Project:	*OpenCRG®*	Document No.	Issue:
Title:	User Manual	VI2009.050	K

Date:	July 30, 2012	no. of pages:	51
Issuing Party:	VIRES Simulationstechnologie GmbH		
Author:	various		
Distribution List:	public		

OpenCRG® Licensing

The following license terms apply to this document, the OpenCRG tools and libraries

Printed in Germany
July 2012

Table of Contents

1 Introduction

1.1 Scope

Thís document provides an overview of the APIs (C, Matlab) which can be used for evaluating and manipulating OpenCRG® data sets. It also contains a brief description of data formats used for storing OpenCRG® data.

1.2 The OpenCRG® C-API

The OpenCRG® C-API consists of a library and a set of tools. The latter also serves as a collection of examples for the usage of the library. All routines are provided in full source code.

The library can be used for reading, modifying and evaluating OpenCRG® data sets.

The API may be used free of charge in accordance with the licensing terms listed at the beginning of this document.

1.2.1 Platforms

The OpenCRG® C-API is written in ANSI-C and is supposed to be independent of hard- and software platforms. It was successfully tested in the following environments:

- Linux on PC
- MS Windows XP on PC
- Irix on sgi workstations

Big and little endian encoding of a given processor is detected automatically in the API.

1.2.2 Status

The OpenCRG® C-API is in release status. This implies that

- all available routines were tested on simplified examples
- performance optimization is in advanced state

1.3 The OpenCRG® Matlab Tools

The OpenCRG® Matlab Tools are provided on any system distributions including Matlab (R14) or higher. No Toolbox or additional extensions are required. It is successfully tested in the following environments:

- Linux on PC (incl. Matlab R 14 or higher)
- MS Windows XP on PC (incl. Matlab R 14 or higher)

1.3.1 Status

The OpenCRG® Matlab Tools are in release status.. This implies that

- all available routines were tested on simplified examples

1.4 Point of Contact

Further assistance concerning OpenCRG® and the C-API is provided

via the OpenCRG® website `www.opencrg.org`

via email `opencrg@opencrg.org`

and via the "classic style" of direct contact with human beings:

VIRES Simulationstechnologie GmbH
Grassinger Strasse 8
83043 Bad Aibling
Germany
phone +49.8061.939093-0
fax +49.8061.939093-13

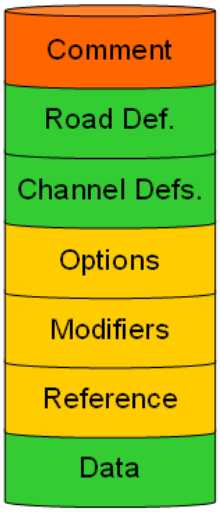
2 OpenCRG® Data Files

2.1 The Basics

OpenCRG® data files consist of various sections which are each enclosed by a line containing the corresponding keyword and a line with a terminating character set. Data files may contain road data in user-readable (i.e. ASCII) format or as binary stream (recommended for large data sets).

The keywords distinguishing the sections start with the $ character followed by the keyword itself. Keywords must be placed at the beginning of a new line and must not be followed by any data but some sort of comment. A section is terminated with a simple $ character at the beginning of a new line.

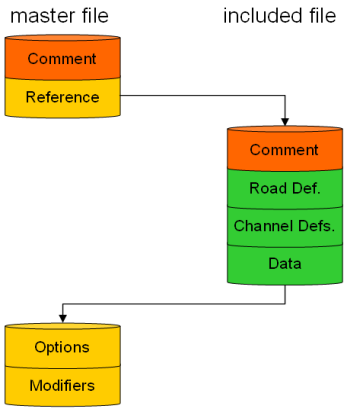
The possible sections within an OpenCRG® data file are shown in the following figure:



`$CT`:	mandatory comment about file's contents
`$ROAD_CRG`:	information about centerline; mandatory if file contains `Data` section,
`$KD_Definition`:	definition of individual channel parameters; mandatory if file contains `Data` section
`$ROAD_CRG_OPTS`:	evaluation options
`$ROAD_CRG_MODS`:	data set modifiers
`$ROAD_CRG_FILE`:	references to other data files
the actual OpenCRG® data	

The sequence of the sections within an OpenCRG® file is irrelevant, except for the `Comment` block which has to be the first section within the file and the `Data` block which, if present, has to be the last. However, it is recommended to use the above scheme for the creation of new files (for user readability).

An OpenCRG® file may contain reference directives pointing to other files. This is usually the case when a control file is used for defining options and modifiers (see below) and the actual data is contained in a separate file. A typical application for referring to other files is shown in the following figure.



2.2 Comments

Comments may be defined in two ways:

- the * character at the beginning of a line makes the entire line a comment
- the ! character within a line will cause all following characters to be treated as comment (until the end of the line)

2.3 The Sections in Detail

2.3.1 Header Information

Each data file MUST start with a comment block (header information) giving some information about the file's contents (preferably) etc.

The keyword is: `$CT`

Example:

```
$CT                                                  ! comment text block
CRG file example for road surface description (width: 3m, length: 22m)
with sloped referenceline and grid of (0.25m...1.0m) x 1.0m

Copyright 2005-2008 OpenCRG - Daimler AG - Jochen Rauh

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

More Information on OpenCRG open file formats and tools can be found at

    http://www.opencrg.org

$!*****************************************************************
```

Note that in the example, the header information block is terminated by the `$` character in the bottom line. The `!` character labels all following characters to be treated as comment and thus to be ignored by the data loader. They have just been added for user-readability of the data file.

2.3.2 Road Parameters

The road parameter section defines various parameters for the reference line and the overall configuration of the longitudinal sections. The minimum content that must be defined is the distance between lateral cross sections (REFERENCE_LINE_INCREMENT). All other parameters are optional. A complete road parameter list including MATLAB-API equivalents and descriptions is listed at ‘./matlab/lib/crg_intro.m’ (see ‘head’ section).

The keyword is: $ROAD_CRG

Example 1:

```
$ROAD_CRG                                          ! crg road parameters
REFERENCE_LINE_START_U   = 0.0
REFERENCE_LINE_START_X   = 0.0
REFERENCE_LINE_START_Y   = 0.0
REFERENCE_LINE_START_PHI = 0.0
REFERENCE_LINE_END_U     = 22.0                     !mandatory content
REFERENCE_LINE_END_PHI   = 0.0
REFERENCE_LINE_INCREMENT = 1.0
LONG_SECTION_V_RIGHT     =-1.50         ! with explicit definition below
LONG_SECTION_V_LEFT      = 1.50         ! with explicit definition below
$!*****************************************************************************
```

Example 2: To avoid potential problems in reference line reconstruction, it is recommended to add start and end positions in the header:

```
$CRG_ROAD
...
REFERENCE_LINE_START_X   = 0.0
REFERENCE_LINE_START_Y   = 0.0
REFERENCE_LINE_END_X     = 220.0
REFERENCE_LINE_END_Y     = 324.0
$
```

Example 3: If GPS (WGS84) co-ordinates of the start and end position are available, these should be provided additionally in the road parameter section (not yet supported by C-API):

```
$CRG_ROAD
...
reference_line_start_lon =  9.12345
reference_line_start_lat =  4.98765
reference_line_end_lon   =  9.22345
reference_line_end_lat   =  4.88765
$
```

2.3.3 Data Definition

The data definition block defines the data format (ASCII / binary) and the sequence of data which is to be expected in the trailing data block. Each definition starts with a `D:` identifier. Reference line and grid data may be defined within the data definition block (again, depending on the contents of the actual data block). Further details can be retrieved from the Matlab routines available for writing OpenCRG® data.

The keyword is: `$KD_Definition`

Example 1: sloped dataset with seven longitudinal sections

```
$KD_Definition                                    ! data definition block
#:LRFI                           ! one of the men readable IPLOS formats
D:reference line slope,m/m                          ! longitudinal slope
D:long section at v = -1.500,m           ! 1.50m right of reference line
D:long section at v = -1.250,m           ! 1.25m right of reference line
D:long section at v = -1.000,m           ! 1.00m right of reference line
D:long section at v =  0.000,m           !             on reference line
D:long section at v =  1.000,m           ! 1.00m left  of reference line
D:long section at v =  1.250,m           ! 1.25m left  of reference line
D:long section at v =  1.500,m           ! 1.50m left  of reference line
$!*************************************************************************
```

Example 2: curved data set with heading angle defined as first item of each data entry

```
$KD_Definition
#:LDFI
D:reference line phi,rad       ! heading angle -pi <= phi <= +pi
D:long section at v = -1.500,m ! longitudinal cuts with position
D:long section at v = -1.250,m
D:long section at v = -1.000,m
...
$
$$$$$$$
```

Example 3: longitudinal sections in fixed lateral spacing (lateral spacing 0.1m, surface width 3m); this example also shows the interaction between the blocks `$CRG_ROAD` and `$KD_Definition`

```
$CRG_ROAD ! minimal header
REFERENCE_LINE_INCREMENT = 0.01 ! 1cm spacing of lateral cuts
LONG_SECTION_V_RIGHT     =-1.50 ! right surface border
LONG_SECTION_V_LEFT      = 1.50 ! left surface border
LONG_SECTION_V_INCREMENT = 0.10 ! 10cm spacing of longitudinal cuts
$
$KD_Definition
#:LDFI
D:reference line phi,rad       ! heading angle -pi <= phi <= +pi
D:long section 1,m             ! 1.5m right of reference line
D:long section 2,m             ! 1.4m right of reference line
...
D:long section 31,m            ! 1.5m left of reference line
```

2.3.4 Modifiers and Options

As shown in the figure at the beginning of this chapter, OpenCRG® data files may also contain information about data evaluation options and data set modifiers. Both may, alternatively, be defined in run-time using the APIs.

The details of options and modifiers are described in the chapters below. Here, it shall only be stated that the keywords are:

Options: `$ROAD_CRG_OPTS`
Modifiers: `$ROAD_CRG_MODS`

Example:

```
$$ROAD_CRG_OPTS                          ! crg runtime evaluation options
*
* BORDER_MODE_U = 0             ! return NaN
* BORDER_MODE_U = 1             ! set zero
* BORDER_MODE_U = 2             ! keep last (default)
* BORDER_MODE_U = 3             ! repeat
* BORDER_OFFSET_U = 1.0         ! z offset beyond border (default: 0)
*
* BORDER_MODE_V = 0             ! return NaN
* BORDER_MODE_V = 1             ! set zero
  BORDER_MODE_V = 2             ! keep last (default)
* BORDER_MODE_V = 3             ! repeat
  BORDER_OFFSET_V = 1.0         ! z offset beyond border (default: 0)
$!*********************************************************************
```

NOTE: In the API routines, default modifiers and options are defined. *Options* provided in an individual data file may alter the default values of the respective options. Defining a *modifier* block in the data file will trigger the removal of **ALL** default modifiers. The user-defined modifiers will be set according to the information provided in the data file. If the user provides an empty "`$ROAD_CRG_MODS`" block, then no modifiers will remain at all (for default modifiers see also 3.2.7.1)

2.3.5 File Reference

OpenCRG® files may contain references to other files, typically containing the actual data. In the "master" file, conventions for modifiers and options will typically be defined. File references may contain absolute or relative paths and environment variables

The keyword is: `$ROAD_CRG_FILE`

Example:

```
$ROAD_CRG_FILE                                  ! crg file reference
* The file name may be split to multiple lines, which must not contain
* leading/trailing blanks, *, or ! to be considered as part of the name.
* Each line length must not exceed 72 characters.
* If a relative path is given, it is evaluated relative to the current
* directory of the reader process.
* If the first character is a $, subsequent characters will be replaced
* by the contents of the equivalent environment variable.
*
* > /home/name/crg-files/handmade_straight.crg  ! absolute path
*
* > /home/name/crg-files/hand                   ! absolute path
* >  made_straight.crg                          ! split in 2 lines
*
* > $crgpath/handmade_straight.crg              ! replace $crgpath by
* >                                             ! it's envvar contents
*
    ./hand                                  ! look in current dir
    made_straight_opts.crg                  ! look in current dir
$!*********************************************************************
```

2.3.6 Road Data

The road data section MUST ALWAYS be the last section of an OpenCRG® data file. It contains the actual road data (typically: elevation). Its formatting depends on the data format defined in the Data Definition section (see above). Since OpenCRG® files will typically be written using the Matlab routines, a further discussion of the various data formats is omitted (for the moment).

2.4 Sample Files

OpenCRG® sample data files are located in the directories

```
OpenCRG/crg-txt
OpenCRG/crg-bin
```

A distinction is made between human-readable text files and binary data files. Each type is located in its respective sub-directory.

The following files are available:

```
|----crg-bin
|    |----belgian_road.crg...binary data set for tests complex tests
|    |----country_road.crg..............a larger stretch of binary test data (*)
|----crg-txt
|    |----fileref.crg..................user-readable data set referring CRG file
|    |----fileref_mods.crg.............user-readable data set referring CRG file w. modifier
|    |----fileref_opts.crg.............user-readable data set referring CRG file w. options|
|----handmade_arc.crg.............user-readable data set for a ~180deg arc
|    |----handmade_banked.crg..........user-readable data set for banked road data
|    |----handmade_circle.crg..........user-readable data set for closed refline|    |----
handmade_curved.crg..........user-readable data set for curved refline
|    |----handmade_curved_banked_sloped.crg.........” data set for curved refline w. opts/mod
|    |----handmade_curved_minimalist.crg............” data set for minimalist curved refline
|    |----handmade_platform.crg.........user-readable data set for constantly elevated road
|    |----handmade_sloped.crg..........user-readable data set for sloped data set
|    |----handmade_sloped_opt.crg.......user-readable data set for sloped data set w. options
|    |----handmade_straight.crg.........user-readable data set for straight refline
|    |----handmade_straight_double.crg..user-readable data set for st. refline in double prec.
|    |----handmade straight_minimalist.crg..........” data set for minimalist straight refline
|    |----handmade_vtest.crg............user-readable data set for tests relating to v-options
|    |----testOptionBorderMode.crg......master file including data file and setting options
* Available as additional sample at www.opencrg.org/download.htm
```

Other files may be added to the data directories without further notice or explicit description in this documentation.

3 The OpenCRG® C-API

3.1 Getting Started

3.1.1 Downloading the Software Package

The latest stable version of the OpenCRG® C-API may be retrieved via the website

`www.opencrg.org`

Go to the `Downloads` area and look for the section `Packages and Tools`. There, you will find a package containing the C-API, MATLAB routines, documentation and sample files. Download this package and unpack it into a dedicated directory on your machine.

The C-API is located in the sub-directory

`OpenCRG/c-api`

`The latest developer version may be rerieved via the OpenCRG® project managing and issue-tracking-system via`

`http://viresftp.dyndns.org (registration required)`

3.1.2 Release Notes

The release notes are part of the software package and will not be transferred to this document. Please look for the "readme.txt" files in the root directory of the OpenCRG package and in the API's sub-directory.

3.1.3 Contents of the Package

The OpenCRG® C-API is delivered as a zipped package which, after unpacking, provides the following file structure (for additional / updated contents, see the `readme.txt` file):

```
|----baselib.................OpenCRG basic library - the core of the toolset
|    |----lib................location of the compiled OpenCRG library
|    |----inc................include files providing the interface to the library
|    |----makefile...........sample makefile for users preferring the make mechanism
|    |----obj................target directory for sources compiled with the make mechanism
|    |----src................the library's sources
|----compileScript.sh........script for the compilation of all demos and tools,
|                             based on simple compiler calls; this is an alternative to
|                             using the make mechanism; all files of the base library are
|                             also compiled with this script, so there is no need for a
|                             separate compilation of the library files.
|----demo....................demo sources showing the usage of the basic library
|    |----Simple.............a really simple application covering all basics of the API,
|    |                       runs with fix data sample "handmade_straight.crg"
|    |----EvalOptions........a set of routines demonstrating the usage of various options
|    |----EvalXYnUV..........a set of routines for the evaluation of OpenCRG reference lines
|    |----EvalZ..............an advanced example for the evaluation of OpenCRG data
|    |----Reader.............a sample application for a CRG file reader
|    |----bin
|    |    |----crgSimple....executable of the very simple example
|    |    |----crgEvalxyuv..executable of the reference line evaluator
|    |    |----crgReader....executable of the sample reader
|    |    |----crgEvalz.....executable of the complex z data evaluator
|    |    |----crgEvalOpts..executable of the option usage example
|    |----makefile...........makefile for all demos (alternative to "compileScript.sh")
|----makefile
|----readme.txt
|----test
|    |----PerfTest...............test tool for evaluating the performance of the library
|    |----bin
|    |    |----testModifiers.sh...script for performing a series of tests using the
```

```
|    |    |                       modifier mechanisms; requires gnuplot
|    |    |----testOptions.sh.....script for performing a series of tests using the
|    |    |                       evaluation option mechanisms; requires gnuplot
|    |    |----crgPerfTest........performance test tool; may not run on all platforms
|    |----makefile...............makefile for all tests (alternative to "compileScript.sh")
```

3.1.4 Compiling the Package

3.1.4.1 Method A – makefiles

On machines with gcc and standard make environment, just type

```
make
```

in the root directory. This should result in a series of executable files in the directories

```
demo/bin/
test/bin/
```

In addition, a library containing all object files of the `baselib/` sources is created in

```
baselib/lib
```

3.1.4.2 Method B – script

On machines having trouble with the provided makefiles, either adapt those files or use the very basic fallback solution which is a compile script. The script `"compileScript.sh"` is located in the root directory

```
OpenCRG/c-api/
```

Open the script, set the compiler variable "COMP" to the name of your compiler, re-save the script and execute it. The results should - again - be found in

```
demo/bin/
test/bin/
```

In contrast to the makefile mechanism, no library is explicitly created from the `baselib/` files.

3.1.4.3 Method C – command line

If you don't like makefiles and our scripts, you may just write your own simple compile instruction at command line level.

For this purpose, please note the following hints:

In order to compile a demo, set your include file search path to
`baselib/inc`

Always compile in combination with all .c-files in
`baselib/src`

Example: For the compilation of EvalXYnUV, use:

```
cc -lm -o EvalXYnUV -I baselib/inc demo/EvalXYnUV/src/main.c baselib/src/*.c
```

3.1.5 Basic Tests

On machines providing a shell environment and gnuplot (e.g. Linux systems), a series of very brief tests can be run. These are also used for acceptance tests of the C-API and have generated the figures in this document showing the various options and modifiers.

In order to run all tests for evaluation options, use the script

```
testOptions.sh
```

In order to run all tests for data set modifiers, use the script

```
testModifiers.sh
```

3.2 The Base Library

3.2.1 Overview

As mentioned above, the base library contains a large set of routines for

- reading
- modifying
- evaluating

OpenCRG® data. If you're looking for tools for creating data sets, please refer to the Matlab routines also available via the OpenCRG® website.

The base library is contained completely in the subdirectory

```
baselib/
```

of the OpenCRG® software package. It has the following structure:

`inc/`	all include files
`obj/`	objects and library resulting from compilation
`src/`	all source files
`makefile`	the makefile for compiling the library

Once compiled with the provided makefiles, the library is stored as archive in the directory

```
lib/
```

below the `OpenCRG/` root directory.

3.2.2 Include Files

The API comes with two include files

```
baselib/inc/crgBaseLib.h
baselib/inc/crgBaseLibPrivate.h
```

The first include file will typically be used when writing own applications featuring the OpenCRG® library. The second include file is used internally by the library's components and, therefore, can be ignored in most cases. It includes itself the first library, so when working with the "private" header, you don't have to include the "public" header explicitly.

3.2.3 Source Files

The library is composed of the following source files which are located in `baselib/src`:

`crgMgr.c`	overall data management
`crgMsg.c`	message / data log handling
`crgStatistics.c`	calculation of data set statistics
`crgContactPoint.c`	contact point management
`crgEvalxy2uv.c`	routines for evaluating x/y -> u/v
`crgEvaluv2xy.c`	routines for evaluating u/v -> x/y
`crgEvalz.c`	routines for evaluating x/y -> z, u/v -> z
`crgEvalpk.c`	routines for evaluating x/y -> phi / kappa, u/v -> phi / kappa
`crgLoader.c`	data file decoder and data loader
`crgOptionMgmt.c`	management of options and modifiers
`crgPortability.c`	collection of routines which might be subject to portability issues (e.g. when integrating the library within FORTRAN)

The interface to the methods in all source files is given in the include files listed in the previous chapter. The actual location of a method within the set of source files is irrelevant to the standard use of the library.

3.2.4 Working with Data Files

3.2.4.1 Simple Use Case

There are many operations that may be applied to OpenCRG® data sets. However, the simple way of using OpenCRG® data files is split into just three stages:

1. load the file
2. create a contact point
3. call the evaluation methods

This use case is shown in a programming example which is located at

```
demo/Simple/
```

3.2.4.2 Data Sets and Contact Points

This document and the entire C-API use two terms which shall be explained here:

Data Set a data set is the instance of data read from an OpenCRG® file into memory. It is identified by a unique integer ID which is returned by the data loader method.

Contact Point a contact point is a "tool" to query data of a given data set. The user must instantiate at least one contact point per data set in order to be able to query its contents. Contact points are identified with unique integer IDs which are returned by the contact point creation method. The number of contact points per data set is not limited.

3.2.4.3 Multi-Thread Applications

To parallelize C-API evaluation calls, contact points MUST NOT be shared between threads. Doing this will destroy history and other data stored within the contact point “object”.

3.2.4.4 Modifiers and Options

In simple cases, data sets will be read from file and evaluations may take place immediately. However, the user may want to vary the way a specific data set is evaluated without having to modify the original data (i.e. the data file) itself.

For this purpose, modifiers and options have been introduced.

- **Modifiers** modify the copy of OpenCRG® data which is stored in memory and, therefore, provide the modified data for all subsequent evaluation requests

- **Options** influence the way a specific query is performed without modifying any of the data stored in memory.

In the data files, modifiers are found in the section `$CRG_MODS` and options are found in `$CRG_OPTS`. The API provides discrete calls for setting modifiers directly and for applying them and the ones defined in the data file to the data set. For reasons of enhanced user control, modifiers defined in a data file are NOT applied automatically to the data set. It is the user's responsibility to apply them explicitly (see below).

The API also provides calls for setting options on a contact point level. When setting an option, its data type is taken into account, resulting in different methods for double and integer modifiers/options.

The set of available modifiers and options is defined as symbolic constants in the include file `crgBaseLib.h`.

Modifiers are set on a data set level by means of the routines

```
crgDataSetModifierSetInt( dataset, … )
crgDataSetModifierSetDouble( dataset, … )
```

After complete definition of all modifiers, they must be applied explicitly to the data set using the method

```
crgDataSetModifiersApply( dataset )
```

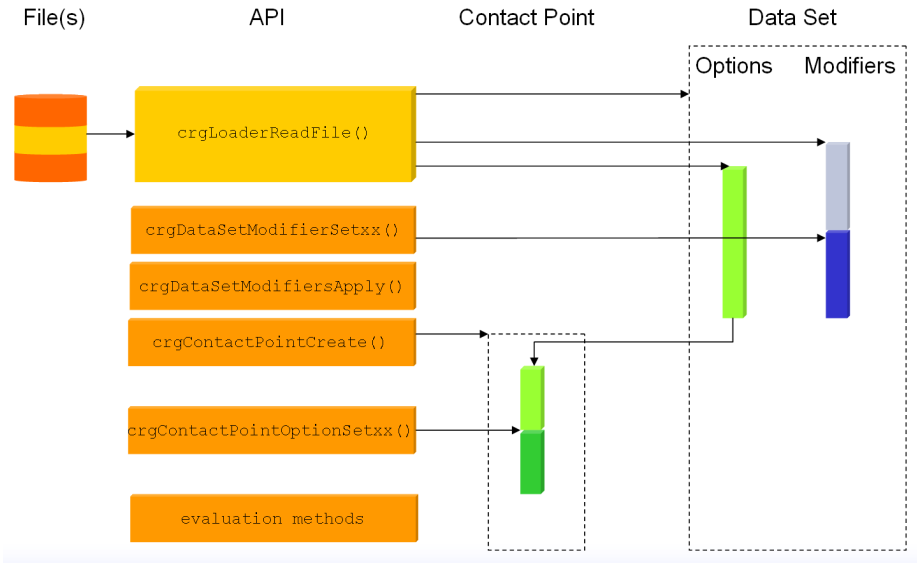
NOTE: This routine must also be called after reading the modifiers from a CRG file. Only defining the section `$ROAD_CRG_MODS` in a CRG files will NOT trigger the application of the modifiers to the data set.

In contrast to modifiers, **options** are applied implicitly each time an evaluation is performed. Therefore, they only have to be defined on a contact point level using the methods

```
crgContactPointOptionSetInt( contactPoint )
crgContactPointOptionSetDouble( contactPoint )
```

3.2.4.5 Sequence of Actions

The following figure gives an overview of the sequence of actions that may be performed when working with OpenCRG® data files.



- first, the user loads a file; the file may already contain definitions for evaluation options and data set modifiers.
- upon reading the file, a data set with a unique ID (returned by the loader method) is created where actual OpenCRG® data is stored as well as modifiers and options defined within the file.
- after reading the file, the modifiers of the data set may be replaced, deleted, extended by means of API calls referring to the data set.
- in order to evaluate data of the data set, the user needs to create a so-called “contact point”; this contact point is dedicated to a given data set; the library creates and returns a unique ID for each contact point
- the contact point inherits all evaluation options from the respective data set
- the user may specify further evaluation options, modify existing ones or delete all evaluation options inherited from the data set
- finally, evaluation of data may begin

3.2.5 Evaluating Data

Once a data set is loaded and a contact point is created, data can be evaluated in various ways (full prototypes of the referenced methods are in `crgBaseLib.h`)

- compute the x/y position of a given u/v position

```
crgEvaluv2xy( … )
```

- compute the u/v position of a given x/y position

```
crgEvalxy2uv( … )
```

- compute the z value at a given u/v position

```
crgEvaluv2z( … )
```

- compute the z value at a given x/y position

```
crgEvalxy2z( … )
```

- compute heading angle and curvature at a given u/v position

```
crgEvaluv2pk( … )
```

3.2.6 Message Printing

The API provides a series of methods for controlling verbosity of the library. Also, the user may use these methods to provide / control own messages which are to be printed to shell output (here: stderr).

The system knows the following message levels (in increasing order) which are controlled by symbolic constants in crgBaseLib.h

```
dCrgMsgLevelNone
dCrgMsgLevelFatal
dCrgMsgLevelWarn
dCrgMsgLevelNotice
dCrgMsgLevelInfo
dCrgMsgLevelDebug
```

The current message level is set by

```
crgMsgSetLevel( int level );
```

A message to be printed at or below a certain level is defined in standard printf() syntax via

```
crgMsgPrint( int level, const char *format, ...);
```

If a user wants to perform custom message handling, a callback may be defined which is invoked instead of the internal print routines. The syntax for setting the callback is: printf() syntax via

```
crgMsgSetCallback( int ( *func )( int level, char* message ) );
```

An example for custom message handling is given in the demo/EvalOptions/src/main.c. Look for the method myMsgHandler(...).

3.2.7 Modifiers

As mentioned above, the OpenCRG® data set stored in memory may be globally modified by means of so-called “modifiers”.

The C-API provides a series of methods for defining, querying, and printing modifiers of a data set. These are described in further detail in the header file `crgBaseLib.h` and are following the naming pattern

```
crgDataSetModifierxxx( dataSetId, … );
```

The following tables and figures give an overview of all applicable modifiers with their respective definitions within OpenCRG® data files and with their addressing from the API. All examples have been computed using the test tool located in `Tools/EvalOptions` via the script `testModifiers.sh` in `Tools/bin`.

3.2.7.1 Default Modifiers

The C-API (similar to the Matlab API) defines a set of default modifiers which are applied to the data set if no other modifiers have been defined in the OpenCRG® data file. However, as pointed out above, application of the modifiers must explicitly be triggered by calling the routine

```
crgDataSetModifiersApply( dataset )
```

The default modifiers may be altered and extended in the following ways:

- Block `"$ROAD_CRG_MODS"` in the OpenCRG® data file
  Upon first occurrence of this block **ALL** default modifiers will be deleted (i.e. even the ones you might have wished to keep). These are replaced with the modifiers defined in the corresponding data block. The removal of all modifiers by means of an OpenCRG® data file is achieved by providing an empty block `"$ROAD_CRG_MODS"`

- Calls to C-API
  By explicitly setting / removing modifiers via the C-API routines

  ```
  crgDataSetModifierSetInt( dataset, … )
  crgDataSetModifierSetDouble( dataset, … )
  crgDataSetModifierRemoveAll( dataset )
  crgDataSetModifierSetDefault( dataset )
  ```

  the user may further customize the list of applicable modifiers

- Combination of both methods

The default modifiers are (details see below):

C-API			data file	
symbolic constant	**data type**	**value**	**name**	**value**
dCrgModRefPointUFrac	double	0.0	REFPOINT_U_FRACTION	0.0
dCrgModRefPointUOffset	double	0.0	REFPOINT_U_OFFSET	0.0
dCrgModRefPointVFrac	double	0.0	REFPOINT_V_FRACTION	0.0
dCrgModRefPointVOffset	double	0.0	REFPOINT_V_OFFSET	0.0
dCrgModRefPointX	double	0.0	REFPOINT_X	0.0
dCrgModRefPointY	double	0.0	REFPOINT_Y	0.0
dCrgModRefPointZ	double	0.0	REFPOINT_Z	0.0
dCrgModRefPointPhi	double	0.0	REFPOINT_PHI	0.0
dCrgModGridNaNMode	int	dCrgGridNaNKeepLast	GRID_NAN_MODE	2

3.2.7.2 Transforming Data by Reference Point

The data set may be “re-located” by defining a reference point in the u/v grid and an inertial target position and orientation.

The reference point may be defined using explicit u/v values:

C-API			data file, section $CRG_MODS	
symbolic constant	data type	values	name	values
dCrgModRefPointU	double		REFPOINT_U	
dCrgModRefPointV	double		REFPOINT_V	

It may also be defined using relative u/v values and explicit offsets (in [m]) from these relative positions (or by mixing the former position mode with the current one):

C-API			data file	
symbolic constant	data type	values	name	values
dCrgModRefPointUFrac	double	0.0..1.0	REFPOINT_U_FRACTION	0.0..1.0
dCrgModRefPointUOffset	double		REFPOINT_U_OFFSET	
dCrgModRefPointVFrac	double	0.0..1.0	REFPOINT_V_FRACTION	0.0..1.0
dCrgModRefPointVOffset	double		REFPOINT_V_OFFSET	

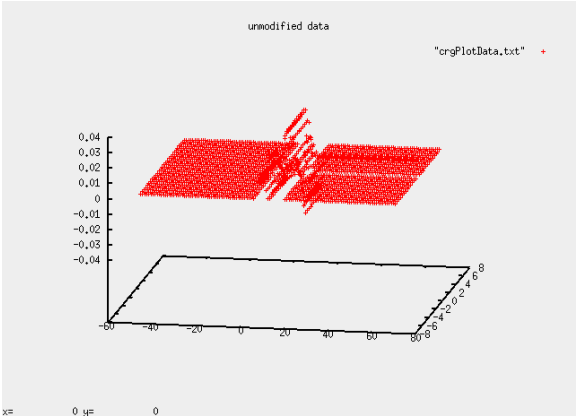
The target position of the data set must be defined in inertial co-ordinates and with absolute heading angle.

C-API			data file	
symbolic constant	data type	values	name	values
dCrgModRefPointX	double		REFPOINT_X	
dCrgModRefPointY	double		REFPOINT_Y	
dCrgModRefPointZ	double		REFPOINT_Z	
dCrgModRefPointPhi	double		REFPOINT_PHI	

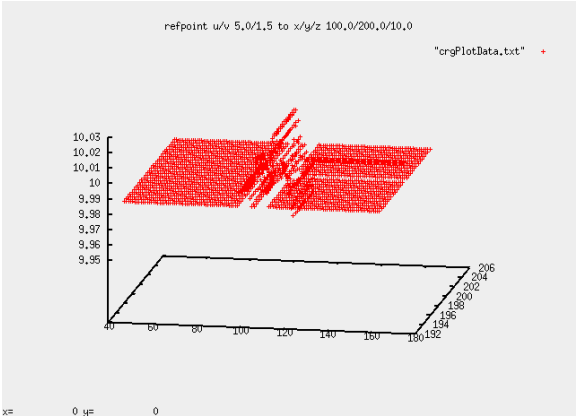
Example 1:

modifier: refpoint u=5.0, v=1.5 to position x=100.0, y=200.0, z=10.0
file: handmade_straight.crg

original data set



modified data set



3.2.7.3 Transforming Data by Offset Position

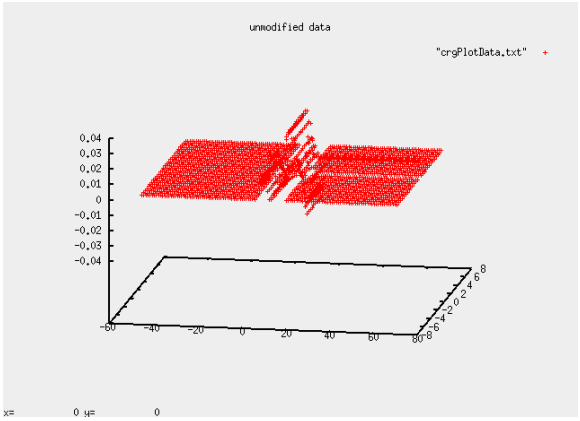
Instead of defining a reference point and its target location, the data set may be shifted and rotated using translation and angle offsets which are applied to the origin of the data set's reference line.

C-API			data file	
symbolic constant	data type	values	name	values
dCrgModRefLineOffsetX	double		REFLINE_OFFSET_X	
dCrgModRefLineOffsetY	double		REFLINE_OFFSET_Y	
dCrgModRefLineOffsetZ	double		REFLINE_OFFSET_Z	
dCrgModRefLineOffsetPhi	double		REFLINE_OFFSET_PHI	

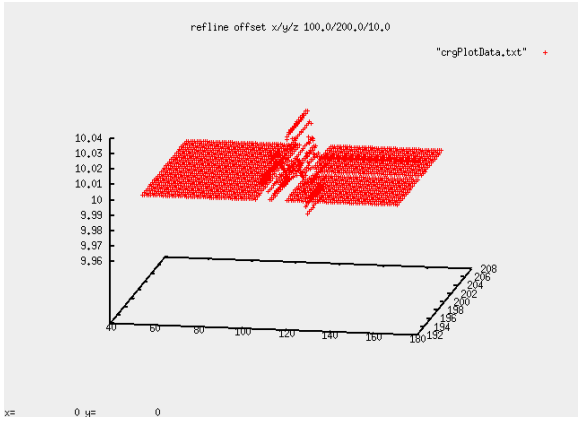
Example 1:

modifier: translation of reference line by x=100.0, y=200.0, z=10.0
file: handmade_straight.crg

original data set



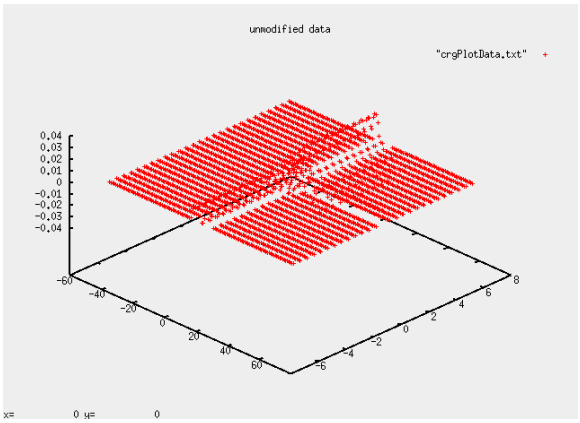
modified data set



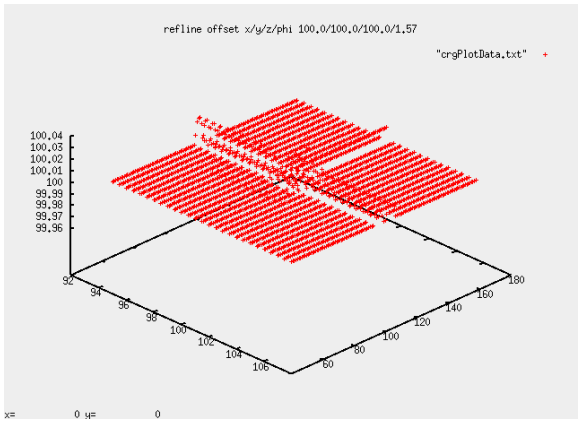
Example 2:

modifier: translation of reference line by x=100.0, y=100.0, z=100.0
rotation by 1.57 (rad)
file: handmade_straight.crg

original data set



modified data set



3.2.7.4 Scaling of Elevation Data

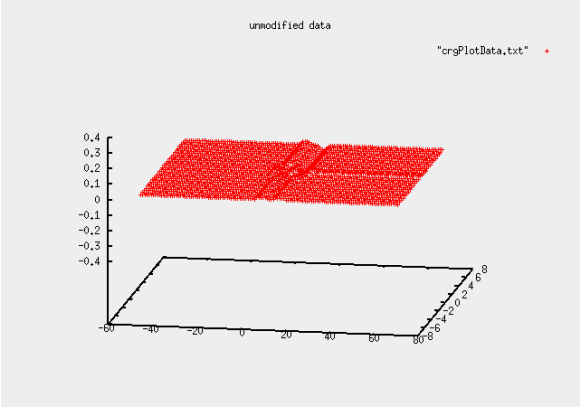
Various properties influencing the elevation of the data set may be scaled. These are listed in the following table. A scale value of 0 will lead to ignoring the respective property.

C-API			data file	
symbolic constant	data type	values	name	values
dCrgModScaleZ	double		SCALE_Z	
dCrgModScaleSlope	double		SCALE_SLOPE	
dCrgModScaleBank	double		SCALE_BANK	

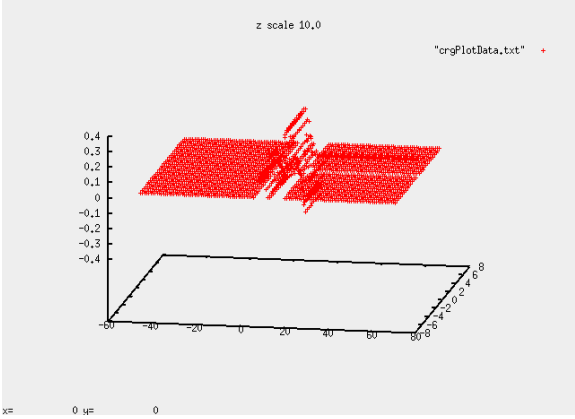
Example 1:

modifier: z scale 10.0
file: handmade_straight.crg

original data set



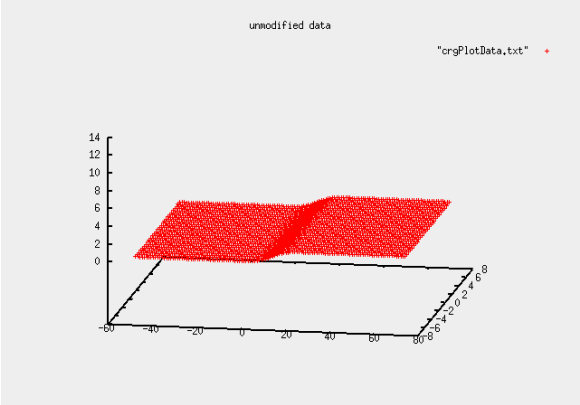
modified data set



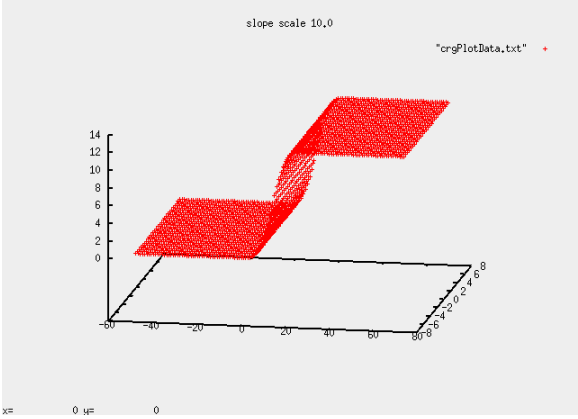
Example 2:

modifier: slope scale 10.0
file: handmade_sloped.crg

original data set



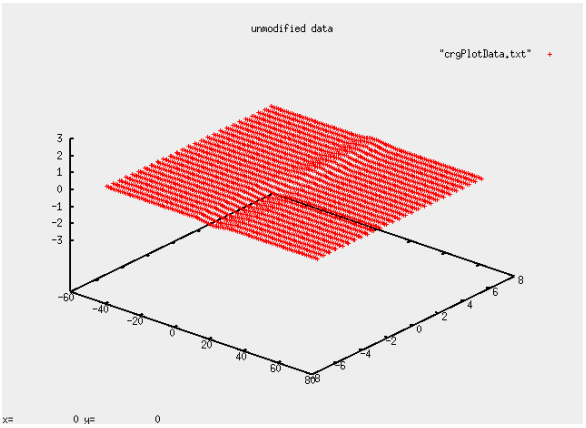
modified data set



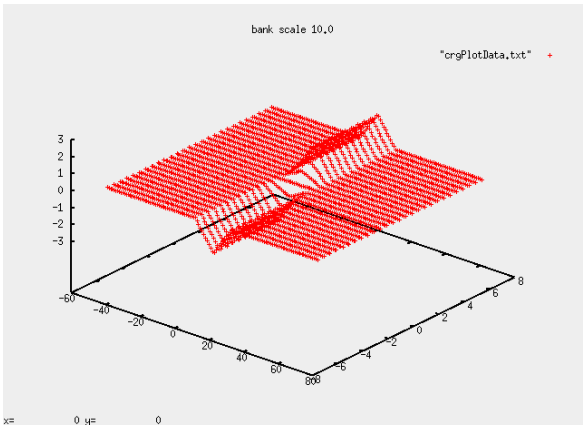
Example 3:

modifier: bank scale 10.0
file: `handmade_banked.crg`

original data set



modified data set



3.2.7.5 Scaling of Data Set Extents

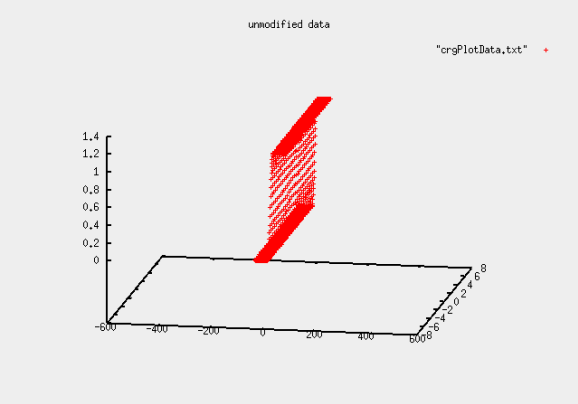
The length and width of the data set may be scaled. Only values greater than zero are valid

C-API			data file	
symbolic constant	data type	values	name	values
dCrgModScaleLength	double	]0.0..]	SCALE_LENGTH	]0.0..]
dCrgModScaleWidth	double	]0.0..]	SCALE_WIDTH	]0.0..]

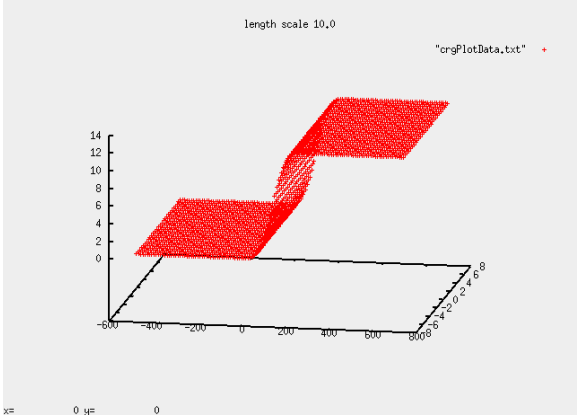
Example 1:

modifier: length scale 10.0
file: handmade_sloped.crg

original data set



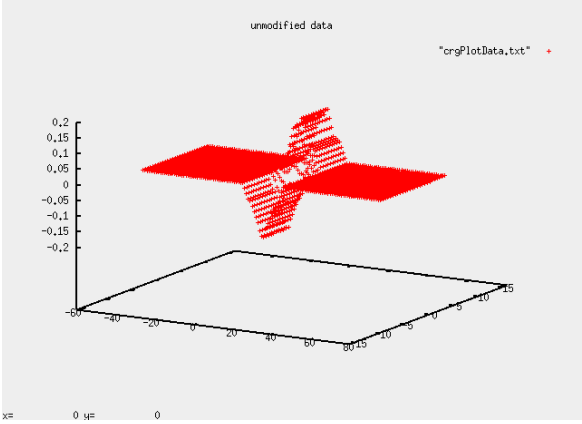
modified data set



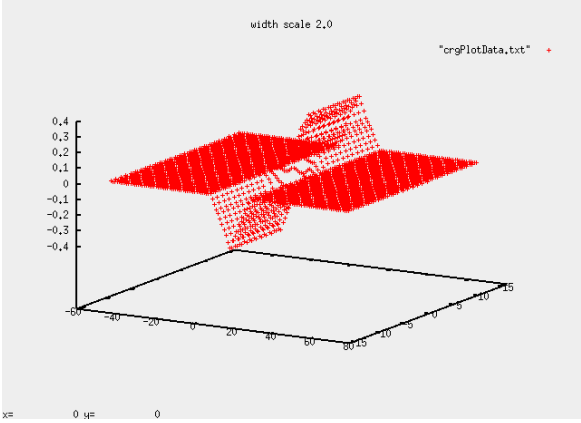
Example 2:

modifier: width scale 2.0
file: handmade_banked.crg

original data set



modified data set



3.2.7.6 Scaling of Curvature

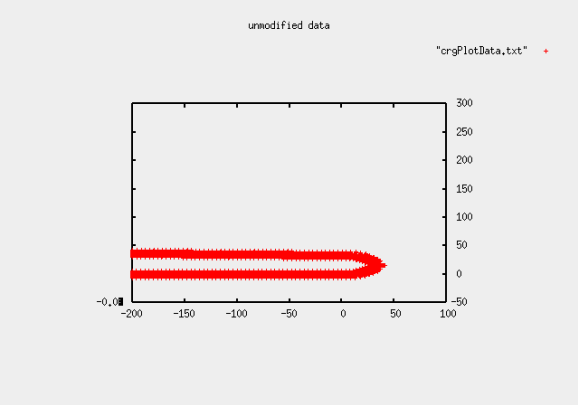
The reference line curvature may be scaled resulting in a modified end point. Scaling by a value of zero will result in a straight reference line.

C-API			data file	
symbolic constant	data type	values	name	values
dCrgModScaleCurvature	double		SCALE_CURVATURE	

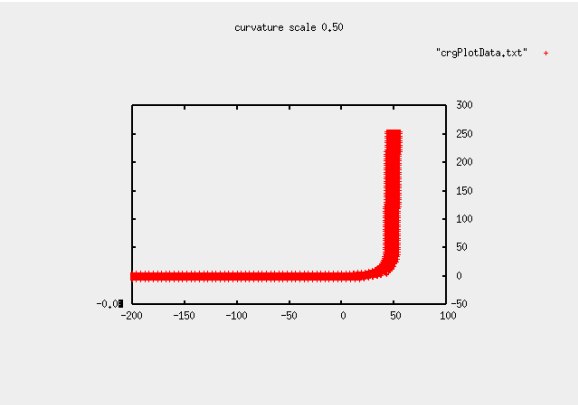
Example 1:

modifier: curvature scale 0.50
file: handmade_arc.crg

original data set



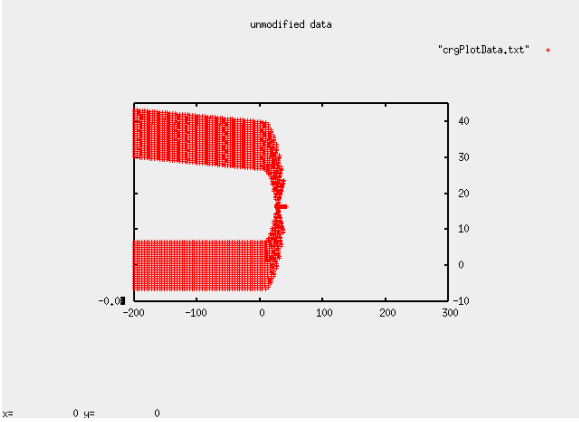
modified data set



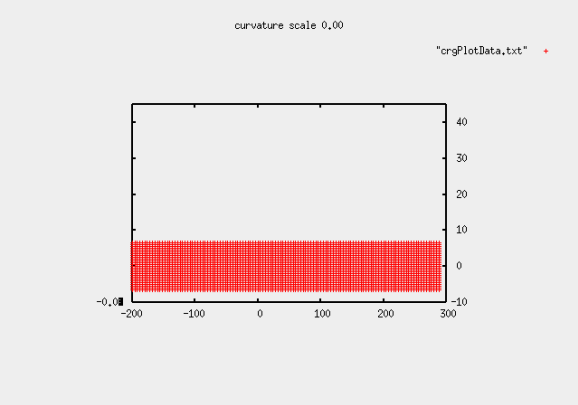
Example 2:

modifier: curvature scale 0.00
file: handmade_arc.crg

original data set



modified data set



3.2.7.7 NaN handling in v Direction

When preparing the data set for queries, the v border values are checked for NaNs. The mode of treating these values can be set by the user.

C-API			data file	
symbolic constant	**data type**	**values**	**name**	**values**
dCrgModGridNaNMode	int	dCrgGridNaNKeep	GRID_NAN_MODE	0
		dCrgGridNaNSetZero		1
		dCrgGridNaNKeepLast		2

When using mode `dCrgGridNanKeep/0`, the option `dCrgCpOptionBorderModeV /BORDER_MODE_V` must also be set to `dCrgBorderModeNone/0`.

Unless NaN is being kept, an explicit offset may be applied to former NaN values in the grid.

C-API			data file	
symbolic constant	**data type**	**values**	**name**	**values**
dCrgModGridNaNOffset	double		GRID_NAN_OFFSET	

3.2.8 Options

As mentioned above, the evaluation methods per contact point may be influenced by so-called "options". This means that options are applied per evaluation, not on a global level.

The C-API provides a series of methods for defining, querying, and printing options of a contact point. These are described in further detail in the header file `crgBaseLib.h` and are following the naming pattern

```
crgContactPointOptionxxx( contactPointId, … );
```

The following tables and figures give an overview of all applicable options with their respective definitions within OpenCRG® data files and with their addressing from the API. All examples have been computed using the test tool located in `Tools/EvalOptions` via the script `testOptions.sh` in `Tools/bin`.

3.2.8.1 Border Modes in u and v Directions

The border modes influence (per direction) the z value returned when the original definition area (i.e. within the min. and max. u- and v-co-ordinates) is exceeded by a query.

For each border, the following behaviors may be defined:

- return NaN/error, i.e. refuse the query
- set the value to zero
- keep the last value
- repeat the core area data
- reflect the core area data

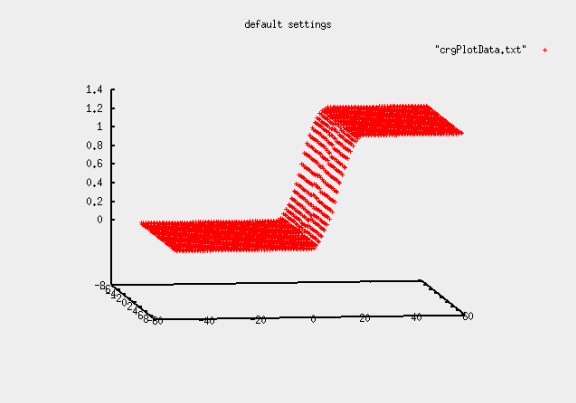
In order to mark the border itself, a z-offset may be applied when the border is crossed in either direction. The interface for defining the respective modes is as follows (default values are highlighted):

C-API			data file	
symbolic constant	**data type**	**values**	**name**	**values**
`dCrgCpOptionBorderModeU`	int	`dCrgBorderModeNone`	`BORDER_MODE_U`	0
		`dCrgBorderModeExZero`		1
		`dCrgBorderModeExKeep`		**2**
		`dCrgBorderModeRepeat`		3
		`dCrgBorderModeReflect`		4
`dCrgCpOptionBorderModeV`	int	`dCrgBorderModeNone`	`BORDER_MODE_V`	0
		`dCrgBorderModeExZero`		1
		`dCrgBorderModeExKeep`		**2**
		`dCrgBorderModeRepeat`		3
		`dCrgBorderModeReflect`		4
`dCrgCpOptionBorderOffsetU`	double		`BORDER_OFFSET_U`	
`dCrgCpOptionBorderOffsetV`	double		`BORDER_OFFSET_U`	

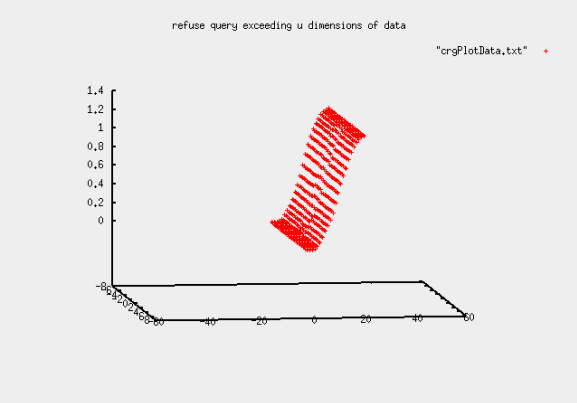
Example 1:

options: border mode U = `dCrgBorderModeNone`
file: `handmade_sloped.crg`

default settings



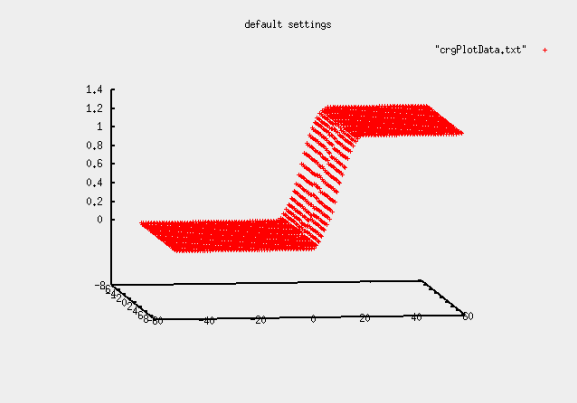
option settings



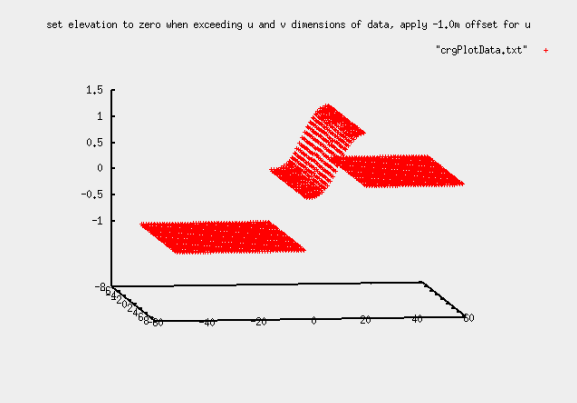
Example 2:

options: border mode U = `dCrgBorderModeExZero`
border mode V = `dCrgBorderModeExZero`
border offset U = `-1[m]`
file: `handmade_sloped.crg`

default settings



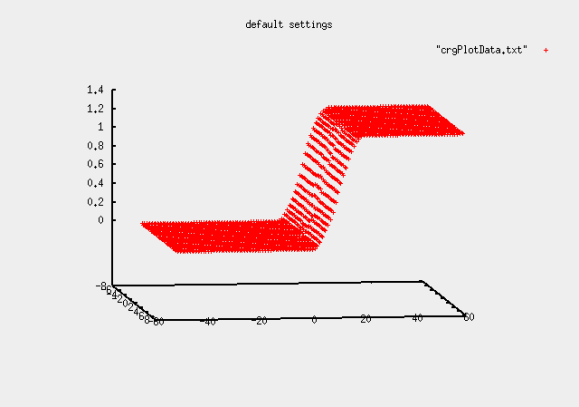
option settings



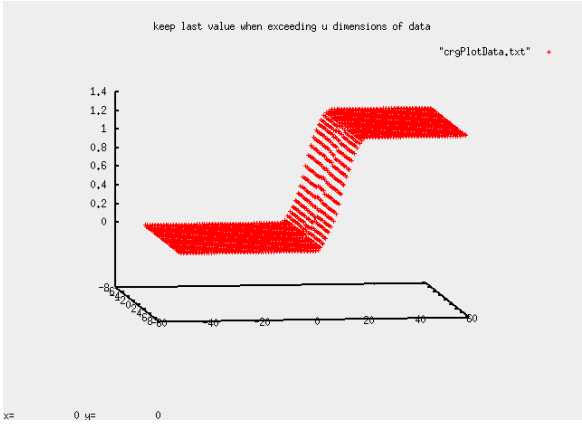
Example_3:

options: border mode U = `dCrgBorderModeExKeep`
file: `handmade_sloped.crg`

default settings



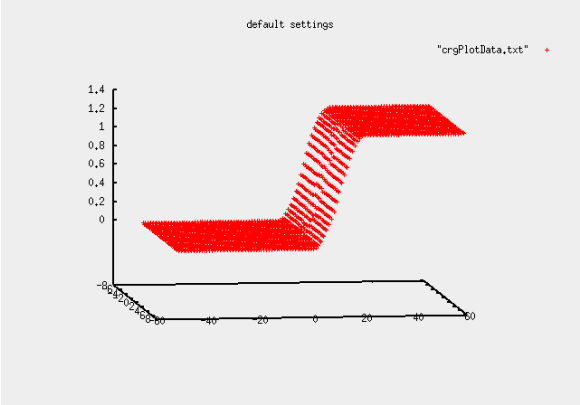
option settings



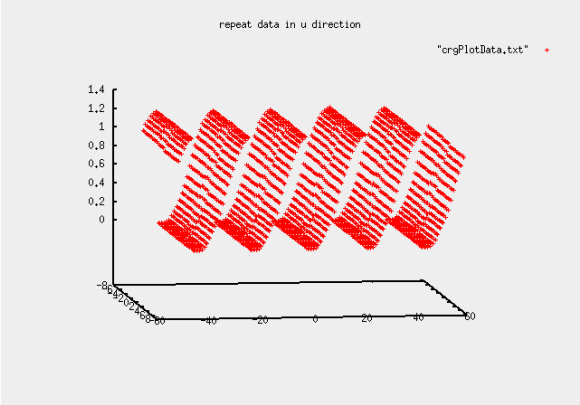
Example 4:

options: border mode U = `dCrgBorderModeExRepeat`
file: `handmade_sloped.crg`

default settings



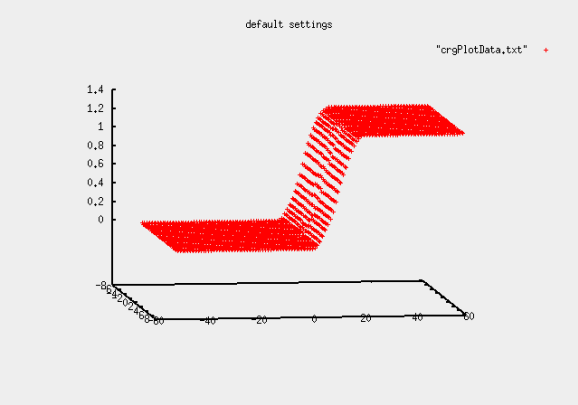
option settings



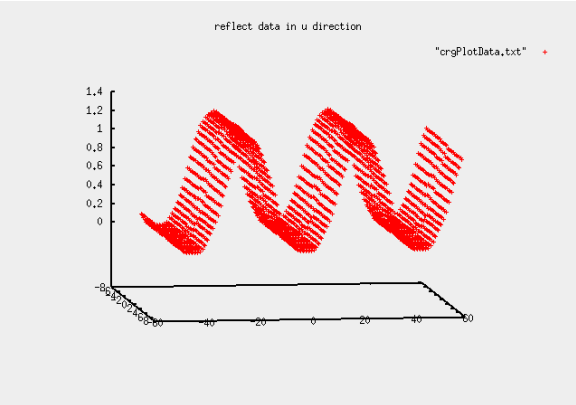
Example 5:

options: border mode U = `dCrgBorderModeExReflect`
file: `handmade_sloped.crg`

default settings



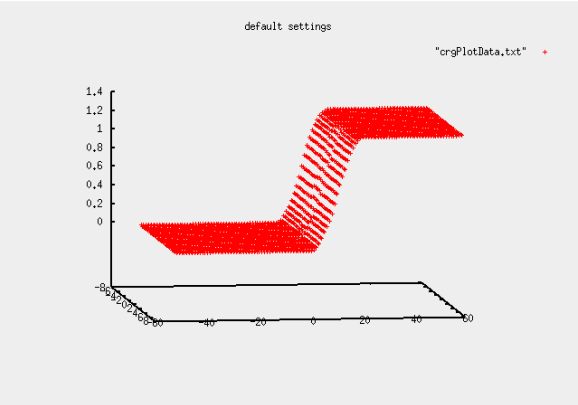
option settings



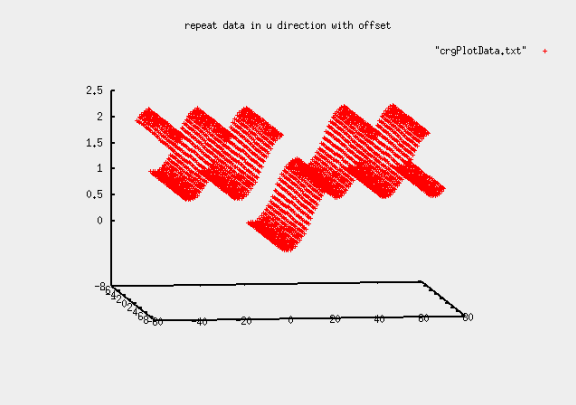
Example 6:

options: border mode U = `dCrgBorderModeExRepeat`
border offset U = `1[m]`
file: `handmade_sloped.crg`

default settings



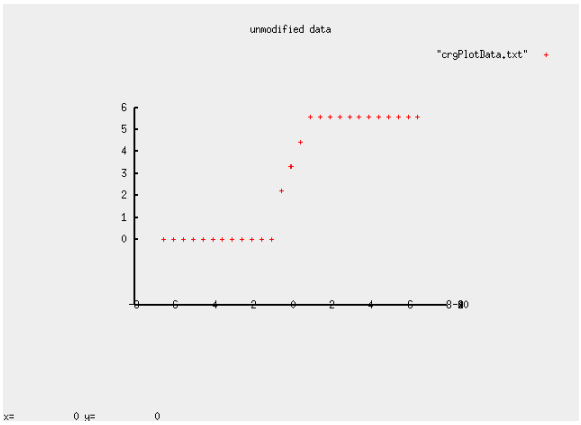
option settings



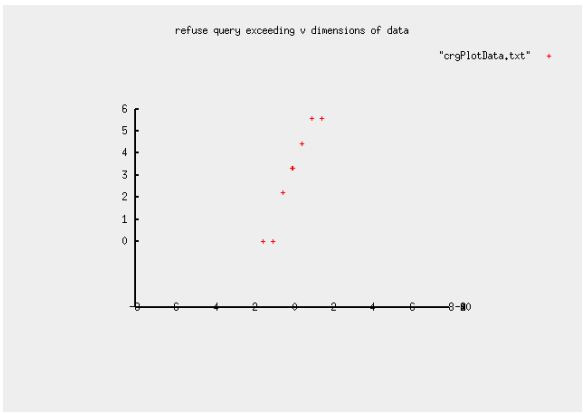
Example 7:

options: border mode V = `dCrgBorderModeNone`
file: `handmade_vtest.crg`

default settings



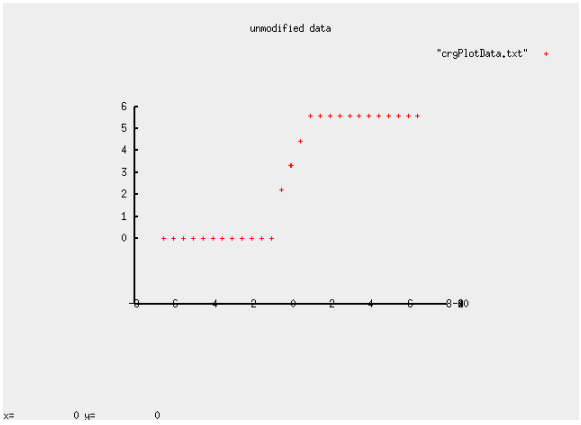
option settings



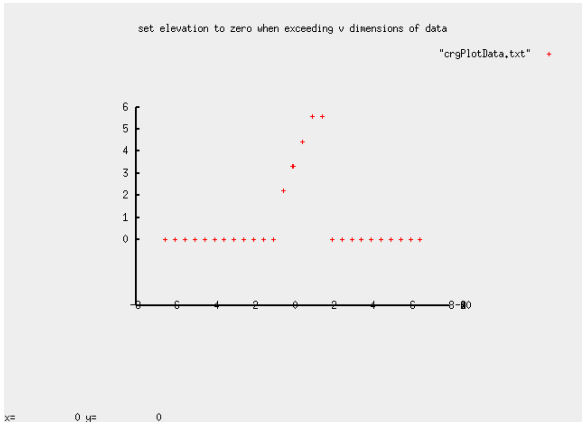
Example 8:

options: border mode V = `dCrgBorderModeExZero`
file: `handmade_vtest.crg`

default settings



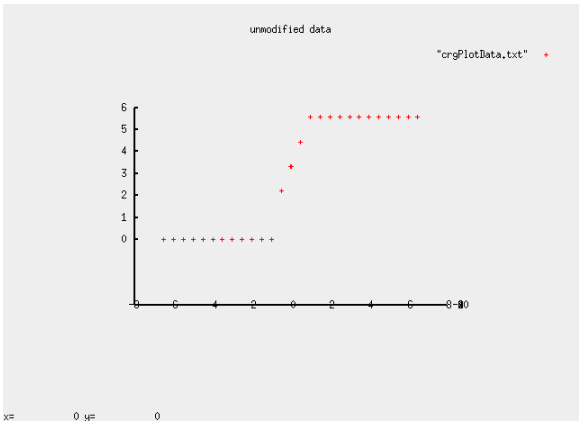
option settings



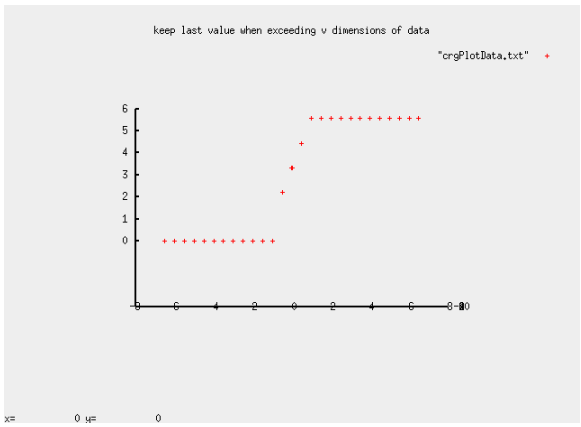
Example 9:

options: border mode V = `dCrgBorderModeExKeep`
file: `handmade_vtest.crg`

default settings



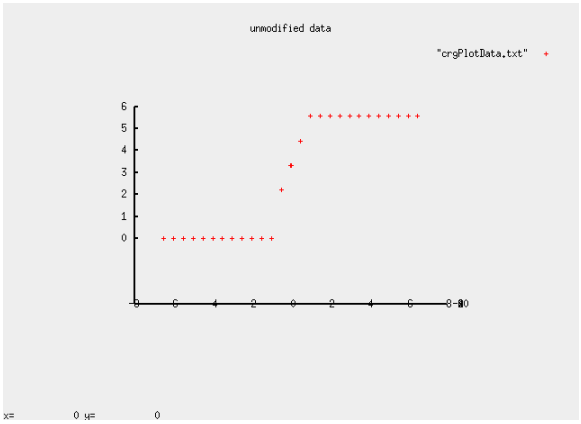
option settings



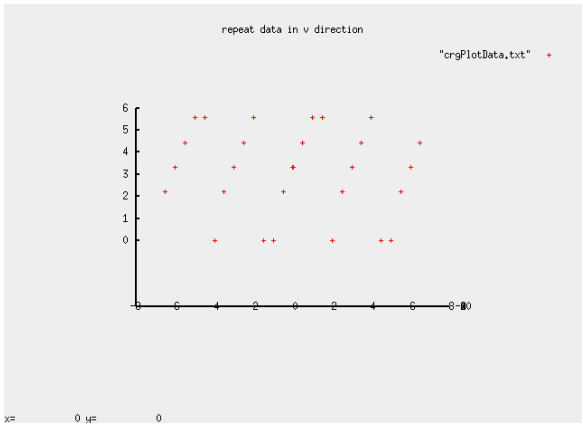
Example 10:

options: border mode V = `dCrgBorderModeExRepeat`
file: `handmade_vtest.crg`

default settings



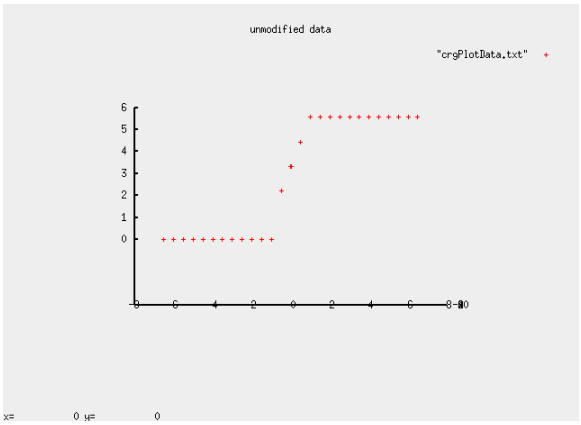
option settings



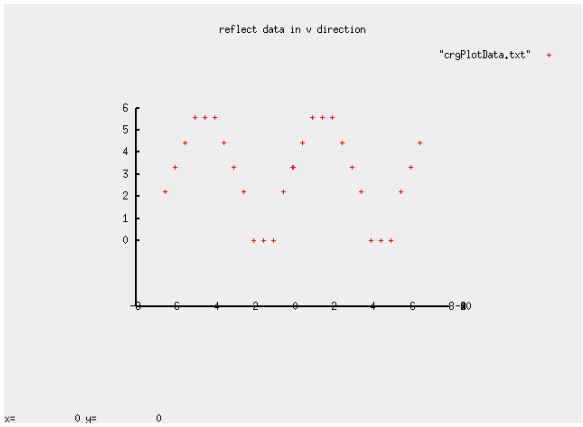
Example 11:

options: border mode V = `dCrgBorderModeExReflect`
file: `handmade_vtest.crg`

default settings



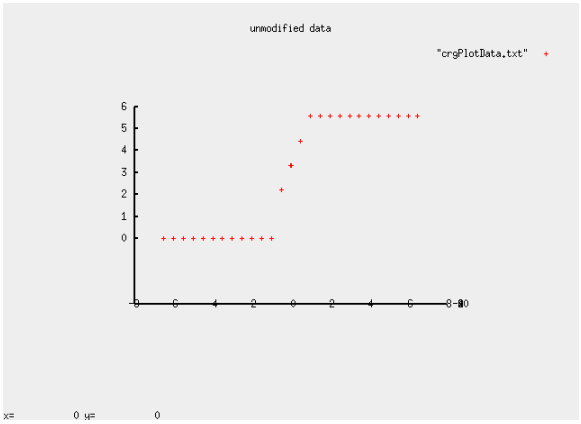
option settings



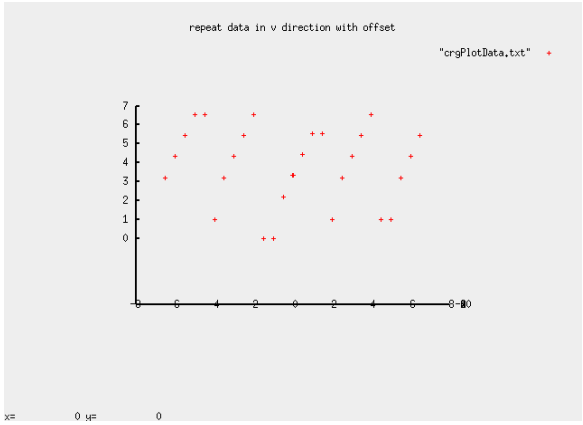
Example 12:

options: border mode V = `dCrgBorderModeExRepeat`
border offset V = `1[m]`
file: `handmade_vtest.crg`

default settings



option settings



3.2.8.2 Smoothing Zones

In order to provide a smooth on-set and/or off-set of z data, the begin and end of the data (including slope and bank) may be scaled linearly from zero to full size within a user-defined range at either end of the core area. Again, core area is the original data extent defined in the data file which is not subject to the evaluation via one of the border modes described in the previous chapter.

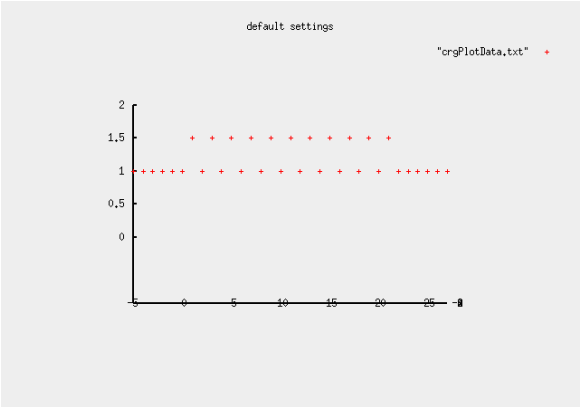
The parameters of the smoothing zone may be defined via the following interfaces:

C-API			data file	
symbolic constant	**data type**	**values**	**name**	**values**
`dCrgCpOptionSmoothUBegin`	double	0.0..u_{max}	`BORDER_SMOOTH_UBEG`	0.0..u_{max}
`dCrgCpOptionSmoothUEnd`	double	0.0..u_{max}	`BORDER_SMOOTH_UEND`	0.0..u_{max}

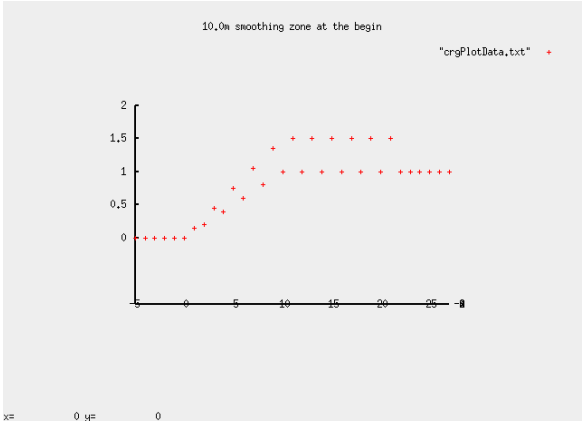
Example 1:

options: smoothing zone at begin = `5[m]`
file: `handmade_platform.crg`

default settings



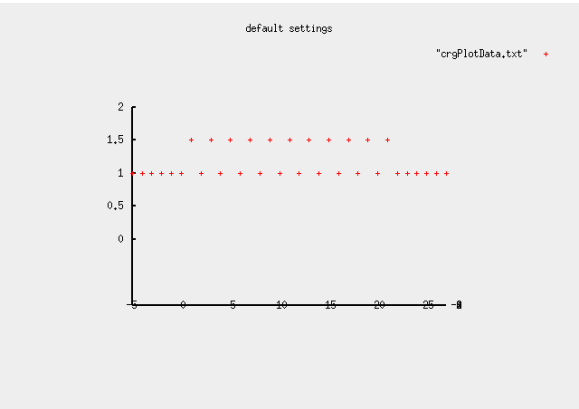
option settings



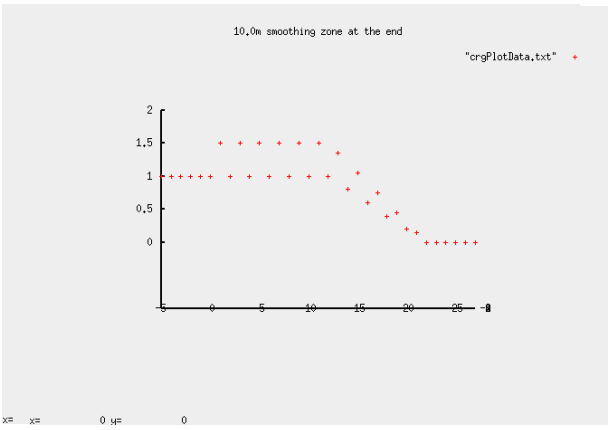
Example 2:

options: smoothing zone at end = `5[m]`
file: `handmade_platform.crg`

default settings



option settings



3.2.8.3 Continuation of Reference Line

For points whose u position exceeds the data set's valid range (i.e. the original reference line), the corresponding x/y position is usually computed by extrapolating the reference line at the respective end. However, reference lines representing a closed track, the extrapolation of the u position may be achieved by trying to close the reference line and, thus, position the point (again) within the core area of the data set.

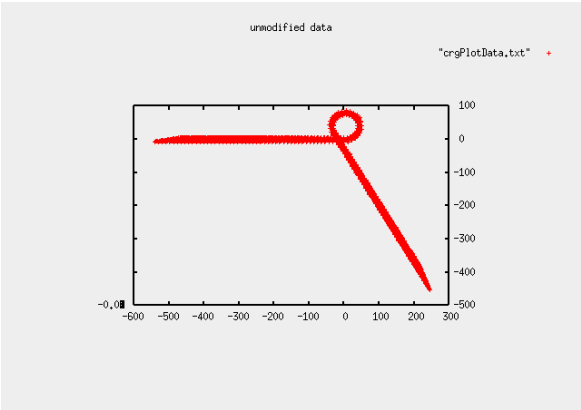
The parameters controlling this behavior are:

C-API			data file	
symbolic constant	**data type**	**values**	**name**	**values**
dCrgCpOptionRefLineContinue	int	**dCrgRefLineExtrapolate** dCrgRefLineCloseTrack	REFLINE_CONTINUATION	**0** 1

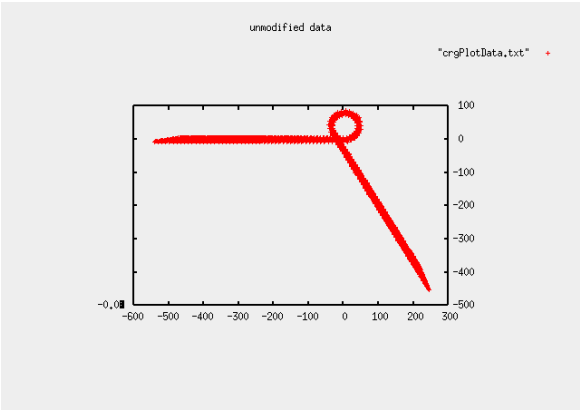
Example 1:

options: reference line continuation = dCrgRefLineExtrapolate
file: handmade_circle.crg

default settings



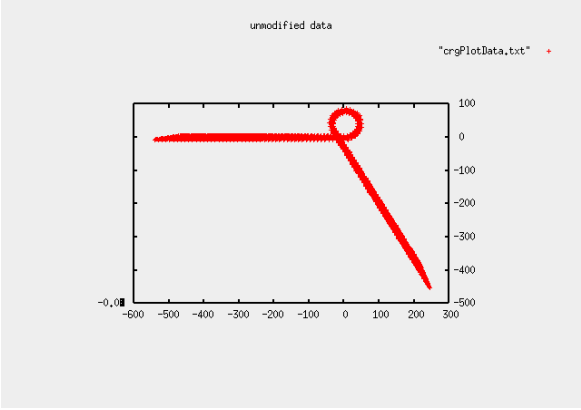
option settings



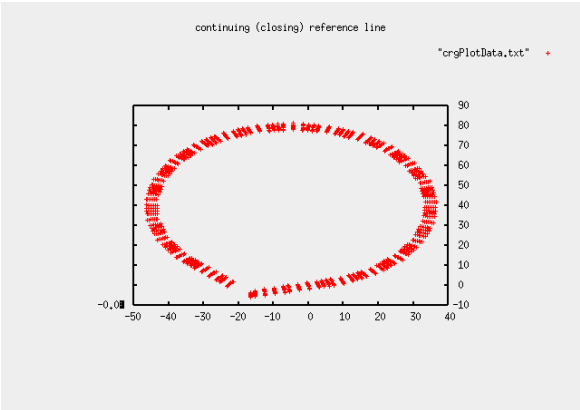
Example 2:

options: reference line continuation = dCrgRefLineCloseTrack
file: handmade_circle.crg

default settings



option settings



3.2.8.4 Curvature Evaluation

When computing the curvature at a given u/v-position, the resulting value is, per default, the actual curvature at the position. An option is available which, when enabled, returns the curvature at the corresponding reference line position instead of the curvature at the actual position.

The parameters controlling this behavior are:

C-API			data file	
symbolic constant	data type	values	name	values
dCrgCpOptionCurvMode	int	**dCrgCurvLateral** dCrgCurvRefLine	n/a	

3.2.8.5 History Manipulation

The performance of data evaluation is increased by using internally a history of recently queried points. It is assumed that new queries typically occur in the vicinity of previous queries.

If the user has more knowledge about the u-positions of the reference line where the next queries will occur, he may pre-load the history with this information and, thereby, increase the performance of the next queries significantly.

The parameters controlling this behavior are:

C-API			data file	
symbolic constant	data type	values	name	values
dCrgCpOptionRefLineSearchU	double	0.0..u_{max}	REFLINE_SEARCH_INIT_U	0.0..u_{max}
dCrgCpOptionRefLineSearchUFrac	double	0.0..1.0	REFLINE_SEARCH_INIT_U_FRACTION	0.0..1.0

3.2.8.6 History Search Criteria

While evaluating history information, the algorithms check for two distances of an actual position to a position stored in the history. Depending on whether a point is within a close or a far range of a point in history, the algorithm may simplify the search and, thus, return much faster with the correct result.

The parameters controlling this behavior are:

C-API			data file	
symbolic constant	data type	values	name	values
dCrgCpOptionRefLineClose	double	default: 0.3	REFLINE_SEARCH_CLOSE	default: 0.3
dCrgCpOptionRefLineFar	double	default: 2.2	REFLINE_SEARCH_Far	default: 2.2

3.3 The Tools

3.3.1 Overview

In the `Tools/` directory, a set of samples for the usage of the C-API is provided as source code. The tools and their purposes are shown in the following list:

```
+--Tools .............collection of tools for an easy entry into the OpenCRG programming
   +--EvalOptions.....a set of routines demonstrating the usage of various options
   +--EvalXYnUV.......a set of routines for the evaluation of OpenCRG reference lines
   +--EvalZ...........an advanced example for the evaluation of OpenCRG data
   +--Reader..........a sample application for a CRG file reader
   +--Simple..........a really simple application covering all basics of the API,
```

Each tool's sub-directory is of identical structure:

```
   +--ToolName ....sub-directory of a tool
   |  +--src .......all source files required for the tool
   |  +--inc .......all include files required for the tool (except for BaseLib includes)
   |  +--obj .......the resulting object files
   |  +--Makefile ..the makefile for compilation of the tool
```

Once a tool is compiled, its executable file will be located in `Tools/bin`.

3.3.2 Simple

This is a very simple example for the usage of the C-API.

executable: `crgSimple`

command line: `crgSimple [options] <CRG data file>`

options: `-h show online help`

3.3.3 Reader

Read a file and print some debug information.

executable: `crgReader`

command line: `crgReader [options] <CRG data file>`

options: `-h show online help`

3.3.4 Co-ordinate conversion x/y and u/v

Perform a series of conversions between x/y and u/v co-ordinate and check for consistency. The routines adapt automatically to the contents (i.e. extents) of the data file.

executable: `crgEvalxyuv`

command line: `crgEvalxyuv [options] <CRG data file>`

options: `-h show online help`

3.3.5 *Evaluation of z Data*

Perform a series of z data queries. The routines adapt automatically to the contents (i.e. extents) of the data file.

executable: `crgEvalz`

command line: `crgEvalz [options] <CRG data file>`

options: `-h show online help`

3.3.6 *Application of Options and Modifiers*

Perform a single test or series of tests involving data set modifiers or evaluation options. This test program is used for acceptance tests before a new version of the C-API is released.

executable: `crgEvalOpts`

command line: `crgEvalOpts [options] <CRG data file>`

options:

```
-h          show online help
-t <n>      run the single test with the given number
            (please check the source code for the numbers
            and the corresponding test routines)
-p          write the results (x/y/z data) into the file
            "crgPlotFile.txt" which may be used as input
            to gnuplot or other data visualizers
```

In order to facilitate the execution of various tests involving modifiers and options, two scripts are provided in the `Tools/bin` directory.

```
testOptions.sh
testModifiers.sh
```

These scripts run all available tests for modifiers and options, asking the user for confirmation of each test. The results are stored in a file named `testReport.txt`. The test tool used by the scripts is `crgEvalOpts`.

4 The OpenCRG® Matlab Tools

4.1 Getting Started

4.1.1 Downloading the Software Package

The latest version of the OpenCRG® Matlab Tools may be retrieved via the website

```
www.opencrg.org
```

Go to the `Downloads` area and look for the section `Packages and Tools`. There, you will find a package containing the C-API, MATLAB routines, documentation and sample files. Download this package and unpack it into a dedicated directory on your machine.

The Matlab Tools are located in the sub-directory

```
OpenCRG/matlab
```

```
The latest developer version may be rerieved via the OpenCRG® project
managing and issue-tracking-system via
```

http://viresftp.dyndns.org (registration required)

4.1.2 Release Notes

The release notes are part of the software package and will not be transferred to this document. Please look for the "readme.txt" files in the root directory of the OpenCRG package and in the Matlab Tools sub-directory.

4.1.3 Contents of the Package

The OpenCRG® Matlab Tools are delivered as a zipped package which, after unpacking, provides the following file structure:

```
|----crg_init.m.............initializes the CRG Matlab environment
|----crg_intro.m............introduction to the basic CRG concept and data structure
|----demo...................demo sources explaining the usage of the basic library
|----lib....................base library including functionality to read, evaluate,
|                           visualize, analyse, generate, modificate and write CRG data files.
|----test...................test sources to validate the basic library
```

4.1.4 Working with OpenCRG® Matlab Tools

4.1.4.1 Initialization

Start OpenCRG environment by running crg_init.m form the Matlab command line

```
run <path-to-OpenCRG>/matlab/crg_init
```

```
To run the tests and demos, it is recommended to change to the work
directory for temporary data
```

```
cd <path-to-OpenCRG>/temp
```

4.1.4.2 Getting Help

The OpenCRG® Matlab files contain the Matlab specific help header as well a copyright notices. Use help to `view declaration and explanation of the methods.`

```
help <OpenCRG-method>
```

To get an overview on CRG data organization, run the 'crg_intro' command.

```
crg_intro
```

4.1.5 Formatting M-File Comments

OpenCRG® uses the same formatting comments like Matlab does for publishing. Please refer to the Matlab product help of 'Formatting M-File Comments for Publishing' for detailed information.

4.1.6 Demos

Several demo scripts are provided in '<path-to-OpenCRG>/matlab/demo'. Running the crg_demo command will generate some CRG data structures, write them to disk, read them again, and optionally visualizes some contents. It may be useful to look into crg_demo to get some first examples how to use the OpenCRG Matlab functions and tools suite. For quite all basic test it is necessary to generate the sample demos by:

```
crg_demos
```

4.1.7 Basic Tests

Several test scripts are provided under '<path-to-OpenCRG>/matlab/tests'. A series of very brief tests can be run. These are also used for acceptance tests of the OpenCRG® Matlab Tools as well as benchmark tests for the C-API. CRG demo files are used to test the functionality. Make sure all sample demos are generated by running crg_demos.

```
crg_demos                % if not done before once
crg_test_<functionname>
```

4.1.8 Publishing script files

The OpenCRG® Tool suites script files are written and comment in Matlab style for publishing. This publishing functionality should be used to produce adequate output. Since R2008a Matlab's “publish configurations” was introduced by Mathworks. It allows to publish to latex, doc and HTML style.

4.1.8.1 Extensible stylesheet language (xsl)

A sample xsl-file is located in '<path-to-OpenCRG>/docsrc/xsl/'. Publishing to latex using the xsl file produces the same formated output like the OpenCRG® tool suite pdf’s.

4.1.8.2 Publish to pdf (Linux)

A simple makefile is provided to generate pdf's from Matlab files in

```
'<path-to-OpenCRG>/docsrc/make/'.
```

Publish your scripts to latex output using the OpenCRG® extensible stylesheet language file

```
'<path-to-OpenCRG>/docsrc/xsl/openCRG_mxdom2latex.xsl'
```

. Make sure you select ‘.jpg’ as image type. Copy the makefile into your destination folder and execute it by :

```
make <function_name>.pdf
```

The Makefile will copy the pdf in a subdirectory '../pdf'. Make sure 'pdflatex' is installed.

Use make clean to delete all latex generated files beside the Makefile.

4.1.8.3 Hint on memory limitations

With 32bit Matlab available memory is sometimes rather limited, especially on WIN32 systems, see Matlab help on memory.

On a typical WIN32 installation with 4GB, only 440MB of continuous memory was available (use feature('memstats') to show on your system). This allows to work on 220MB CRG files, with e.g. 1cm x 1cm resolution, 4m with and 1.4km length. On Linux32, at least double file sizes can be handled. On all 64bit installations, there is virtually no limitations in CRG size.

4.2 The Base Library

4.2.1 Overview

The base library contains a large set of routines similar to the C-API for

- reading
- modifying
- evaluating

and is extended by methods to

- visualize
- generate
- analyse

OpenCRG® data.

4.2.2 Demo Files

The Tool suite comes with basic script files

```
matlab/demo/crg_demo.m
matlab/demo/crg_demo_gen*.m
```

Run 'crg_demo.m' to generate a set of simple crg-files. These are used to execute the test files. The second set of demo files show how to generate a crg-file with a smooth refline and surface data.

4.2.3 Test Files

A set of test files are provided to verify the functionality of the tool suite and are located in '/matlab/test':

```
crg_test_append.m            concatenate two crg files
crg_test_continuesTrack.m    road continuation
crg_test_eval_uv2iuiv.m      cut crg file
crg_test_ext_sb.m            extract slope and/or banking
crg_test_filter.m            filter crg file
crg_test_gen_csb2crg.m       additional synthetically generated crg files
crg_test_gen_road.m          generate and write synthetically crg file
crg_test_isequal.m           compare two crg files if equal
crg_test_limiter.m           limits crg z-values
crg_test_map_uv2uvAxy2xy.m   uv and inertial mapping
crg_test_options.m           add options
crg_test_peakfinder.m        find peaks
crg_test_rerender.m          re-render crg-file
```

4.2.4 Library Files

The library is composed of the following Matlab files which are located in 'matlab/lib'.

4.2.4.1 Analysis tools:

```
crg_isequal.m                comparison if two crg files are equal
crg_peakfinder.m             find peaks
```

4.2.4.2 Evaluation tools:

```
crg_eval_u2crv.m             evaluate curvature crv in reference line u position
crg_eval_u2phi.m             evaluate heading phi in reference line u position
crg_eval_uv2iuiv.m           evaluate index iu, iv of distance u,v
crg_eval_uv2xy.m             transform point in uv to xy
crg_eval_uv2z.m              evaluate z at grid position uv
crg_eval_xy2uv.m             transform point in xy to uv
crg_eval_xy2z.m              evaluate z at position xy
crg_wgs84_crg2html.m         generate HTML file to show wgs info in Google maps
crg_wgs84_dist.m             evaluate distance and bearing between WGS84 positions
crg_wgs84_invdist.m          calculate WGS84 position by distance and bearing
crg_wgs84_setend.m           set WGS84 end coordinate
crg_wgs84_wgs2url.m          generate url string to show wgs info
crg_wgs84_wgsxy2wgs.m        transform point in xy to WGS84 using 2 references
crg_wgs84_xy2wgs.m           transform point in xy to WGS84
```

4.2.4.3 Generation tools:

```
crg_check_uv_descript.m      creates a cross section (v-) profile.
crg_gen_csb2crg0.m           generate synthetically crg data
crg_gen_ppxy2phi.m           generate refline heading out of polynomial
crg_gen_pxy2ppxy.m           generates (smooth) polynomial through ref-points
crg_perform2surface.m        add cell array to surface data
```

4.2.4.4 Modification tools:

```
crg_append.m                 appends two crg data sets
crg_b2z.m                    add banking to z-values of crgdata
crg_cut_iuiv.m               separate a crg file
crg_ext_banking.m            extract banking
crg_ext_slope.m              extract slope
crg_filter.m                 filter crg data
crg_flip.m                    flips the crg contents
crg_generate_sb.m             finds and generates sloope and banking
crg_limiter.m                limit crg data
crg_map_uv2uv.m              map crg data in uv
crg_map_xy2xy.m              inertial mapping
crg_mods.m                   apply modifiers on data
crg_rerender.m               re-render crg data
crg_s2z.m                    add slope to z-values of crg data
```

4.2.4.5 Io-files tools:

```
crg_check.m                  base check class
crg_check_data.m             check, fix and complement data
crg_check_head.m             verifies crg struct head data
crg_check_mods.m             verifies crg struct modification data
crg_check_opts.m             verifies crg struct options data
crg_check_single.m           single type check of core data
crg_read.m                   read crg file
crg_single.m                 single type conversion of core data
crg_wrap.m                   re-wraps heading angles to +/- pi range
crg_write.m                  road file writer
ipl_demo.m                   IPLOS write/read demonstration
ipl_read.m                   IPLOS file reader
```

```
ipl_write.m                  IPLOS file writer
sdf_add.m                    (structured data file) block add
sdf_cut.m                    (structured data file) block cut
str_num2strn.m               convert number to string of given length
```

4.2.4.6 Visualization tools:

```
copy_ax2fig.m                          copy current axes object to new figure
crg_figure.m                           figure setup
crg_plot_elgrid_cross_sect.m           plot road elevation grid cross sections
crg_plot_elgrid_limits.m               plot road elevation grid limits
crg_plot_elgrid_long_sect.m            plot road elevation grid long sections
crg_plot_elgrid_uvz_map.m              plot road elevation grid UVZ map
crg_plot_elgrid_xyz_map.m              plot road elevation grid XYZ map
crg_plot_refline_curvature.m           road refline curvature plot
crg_plot_refline_elevation.m           road refline elevation plot
crg_plot_refline_heading.m             road refline heading plot
crg_plot_refline_slope_bank.m          road refline slope and banking plot
crg_plot_refline_xy_map_and_curv.m     road refline XY map and curv plot
crg_plot_refline_xy_overview_map.m     road refline XY overview map plot
crg_plot_refline_xyz_map.m             road refline XYZ map 3D plot
crg_plot_refpnt_distance.m             road refpoint distance plot
crg_plot_road_uv2uvz_map.m             plot road UVZ map
crg_plot_road_uv2xyz_map.m             plot road XYZ map
crg_plot_road_uvz_map.m                road UVZ map
crg_plot_road_xyz_map.m                road XYZ map
crg_show.m                             road visualizer
crg_show_elgrid_cuts_and_limits.m      road elevation grid 2D visualizer
crg_show_elgrid_surface.m              road elevation grid 3D surface visualizer
crg_show_isequal.m                     comparison of two crg_files data core visualizer
crg_show_info.m                        read text info visualizer
crg_show_peaks.m                       peak visualizer
crg_show_refline_elevation.m           road refline elevation visualizer
crg_show_refline_map.m         road refline map visualizer
crg_show_refpnts_and_refline.m         road refpnts and refline visualizer
crg_show_road_surface.m                road 3D surface visualizer
crg_show_road_uv2surface.m             road 3D surface visualizer by u,v grid
crg_surf.m                             3D surface plot
```

4.2.5 Working with Data Files

4.2.5.1 Simple Use Case

There are many operations that may be applied to OpenCRG® data sets. However, a simple way of using OpenCRG® data files is shown in just 2 stages:

```
1.    load a crg file
2.    call the evaluation methods
```

```
crg_demo                          % if not done before once
data = crg_read('demo1.crg');       % load crg file
pz = crg_eval_uv2z(data, [100 0]);  % z-elevation on (u,v) = [100 0]
```

4.2.5.2 CRG data structure

This Matlab tools suite uses one structure which shall be briefly explained here. For further explanation run crg_intro.

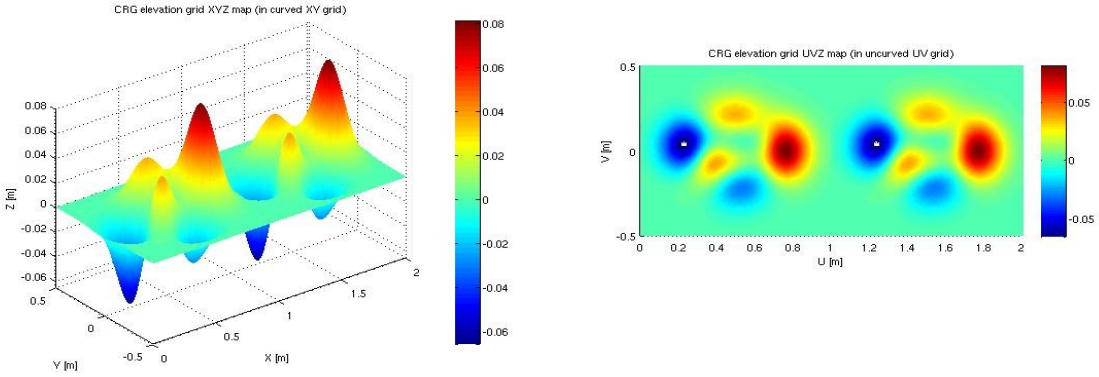
```
|data = ...................variable name
|----head..................struct array of data scalars
|----mods..................data, which defines further CRG modifiers
|----opts..................data, which defines further CRG processing options
|----ct....................cell array of comment text, required for file output
|----struct................(optional) cell array of further structured data, used
|.........................for file output, may contain opts data for later CRG
```

```
|.........................file processing.
|----filenm................name of read crg data file
|----z.....................array(nu, nv) of z (height) values (single)
|----v.....................vector of v values (single)
|----b.....................(optional) vector of banking values (single)
|----u.....................vector of u values (single)
|----p.....................(optional) vector of phi values (single)
|----s.....................(optional) vector of slope values (single)
```

Not all structure fields are required and some are derived by 'crg_check_*' routines. A minimum CRG content consist of struct fields u,v and z. All necessary extended data can than be derived by the 'crg_check' routine.

```
data = struct;                                      % empty struct
data.z = single(repmat(0.01*peaks(101), 2, 1));% z values
data.u = 2.01;                                      % set u-space
data.v = 0.5;                                       % set v-space

data = crg_check(data);                             % get derived data
crg_show(data);                                     % display result
```



4.2.5.3 Modifiers and Options

In simple cases, data sets will be read from file and evaluations may take place immediately. However, the user may want to vary the way a specific data set is evaluated without having to modify the original data (i.e. the data file) itself.

For this purpose, modifiers and options have been introduced.

- **Modifiers** modify a copy of OpenCRG® data which is stored in memory and, therefore, provide the modified data for all subsequent evaluation requests
- **Options** influence the way a specific query is performed without modifying any of the data stored in memory.

In the data struct, modifiers are found in the substructure crg.mods and options are found in crg.opts. The tool suite provides discrete calls for setting modifiers and options. Evaluation methods take account if options are set..

Modifiers have to be executed by calling the crg_mods() routine, which will affect the modifier settings on crg-data.

4.2.6 Modifiers

As mentioned above, the OpenCRG® data struct contains the so called 'modifiers' which globally modify the crg-data. All modifier functionalities are equal to the C-API and explained there. Further details about the Matlab API are located in crg_intro or in the help content of the function..

```
CRG scaling (default: no scaling)
       - scale elevation data
              szgd           scale elevation grid
              sslp           scale slope information
              sbkg           scale banking information

       - scale refline data (resets refline position to origin)
              slth           scale u information
              swth           scale v information
              scrv           scale reference line's curvature
CRG elevation grid NaN handling
              gnan           how to handle NaN values (default: 2)
                     = 0: keep NaN
                     = 1: set zero
                     = 2: keep last value in cross section
              gnao           z offset to be applied at NaN positions (default: 0)
CRG re-positioning: refline by offset (default: "by refpoint")
              rlox           translate by rlox
              rloy           translate by rloy
              rloz           translate by rloz
              rlop           rotate by rlop around (xbeg, ybeg)
CRG re-positioning: refline by refpoint (overwrites "by offset")
       - position (u,v) on reference line:
              rpfu           relative u (default: 0)
              or: rptu       absolute u
              [with optional: rpou (default: 0)]
              rptv           absolute v (default: 0)
              or: rpfv       relative v
              [with optional rpov (default: 0)]
       - position (x,y,z) and orientation (phi) in inertial frame:
              rptx           target position (default: 0)
              rpty           target position (default: 0)
              rptz           target position (default: 0)
              rptp           target orientation (default: 0)
```

Modifiers are evaluated by the ‘crg_mods()’ routine. The mods are evaluated in sequence as they appear below, and are cleared after they are applied. An empty mods array inhibits any default settings. Make sure that you create a new struct for a new modifier set. The example below shows how to set up a modification.

```
data = crg_read('demo1.crg');       % read crg file
data.mods = struct;                 % create new struct
data.mods.gnan = 1;                 % set modifier
data = crg_mods(data);              % apply mods on crg data
```

4.2.7 Options

As mentioned above, the OpenCRG® data struct contains also the so called 'options' which are applied during evaluation, not globally. All modifier functionalities are equal to the C-API and explained there. Further details about the Matlab API are located in ‘crg_intro’ or in the help content of the function.

```
CRG elevation grid border modes in u and v directions
       bdmu           at beginning and end (default: 2)
       bdmv           at left and right side (default: 2)
              = 0: return NaN
              = 1: set zero
              = 2: keep last
              = 3: repeat
              = 4: reflect
       bdou           z offset beyond border (default: 0)
       bdov           z offset beyond border (default: 0)
       bdss           smoothing zone length at start
       bdse           smoothing zone length at end
CRG reference line continuation
       rflc           how to extrapolate (default: 0)
              = 0: follow linear extrapolation
              = 1: close track
CRG reference line search strategy
       sfar           far value (default: 1.5)
       scls           close value (default: sfar/5)
CRG message options
       wmsg           warning messages (default: -1)
       wcvl           local curvature limit exceeded (d:-1)
       wcvg           global curvature limit exceeded (d:-1)
       lmsg           log messages (default: -1)
       leva           evaluation inputs and results (default: 20)
       levf           how often (default: 1)
       lhst           refline search history (default: -1)
       lhsf           how often (default: 100000)
       lsta           evaluation statistics (default: -1)
       lstf           how often (default: 100000)
CRG check options
       ceps           expected min. accuracy (default: 1e-6)
       cinc           expected min. increment (default: 1e-3)
       ctol           expected abs. tolerance (default: 0.1*cinc)
```

4.3 Tools

4.3.1 crg_gen_csb2crg0: Generate synthetic roads

‘crg_gen_csb2crg0’_allows to simple generate crg-files including slope [s], banking [b] and curvature [c] in MATLAB.

```
crg_gen_csb2crg0( inc, u, v, c, s, b )
```

The minimal input parameter are increment [inc] and length information in u,v direction.

```
data = crg_gen_csb2crg0( [1,0.5], 100, 2);
```

Please also see ‘crg_test_gen_csb2crg0’, ‘crg_test_gen_road’, ‘crg_demo_gen_sl*’ and ‘crg_demo_gen_syntheticStraight’ for additional examples handling synthetic road generation.

4.3.1.1 Cross sections

Creating non equidistant cross sections (v) is up to the developer. Nevertheless the OpenCRG – Matlab Api provides several demonstrations (‘crg_demo_gen_sl*’) how a readable v-profile could be set up.

The task is devided into two steps.

1. Create readable profiles (yourself)
2. Generate v-profile for crg_gen_csb2crg0.m (‘crg_check_uv_descript.m’)

4.3.1.1.1 Create profile(s)

The next example show one possibility to create profiles. Please note the comments for explanation.

```
%% STEP 1: create longitudinal and lateral profile(s) like this

% u ----> coordinate          begin                                middle sections                      end
% v ----> coordinate          left                                 origin                               right
%  l ---> left  hand side
%  r ---> right hand side
%  w ---> width of whole road
%  c ---> center of road
%   p --> profile
%    r --> raughness
%     _ -> name_sect
%     _ -> name_prof
%                 offset or amplitude to origin
%                 |
uwp_road_sect  =          [  ubeg                                                                    uend  ];    % road width u sections
uwp_road_prof  =          [  1                                                                       1     ];
vwp_road_sect  =  0     + [  3                                    : -0.5 :                         -3     ];    % road width v sect
vwp_road_prof  =  1     * [  ones(size(vwp_road_sect))                                             ];

ulp_lane_sect  =          [  ubeg                                                                    uend  ];    % left lane  u sections
ulp_lane_prof  =          [  1                                                                       1     ];
vlp_lane_sect  =  1.25  + [  1                                  : -0.2 :                           -1     ];    % left lane  v sect
vlp_lane_prof  =  1     * [  1            0.8 0.8 0.6 0.6    0.6       0.6 0.6 0.8 0.8             1     ];

ulr_lane_sect  =  100   + [  0    10                                                          50     60     ];    % left lane  u sections
ulr_lane_prof  =          [  1     0.8                                                        0.8    1     ];
vlr_lane_sect  =  1.25  + [  1                                  : -0.2 :                           -1     ];    % left lane  v sect
vlr_lane_prof  =  1     * [  ones(size(vlr_lane_sect))                                             ];

urp_lane_sect  =          [  ubeg                  30  40    50   80     90                        uend  ];    % right lane u sections
urp_lane_prof  =          [  1                      1  0.6   0.8  0.5    1                         1     ];
vrp_lane_sect  = -1.25  + [  1                                                                    -1     ];    % right lane v sect
vrp_lane_prof  =  1     * [  1                                                                     1     ];

%             mode one of {'Profile' 'Random' 'Ignore' }
%             |          u section       u profile         v section       v profile        valid only for random profile(s)
%             |          |               |                 |               |                /----------^----------\
uv_mue = { ...
         ; { 'Profile' [ uwp_road_sect ; uwp_road_prof ] [ vwp_road_sect ; vwp_road_prof  ]                          } ...
         ; { 'Random'  [ ulr_lane_sect ; ulr_lane_prof ] [ vlr_lane_sect ; vlr_lane_prof  ] 2010  0.5  1  0.4  0.4 } ...
         ; { 'Profile' [ ulp_lane_sect ; ulp_lane_prof ] [ vlp_lane_sect ; vlp_lane_prof  ]                          } ...
         ; { 'Profile' [ urp_lane_sect ; urp_lane_prof ] [ vrp_lane_sect ; vrp_lane_prof  ]                          } ...
         };
%                                                                                          ^     ^   ^   ^    ^
%                                                                                          |     |   |   |    |
%                                                                                          |     |   |   |    v_smooth (random) in range 0..1
%                                                                                          |     |   |   u_smooth (random) in range 0..1
%                                                                                          |     |    max amplithude random
%                                                                                          |     min amplithude random
%                                                                                          initialize state of random generator

%% end of user settings

%% STEP 2: create and check lateral profile vector

v = crg_check_uv_descript(uv_mue, {'Ignore' 'Profile' 'Random'});
```

4.3.1.1.2 Create profile vector

OpenCRG provides a functionality to extract the cross section vector defined above. For further details see ‘crg_check_uv_descript.m’.

4.3.1.2 Curvature

Curvature can be divided into sections independent of banking and slope. It is described by a polynomial of grade 2. The following represents a possible curvature matrix definition.

c = { LC1 { 1/R1s (1/R1e - 1/R1s) / LC1 } …
; LC2 { 1/R2s (1/R2e - 1/R2s) / LC2 } …

```
                ; ……
                };
```

LC1, LC2 : section length
R1s, R2s : section starting radius
R1e, R2e : section ending radius

See also: ‘crg_test_gen_csb2crg0’

4.3.1.3 Slope

Slope can be divided into sections independent of banking and curvature. It is described by a polynomial of grade 2. The following represents a possible slope matrix definition.

```
s       =       { LS1   {  S1s   (S1e - S1s) / LS1  }  …
                ; LS2   {  S2s   (S2e - S2s) / LS2  }  …
                ; ……
                };
```

LS1, LS2 : section length
S1s, S2s : section starting slope
S1e, S2e : section ending slope

See also: ‘crg_test_gen_csb2crg0’

4.3.1.4 Banking

Banking can be divided into sections independent of slope and curvature. It is described by a polynomial of grade 2 or grade 3. The following represent possible slope matrix definitions.

```
b       =       { LB1   {  B1s   (B1e - B1s) / LB1  }  …
                ; LB2   {  B2s   (B2e - B2s) / LB2  }  …
                ; ……
                };
```

or (non-linear for section LB2)

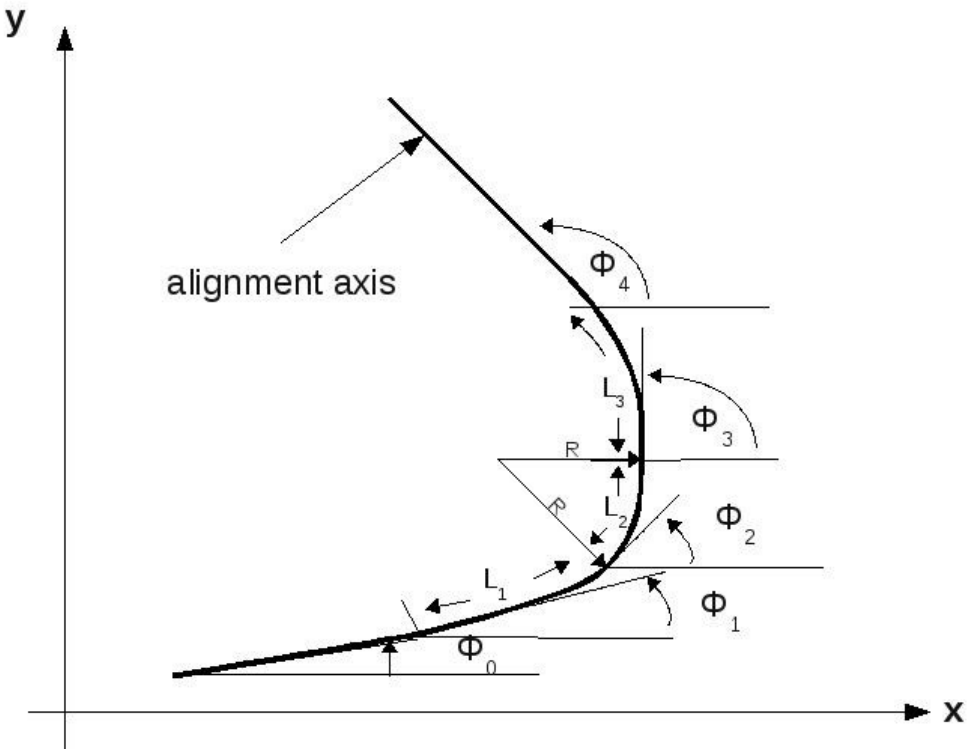
```
b       =       { LB1   {  B1s   (B1e - B1s) / LB1  }  …
                ; LB2   {  B2_a  B2_b  B2_c  B2_d       }  …
                ; ……
                };
```

LB1, LB2 : section length
B1s, B2s : section starting banking
B1e, B2e : section ending banking
B2_a, B2_b, B2_c , B2_d: section polynomial coefficients

See also: ‘crg_test_gen_csb2crg0’

4.3.1.5 Coherence of azimuth direction angle alteration, curve radius and clothoid parameter

(Based on Dr. Klaus Müller (Daimler AG) considerations to calculate sensible parameter for road construction)



$$\Delta\Phi = \Phi_4 - \Phi_1 = (\Phi_2 - \Phi_1) + (\Phi_3 - \Phi_2) + (\Phi_4 - \Phi_3)$$

$$\Delta\Phi = \frac{L_1}{2R} + \frac{L_2}{R} + \frac{L_3}{2R}$$

with R = radius
L_1 = length clothoid 1
L_2 = length circular arc
L_3 = length clothoid 2

common in road construction:

$$\frac{1}{3}R \le L_1 \le R$$

$$\frac{1}{3}R \le L_3 \le R$$

hence $L_{1,2} = f_{1,2} * R$
choose $0 <= f_{1,2}$
yields to

$$\Delta\,\Phi = \frac{f_1}{2} + \frac{L_2}{R} + \frac{f_3}{2} \quad , \qquad \frac{L_2}{R} = \Delta\,\Phi - \frac{1}{2}(f_1 + f_3)$$