RODIS

rigorous optical diffration software

Introduction

this manuals explains some hidden code

1 Install & Build

Installing of a windows version is simple, just clicking the .exe file and it installs itself. If python is installed on the computer of course.

Installing it on other systems requires a building too. In the best case, all works fine with the `python setup.py` command on the command line.

Everything is explaned in the building instructions.

1.1 short building instructions

Have python installed on your pc (www.python.org (`http://www.python.org`))

Get the boost_python-library (www.boost.org (`http://www.boost.org`))

'SCons' should be installed. (www.scons.org (`http://www.scons.org`))

Have the right `util/cmatrix_inv.cxx` there is one for windows and one for unix

Change the machine_cfg.py script (addapt it to your machine)

`python setup.py` does the building and the installation

if something isn't working properly, see next chapter, the longer building instructions..

1.2 building instructions

Have python installed on your pc (www.python.org (`http://www.python.org`))

You can either have the original python or the active-state version. The later has a good editor window to write python-scripts. (for windows, linux users will use xemacs or...?)

Get the boost_python-library (www.boost.org (`http://www.boost.org`))

Boost.python is a c++ library which enables seamless interoperability between c++ and the python programming language. Don't install the whole boost_library! Boost.python does the job. install instructions are on the boost site, see install -> python.

'SCons' should be installed. (www.scons.org (`http://www.scons.org`))

Scons is a python program that acts like the classic '(Make)' utility. It constructs projects. Rodis has a Sconstruct and a SConscript file. Sconscript holds the name of all the files which need to be compiled. Sconscript passes the compiler variables to scons. Those variables need to be adapted in the machine_cfg.py file.

Have the right `util/cmatrix_inv.cxx`

there is one for windows and one for unix. Because of awefull hacking. Hope this will change once, but is not prior at the moment.

Change the machine_cfg.py script (addapt it to your machine)

You should change the compiler commands, flags and adapt the directories where the include files and the libraries are.

`python setup.py` does the building and the installation

If you type '`scons`', the library will be made. e.g rodis/_rodis.so. If this one is in the `PYTHONPATH`, you will be able to import this code in python '`>>> import _rodis`'

The '`scons`' -command is incorporated in the setup.py script. This script will do more than building the .so or .dll. It will install the top-layer-scripts too (rodis_ui.py, rodis_version.py, __init__) and set the pythonpath! That's all done by the '`python setup.py`' command.

To make the windows self-installer, I used `python setup.py bdist_wininst --bitmap="rodis.1.2.bmp" --target-version="2.2"`. This command will make a new directory '`dist`' and put one exe-file in it. This file includes some libraries and documentation too. (e.g the boost library).

2 Files

The rodis code is not that big, and it is quite easy to understand the major meaning of all files. Some Basic code however is changed and rewritten over the past 10 years, without the aim of distrubution, what makes it a slightly more difficult to get.

‘`rodis/`’ This directory holds all the core-rodis cpp and header files

‘`util/`’ Holds some ccp files rodis needs for common mathematics. Some files are translated from fortran and pretty incomprehensive.

‘`rodis/rodis_wrap.cpp`’
This file wraps the rodis-cpp code into a rodis-python code. It gives the cpp variables a python equivalent.

‘`rodis_ui.py`’
This file is a pure python script. This files gives the program a ’camfr’-look and feel. E.g.,it gives the rodis variables from rodis_wrap.cpp a new name, creates new classes and new functions. All this work could also be provided on cpp-level as it is the case for ’camfr’. Isn’t done though.

‘`rodis_version.py`’
Is the version number!

‘`setup.py`’
Is used to build the python-project as explained in the building instructions

‘`SConstruct`’
This file is used by scons to compile all the files.

‘`rodis/SConscript`’
A list of all the files which need to be compiled

‘`machine_cfg.py`’
Holds the variables, used by the setup.py and SConsstruct

3 Gratings

The python-user-interface-syntax (see the rodis_ui.py file) hides the distinct gratings which RODIS uses. Depending on the settings, the ui will chose an aproppriate grating. Those are discused in this chapter

Three conditions decide what grating will be used:

Dimension (1D or 2D)
Incident angle (priciple plane or conical)
Lossy (material with complex refractive index?)

3.1 1DM & 1DML

1DM is the most simple grating. Only incident beams in the pricipal plane can be used. So, this one is chosen in case of a 1 Dimensional structure where delta is zero (by default, or by set_delta(0.0))

settings:

set_alpha()
set_N()
set_lambda()
set_polarisation()

Build a structure with Material, Slab and Stack.

gettings:

diff_reff().R()
diff_reff().T()
field().R()
field().T()

Now by changing the material refractive index from real into a complex, a 1DML grating will be build, using the same settings and gettings as for a 1DM.

3.2 1DC & 1DCL

In case of one dimensional gratings with a conical incident beam, 1DC gratings are used. If your script holds a set_delta(x) with 0 < x, and the grating is constructed as a 1D grating, you're working with a 1DC, and polarization setting is no longer needed

settings:

set_alpha()
set_delta()
set_N()
set_lambda()
set_phi()
set_psi()

Build a structure with Material, Slab and Stack.

gettings:

diff_reff().R()
diff_reff().T()
field().R_TM()
field().R_TE()
field().T_TM()
field().T_TE()

Now by changing the material refractive index from real into a complex, a 1DCL grating will be build, using the same settings and gettings as for a 1DC.

3.3 2D & 2DL

Very easy: if you build a two dimensional grating, 2D is used, no matter what incident beam is used.

settings:

set_alpha()
set_delta()
set_Nx()
set_Ny()
set_lambda()
set_phi()
set_psi()
set_dzeta()

Build a structure with Material, Slab, Section and Stack.

gettings:

diff_reff().R()
diff_reff().T()
field().R_TM()
field().R_TE()
field().T_TM()
field().T_TE()

Now by changing the material refractive index from real into a complex, a 2DL grating will be build, using the same settings and gettings as for a 2D.

3.4 Show Grating

you can see what kind of grating you're using, look at C++ level and type `device.showgrating()`. The first line will inform you whether the grating is 1DM, 1DC, 2D 1DML, 1DCL, 2DL

4 About

RODIS is written as part of the PhD-thesis of Bart Dhoedt (`"`Theoretical and Experimental Study of Free Space Optical Interconnections Based on Diffractive Lens Arrays`"` 1995). The 2D gratings were added by Danae Delbeke during her PhD-work (`"`Design and fabrication of a highly efficient light-emitting diode: The grating-Assisted Resonant-Cavity Light-Emitting Diode`"`, 2002). In order to have a more user-friendly access to the `c++` files, Lieven wrote the python-wrapper, with big support from Peter Bienstman, whose camfr (`http://camfr.sourceforge.net/`) -syntax was used as a model for the rodis-syntax.

All work is done at the photonics group (`http://photonics.intec.ugent.be/`) from the Department of Informationtechnology (INTEC (`http://www.intec.ugent.be/`)) of Ghent University, Flanders, Europe.

Index