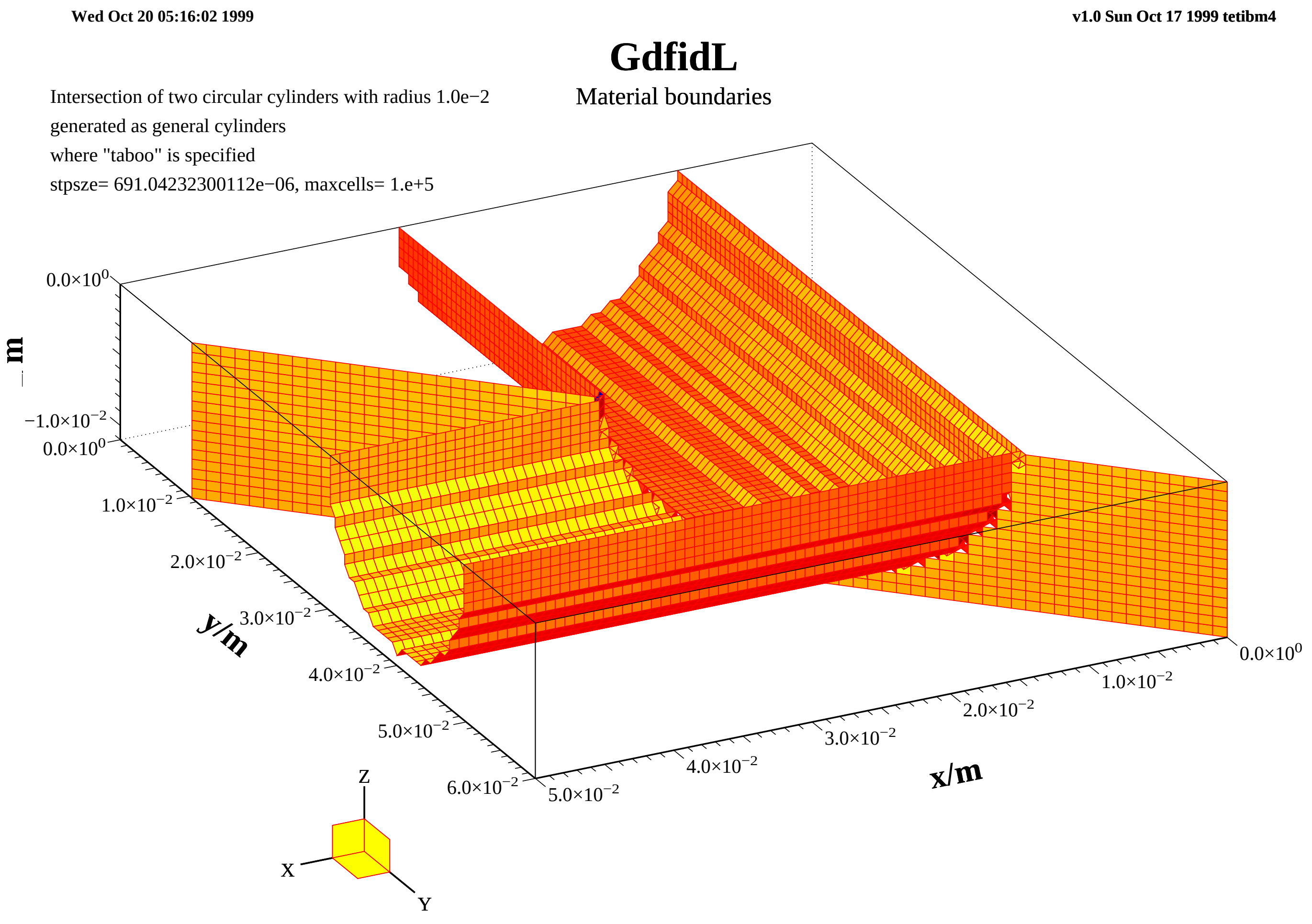
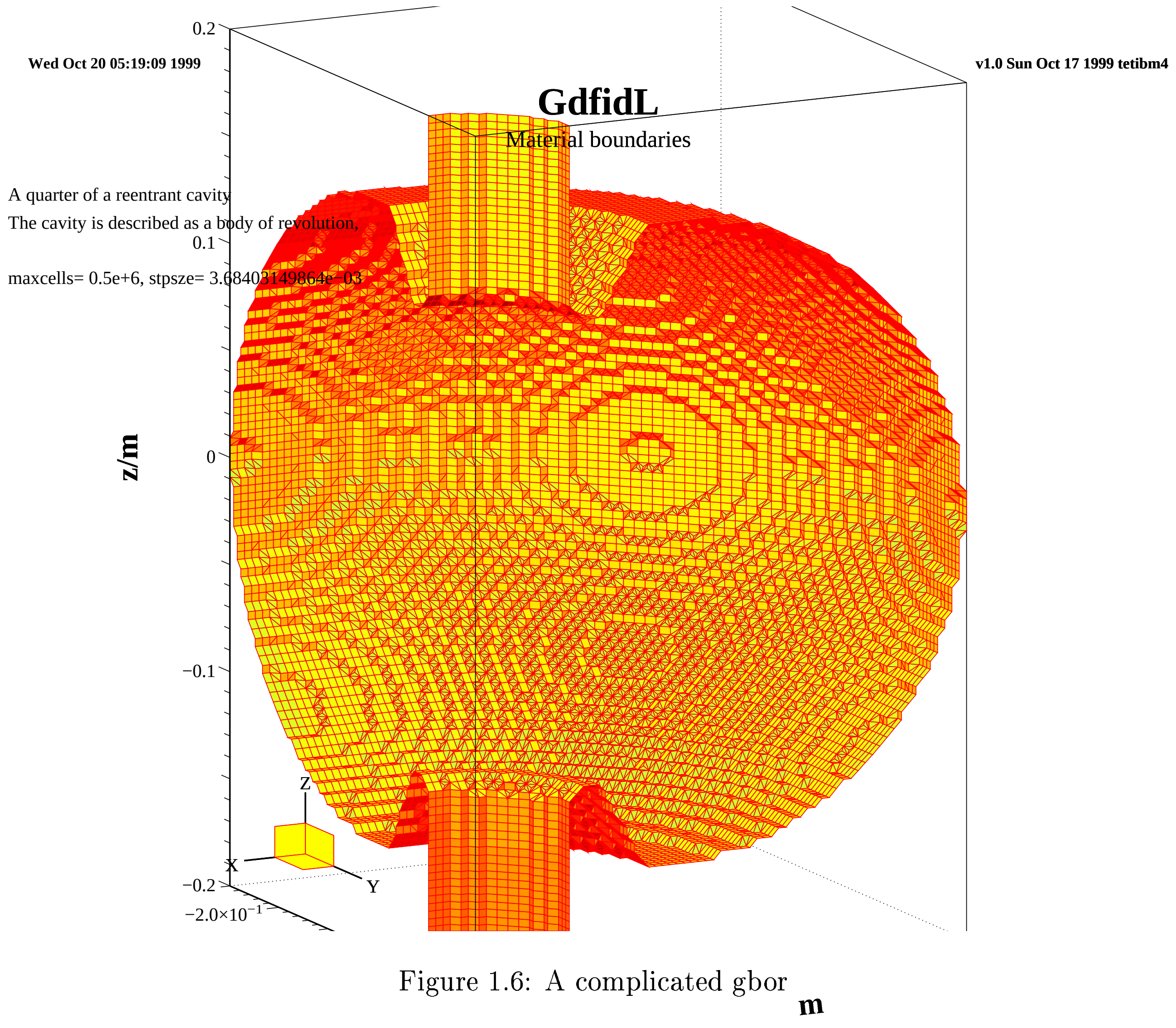


Figure 1.5: The intersection of two circular cylinders, where **whichcells** and **taboo** were specified.

Example

```
 define(MaxCells, 0.5e+6)

 #
 # define the geometry parameters
 #
 define(RBeamTube, 4.7625e-2)
 define(RCurve,  1.0e-2)
 define(ZGapNose, 10.9e-2)
define(RLarge, 25.0e-2)     define(RSmall, 15.0e-2)  define(RCenter, 10.0e-2)

define(INF,10000) define(EL,1) define(MAG,2)

define(XLOW,-0.250) define(XHIGH,0)
define(YLOW,-0.250) define(YHIGH,0)
define(ZLOW,-0.200) define(ZHIGH,0.200)

define(STPSZE,eval( ((XHIGH-XLOW)*(YHIGH-YLOW)*(ZHIGH-ZLOW)/MaxCells)**(1/3) ))

 #
 # gdfidl can evaluate sin(), cos(), atan() and X**Y
 # definition of functions degsin() and degcos()
 #
 define(degsin,[eval(sin((@arg1)*@pi/180))])
 define(degcos,[eval(cos((@arg1)*@pi/180))])
 define(degtan,[eval(sin((@arg1)*@pi/180)/cos((@arg1)*@pi/180))])

-general
   outfile= /tmp/bruw1931/garbage/example
   scratch= /tmp/bruw1931/garbage/scratch-

   text(3)= A quarter of a reentrant cavity
   text()= The cavity is described as a body of revolution,
   text()=
   text()= maxcells= MaxCells, stpsze= STPSZE
return

-mesh
   pxlow= XLOW, pxhigh= eval(1*XHIGH)
   pylow= eval(1*YLOW)
   pyhigh= eval(0*YHIGH)
   pzlow= eval(1*(ZLOW))
   pzhigh= eval(1*ZHIGH)
```

```
    cxlow= mag, cxhigh= mag
    cylow= mag, cyhigh= mag
    czlow= el,  czhigh= el

    spacing= eval(1*STPSZE)
    minspacing= eval(STPSZE/10)

  -brick
     #
     # we fill the universe with metal
     #
     material= EL
        volume= ( -INF,INF, -INF,INF, -INF,INF)
     doit

  #
  # carve out the cavity itself
  #
  -gbor
      material= 0, range= (0, 360)
      origin= (0,0,0)
      show= later,      # don't show now, but show later
#

      clear   # clear a possible previous polygon list

         point= (     0,                  0),
         point= ( eval(ZHIGH+2*STPSZE),                  0),
         point= ( eval(ZHIGH+2*STPSZE), RBeamTube),
         point= ( eval(ZGapNose+RCurve), RBeamTube),
            arc, radius= RCurve, size= small, type= clockwise,
            deltaphi= 10
define(rdum, eval(RBeamTube+(1+degcos(30))*RCurve))
define(zdum, eval(ZGapNose +(1-degsin(30))*RCurve))
         point= ( zdum, rdum)
define(deltaz, eval(RSmall-zdum))
define(deltar, eval(deltaz*degtan(30)))
 define(ffac, 0.85)   ## adjust this for a smooth transition
define(zdum2, eval(zdum+ffac*deltaz))
define(rdum2, eval(rdum+ffac*deltar))
         point= (zdum2, rdum2)
           arc, radius= RCurve, size= small, type= counterclockwise,
           deltaphi 10
         point= (RSmall, eval(RCenter-0.8*RCurve)),
         point= (RSmall, RCenter),
           arc, radius= RSmall, size= small, type= counterclockwise,
           delta= 3
        point= (-RSmall, RCenter),
        point= (-RSmall, eval(RCenter-0.8*RCurve)),
           arc, radius= RCurve, size= small, type= counterclockwise,
           deltaphi= 10
        point= (eval(-zdum2), rdum2)
        point= (eval(-zdum), rdum)
           arc, radius= RCurve, size= small, type= clockwise, delta 10
        point= ( eval(-(ZGapNose+RCurve)), RBeamTube),

        point= (eval(ZLOW-2*STPSZE), RBeamTube),
        point= (eval(ZLOW-2*STPSZE), 0)
     list
     doit
 return
#
# enforce some meshplanes:
#
-mesh
   zfixed(2, eval(-(ZGapNose+RCurve)), eval(-ZGapNose) ) # at the noses
   zfixed(2, eval( (ZGapNose+RCurve)), eval( ZGapNose) ) # at the noses

   zfixed(2, -RSmall, RSmall)                # at the z-borders of the cavity

-volumeplot
   scale= 3
   doit
```



Wed Oct 20 05:19:09 1999
v1.0 Sun Oct 17 1999 tetibm4
GdfidL
Material boundaries
A quarter of a reentrant cavity
The cavity is described as a body of revolution.
maxcells= 0.5e+6, stpsze= 3.68403149864e-03
0.2
0.1
0
−0.1
−0.2
z/m
−2.0×10^{-1}
m
Z
X
Y

Figure 1.6: A complicated gbor

1.5.5 -stlfile: CAD import via STL-file

This section is for importing a geometry description from a CAD system via a STL-file (STereo-Lithography). A STL-file describes a closed body via a set of triangles.

The so described body can be rotated, shrunk or expanded and shifted.

```
##############################################################################
# Flags: nomenu, noprompt, nomessage,                                        #
##############################################################################
# section -stlfile                                                           #
##############################################################################
# file= /usr/local/gd1/examples/woman.stl                                    #
# material  =  1                                                             #
# whichcells= all, taboo= none                                               #
# originprime    = ( 0.0, 0.0, 0.0 )                                         #
# xprimedirection= ( 1.0, 0.0, 0.0 )                                         #
# yprimedirection= ( 0.0, 1.0, 0.0 )                                         #
# xscale= 1.0                                                                #
# yscale= 1.0                                                                #
# zscale= 1.0                                                                #
# show      = no    ( yes | no )                                             #
##############################################################################
# doit, return, help                                                         #
##############################################################################
```

- `file= NAME_OF_STLFILE`:
  The name of the STL-File (in ASCII) that shall be imported.

- `material= MAT`:
  The material index that will be assigned to cells inside the volume.

- **whichcells**
  Possible values are `all`, or a material-index.
  If `whichcells= all`, all volume inside the stl-body is assigned the material–1, **provided** the former material is not `taboo`. If `whichcells` is a material-index, only the parts of the stl-body that are currently filled with the given -1 are assigned the new material-index.

- **taboo**
  Possible values are `none`, or a material-index.
  If `taboo= all`, all volume inside the stl-body is assigned the material–1. If `taboo` is a material-index, only the parts of the stl-body that are currently filled with another -1 than the given index are assigned the new material-index.

- `originprime= (X0, Y0, Z0)`:

- `xprimedirection= (XXN, XYN, XZN)`:
- `yprimedirection= (YXN, YYN, YZN)`:
- `xscale= XS, yscale= YS, zscale= ZS`:
  The coordinates of the STL-Data are transformed as follows: A 3x3 matrix (A) is built, such that A11, A21, A31 are the components of the normalized "xprimedirection". The direction "yprimedirection" is enforced to be perpendicular to "xprimedirection". The result is normalized and taken as the second column of (A). The third column of (A) is the normalized cross product of "xprimedirection" and "yprimedirection". (The matrix (A) is a rotation matrix.)

  The coordinates of a point P of the STL-set are now transformed via

  ```
  P <= (A) * P
  P_x <= P_x * xscale + X0
  P_y <= P_y * yscale + Y0
  P_z <= P_z * zscale + Z0
  ```

- `show= {yes|no}` :
  Flag, specifying whether a plot of the transformed triangles shall be shown.

Example

```
-mesh
   pxlow= -0.3, pxhigh= 0.3
   pylow= -0.4, pyhigh= 0.3
   pzlow= -0.4, pzhigh= 0.35

   spacing= eval(0.5/70)

-stlfile
   material= 1
   file= /nfs/wilson/u5/bruns/gd1/examples/brain.stl
define(SCALE, 0.1)
      xscale= SCALE, yscale= SCALE, zscale= SCALE
      origin= (0,0,-2.2)
      xprimedirection= (1,0,0), yprimedirection= (0,0,1)
   doit

-volumeplot
   scale= 3
   doit
```

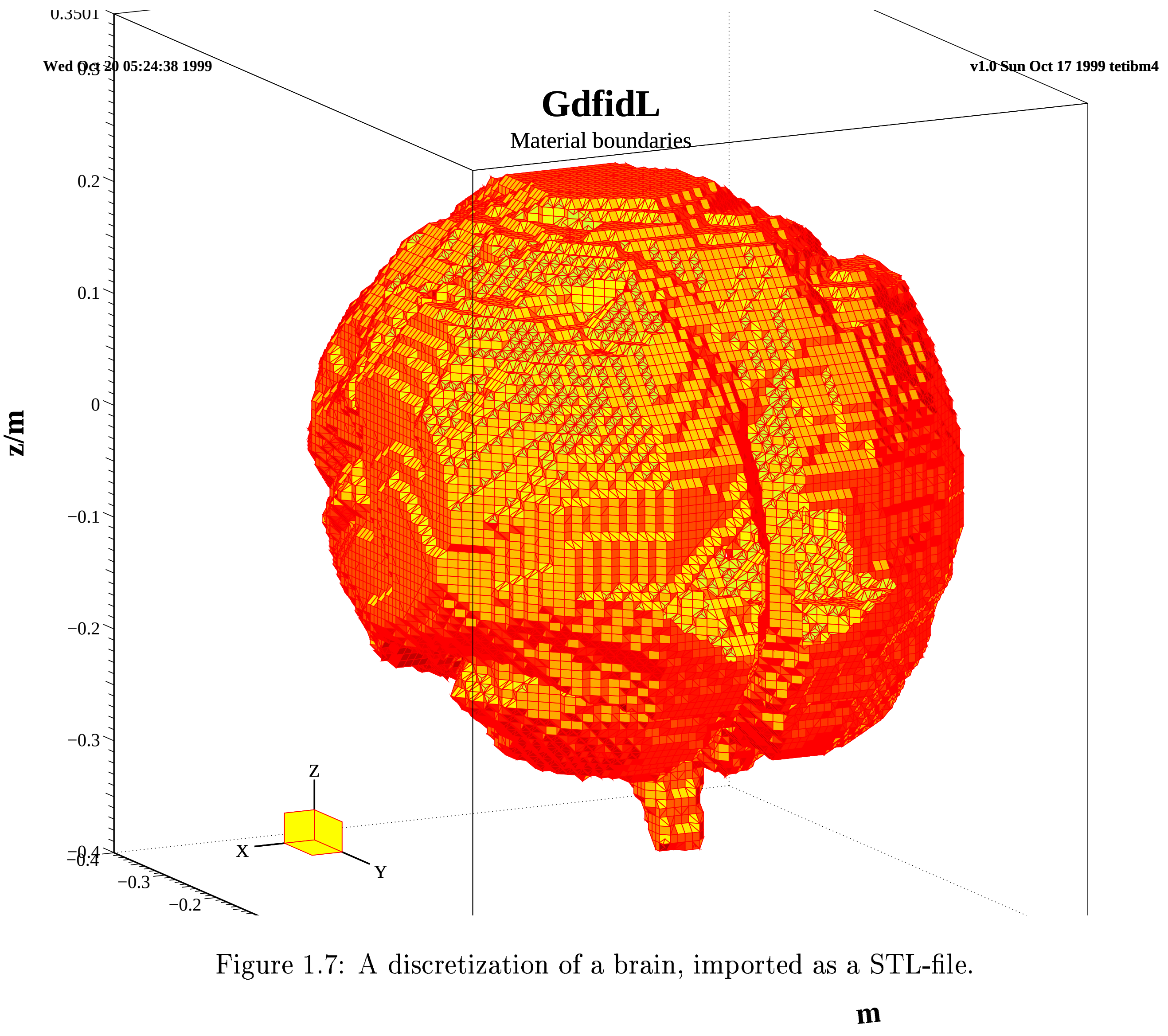


Figure 1.7: A discretization of a brain, imported as a STL-file.

1.6 -volumeplot: shows the generated mesh

This section enables the regeneration of the grid and shows the dicretized material boundaries as generated from the geometric primitives specified so far.

```
###############################################################################
# Flags: nomenu, noprompt, nomessage,                                         #
###############################################################################
# section: -volumeplot                                                        #
###############################################################################
# text     = yes                                                              #
# scale    = 1.80                                                             #
# plotopts = -geometry 1100x900+180+10 -landscape                             #
# eyeposition  = ( 1.0, 2.0, 500.0e-03)                                       #
# lightposition= ( 1.0, 1.0, 1.0)                                             #
# ncolors     = 20                                                            #
# icoloroffset=  0                                                            #
# bbxlow =    -1.0000e+30, bbylow =    -1.0000e+30, bbzlow =    -1.0000e+30  #
# bbxhigh=     1.0000e+30, bbyhigh=     1.0000e+30, bbzhigh=     1.0000e+30  #
#                                                                             #
###############################################################################
# doit, ?, return, help                                                       #
###############################################################################
```

- `text= [yes|no]`:
  Flags, whether the annotation-text that has been specified via `text()= bla bla` in the section `-general` should be plotted together with the material boundaries.

- `scale= SCALE`:
  The initial zoom factor of the resulting plot.

- `plotopts= ANY STRING CONTAING OPTIONS FOR mymtv2`:
  gd1 does not display the data itself, but writes a datafile for **mymtv2** and starts **mymtv2** to display these data.
  Useful options are:

  – -landscape : Produce the PostScript plot in landscape mode

  – -colorps : Produce colour PostScript

  – -printcmd command : Use as printcommand `command`

  – -Pprinter: Print to printer `printer`

  – -pbg background_color : Use as X11-background colour the colour `background_color` (grey | white | ..)

  – -nopixmap : Good for very dumb terminals. If only the first one or two plots appear, try this.

  - -geometry X11-GEOMETRY Initial geometry for the X11 window of **mymtv2**.

- `eyeposition= (XEYE, YEYE, ZEYE)`:
  This specifies the initial eyeposition for the 3D plot that will be produced. As soon as the plot appears, you can interactively change the eyposition with the buttons of **mymtv2**, but for a complex geometry, this takes a lot of time.

- `lightposition= (XL, YL, ZL`:
  Specifies the illumination of the plot.

- `ncolors= N`:
  Specifies the maximum number of colours in the plot.

- `icoloroffset= N`:
  Shifts the colour palette. When icoloroffset is greater than 100, the materialplot appears without the lines that indicate the borders of the grid cells.

- `bbxlow=,bbxhigh=,bbylow=bbyhigh=,bbzlow=,bbzhigh=`:
  Specifies the coordinates of a bounding box. Only the materialboundaries that lie within the box are plotted. Use this if you want to see the materialdistribution in a plane.

- `doit`:
  If you say "doit", the relevant settings in "-mesh" are checked, the mesh is re-generated, and the material boundaries are plotted.

1.7 Solver sections: Eigenvalues and driven time domain problems

1.7.1 -eigenvalues

```
###############################################################################
# Flags: nomenu, noprompt, nomessage,                                         #
###############################################################################
# section -eigenvalues                                                        #
###############################################################################
# solutions    = 15                                                           #
# estimation   = undefined                                                    #
# storeallmodes= no                                                           #
# passes       =  2                                                           #
# pfac2        = 1.0e-03                                                      #
#                                                                             #
#                                                                             #
#                                                                             #
#                                                                             #
###############################################################################
# return, doit, help, ?                                                       #
###############################################################################
```

- `solutions= NSOL`:
  The number of basis vectors to use for solving the eigenvalue problem. This number should better not be smaller than, say, 10. The more basis vectors you allow, the more accurate will the solutions be. But: the memory requirement is proportional to this number.

- `estimation= FUP`:
  Estimation of the resonant frequency of the highest mode, i. e. the `NSOL`.st mode.
  This parameter is MANDATORY.
  A common error, that even we commit, is, to specify this estimation badly wrong.

- `storeallmodes= [yes | no]`:
  Selects whether even the static modes shall be saved.
  gd1 computes normally 20 to 30 % static modes (when `passes=1`, but does not store them. If you do not believe that these static modes are really static, you can enforce the storing to file with this option. When the static solutions are in the file, you can view these 'solutions' with **gd1.pp**.

- `passes= NPASS`:
  gd1 uses an algorithm of Tueckmantel to solve the eigenproblem. This algorithm establishes a set of `NSOL` basisvectors that are mutually orthogonal

and (hopefully) span only the subspace also spanned by the eigenvectors corresponding to the `NSOL` lowest eigenvalues.
But since **gd1** solves the eigenproblem resulting from the discretized version of $[\varepsilon]^{-1}\nabla\times[\mu]^{-1}\nabla\times\vec{E} = \omega^2\vec{E}$, **gd1** normally finds 20 to 30 % eigenvalues very near to zero. These eigenvalues belong to static fields, $\omega = 0$.
If you specify `passes`>1, **gd1** throws away the static components in its already established basis vectors and can find more resonant modes with the same specified `NSOL`. Also, the accuracy of the fields becomes better.

- `pfac2= PFAC2`:
  gd1 uses an algorithm of Tückmantel to solve the algebraic eigenproblem resulting from the discretized MAXWELLian equations. This algorithm has an internal accuracy factor, called `pfac2`, that controls the relative cleaning of the basisvectors from eigenvectors outside the range 0 to `FUP`, in which the `NSOL` resonant fields are expected.
  A smaller `pfac2` leads to increased cpu-time, but better accuracy for the lower modes. A good value is `pfac2=1e-3`.

- `doit`:
  If you say "doit", the settings in "-general, -mesh" are checked, the mesh is re-generated, and the iteration for the resonant fields is performed.
  After the field computation, the fields are written to file and the program stops.

Example:
The following specifies that we want to compute with 15 basisvectors, we want to perform two passes to search for the eigenvalues, and we estimate that the highest resonant frequncy of the 15 to be found will be about 1.3 GHz. The final `doit` starts the eigenvalue computation.

```
solutions= 15
passes= 2
estimation= 1.3e9
doit
```

1.7.2 -fdtd: Compute time dependent fields

This section enables the branching to subsections that only make sense for time domain computations. The "doit" in this section also starts the time domain computation.

```
###############################################################################
# Flags: nomenu, noprompt, nomessage,                                         #
###############################################################################
# section: -fdtd                                                              #
###############################################################################
# -ports                                                                      #
# -pexcitation                                                                #
# -lcharge                                                                    #
# -time                                                                       #
###############################################################################
# doit, ?, return, help                                                       #
###############################################################################
```

- `-ports`:
  Branches to the sub-section `-fdtd/-ports` where you specify the location of "ports" and some properties of these ports.

- `-pexcitation`:
  Branches to the sub-section `-fdtd/-pexcitation`, where you specify the center frequency, the bandwidth and the amplitude of a selected mode of a selected port.

- `-lcharge`:
  Branches to the sub-section `-fdtd/-lcharge`, where you specify the properties of a relativistic line charge.

- `-time`:
  Branches to the sub-section `-fdtd/-time`, where you specify the minimum and maximum allowed simulation time.
  You can also specify the times where the electric and magnetic fields are written.

- `doit`:
  If you say "doit", the settings in "-general, -mesh" are checked, the mesh is re-generated, and the iteration for the time dependent fields is performed.
  While the field computation is running, the amplitudes of selected port modes are written, and selected electric and magnetic fields are written. If a wake computation is performed, the data required for the computation of wakepotentials is also written. These amplitudes and fields (and wake-data) can be processed further by **gd1.pp**. After the field computation, the program stops.

1.7.3 -fdtd/-ports

A "port" is part of the border of the computational volume, that shall be treated as an infinitely long waveguide. In this section you specify the location of ports. You also specify the number of modes whose time amplitudes shall be written to file.

```
################################################################################
# Flags: nomenu, noprompt, nomessage,                                          #
################################################################################
# section: -ports                                                              #
################################################################################
# name  = noname-001                                                           #
# plane = xlow                                                                 #
# modes = 1                                                                    #
#(pxlow =      -1.0e+30    , pxhigh =      1.0e+30    ) Ignored..              #
# pylow =      -1.0e+30    , pyhigh =      1.0e+30                             #
# pzlow =      -1.0e+30    , pzhigh =      1.0e+30                             #
# epsmaximum=        2.0                                                       #
# muemaximum=        1.0                                                       #
# npml      = 15                     ( no of PML-planes )                      #
# dampn     =       50.0e-03         ( damping factor in last plane)           #
################################################################################
# doit, list, ?, return, help                                                  #
################################################################################
```

- `name = ANY-STRING-WHICH-COULD-SERVE-AS-FILENAME`:
  Specifies the name of the port.
  This name is used eg. to identify the port later on. If you enter the name of an already defined port, you can edit the parameters for this already defined port.
  The length of the name has to be less or equal 64 characters.

- `plane= [xlow|xhigh|ylow|yhigh|zlow|zhigh]`:
  Specifies at which of the six bounding planes of the computational volume the port is located on.

- `modes= NMODES`:
  Number of orthogonal modes whose amplitudes shall be monitored. This number can be zero. The absorbing boundary conditions work independently of the orthogonal modes, so there is no need to specify a large number here.

- `pxlow, pxhigh, pylow, pyhigh, pzlow, pzhigh`:
  Specifies the rectangle where the port is in. These parameters are needed if there is more than one port at one of the possible planes `[x|y|z][low|high]`.
  If the port is at xlow or at xhigh, the values for pxlow and pxhigh are ignored.
  If the port is at ylow or at yhigh, the values for pylow and pyhigh are ignored.
  If the port is at zlow or at zhigh, the values for pzlow and pzhigh are ignored.

- `epsmaximum, muemaximum`:
  Values of the "densiest" materials at the port. These values are needed to compute the patterns of the port-modes.

- `npml`:
  The number of "Perfectly Matched Layers" to use as absorbing boundary conditions.
  15 is a good value.
  If a relativistic charge enters or exits the computational volume, you should choose a value of 40 or more.

- `dampn`:
  The damping factor of the outermost "Perfectly Matched Layer".

- `doit`:
  Stores the current data and enables the editing of the parameters of a port that is not yet defined (a new port).

- `list`:
  Lists the names of the already defined ports.

Example:
The following specifies that we want to have attached four ports: Two are at the lower x-boundary, the names are `xlow1, xlow2`. Two ports are at the upper x-boundary of the computational volume, the names are `xhigh1, xhigh2`. The ports at the same boundary are distinguished by their `pzlow, pzhigh` parameters.

```
-fdtd
  -ports
     name= xlow1,  plane= xlow , pzlow= 0,  modes= 10, doit
     name= xhigh1, plane= xhigh, pzlow= 0,  modes= 2, doit
     name= xlow2,  plane= xlow , pzhigh= 0, modes= 2, doit
     name= xhigh2, plane= xhigh, pzhigh= 0, modes= 2, doit
```

1.7.4 -fdtd/-pexcitation: What port-mode should be excited

Here you specify what port-excitation should be used.

```
###############################################################################
# Flags: nomenu, noprompt, nomessage,                                         #
###############################################################################
# section: -pexcitation                                                       #
###############################################################################
# port  =  undefined                                                          #
# mode  = 1                                                                   #
# amplitude =       1.0                                                       #
# frequency =  undefined                                                      #
# bandwidth =  undefined                                                      #
# risetime  =       0.0                                                       #
###############################################################################
# list, ?, return, help                                                       #
###############################################################################
```

- `port= NAME-OF-AN-ALREADY-DEFINED-PORT`:
  The name of the port where the mode should be launched. This is a name that you have used in the section "-fdtd/-ports".
- `mode`:
  The number of the mode to launch. The first mode is the one with the lowest cut-off-frequency.
- `amplitude`:
  The amplitude of the mode to launch. If the amplitude is zero, no mode is launched.
- `bandwidth`:
  The frequency bandwidth of the time signal to use as excitation.
- `risetime`:
  If specified as nonzero, a monochromatic excitation is used.
- `list`:
  Lists the names and planes of the ports as specified in section "-fdtd/-ports".

Example:
The following specifies that the fundamental mode of the port with name `InputPort` shall be excited. Its amplitude shall be '1', the centerfrequency of the excited pulse shall be 1.2 GHz, and the bandwidth of the excited pulse shall be 0.7 GHz.

```
-fdtd,
   -pexcitation
       port= InputPort
       mode= 1
       amplitude= 1
       frequency= 1.2e+9
       bandwidth= 0.7e+9
```

1.7.5 -fdtd/-lcharge: Properties of a relativistic line charge

```
###############################################################################
# Flags: nomenu, noprompt, nomessage,                                        #
###############################################################################
# section: -lcharge                                                          #
###############################################################################
# charge    =        0.0              [As]                                   #
# sigma     =  undefined              [m]                                    #
# xposition =        0.0              [m]                                    #
# yposition =        0.0              [m]                                    #
# ds        =       auto              [m]                                    #
# shigh     =       auto              [m]                                    #
# showdata  = no                       # ( yes | no )                        #
###############################################################################
# ?, return, help                                                            #
###############################################################################
```

- `charge`:
  The total charge of the gaussian line charge.
  If the position of the charge is specified such that the charge travels along a magnetic plane, the charge taken for the computational volume is halfed.
  If the charge travels along two symmetry planes simultaneously, the charge in the volume is a quarter of the specified charge.
  This way, the excited wakefields in a half or a quarter of the structure are the same as if the charge has traveled in a full structure.

- `sigma`:
  The width of the gaussian line charge.

- `xposition, yposition`:
  The position of the line charge in the x-y-plane.

- `ds`:
  The s-resolution of the wakepotentials.
  Possible values are "`auto`" or a positive real. If "`ds= auto`", a value of "spacing/2" is used. ("spacing" is defined in `-mesh`)

- `shigh`:
  The s-coordinate of the last wakepotential wanted. Possible values are "`auto`" or a positive real. If "`shigh= auto`", a value of "20 * sigma" is taken.

- `showdata= [yes|no]`:
  If "yes", some diagnostic plots concerning the wake-potential are shown when the time domain computation starts.

Example:
The following specifies that we want to model a line charge traveling with the speed of light along the axis (x,y)=(0,0). The total charge of the line-charge shall be 1 pC, and its gaussian width sigma shall be 1 mm. We want to have a s-resolution of the wakepotentials of 1/10 mm, and we are interested in s-values up to 100 mm.

```
 -lcharge
    xpos= 0
    ypos= 0
    charge= 1e-12
    sigma= 1e-3

    ds= 0.1e-3
    shigh= 100e-3
```

1.7.6 -fdtd/-time: minimum / maximum time, when to store

```
##############################################################################
# Flags: nomenu, noprompt, nomessage,                                        #
##############################################################################
# section -time                                                              #
##############################################################################
# firstsaved    = undefined                                                  #
# lastsaved     = undefined                                                  #
# distancesaved = undefined                                                  #
# tminimum      = undefined                                                  #
# tmaximum      = undefined                                                  #
# decaytime     =       1.0e+30                                              #
##############################################################################
# amptresh =       3.0e-03                                                   #
# dtsafety =      950.0e-03                                                  #
# ndt      = auto                                                            #
##############################################################################
# return, help, ?                                                            #
##############################################################################
```

- `firstsaved`:
  The first time where the electric and magnetic fields shall be stored on files.

- `lastsaved`:
  The last time where the fields shall be stored on files.

- `distancesaved`:
  The time-distance between fields to be saved.

- `tminimum= TMIN, tmaximum= TMAX`:
  The minimum and maximum time to simulate. **gd1** will simulate minimum a simulation time of `TMIN` and maximum one of `TMAX`. If after `TMIN` no power is flowing through any port, the simulation is stopped. No matter what the status of the fields in the computational volume is after a simulation time of `TMAX`, the simulation will be stopped.

- `amptresh= TRESH`:
  The treshold value for exiting the FDTD-Loop. when all port-amplitudes have decayed down to TRESH times the maximum of all monitored modes somewhere in the time, the computation stops.

- `dtsafety`:
  The safety factor for the time step.
  gd1 computes the largest stable timestep as
  $dtmax = \sqrt{\frac{4}{\text{"highest eigenvalue of the closed volume"}}}$,

This is normally reliable and gives the minimum number of timesteps required for a wanted simulation time. The used timestep is
dt = dtsafety * dtmax

- `ndt`:
  This enforces the used time step "dt" to be such that "dt" is smaller than the normally used timestep and that an integer multiple of "dt" gives the value of "ndt".

Example:
The following specifies that we want to simulate minimum a time span of 100 periods of a frequency of 1 GHz. We are absolutely shure that after a timespan of 1000 periods the fields have decayed sufficiently, and we want to store the fields between 10 periods and 20 periods, in a distance of 1/2 period.

```
define(FREQ, 1e9)
tmin= eval(100/FREQ)
tmax= eval(1000/FREQ)
firstsaved= eval(10/FREQ)
lastsaved=  eval(20/FREQ)
distance=   eval(1/2/FREQ)
```

Chapter 2

gd1.pp

gd1.pp is the postprocessor. **gd1.pp** loads the data that were computed by **gd1** and displays them. **gd1.pp** also computes integrals over the fields, like wall losses and voltages. Together with the macro facility, this allows easy computation of figures of merit like Q-values and shunt-impedances.
A special section (`-sparameter`) computes scattering parameters from the amplitudes of port-modes that were computed and stored by **gd1**.
Wakepotentials and impedances are computed in the section `-wakes`.

Before you can do anything useful, you have to specify from what database **gd1.pp** shall take the fields from. Therefore the first thing you do is: enter the section `-general`.

2.1 -base

```
##############################################################################
# Flags: nomenu, noprompt, nomessage,                                        #
##############################################################################
# section: -base                                                             #
##############################################################################
# -general        -- define database                                         #
# -3darrowplot                                                               #
# -lineplot                                                                  #
# -energy         -- compute stored energy                                   #
# -lintegral      -- compute voltages                                        #
# -losses         -- compute wall losses                                     #
# -sparameter                                                                #
# -material       -- specify conductivities for loss computations            #
# -wakes          -- wakepotentials, impedances                              #
# ***** Miscalleneous ******                                                 #
#  -debug       : specify debug levels                                       #
#                                                                            #
##############################################################################
# ?, help, end, ls                                                           #
##############################################################################
```

- `-general`:
  Branches to the -general section, where you load the database where **gd1** has written its data to.

- `-3darrowplot`:
  Branches to the section -3darrowplot, where you can plot 3D-fields together with the material-boundaries.
  You can also plot the computed portmodes that were used in a time-domain computation.

- `lineplot`:
  Branches to the section lineplot, where you can plot selected components of 3D-Fields along selected lines.

- `-energy`:
  Branches to the section -energy, where you can compute the stored energy in resonant or time dependent fields.

- `-losses`:
  Branches to the section -losses, where you can compute wall losses from magnetic fields via the perturbation formula.

- `-sparameter`:
  Branches to the section -sparameter, where you can compute and plot the scattering parameters from the time data that were computed by **gd1**.

- `material`:
  Branches to the section -material, where you specify the conductivities of electric materials. These conductivities are used for the perturbation formula that is applied in the section -losses.

- `-wakes`:
  Branches to the section -wakes, where you can compute longitudinal and transverse wakepotentials from data that were recorded by **gd1** during a time domain computation with a relativistic charge.

2.2 -general

```
###############################################################################
# Flags: nomenu, noprompt, nomessage,                                         #
###############################################################################
# section: -general                                                           #
###############################################################################
# infile     = ./results                                                      #
# scratchbase= /tmp/bruw1931/garbage/gdfidl-scratch-pid=10958-                #
# text( 1)=' '                                                                #
###############################################################################
# plotopts    =  -landscape -geometry 640x530                                 #
# showtext    = yes           # (yes | no)                                    #
# onlyplotfile= no            # (yes | no)                                    #
###############################################################################
# ?, return, end, help                                                        #
###############################################################################
```

- `infile`:
  The name of the resultfile from **gd1**. "infile= @last" is special: the name of the last computed run is taken (this value is read from the file $HOME/name.of.last.gdfidl.file)

- `scratchbase`:
  The base name of scratchfiles that **gd1.pp** needs for its operation. These will be mainly plotfiles. If you have an environment variable `TMPDIR` set, SOLVER will as default value set `scratchbase` to the string

  ```
  $TMPDIR/gdfidl-scratch-pid=XXXXX-
  ```

  Here `$TMPDIR` is the value of the environment variable `TMPDIR`, and `XXXXX` is the number of the process-id of **gd1**.

- **text(*)=** This annotation text is plotted together with the field plots. **syntax:**
  ` text()= ANY STRING, NOT NECESSARILY QUOTED`
  or
  ` text(NUMBER)= ANY STRING, NOT NECESSARILY QUOTED`

  - `ANY STRING, NOT NECESSARILY QUOTED`:
    The string to be included in the plots,
  - `NUMBER`:
    Optionally, the line number, where the text should be plotted.

In the first case, without `NUMBER`, the string following `text()=` is placed in the next free line. In the case with `NUMBER`, it is guaranteed, that the string is placed in the `NUMBER`.st line. You can specify up to 20 annotation strings, the maximum length of each annotation string is is 80 characters.

When a new database is specified, the values of `text` are overwritten by the values that are found in the database.

- `plotopts= ANY STRING CONTAING OPTIONS FOR mymtv2`:
  gd1.pp does not display the data itself, but writes a datafile for **mymtv2** and starts **mymtv2** to display these data.
  Useful options are:
  - -landscape : Produce the PostScript plot in landscape mode
  - -colorps : Produce colour PostScript
  - -printcmd command : Use as printcommand `command`
  - -Pprinter: Print to printer `printer`
  - -pbg background_color : Use as X11-background colour the colour `background_color` (grey | white | ..)
  - -nopixmap : Good for very dumb terminals. If only the first one or two plots appear, try this.
  - -geometry X11-GEOMETRY Initial geometry for the X11 window of **mymtv2**.
- `showtext= [yes|no]`:
  This flags, whether the annotation text shall appear in the plots.
- `onlyplotfile= [yes|no]`:
  This flags, whether plotfiles shall be written AND **mymtv2** shall be started to display them on an X11 display, or whether only the plotfiles shall be produced.

Example:
The following specifies that we want to work with the data that were generated by the last run of **gd1**. We want to have the scratchfiles written into the directory '/tmp/garbage/' and there the files shall be called 'delete-me-*'. We want to have **mymtv2** started with an X11-geometry of 1024x768, and **mymtv2** shall produce colour-PostScript.

```
infile= @last
scratchbase= /tmp/garbage/delete-me-
plotopts= -geometry 1024x768 -colorps
```

2.3 -3darrowplot: Plots 3D Fields together with the material boundaries.

```
################################################################################
# Flags: nomenu, noprompt, nomessage,                                          #
################################################################################
# section: -3darrowplot                                                        #
################################################################################
# symbol      = e_1                                                            #
# quantity    = e                                                              #
# solution    = 1                                                              #
#    phase    = 45.0                                                           #
# arrows      = 1000                                                           #
# lenarrows   = 2.0                                                            #
# maxlenarrows= 2.0                                                            #
#   fcolour   =  5                                                             #
# materials   = yes          # (yes | no)                                      #
# scale       = 1.80                                                           #
# fscale      = auto                                                           #
# eyeposition = ( 1.0, 2.0, 500.0e-03)                                         #
# lightposition= ( 1.0, 1.0, 1.0)                                              #
# ncolors     = 20                                                             #
# icoloroffset=  0                                                             #
# bbxlow =    -1.0000e+30, bbylow =    -1.0000e+30, bbzlow =    -1.0000e+30  #
# bbxhigh=     1.0000e+30, bbyhigh=     1.0000e+30, bbzhigh=     1.0000e+30  #
################################################################################
# plotopts    =  -landscape -geometry 640x530                                  #
# showtext    = yes         # (yes | no)                                       #
# onlyplotfile= no          # (yes | no)                                       #
################################################################################
# doit, ?, return, end, help                                                   #
################################################################################
```

- `symbol= QUAN_ISOL`:
  This is the full name of the symbol to be processed.

- `quantity= QUAN`:
  This is the first part of the ”symbol”.

- `solution= ISOL`:
  This is the last part of the ”symbol”, the index of the ”symbol”.

- `phase`:
  This parameter is used only when you want to plot the fields of portmodes.
  This specifies the phase of the phasor to be shown.

- `arrows`:
  The arrows that make up the plot are distributed equidistant over the whole computational volume. The number of points where an arrow is possible is given here.

- `lenarrows`:
  The relative length of the arrows that make up the plot. If `lenarrows= 1`, the arrows are scaled such, that for an homogeneous field the end of one arrow is past the end of the next arrow.

- `maxlenarrows`:
  No arrow shall be larger than this value.
  This is useful for fields where a strong field concentration happens, e.g. wake-fields.

- `fscale`:
  If a value $\neq$ `auto` is specified, the vectorfield is scaled by this value. No further scaling takes place. This is useful for generating movies, where every frame of the movie should be scaled by the same factor.

- `bbxlow=,bbxhigh=,bbylow=bbyhigh=,bbzlow=,bbzhigh=`:
  Specifies the coordinates of a bounding box. Only the material-boundaries and the field-arrows that lie within the box are plotted.

- `eyeposition= (XEYE, YEYE, ZEYE)`:
  This specifies the initial eyeposition for the 3D plot that will be produced. As soon as the plot appears, you can interactively change the eyposition with the buttons of **mymtv2**, but for a complex geometry, this takes a lot of time.

- `plotopts= ANY STRING CONTAINING OPTIONS FOR mymtv2`:
  gd1.pp does not display the data itself, but writes a datafile for **mymtv2** and starts **mymtv2** to display these data.
  Useful options are:

  - -landscape : Produce the PostScript plot in landscape mode
  - -colorps : Produce colour PostScript
  - -printcmd command : Use as printcommand `command`
  - -Pprinter: Print to printer `printer`
  - -pbg background_color : Use as X11-background colour the colour `background_color` (grey | white | ..)
  - -nopixmap : Good for very dumb terminals. If only the first one or two plots appear, try this.
  - -geometry X11-GEOMETRY Initial geometry for the X11 window of **mymtv2**.

- `showtext= [yes|no]`:
  This flags, whether the annotation text shall appear in the plots.

- `onlyplotfile= [yes|no]`:
  This flags, whether plotfiles shall be written AND **mymtv2** shall be started to display them on an X11 display, or whether only the plotfiles shall be produced.

Example:
To generate a plot of the first electric field found in the database, we say:

```
symbol= e_1
doit
```

2.4 -lineplot: Plots a field component along an axis.

```
##############################################################################
# Flags: nomenu, noprompt, nomessage,                                        #
##############################################################################
# section: -lineplot                                                         #
##############################################################################
# symbol    = e_1                                                            #
# quantity  = e                                                              #
# solution  = 1                                                              #
# component = z                                                              #
#                                                                            #
# direction = z                                                              #
# startpoint= ( undefined, undefined, undefined )                            #
# used:       ( undefined, undefined, undefined )                            #
##############################################################################
# plotopts =  -landscape -geometry 640x530                                   #
# showtext = yes             # (yes | no)                                    #
# onlyplotfile= no           # (yes | no)                                    #
##############################################################################
# doit, ?, return, end, help                                                 #
##############################################################################
```

- `symbol= QUAN_ISOL`:
  This is the full name of the symbol to be processed.
- `quantity= QUAN`:
  This is the first part of the "symbol".
- `solution= ISOL`:
  This is the last part of the "symbol", the index of the "symbol".
- `component= [x|y|z|]`:
  The wanted component of the 3D-field to be plotted.
- `direction= [x|y|z]`:
  The direction along which the component shall be plotted.
- `startpoint= (X0, Y0, Z0)`:
  The startpoint from which the component shall be plotted.
- `plotopts= ANY STRING CONTAING OPTIONS FOR mymtv2`:
  gd1.pp does not display the data itself, but writes a datafile for **mymtv2** and starts **mymtv2** to display these data.
  Useful options are:
  - -landscape : Produce the PostScript plot in landscape mode

  - -colorps : Produce colour PostScript
  - -printcmd command : Use as printcommand `command`
  - -Pprinter: Print to printer `printer`
  - -pbg background_color : Use as X11-background colour the colour `background_color` (grey | white | ..)
  - -nopixmap : Good for very dumb terminals. If only the first one or two plots appear, try this.
  - -geometry X11-GEOMETRY Initial geometry for the X11 window of **mymtv2**.

- `showtext= [yes|no]`:
  This flags, whether the annotation text shall appear in the plots.

- `onlyplotfile= [yes|no]`:
  This flags, whether plotfiles shall be written AND **mymtv2** shall be started to display them on an X11 display, or whether only the plotfiles shall be produced.

Example:
To generate a plot of the x-component of the electric field of the third field in the database, starting from the lowest z-coordinate in the grid, up to the highest z-coordinate at the position (x,y)= (0,0), we say:

```
symbol= e_3
comp x
dir z
start 0 0 @zmin
doit
```

2.5 -energy: compute energy in E or H fields

```
################################################################################
# Flags: nomenu, noprompt, nomessage,                                          #
################################################################################
# section: -energy                                                             #
################################################################################
# symbol   = h_1                                                               #
# quantity = h                                                                 #
# solution = 1                                                                 #
#                                                                              #
#                                                                              #
# @henergy : undefined                (symbol: undefined, m: 1)               #
# @eenergy : undefined                (symbol: undefined, m: 1)               #
################################################################################
# doit, ?, return, end, help                                                   #
################################################################################
```

In this section you may compute the stored energy for an electric or magnetic field. The result of the computation is stored in symbolic variables that may be used eg. for computation of user defined figures of merit.

- `symbol= QUAN_ISOL`:
  This is the full name of the symbol to be processed.
- `quantity= QUAN`:
  This is the first part of the "symbol".
- `solution= ISOL`:
  This is the last part of the "symbol", the index of the "symbol".
- **doit**: Performs the computation. The specified field is loaded from file and its energy density is integrated over the computational volume.

  If the integrated field was an electric field, the result of the compuation is stored in the symbolic variable **@eenergy**. If the integrated field was a magnetic field, the result of the compuation is stored in the symbolic variable **@henergy**.

The energy for a magnetic fields is computed as:

$$@henergy = \frac{1}{2m} \int \mu H^2 \, dV \tag{2.1}$$

The energy for an electric fields is computed as:

$$@eenergy = \frac{1}{2m} \int \varepsilon E^2 \, dV \tag{2.2}$$

The time-averaging factor "m" is "2" for resonant fields, and "1" for nonresonant fields.

Example:
To compute the stored energy in the electric field of the first field found in the database, we say:

```
 symbol= e_1 doit
```

2.6 -lintegral: computes line integrals

This section computes the integral $\int F \exp(j\omega/(\beta c_0))\, ds$.

```
################################################################################
# Flags: nomenu, noprompt, nomessage,                                          #
################################################################################
# section: -lintegral                                                          #
################################################################################
# symbol    = e_1                                                              #
# quantity  = e                                                                #
# solution  = 1                                                                #
#                                                                              #
# direction = z                                                                #
# component = z                                                                #
# startpoint= ( undefined, undefined, undefined )                              #
#    (used) : ( @x0: undefined, @y0: undefined, @z0: undefined )               #
# length    = auto                                                             #
#    (@length) : undefined                                                     #
# beta      = 1.0                                                              #
################################################################################
# @vreal= undefined        @vimag= undefined       @vabs= undefined            #
################################################################################
# doit, ?, return, end, help                                                   #
################################################################################
```

- `symbol= QUAN_ISOL`:
  This is the full name of the symbol to be processed.
- `quantity= QUAN`:
  This is the first part of the "symbol".
- `solution= ISOL`:
  This is the last part of the "symbol", the index of the "symbol".
- `direction= [x|y|z]`:
  The cartesian direction along which the integral shall be performed.
- `component= [x|y|z]`:
  The cartesian component that shall be integrated.
- `length`:
  Possible values are `auto`, or a real number.
  If `length= auto`, the integral is performed from the startpoint to the end of the computational domain (in `direction`-direction).
- `beta`:
  The value of β in the formula $\int F \exp(j\omega/(\beta c_0))\, ds$. If you are interested in computing $\int F\, ds$, then choose `beta` as a very large number, say 1e+10.

- `doit`:
  The integral is performed, the results are shown in the menu. The value of the integral is also accessible as the symbolic variables **@vreal, @vimag, @vabs**. The length of the used integration path is accessible as the variable **@length**. The coordinates of the used startpoint are accesible as **@x0, @y0, @z0**.

2.7 -losses: Compute wall losses from H-fields

```
################################################################################
# Flags: nomenu, noprompt, nomessage,                                          #
################################################################################
# section: -losses                                                             #
################################################################################
# symbol   = h_1                                                               #
# quantity = h                                                                 #
# solution = 1                                                                 #
#                                                                              #
#                                                                              #
# @metalpower : undefined            (symbol: undefined)                       #
################################################################################
# doit, ?, return, end, help                                                   #
################################################################################
```

In this section you may compute the wall losses that stem from the induction of surface currents from magnetic fields. This is a power loss perturbation computation.

The conductivities that are used in the pertubation formula may be entered in the section **-material**. The result of the computation is available as the symbolic variable **@metalpower**.

The wall losses are computed as:

$$\int\int \frac{H^2}{1/(2\kappa\delta)} dF \tag{2.3}$$

$$\frac{1}{\delta} = \sqrt{\pi f \mu_0 \kappa} \tag{2.4}$$

The integral is performed over all metallic surfaces that would appear in a plot as produced by the section **-3darrowplot**. This implies, that wall losses are NOT computed for electric symmetry planes, since the material on the symmetry planes are not shown in **-3darrowplot**.

- `symbol= QUAN_ISOL`:
  This is the full name of the symbol to be processed. This field has to be a 3D-H-field.

- `quantity= QUAN`:
  This is the first part of the "symbol".

- `solution= ISOL`:
  This is the last part of the "symbol", the index of the "symbol".

- `doit`:
  The integral is performed, the result is shown in the menu. The result is available as the symbol `@metalpower` for subsequent calculations.

2.8 -sparameter: Computes scattering parameters from time dependent data

```
###############################################################################
# Flags: nomenu, noprompt, nomessage,                                         #
###############################################################################
# section -sparameter                                                         #
###############################################################################
# ports      = all                                                            #
#              # (all | LIST )                                                #
# modes      = 1                              # (all | LIST )                 #
# timedata   = no                             # ( yes | no )                  #
#   tsumpower= no                             # ( yes | no )                  #
#   tintpower= no                             # ( yes | no )                  #
# freqdata   = yes                            # ( yes | no )                  #
#   fsumpower= yes                            # ( yes | no )                  #
#   magnitude= yes                            # ( yes | no )                  #
#   fintpower= no                             # ( yes | no )                  #
#     slog   = no                             # ( yes | no )                  #
#   phase    = no                             # ( yes | no )                  #
#   smithplot= no                             # ( yes | no )                  #
# upto       = auto                     [s]   # ( auto | REAL )               #
# flow       = auto                     [1/s] # ( auto | REAL )               #
# fhigh      = auto                     [1/s] # ( auto | REAL )               #
# ignoreexc  = no                             # ( yes | no )                  #
# details    = no                             # ( yes | no )                  #
###############################################################################
# plotopts =  -landscape -geometry 640x530                                    #
# showtext    = yes         # (yes | no)                                      #
# onlyplotfile= no          # (yes | no)                                      #
###############################################################################
# return, help, end, doit                                                     #
###############################################################################
```

- `ports`:
  Possible values are `all`, or a list of names of ports.
  If `ports= all`, the time dependent data of all ports found in the data-base are processed.
  If `ports` is a list of names, only the time dependent data of the ports whose names are found in the list are processed.

- `modes`:
  Possible values are `all`, or a list of mode-numbers.
  If `modes= all`, the time dependent data of all modes of the selected ports found in the data-base are processed.

If `modes` is a list of mode-numbers, only the time dependent data of the modes with numbers in the list are processed.

- `timedata`:
  If `timedata= yes`, the time dependent amplitudes of the selected modes are plotted.

- `tsumpower`:
  If "yes", the sum of the power of the selected modes is computed as a function of time.

- `tintpower`:
  If "yes", the sum of the power of the selected modes is computed as a function of time and integrated over time.

- `freqdata`:
  If `freqdata= no`, no scattering parameters are plotted. This is: no magnitudes, no phase and no smithplots.

- `fsumpower`:
  If "yes", the sum of the power of the selected modes is computed as a function of frequency.

- `magnitude`:
  If `magnitude= yes`, the absolute value of the scattering parameters are plotted.

- `fintpower`:
  If "yes", the sum of the power of the selected modes is computed as a function of frequency and integrated over frequency.

- `slog`:
  If `slog= yes`, the magnitude of the scattering parameters are initially plotted in a logarithmic plot.

- `phase`:
  If `phase= yes`, the phase of the scattering parameters are plotted.

- `smithplot`:
  If `smithplot= yes`, the scattering parameters are plotted in a smith-plot.

- `upto`:
  Possible values are `auto` or a time-value.
  If `upto= auto`, the time data upto the last found simulated time are considered for the fourier-transforms.
  If `upto= TMAX`, only the time-data upto the time-value `TMAX` are considered for the fourier-transforms. This is useful, for the case that a late time instability of the time domain computation has occured, and you want to know the

scattering parameters of a shorter simulation. THIS SHOULD NOT HAPPEN. IF YOU ENCOUNTER A LATE TIME INSTABILITY, PLEASE SEND THE INPUTFILE WHERE THIS INSTABILITY HAS OCCURED TO "bruns@tetibm2.ee.TU-Berlin.de".

—

If "upto" is specified as a value larger than the time simulated by the FDTD-solver, then the time values up to "upto" are filled up with zeroes. This has the effect that additional frequency points are computed by FFTing the padded data set. The scattering parameters at the additional frequency points are interpolated from the primary frequency dependent values by a sin(x)/x interpolation.

- `flow`:
  If `flow= auto`, the lower boundary of the frequency range of the plots is derived from the centerfrequency and the bandwidth of the excitation as they were specified in the input for **gd1**. If `flow= FMIN`, the lower boundary of the frequency range is `FMIN`.

- `fhigh`:
  If `fhigh= auto`, the upper boundary of the frequency range of the plots is derived from the centerfrequency and the bandwidth of the excitation as they were specified in the input for **gd1**. If `fhigh= FMAX`, the lower boundary of the frequency range is `FMAX`.

- `ignoreexc`:
  If `ignoreexc= yes`, the spectrum of the excited mode is ignored. Instead a flat spectrum is assumed. This is useful e.g. to compute the coupling from an relativistic charge to the port-modes.

- `doit`:
  The scattering parameters are computed from the time dependent amplitudes of the port-modes.

Example
To compute and plot the scattering parameters of only the first two modes of the ports wth names `InPut, Output`, we say:

```
modes= ( 1, 2 )
ports= ( InPut, OutPut )
doit
```

2.9 -material: Conductivities of electric materials

This section enables the changing of the conductivities of materials with **type= electric**. These conductivities are used for computing the wall losses in the section **-losses**.

```
##############################################################################
# Flags: nomenu, noprompt, nomessage,                                        #
##############################################################################
# section -material                                                          #
##############################################################################
# material=  1        # epsr....: infinity       # kappa    =      58.0e+06  #
# type: electric      #    xepsr: infinity       #   xkappa =      58.0e+06  #
#                     #    yepsr: infinity       #   ykappa =      58.0e+06  #
#                     #    zepsr: infinity       #   zkappa =      58.0e+06  #
#                     # muer....:       0.0      # mkappa   :       0.0      #
#                     #    xmuer:       0.0      #   xmkappa:       0.0      #
#                     #    ymuer:       0.0      #   ymkappa:       0.0      #
#                     #    zmuer:       0.0      #   zmkappa:       0.0      #
##############################################################################
# return                                                                     #
##############################################################################
```

- **material**
  The material index of the material whose conductivity is to be changed.
- **kappa**
  The electric conductivity of the material in MHO/m (1/Ohm/m).
- **xkappa, ykappa, zkappa**
  The x-, y-, z-value of an anisotropic material. Only diagonal kappa matrices can be specified.

The entries for `type, epsr, muer, mkappa` cannot be changed.

Example:
To specify that wall loss computations in the section `-losses` shall use a conductivity of 30e6 for the material '10', we say:

```
material= 10
kappa= 30e6
```

2.10 -wakes: longitudinal and transverse wakepotentials

This section enables the computation of longitudinal and transverse wake potentials from data that were computed by **gd1**. These data are only recorded by **gd1** when you did specify a charge in the section **-lcharge** of **gd1**.

gd1 computes the integral of the E_z component along the outermost paths where a beam can travel and stores the result in the database. Since from these data the longitudinal and transverse wakepotentials everywhere in the beam pipe can be computed, you can specify an unlimited number of positions (x,y) where you are interested in the wakepotentials. The (x,y) position of the exciting charge cannot be changed afterwards, though.

```
###############################################################################
# Flags: nomenu, noprompt, nomessage,                                         #
###############################################################################
# section -wakes                                                              #
###############################################################################
# watq      = yes  # Process all wakes at positions of line-charges          #
# awtatq    = yes  # Use the average of the two nearest transverse wakes     #
#                  # as the transverse wakes at the positions of charges     #
# impedances= no   # Compute impedances.                                     #
# window    = yes  # Apply hann-window when computing impedances.            #
# watxy = ( undefined, undefined)                   # want wz(xi,yi,s), i= 1 #
# wxatxy= ( undefined, undefined)                   # want wx(xi,yi,s), i= 1 #
# wyatxy= ( undefined, undefined)                   # want wy(xi,yi,s), i= 1 #
# watsi = undefined                   # want w(x,y,si), i= 1                 #
# watxi = undefined                   # want w(xi,y,s), i= 1                 #
#   liny= 20                          # number of lines in y-direction.      #
# watyi = undefined                   # want w(x,yi,s), i= 1                 #
#   linx= 20                          # number of lines in x-direction.      #
# wxatxi= undefined                   # want wx(xi,y,s), i= 1                #
# wxatyi= undefined                   # want wx(x,yi,s), i= 1                #
# wyatxi= undefined                   # want wy(xi,y,s), i= 1                #
# wyatyi= undefined                   # want wy(x,yi,s), i= 1                #
# istrides= 3                         # distance of s-points of the plots    #
#                                     # in units of "ds".                    #
###############################################################################
# plotopts =  -landscape -geometry 640x530                                    #
# showtext    = yes          # (yes | no)                                     #
# onlyplotfile= no           # (yes | no)                                     #
###############################################################################
# return, help, end, clear, doit                                              #
###############################################################################
```

- **watq=** [**yes** | **no**]
  watq stands for "Wake at Q-position". If **watq= yes**, then the longitudinal and transverse wakepotentials at the x-y-position of the exciting charge are computed. You do not have to specify these position by yourself.

- **awtatq=** [**yes** | **no**]
  awtatq stands for "Average Wakes (transverse) at Q-position". In a finite difference grid, the transverse wakepotentials are best defined just in between the grid planes. But the exciting charge can only travel along a grid line (the crossing line of two grid planes). So the transverse wakepotential just at the position of a charge is not well defined.

  If **awtatq= yes**, **gd1.pp** computes the transverse wakepotentials at the two positions between the grid planes that are nearest to the position of the exciting charge. The average of the two potentials are then assumed to be the transverse wakepotential at the position of the charge.

- **impedances=** [**yes** | **no**]
  If **impedances= yes**, the spectrum of wakepotentials at points (x,y) are computed and divided by the spectrum of the exciting current.

- **clear**
  Clear the list of positions where to plot wakepotentials in addition to the **watq**'s. See the explanation for **watxy, wxatxy, wyatxy, watsi, watxi, watyi, wxatxi, wxatyi, wyatxi, wyatyi** below.

- **watxy= (Xi, Yi), wxatxy= (Xi, Yi), wyatxy= (Xi, Yi)**
  watxy, wxatxy, wyatxy stands for "Wake at xy-Position, Wake in x at xy-Position, Wake in y at xy-Position". If you specify **watxy= (Xi, Yi)**, the longitudinal wakepotential at the gridpoint nearest to the specified position (Xi, Yi) will be computed. Similiar, when you specify **wxatxy= (Xi, Yi)** or **wyatxy= (Xi, Yi)**, the transverse wakepotentials at the midpoints between gridpoints nearest to the spicified position (Xi, Yi) will be computed.

  You can specify an unlimited number of (x,y)-positions where you want to know the longitudinal or transverse wakepotentials.

- **watsi= Si**
  watsi stands for "Wake at S-Position". If you specify **watsi= Si**, the longitudinal in the whole (x,y) region of the beam pipe at the s-value Si will be computed.

  You can specify an unlimited number of s-positions where you want to know the longitudinal or transverse wakepotentials.

- **watxi= Xi, watyi= Yi watxi** stands for "Wake at x-Position", **watyi** stands for "Wake at y-Position", If you sepcify **watxi= Xi** or **watyi= Yi**, the longitudinal wakepotential at the position Xi, or Yi will be plotted as a function of (y,s) or (x,s), respectively. These function will be plotted with **liny** or **linx** lines respectively.

You can specify an unlimited number of y- or y-positions where you want to know the longitudinal wakepotentials.

- **linx= LX, liny= LY**
  Number of lines to use to plot the data requested with **watxi= Xi, watyi= Yi**.

- **wxatxi= Xi, wxatyi= Yi, wyatxi= Xi, wyatyi= Yi**
  wxatxi, wxatyi, wyatxi, wyatyi stands for "Wake in x at x-Position, Wake in x at y-Position, Wake in y at x-Position, Wake in y at y-Position".

  If you specify **wxatxi= Xi**, the transverse wakepotential in x-direction will be plotted in the y-s plane at the x-coordinate between meshplanes nearest to Xi.

  If you specify **wxatyi= Yi**, the transverse wakepotential in x-direction will be plotted in the x-s plane at the y-coordinate between meshplanes nearest to Yi.

  If you specify **wyatxi= Xi**, the transverse wakepotential in y-direction will be plotted in the y-s plane at the x-coordinate between meshplanes nearest to Xi.

  If you specify **wyatyi= Yi**, the transverse wakepotential in y-direction will be plotted in the x-s plane at the y-coordinate between meshplanes nearest to Yi.

  You can specify an unlimited number of y- or y-positions where you want to know the transverse wakepotentials.

- **istrides= IS**
  Specifies the distance of s-values in the plots requested via **watxy, wxatxy, wyatxy, watsi, watxi, watyi, wxatxi, wxatyi, wyatxi, wyatyi**. These plots contain a huge amount of data when large s-values are present. It may happen that **mymtv2** needs a long time to load these datasets, and also there may be way too much s-values in the plots. With a value greater than 1 of this parameter you can reduce the information in these plots.

- `plotopts= ANY STRING CONTAING OPTIONS FOR mymtv2`:
  gd1.pp does not display the data itself, but writes a datafile for **mymtv2** and starts **mymtv2** to display these data.
  Useful options are:

  - -landscape : Produce the PostScript plot in landscape mode
  - -colorps : Produce colour PostScript
  - -printcmd command : Use as printcommand `command`
  - -Pprinter: Print to printer `printer`
  - -pbg background_color : Use as X11-background colour the colour `background_color` (grey | white | ..)

  - -nopixmap : Good for very dumb terminals. If only the first one or two plots appear, try this.
  - -geometry X11-GEOMETRY Initial geometry for the X11 window of **mymtv2**.

- `showtext= [yes|no]`:
  This flags, whether the annotation text shall appear in the plots.

- `onlyplotfile= [yes|no]`:
  This flags, whether plotfiles shall be written AND **mymtv2** shall be started to display them on an X11 display, or whether only the plotfiles shall be produced.

- **doit**
  The requested wakepotentials are computed from the data in the database, for each dataset an instance of **mymtv2** is started to plot the data.

Example
In order to have plotted the longitudinal wakepotential at the planes x=1e-3 and x=2e-3:

```
watxi= 1e-3
watxi= 2e-3
```

Chapter 3

GdfidLs command language

3.1 Variables

A variable has a name and a value. You define or redefine a variable with the sequence `define(name, value)`.

- The name of the variable can be up to 32 characters long. The name must begin with an alphabetic character and may contain numbers and alphabetic characters.
- The value of the variable may be up to 132 characters long. It may contain any characters inside, except for '`(\)`'. Leading and trailing blanks in the value are ignored.

Whenever **gd1** or **gd1.pp** encounter the name of an already defined variable, the name is substituted by the value of the variable, and the line is interpreted again.

3.1.1 Defining variables from outside

Both **gd1** and **gd1.pp** can be supplied options that define variables from outside an inputfile. The syntax is `gd1 -Dname=value`. This way, you can eg. compute dispersion relations with simple shell scripts.

Example

```
#!/bin/sh
  # Given the proper "inputfile.gdf", this shell-script computes the
  # dispersion relation of some periodic structure.
  for PHASE in 0 20 40 60 80 100 120 140 160 180
  do
      gd1 -DThisPhase=$PHASE < inputfile.gdf > out.Phase=$PHASE
  done
```

3.2 Arithmetic expressions

Whenever **gd1** or **gd1.pp** encounter the string `eval(`, the matching closing brace is searched and the string inside the enclosing braces is interpreted as an arithmetic expression. The value of the expression is transformed to a string and substituted for `eval(expression)`.

Example :

```
 echo (2*3)      # this outputs "(2*3)"
 echo eval(2*3)  # this outputs "6"
```

The arithmetic expression may contain the arithmetic operators `+,-,*,/,**,%`. In addition to that, the boolean operators `==,!=,<,>,<=,>=` are handled. The result of applying a boolean operator is an integer 0 or 1. Zero stands for false, and 1 for true.

The functions `cos(), sin(), atan()` are recognised and evaluated.

3.3 do-loops

Sections of the inputfile can be interpreted repeatedly via do loops: The structure of a do loop is the same as in Fortran.

```
do M1, M2, M3
   # Loop-Body
enddo
```

The loop-body may itself contain do-loops, macro calls, whatever. The iteration variable `M1` is not restricted to integer values.

```
do i= 1, 100, 1   # count upwards
   echo I is i
enddo
do i= 100, 1, -1  # count downwards
   echo I is i
enddo
do i= 1, 2, 0.1   # non integer step
   echo I is i
   echo 2*I is eval(2*i)
enddo
```

3.4 if-then

Conditional execution of part of an inputfile is possible with `if` blocks.
An if block is:

```
  if (ARITHMETIC-EXPRESSION) then
     #
     # If-Block-Body
     #
  endif
```

If the `ARITHMETIC-EXPRESSION` evaluates to something else than '0' then the body of the If-Block is executed.

3.5 Macros

Anywhere in your inputfile you can define macros. A macro is enclosed between two lines: The first line contains the keyword `macro` followed by the name of the macro. All lines until a line with only the keyword `endmacro` are considered the body of the macro. When **gd1** or **gd1.pp** find such a macro, they read it and store the body of the macro in an internal buffer.

Example :

```
   #
   # This defines a macro with name 'foo'
   #
   macro foo
      echo I am foo, my first argument is @arg1
      echo The total number of arguments supplied is @nargs
   endmacro
```

When **gd1** or **gd1.pp** find a call of the macro, the number of the supplied arguments is assigned to the variable `@nargs`, and the variables `@arg1, @arg2, ..` are assigned the values of the supplied parameters of the call. Similiar to the user definable variables, the values of the arguments are strings. Of course it is possible to have a string eg. '1e-4' which happens to be interpreted in the right context as a real number.

Example :

```
   #
   # this calls 'foo' with the arguments 'hi', 'there'
   #
   call foo(hi, there)
```

Macro calls may be nested. The body of a macro may call another macro.

3.6 Result-variables

gd1.pp makes its results accessible as symbolic variables. The names of these variables all start with `@`. The exact name can be found in the description of the sections of **gd1.pp**. There are some other variables as well that have not been described.

- `@pi, @clight`: These are the values of π and of the velocity of light.

The following variables are defined as soon as a database has been specified:

- `@nx, @ny, @nz`: These contain the number of grid planes in the three coordinate directions.
- `@x(i), @y(i), @z(i)`: These are the positions of the i.th gridplane.
- `@xmin, @xmax, @ymin, @ymax`: These are the extreme coordinates of the computational volume.

gd1.pp has a special variable `@path`. Its value is a command string that would enter the current section.

This is the end of this document