radmc3d

Release 2.0

Cornelis Dullemond

Jan 29, 2022

CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 What is RADMC-3D?

RADMC-3D is a software package for astrophysical radiative transfer calculations in arbitrary 1-D, 2-D or 3-D geometries. It is mainly written for continuum radiative transfer in dusty media, but also includes modules for gas line transfer. Typical applications would be protoplanetary disks, pre- and proto-stellar molecular cloud cores, and similar objects. It does not treat photoionization of gas, nor does it treat chemistry. It can self-consistently compute dust temperatures for the radiative transfer, but it is not equipped for self-consistent gas temperature computations (as this requires detailed coupling to photochemistry). The main strength of RADMC-3D lies in the flexibility of the spatial setup of the models: One can create or use parameterized dust and/or gas density distributions, or one can import these from snapshots of hydrodynamic simulations.

1.2 Capabilities

Here is a list of current and planned features. Those features that are now already working are marked with [+], while those which are not yet (!!) built in are marked with [-]. Those that are currently being developed are marked with [.] and those that are ready, but are still in the testing phase are marked with [t].

1. Coordinate systems:
    1. [+] Cartesian coordinates (3-D)
    2. [+] Spherical coordinates (1-D, 2-D and 3-D)
2. Gridding systems (regular and adaptive mesh refinement grids are available for cartesian *and* spherical coordinates):
    1. [+] Regular
    2. [+] Adaptive Mesh Refinement: oct-tree style
    3. [+] Adaptive Mesh Refinement: layered ('patch') style
    4. [+] Delaunay gridding
    5. [+] Voronoi gridding
    6. [+] Flexible unstructured gridding
3. Radiation mechanisms:
    1. [+] Dust continuum, thermal emission
    2. [+] Dust continuum scattering:
        1. [+] . . . in isotropic approximation

    2. [+] ... with full anisotropy
    3. [+] ... with full Stokes and Polarization
  3. [-] Dust quantum heated grains *[To be implemented on request]*
  4. [t] Polarized dust emission by aligned grains *[first test version]*
  5. [+] Gas line transfer (LTE)
  6. [+] Gas line transfer (non-LTE: LVG)
  7. [+] Gas line transfer (non-LTE: LVG + Escape Probability)
  8. [-] Gas line transfer (non-LTE: full transfer)
  9. [+] Gas line transfer with user-defined populations
  10. [+] Gas continuum opacity and emissivity sources
4. Radiation netto sources for continuum:
  1. [+] Discrete stars positioned at will
  2. [t] Continuous 'starlike' source
  3. [t] Continuous 'dissipation' source
  4. [t] External 'interstellar radiation field'
5. Imaging options:
  1. [+] Observer from 'infinite' distance
  2. [+] Zoom-in at will
  3. [+] Flux-conserving imaging, i.e. pixels are recursively refined
  4. [+] A movie-making tool
  5. [+] Multiple wavelengths in a single image
  6. [+] Local observer with perspective view (for PR movies!)
6. Spectrum options:
  1. [+] SED spectrum (spectrum on 'standard' wavelength grid)
  2. [+] Spectrum on any user-specified wavelength grid
  3. [+] Spectrum of user-specified sub-region (pointing)
  4. [t] Specification of size and shape of a primary 'beam' for spectra
7. User flexibility:
  1. [+] Free model specification via tabulated input files
  2. [+] Easy special-purpose compilations of the code (optional)
8. Front-end Python packages:
  1. [+] Python simple tools for RADMC-3D
  2. [+] Python RADMC-3D library {smalltt radmc3dPy} (author: A. Juhasz)
9. Miscellaneous:
  1. [+] Stars can be treated as point-sources or as spheres
  2. [+] Option to calculate the mean intensity $J_\nu(\vec{x})$ in the model

3. [+] OpenMP parallellization of the Monte Carlo

1.3 Version tracker

The RADMC-3D software package in under continuous development. A very detailed development log-book is found in the git repository. A more user-friendly overview of the development history can be found in this manual, in appendix Section 21.

1.4 Copyright

RADMC-3D was developed from 2007 to 2010/2011 at the Max Planck Institute for Astronomy in Heidelberg, funded by a Max Planck Research Group grant from the Max Planck Society. As of 2011 the development continues at the Institute for Theoretical Astrophysics (ITA) of the Zentrum für Astronomy (ZAH) at the University of Heidelberg.

1.5 Contributing authors

The main author of RADMC-3D is Cornelis P. Dullemond. However, the main author of the `radmc3dPy` Python package is Attila Juhasz.

Numerous people have made contributions to RADMC-3D. Major contributions are from:

- Michiel Min
- Attila Juhasz
- Adriana Pohl
- Rahul Shetty
- Farzin Sereshti
- Thomas Peters
- Benoit Commercon
- Alexandros Ziampras

The code profited from testing, feedback and bug reports from (incomplete list):

- Daniel Harsono
- Rainer Rolffs
- Laszlo Szucs

- Sean Andrews
- Stella Offner
- Chris Beaumont
- Katrin Rosenfeld
- Soren Frimann
- Jon Ramsey
- Seokho Lee
- Blake Hord
- Tilman Birnstiel
- Uma Gorti

and others.

1.6 Disclaimer

IMPORTANT NOTICE 1: I/We reject all responsibility for the use of this package. The package is provided as-is, and we are not responsible for any damage to hardware or software, nor for incorrect results that may result from the software. The user is fully responsible for any results from this code, and we strongly recommend thorough testing of the code before using its results in any scientific papers.

IMPORTANT NOTICE 2: Any publications which involve the use of this software must mention the name of this software package and cite the accompanying paper once it is published (Dullemond et al.in prep), or before that the above mentioned web site.

CHAPTER

TWO

QUICKSTARTING WITH RADMC-3D

In general I recommend reading the manual fully, but it is often useful to get a quick impression of the package with a quick-start. To make your first example model, this is what you do:

1. When you read this you have probably already unzipped this package, or cloned the git repository. You should find, among others, a `src/` directory and a `examples/` directory. Go into the `src/` directory.
2. Edit the `src/Makefile` file, and make sure to set the `FF` variable to the Fortran-90 compiler you have installed on your system.
3. Type `make`. If all goes well, this should compile the entire code and create an executable called `radmc3d`.
4. Type `make install`. If all goes well this should try to create a link to `radmc3d` in your `$HOME/bin/` directory, where `$HOME` is your home directory. If this `$HOME/bin/` directory does not exist, it will ask to make one.
5. Make sure to have the `$HOME/bin/` directory in your path. If you use, for instance, the `bash` shell, you do this by setting the `PATH` variable by adding a line like `export PATH=$HOME/bin:\$PATH` to your `$HOME/.bashrc` file. If you change these things you may have to open a new shell to make sure that the shell now recognizes the new path.
6. Check if the executable is OK by typing `radmc3d` in the shell. You should get a small welcoming message by the code.
7. Now enter the directory `examples/run_simple_1/`. This is the simplest example model.
8. Type `python problem_setup.py` (Note: you must have a working Python distribution on your computer, which is reasonably up to date, with `numpy` and `matplotlib` libraries included). This will create a series of input files for RADMC-3D.
9. Type `radmc3d mctherm`. This should let the code do a Monte Carlo run. You should see `Photon nr 1000`, followed by `Photon nr 2000`, etc until you reach `Photon nr 1000000`. The Monte Carlo modeling for the dust temperatures has now been done. A file `dust_temperature.dat` should have been created.
10. Type `radmc3d image lambda 1000 incl 60 phi 30`. This should create an image with the camera at inclination 60 degrees (from pole-on), and rotated 30 degrees (along the polar axis, clockwise, i.e.the object rotating counter-clockwise), at wavelength $\lambda = 1000\,\mu$m (i.e. at 1 millimeter wavelength). The file that contains the image is `image.out`. It is a text file that can be read with the `simpleread.py` tool in the directory `python/radmc3d_tools/`.

If you experience troubles with the above steps, and you cannot fix it, please read the next chapters for more details.

CHAPTER

THREE

OVERVIEW OF THE RADMC-3D PACKAGE

3.1 Introduction

The RADMC-3D code is written in fortran-90 and should compile with most f90 compilers without problems. It needs to be compiled only once for each platform.

The executable is called `radmc3d` and it performs all the model calculations of the RADMC-3D package, for instance the Monte Carlo simulations, ray-tracing runs (images, spectra), etc. There is also a set of useful subroutines written in the Python language to use the `radmc3d` code, but `radmc3d` can also run without it. In that case the user will have to write his/her own pre- and post-processing subroutines.

3.2 Requirements

The following pre-installed software is required:

1. *Operating system: Unix-like (e.g. Linux or MacOSX)*

   This package runs under Unix-like environment (e.g. Linux or MacOSX), but has not been tested under Windows. There is no particular reason why it should not also run under Windows, but it would require different ways of file handling. *In this manual we always assume a Unix-like environment, in which we will make use of a bash command-line interface (CLI).* We will call this the *shell*.

2. `make` or `gmake`

   This is the standard tool for compiling packages on all Unix/Linux/MacOS-based systems.

3. `perl`

   This is a standard scripting language available on most or all Unix/Linux-based systems. If you are in doubt: type `which perl` to find the location of the `perl` executable. See http://www.perl.org/ for details on perl, should you have any problems. But on current-day Unix-type operating systems perl is nearly always installed in the `/usr/bin/` directory. If you do not have Perl installed, you can also do without. Its sole use is to copy the executables into your home `$HOME/bin/` directory for quick access from the Unix/Linux/MacOS command line. You can work around that, if necessary.

4. *A fortran-90 compiler*

   Preferably the `gfortran` compiler (which the current installation assumes is present on the system). Website: http://gcc.gnu.org/fortran/. Other compilers may work, but have not been tested yet.

5. *An OpenMP-fortran-90 compiler (optional)*

   Only needed if you want to use the parallelized OpenMP version for the thermal Monte Carlo (for faster execution). Preferably the `GNUOpenMP/GOMP` compiler which is an implementation of OpenMP for the Fortran

compiler in the GNU Compiler Collection. Websites: http://openmp.org/wp and http://gcc.gnu.org. Other compilers may work, but we give no guarantee.

6. *Python version 3 with standard libraries*

   The core RADMC-3D code `radmc3d` (written in Fortran-90) is the raw workhorse code that reads some input files and produces some output files. Typically you will not need to worry about the internal workings of the RADMC-3D code. All you need to do is produce the proper input files, run `radmc3d`, and read the output files for post-processing (such as displaying and analyzing the results). This pre- and post-processing is done in Python. This RADMC-3D distribution provides you with the Python tools you need, though you will likely want to program your own additional Python code to adjust the models to your own needs. To use the Python tools provided in this RADMC-3D distribution, you need Python version 3 (though most things should also work with the depricated Python 2), with a set of standard libraries such as `numpy` and `matplotlib`. Typically we will assume that Python is used as a Python or iPython command-line interface, which we shall call the *Python command line* (as opposed to the *shell*). The user can, of course, also use Jupyter Notebooks instead. But for the sake of clarity, in this manual we assume the use of Python the Python command line.

Note that the Monte Carlo code RADMC-3D itself (`radmc3d`) is in Fortran-90. Only the creation of the input files (and hence the problem definition) and the analysis of the output files is done in Python. The user is of course welcome to use other ways to create the input files for RADMC-3D if he/she is not able or willing to use Python for whatever reason. Therefore Python is not strictly required for the use of this code. However, all examples and support infrastructure is provided in Python.

3.3 Contents of the RADMC-3D package

3.3.1 RADMC-3D package as a .zip archive

If you obtain RADMC-3D from its website, it will be packed in a zip archive called `radmc-3d_v*.*_dd.mm.yy.zip` where the `*.*` is the version number and `dd.mm.yy` is the date of this version. To unpack on a linux, unix or Mac OS X machine you type:

```
unzip <this archive file>
```

i.e. for example for `radmc-3d_v2.0_25.08.20.zip` you type:

```
unzip radmc-3d_v2.0_25.08.20.zip
```

3.3.2 RADMC-3D package from the github repository

If you obtain RADMC-3D by cloning its github repository, you will get a copy of the full git repository of RADMC-3D. In principle this is not much different from unzipping the .zip archive. But it is more powerful: You can more easily stay up to date with the latest bugfixes, and you can see the entire development history of this version of the code. See https://git-scm.com/book/en/v2 for an extensive documentation of how to use git.

The way to produce a clone of RADMC-3D in the directory where your shell is currently is, is like this:

```
git clone https://github.com/dullemond/radmc3d-2.0.git
```

This will create the directory `radmc3d-2.0/`. At any time you can pull the latest version from the repository like this:

```
cd radmc3d-2.0/
git pull
```

Keep in mind, however, that while the repository is always the very latest version, this comes with a (small) risk that some new features may not have been tested well, or (new) bugs may have been introduced. Overall, however, we advise to use the github repository instead of the .zip archive from the website.

3.3.3 Contents of the package

The RADMC-3D package has the following subdirectory structure:

```
src/
python/
examples/
   run_simple_1/
   run_simple_1_userdef/
   run_simple_1_userdef_refined/
   .
   .
   .
opac/
manual/
```

plus some further directories.

The first directory, `src/`, contains the fortran-90 source code for RADMC-3D. The second directory, `python/`, contains two sets of Python modules that are useful for model preparation and post-processing. One is a directory called `radmc3d_tools/`, which contains some simple Python tools that might be useful. The other is a directory called `radmc3dPy/`, which is a high-level stand-alone Python library developed by Attila Juhasz for RADMC-3D. The third directory contains a series of example models. The fourth directory, `opac/` contains a series of tools and data for creating the opacity files needed by RADMC-3D (though the example models all have their own opacity data already included), The fifth directory contains this manual.

3.4 Units: RADMC-3D uses CGS units

The RADMC-3D package is written such that all units are in CGS (length in cm, time in sec, frequency in Hz, energy in erg, angle in steradian). There are exceptions:

- Wavelength is usually written in micron
- Sometimes angles are in degrees (internally in radian, but input as degrees)

CHAPTER

FOUR

INSTALLATION OF RADMC-3D

Although the RADMC-3D package contains a lot of different software, the main code is located in the `src/` directory, and is written in Fortran-90. The executable is `radmc3d`. Here we explain how to compile the fortran-90 source codes and create the executable `radmc3d`.

4.1 Compiling the code with 'make'

To compile the code, enter the `src/` directory in your shell. You now *may* need to edit the `Makefile` in this directory using your favorite text editor and replace the line

```
FF = gfortran -fopenmp
```

with a line specifying your own compiler (and possibly OpenMP directive, if available). If, of course, you use gfortran, you can keep this line. But if you use, e.g., ifort, then replace the above line by

```
FF = ifort -openmp
```

(note the slightly different OpenMP directive here, too). If you save this file, and you are back in the shell, you can compile the radmc3d code by typing

```
make
```

in the shell. If all goes well, you have now created a file called `radmc3d` in the `src/` directory.

If, for whatever reason, the OpenMP compilation does not work, you can also compile the code in serial mode. Simply remove the `-fopenmp` directive.

4.2 The install.perl script

If instead of typing just `make` you type

```
make install
```

(or you first type `make` and then `make install`, it is the same), then in addition to creating the executable, it also automatically executes a perl script called `install.perl` (located also in the `src/` directory). This PERL script installs the code in such a way that it can be conveniently used in any directory. What it does is:

- It checks if a `bin/` directory is present in your home directory (i.e. a `$HOME/bin/` directory). If not, it asks if you want it to automatically make one.

- It checks if the `$HOME/bin/` directory is in the *path* of the currently used shell. This is important to allow the computer to look for the program `radmc3d` in the `$HOME/bin/` directory. If you use a bash shell, then you can add the following line to your `$HOME/.bashrc`:

```
export PATH=/myhomedirectory/bin/python:$PATH
```

- It creates a file `radmc3d` in this `$HOME/bin/` directory with the correct executable permissions. This file is merely a dummy executable, that simply redirects everything to the true `radmc3d` executable located in your current `src/` directory. When you now open a new shell, the path contains the `$HOME/bin/` directory, and the command `radmc3d` is recognized. You can also type `source $HOME/.bashrc` followed by `rehash`. This also makes sure that your shell recognizes the `radmc3d` command.
- It checks if a `python/` subdirectory exists in the above mentioned `bin/` directory, i.e.a `$HOME/bin/python/` directory. If not, it asks if you want it to automatically create one.
- If yes, then it will copy all the files ending with `.py` in the `python/radmc3d_tools/` directory of the distribution to that `$HOME/bin/python/radmc3d_tools/` directory. This is useful to allow you to make an `PYTHONPATH` entry to allow python to find these python scripts automatically.

Note that this perl script installs the code only for the user that installs it. A system-wide installation is not useful, because the code package is not very big and it should remain in the control of the user which version of the code he/she uses for each particular problem.

If all went well, then the `perl.install` script described here is called automatically once you type `make install` following the procedure in Section *Compiling the code with 'make'*.

Before the installation is recognized by your shell, you must now either type `rehash` in the shell or simply open a new shell.

How do you know that all went OK? If you type `radmc3d` in the shell the RADMC-3D code should now be executed and give some comments. It should write:

```
================================================================
     WELCOME TO RADMC-3D: A 3-D CONTINUUM AND LINE RT SOLVER

                          VERSION 2.0

               (c) 2008-2020 Cornelis Dullemond

      Please feel free to ask questions. Also please report
       bugs and/or suspicious behavior without hestitation.
     The reliability of this code depends on your vigilance!
                    dullemond@uni-heidelberg.de

  To keep up-to-date with bug-alarms and bugfixes, register to
                     the RADMC-3D forum:
           http://radmc3d.ita.uni-heidelberg.de/phpbb/

              Please visit the RADMC-3D home page at
 http://www.ita.uni-heidelberg.de/~dullemond/software/radmc-3d/
================================================================

Nothing to do... Use command line options to generate action:
  mctherm        : Do Monte Carlo simul of thermal radiation
  mcmono         : Do Monte Carlo simul only for computing mean intensity
  spectrum       : Make continuum spectrum
  image          : Make continuum image
```

on the screen (or for newer versions of RADMC-3D perhaps some more or different text). This should also work from any other directory.

4.3 What to do if this all does not work?

In case the above compilation and installation does not work, here is a proposed procedure to do problem hunting:

1. First, answer the following questions:
   - Did you type `make install` in the `src/` directory? I mean, did you not forget the `install` part?
   - Did you put `$HOME/bin/` in your path (see above)?
   - If you just added `$HOME/bin/` to your path, did you follow the rest of the procedure (either closing the current shell and opening a new shell or typing the `source` and `rehash` commands as described above)?

   If this does not help, then continue:
2. Close the shell, open a new shell.
3. Go to the RADMC-3D `src/` directory.
4. Type `./radmc3d`. This should give the above message. If not, then make sure that the compilation went right in the first place:
5. Type `rm -f radmc3d`, to make sure that any old executable is not still present.
6. Type `make clean`. This should return the sentence `OBJECT and MODULE files removed.`
7. In case the problem lies with the OpenMP parallellization, you could do `cp Makefile_normal Makefile`, which switches off the OpenMP compilation.
8. Then type `make`. This should produce a set of lines, each representing a compilation of a module, e.g. `gfortran -c -O2 ./amr_module.f90 -o amr_module.o`, etc. The final line should be something like `gfortran -O2 main.o ..... gascontinuum_module.o -o radmc3d`. If instead there is an error message, then do the following:
   - Check if the compiler used (by default `gfortran`) is available on your computer system.
   - If you use an other compiler, check if the compiler options used are recognized by your compiler.
   - Check if the executable `radmc3d` is now indeed present. If it is not present, then something must have gone wrong with the compilation. So then please check the compilation and linking stage again carefully.

   If you followed all these procedures, but you still cannot get even the executable in the `src/` directory to run by typing (in the `src/` directory) `./radmc3d` (don't forget the dot slash!), then please contact the author.
9. At this point we assume that the previous point worked. Now go to another directory (any one), and type `radmc3d`. This should also give the above message. If not, but the `radmc3d` executable was present, then apparently the shell path settings are wrong. Do this:
   - Check if, in the current directory (which is now not `src/`) there is by some accident another copy of the executable `radmc3d`. If yes, please remove it.
   - Type `which radmc3d` to find out if it is recognized at all, and if yes, to which location it points.
   - Did you make sure that the shell path includes the `$HOME/bin/` directory, as it should? Otherwise the shell does not know where to find the `$HOME/bin/radmc3d` executable (which is a perl link to the `src/radmc3d` executable).
   - Does the file `$HOME/bin/radmc3d` perl file exist in the first place? If no, check why not.

- Type `less $HOME/bin/radmc3d` and you should see a text with first line being `#!/usr/bin/perl` and the second line being someting like `system("/Users/user1/radmc-3d/version_2.0/src/radmc3d @ARGV");` where the `/Users/user1` should of course be the path to your home directory, in fact to the directory in which you installed RADMC-3D.

If this all brings you no further, please first ask your system administrators if they can help. If not, then please contact the author.

4.4 Installing the simple Python analysis tools

RADMC-3D offers (in addition to the model setup scripts in the `examples/` subdirectories) two Python support libraries:

1. `python/radmc3d_tools/`

   This library contains only some bare-bones small Python scripts.

2. `python/radmc3dPy/`

   This library is a sophisticated stand-alone library developed by Attila Juhasz, and further maintained together with the RADMC-3D main author.

4.4.1 How to install and use the `python/radmc3d_tools/`

The installation of the `python/radmc3d_tools` should be automatic when you type `make install` in the `src/` code directory (see above). It will copy the files to the `bin/python/radmc3d_tools/` directory in your home directory. If this directory does not exist, you will be asked if you want it to be created. If you confirm (typing 'y'), then the files from the `python/radmc3d_tools/` directory will be copied into the `$HOME/bin/python/radmc3d_tools/` directory.

Now you need to make sure that Python knows that these tools are there. In Python here are two ways how you can make sure that Python automatically finds these scripts:

1. Under Unix/Linux/MacOSX you can set the `PYTHONPATH` directly in your `.bashrc` file. For example: in `.bashrc` (if you use the bash shell) you can write:

```
export PYTHONPATH=$HOME/bin/python:$PYTHONPATH
```

(where `$HOME` is your home directory name).

1. Alternatively you can set the `PYTHONPATH` directly from within Python with the python command:

```
import os
import sys
home = os.environ["HOME"]
sys.path.append(home+'/bin/python')
```

If all goes well, if you now start Python you should be able to have access to the basic Python tools of RADMC-3D directly. To test this, try typing `from radmc3d_tools.simpleread import *` in Python. If this gives an error message that `simpleread.py` cannot be found, then please ask your system administrators how to solve this.

You may ask why first copy these files to `$HOME/bin/python/radmc3d_tools/` and not point PYTHONPATH directly to the `python/radmc3d_tools` in your RADMC-3D distribution? The reason is that if you have multiple versions of RADMC-3D on your computer system, you always are assured that Python finds the python routines belonging to the latest installation of RADMC-3D (note: only assured if that latest compilation was done with `make install`).

Now you should be ready to use the tools. The most important one would be the `simpleread.py` tool, which contains a set of functions for reading typical RADMC-3D input and output files (though only for regular model grid, not for octree grids). In a Python command line interface you can import them by:

```
from radmc3d_tools import simpleread
```

And you can then, for instance, read the dust density file with:

```
d = simpleread.read_dustdens()
```

Here, `d` is now an object containing a `d.grid` subobject (which contain information about the grid) and the dust density array `d.rhodust`. Have a look at the various functions in `simpleread`, to see what is available.

4.4.2 How to install and use the `python/radmc3dPy` library

The installation of the `python/radmc3dPy` package is described in the `python/radmc3dPy/README` file. In short, by going into the `python/radmc3dPy/` directory and typing in the shell:

```
python setup.py install --user
```

it should install itself right into your Python distribution. For instance, if you have `anaconda3` on a Mac, it would copy the files into the directory

```
$HOME/.local/lib/python3.7/site-packages/radmc3dPy/
```

Python knows where to find it there.

Now you should be ready to use `radmc3dPy`, by importing it:

```
import radmc3dPy
```

`radmc3dPy` consists of several sub libraries such as `radmc3dPy.analyze` and `radmc3dPy.image`. For instance, to read the dust density distribution, you could do this:

```
from radmc3dPy import analyze
d = analyze.readData(ddens=True)
```

The `d.rhodust` array now contains the dust density.

For more information, please consult the `radmc3dPy` documentation in the `python/radmc3dPy/doc/` directory.

4.5 Making special-purpose modified versions of RADMC-3D (optional)

For most purposes it should be fine to simply compile the latest version of RADMC-3D once-and-for-all, and simply use the resulting `radmc3d` executable for all models you make. Normally there is no reason to have to modify the code, because models can be defined quite flexibly by preparing the various input files for RADMC-3D to your needs. So if you are an average user, you can skip to the next subsection without problem.

But sometimes there *is* a good reason to want to modify the code. For instance to allow special behavior for a particular model. Or for a model setup that is simply easier made internally in the code rather than by preparing large input files. One can imagine some analytic model setup that might be easier to create internally, so that one can make use of the full AMR machinery to automatically refine the grid where needed. Having to do so externally from the code would require you to set up your own AMR machinery, which would be a waste of time.

The problem is that if the user would modify the central code for each special purpose, one would quickly lose track of which modification of the code is installed right now.

Here is how this problem is solved in RADMC-3D:

- For most purposes you can achieve your goals by only editing the file `userdef_module.f90`. This is a set of standard subroutines that the main code calls at special points in the code, and the user can put anything he/she wants into those subroutines. See Chapter *Modifying RADMC-3D: Internal setup and user-specified radiative processes* for more information about these standard subroutines. This method is the safest way to create special-purpose codes. It means (a) that you know that your modification cannot do much harm unless you make really big blunders, because these subroutines are meant to be modified, and (b) you have all your modifications *only* in one single file, leaving the rest of the code untouched.
- You can create a *local* version of the code, without touching the main code. Suppose you have a model directory `run_mymodel` and for this model you want to make a special-purpose version of the code. This is what you do:
  1. Copy the Makefile from the `src/` directory into `run_mymodel`.
  2. Copy the `.f90` file(s) you want to modify from the `src/` directory into `run_mymodel`. Usually you only want to modify the `userdef_module.f90` file, but you can also copy any other file if you want.
  3. In the `run_mymodel/Makefile` replace the `SRC = .` line with `SRC = XXXXXX`, where `XXXXXX` should be the *full* path to the `src/` directory. An example line is given in the Makefile, but is commented out.
  4. In the `run_mymodel/Makefile` make sure that all the `.f90` files that should remain as they are have a `$(SRC)/` in front of the name, and all the `.f90` files that you want to modify (and which now have a copy in the `run_mymodel` directory) have a `./` in front of the name. By default all `.f90` files have `$(SRC)/` in front of the name, except the `userdef_module.f90` file, which has a `./` in front of the name because that is the file that is usually the one that is going to be edited by you.
  5. Now edit the local `.f90` files in the `run_mymodel` directory in the way you want. See Chapter *Modifying RADMC-3D: Internal setup and user-specified radiative processes* for more details.
  6. Now *inside* the `run_mymodel` directory you can now type `make` and you will create your own local `radmc3d` executable. NOTE: Do not type `make install` in this case, because it should remain a local executable, only inside the `run_mymodel` directory.
  7. If you want (though this is not required) you can clean up all the local `.o` and `.mod` files by typing `make clean`, so that your `run_mymodel` directory is not filled with junk.
  8. You can now use this special purpose version of `radmc3d` by simply calling on the command line: `./radmc3d`, with any command-line options you like. Just beware that, depending on the order in which you have your paths set (in tcsh or bash) typing just `radmc3d` *may* instead use the global version (that you may have created in the `src/` directory with `make install`). So to be sure to use the *local* version, just put the `./` in front of the `radmc3d`.

Note: In chapter *Modifying RADMC-3D: Internal setup and user-specified radiative processes* there is more information on how to set up models internally in the code using the method described here.

Note: You can use `make clean` to remove all the .o and .mod files from your model directory, because they can be annoying to have hanging around. By typing `make cleanmodel` you remove, in addition to the .o and .mod files, also all model input and output files, with the exception of dust opacity or molecular data files (because these latter files are usually not created locally by the `problem_setup.py` script). By typing `make cleanall` you remove everything *except* the basic files such as the `Makefile`, any `.f90` files, any `.py` files, the dust opacity or molecular data files and `README` files.

CHAPTER

FIVE

BASIC STRUCTURE AND FUNCTIONALITY

RADMC-3D is a very versatile radiative transfer package with many possibilities. As a consequence it is a rather complex package. However, we have tried to keep it still as easy as possible to use as a first-time user. We tried to do so by keeping many of the sophisticated options 'hidden' and having many default settings already well-chosen. The idea is that one can already use the code at an entry level, and then gradually work oneself into the more fancy options.

RADMC-3D is a general-purpose package, so there are no 'built-in' models inside the `radmc3d` executable (Except if you insert one yourself using the userdef module, see Chapter *Modifying RADMC-3D: Internal setup and user-specified radiative processes*). For instance, if you want to model a protoplanetary disk, then you would have to design the grid and density structure of the disk on this grid yourself. To make it easier for the user, we have provided several Python-scripts as examples. Among these examples is indeed a protoplanetary disk model. So this is as close as we go to 'built-in' models: we provide, for some cases, already well-developed example models that you, the user, can use out-of-the-box, or that you can adapt to your needs.

In this chapter we give an overview of the rough functionality of the code in its simplest form: ignoring all the hidden fancy options and possibilities. For the details we then refer to the chapters ahead.

5.1 Basic dataflow

Let us first clarify the basic philosophy of the code package (details will be done later). When we talk about RADMC-3D we talk about the fortran-90 program. The source codes are in the directory `src/` and the executable is called `radmc3d`. This is the code that does all the main calculations. You can call the code from the bash shell (in Unix/Linux/MacOSX systems) and you can specify command-line options to tell RADMC-3D what you want it to do.

The code RADMC-3D is in a way just a dumb computational engine. It has no physical data (such as opacities or material properties) implemented, nor does it have any model implemented. It is totally dependent on input files of various kinds. These input files have filenames that end in `.inp`, or `.binp`, dependent on whether the data in ASCII, or binary form. You, the user, will have to create these input files. RADMC-3D will simply look if an `.inp`, or a `.binp` file is present, and will switch to ASCII, dependent on which file-extension it finds.

After you run RADMC-3D (by calling `radmc3d` with the appropriate command-line options) you will see that the code will have produced one or more output files, with filenames ending in `.out` or `.bout`. Whether RADMC-3D produces ASCII or binary files, depends on a flag called `rto_style` that you can set (see Chapter *Binary I/O files*).

*IMPORTANT NOTE: In this manual we will mostly refer to the ASCII form of input and output files for convenience. But each time we refer to an *.inp, *.dat or *.out file, we implicitly assume that this could also be a *.binp, *.bdat or *.bout file.*

This basic dataflow is shown in Fig. *Pictographic representation of the basic dataflow of RADMC-3D. The user produces the input files; RADMC-3D reads them, performs the calculation, and produces output files. The user can then analyze the output files.*.

Not always can RADMC-3D produce its output files in one go. Sometimes it has to use a two-stage procedure: For dust continuum radiative transfer the dust temperatures are computed first (stage 1), and the images and/or spectra

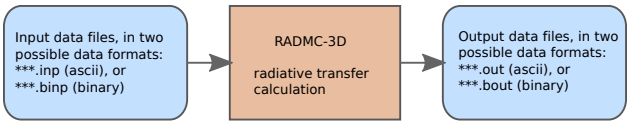

Fig. 5.1: Pictographic representation of the basic dataflow of RADMC-3D. The user produces the input files; RADMC-3D reads them, performs the calculation, and produces output files. The user can then analyze the output files.

are rendered after that (stage 2). Between stage 1 and stage 2 an intermediate file is then produced (with filename ending in `.dat` or `.bdat`), which in the case of dust continuum radiative transfer is `dust_temperature.dat` (or `*.bdat`).

This basic dataflow is shown in Fig. *Pictographic representation of the dataflow of RADMC-3D for the case of a 2-stage procedure, such as for dust continuum transfer. An intermediate file is produced that will be used by stage 2, but of course the user can also analyze the intermediate file itself.*.

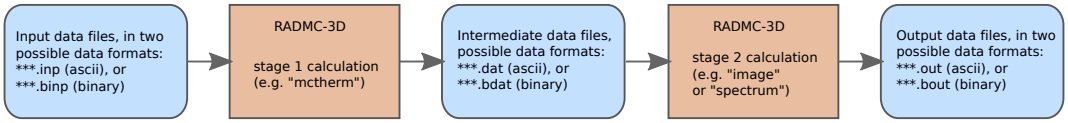

Fig. 5.2: Pictographic representation of the dataflow of RADMC-3D for the case of a 2-stage procedure, such as for dust continuum transfer. An intermediate file is produced that will be used by stage 2, but of course the user can also analyze the intermediate file itself.

Several of these input files contain large tables, for instance of the density at each grid point, or the stellar flux at each wavelength bin. It is, of course, impossible to create these datafiles by hand. The idea is that you design a program (in any language you like) that creates these datafiles. In that program you essentially 'program the model'. We have provided a number of example model setups in the `examples/` directory. For these examples models the setup programs were written in Python (their filenames all start with `problem_` and end with `.py`). For you as the user it is therefore the easiest to start from one of these examples and modify the Python code to your needs. However, if you prefer to use another language, you can use the examples to see how the input files were generated and then program this in another programming language.

Note: The Python files called `problem_*.py` *are meant to be edited and changed by you! They are templates from which you can create your own models.*

For the analysis of the output files created by RADMC-3D you can use your own favorite plotting or data-analysis software. But also here we provide some tools in Python. These Python routines are in the `python/` directory. Typically you will create your own program, e.g. `plot_model.py` or so, that will use these subroutines, e.g. by putting in the first line: `from radmc3dPy import *`. In this way Python is used also as a post-processing tool. But again: this can also be done in another language.

This procedure is shown in Fig. *Pictographic representation of how the Python programs in the example directories are used to create the input files of RADMC-3D.* for the single-stage dataflow and in Fig. *Pictographic representation of the dataflow of RADMC-3D for the case of a 2-stage procedure, such as for dust continuum transfer. An intermediate file is produced that will be used by stage 2, but of course the user can also analyze the intermediate file itself.* for the two-stage dataflow.

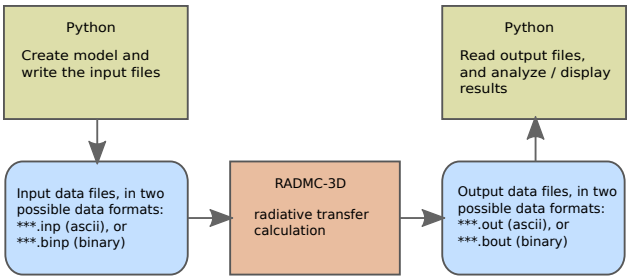

Fig. 5.3: Pictographic representation of how the Python programs in the example directories are used to create the input files of RADMC-3D.

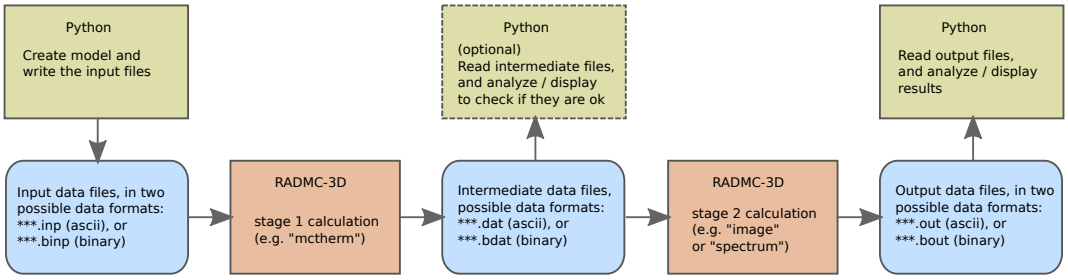

Fig. 5.4: Pictographic representation of the dataflow of RADMC-3D for the case of a 2-stage procedure, such as for dust continuum transfer. An intermediate file is produced that will be used by stage 2, but of course the user can also analyze the intermediate file itself.

5.2 Radiative processes

Currently RADMC-3D handles the following radiative processes:

- Dust thermal emission and absorption

  RADMC-3D can compute spectra and images in dust continuum. The dust temperature must be known in addition to the dust density. In typical applications you will know the dust density distribution, but not the dust temperature, because the latter is the results of a balance between radiative absorption and re-emission. So in order to make spectra and images of a dusty object we must first calculate the dust temperature consistently. This can be done with RADMC-3D by making it perform a 'thermal Monte Carlo' simulation (see Chapter *Dust continuum radiative transfer*). This can be a time-consuming computation. But once this is done, RADMC-3D writes the resulting dust temperatures out to the file `dust_temperature.dat`, which it can then later use for images and spectra. We can then call RADMC-3D again with the command to make an image or a spectrum (see Chapter *Dust continuum radiative transfer*). To summarize: a typical dust continuum radiative transfer calculation goes in two stages:

  1. A thermal Monte Carlo simulation with RADMC-3D to compute the dust temperatures.
  2. A spectrum or image computation using ray-tracing with RADMC-3D.

- Dust scattering

  Dust scattering is automatically included in the thermal Monte Carlo simulations described above, as well as in the production of images and spectra. For more details, consult Chapter *Dust continuum radiative transfer*.

- Gas atomic/molecular lines

  RADMC-3D can compute spectra and images in gas lines (see Chapter *Line radiative transfer*). The images are also known as *channel maps*. To compute these, RADMC-3D must know the population densities of the various atomic/molecular levels. For now there are the following options how to let RADMC-3D know these values:

- Tell RADMC-3D to assume that the molecules or atoms are in *Local Thermodynamic Equilibrium* (LTE), and specify the gas temperature at each location to allow RADMC-3D to compute these LTE level populations. *Note that in principle one is now faced with the same problem as with the dust continuum: we need to know the gas temperature, which we typically do not know in advance.* However, computing the gas temperature self-consistently is very difficult, because it involves many heating and cooling processes, some of which are very complex. That is why most line radiative transfer codes assume that the user gives the gas temperature as input. We do so as well. If you like, you can tell RADMC-3D to use the (previously calculated) dust temperature as the gas temperature, for convenience.
- Deliver RADMC-3D an input file with all the level populations that you have calculated youself using some method.
- Tell RADMC-3D to compute the level populations according to some simple local non-LTE prescription such as the Sobolev approximation (*Large Velocity Gradient method*) or the Escape Probability Method.

Currently RADMC-3D does not have a full non-local non-LTE computation method implemented. The reason is that it is very costly, and for many applications presumably not worth the computational effort.

5.3 Coordinate systems

With RADMC-3D you can specify your density distribution in two coordinate systems:

- Cartesian coordinates: 3-D

  The simplest coordinate system is the Cartesian coordinate system (x, y, z). For now each model must be 3-D (i.e. you must specify the densities and other quantities as a function of x, y and z).

- Cartesian coordinates: 1-D plane-parallel

  This is like the normal cartesian coordinates, but now the x- and y- directions are infinitely extended. Only the z-direction has finite-size cells, and hence the grid is only in z-direction. This mode is the usual plane-parallel mode of radiative transfer. See Section *1-D Plane-parallel models* for more details on this mode.

- Cartesian coordinates: 2-D pencil-parallel

  This is the intermediate between full 3-D cartesian and 1-D plane-parallel. In this mode only the x-direction is infinitely extended and a finite grid is in both y and z directions. This mode is only useful in very special cases, and is much less familiar to most - so use only when you are confident.

- Spherical coordinates

  You can also specify your model in spherical coordinates (r, θ, ϕ). These coordinates are related to the cartesian ones by:

$$x = r \sin\theta \cos\phi$$
$$y = r \sin\theta \sin\phi$$
$$z = r \cos\theta$$

  This means that the spatial variables (density, temperature etc) are all specified as a function of (r, θ, ϕ). However, the location of the stars, the motion and direction of photon packages etc. are still given in cartesian coordinates (x, y, z). In other words: any function of space $f(\vec{x})$ will be in spherical coordinates $f(r, \theta, \phi)$, but any point-like specification of position $\vec{x}$ will be given as Cartesian coordinates $\vec{x} = (x, y, z)$. This hybrid method allows us to do all physics in cartesian coordinates: photon packages or rays are treated always in cartesian coordinates, and so is the physics of scattering, line emission etc. Only if RADMC-3D needs to know what the local conditions are (dust temperature, gas microturbulence, etc) RADMC-3D looks up which coordinates (r, θ, ϕ) belong to the current (x, y, z) and looks up the value of the density, microturbulence etc.at that location in the (r, θ, ϕ) grid. And the same is true if RADMC-3D updates or calculates for instance the dust temperature: it will compute the (r, θ, ϕ) belong to the current (x, y, z) and update the temperature in the cell belonging to

(r, θ, ϕ). For the rest, all the physics is done in the Cartesian coordinate system. This has the major advantage that we do not need different physics modules for cartesian and spherical coordinates. Most parts of the code don't care which coordinate system is used: they will do their own work in Cartesian coordinates. When using spherical coordinates, please read Section *Separable grid refinement in spherical coordinates (important!)*.

5.4 The spatial grid

To specify the density or temperature structure (or any other spatial variable) as a function of spatial location we must have a grid. There are two basic types of grids:

The standard gridding is a simple rectangular grid.

- Cartesian coordinates

  When cartesian coordinates are used, this simply means that each cell is defined as $x_l < x < x_r$, $y_l < y < y_r$ and $z_l < z < z_r$, where l and r stand for the left and right cell walls respectively.

- Spherical coordinates

  When spherical coordinates are used, this simply means that each cell is defined as $r_l < r < r_r$, $\theta_l < \theta < \theta_r$ and $\phi_l < \phi < \phi_r$. Note therefore that the shape of the cells in spherical coordinates is (in real space) curved. For spherical coordinates the following four modes are available:

  - 1-D Spherical symmetry:

    All spatial functions depend only on r.

  - 2-D Axial symmetry:

    All spatial functions depend only on r and θ.

  - 2-D Axial symmetry with mirror symmetry:

    All spatial functions depend only on r and θ, where the θ grid only covers the part above the $z = 0$ plane. Internally it is in this mode assumed that all quantities below the $z = 0$ plane are equal to those above the plane by mirror symmetry in the $z = 0$ plane. This saves a factor of two in computational effort for Monte Carlo calculations, as well as in memory useage. Note that also the resulting output files such as `dust_temperature.dat` will only be specified for $z > 0$.

  - 3-D:

    All spatial functions depend on all three variables r, θ and ϕ.

  - 3-D with mirror symmetry:

    All spatial functions depend on all three variables r, θ and ϕ, but like in the 2-D case only the upper part of the model needs to be specified: the lower part is assumed to be a mirror copy.

  When using spherical coordinates, please read Section *Separable grid refinement in spherical coordinates (important!)*.

In all cases these structured grids allow for oct-tree-style grid refinement, or its simplified version: the layer-style grid refinement. See Section *INPUT (required): amr_grid.inp or unstr_grid.inp* and Chapter *More information about the gridding* for more information about the gridding and the (adaptive) mesh refinement (AMR).

5.5 Computations that RADMC-3D can perform

The code RADMC-3D (i.e. the executable `radmc3d`) is *one* code for *many* actions. Depending on which command-line arguments you give, RADMC-3D can do various actions. Here is a list:

1. Compute the dust temperature:

   With `radmc3d mctherm` you call RADMC-3D with the command of performing a thermal Monte Carlo simulation to compute the dust temperature under the assumption that the dust is in radiative equilibrium with its radiation field. This is normally a prerequisite for computing SEDs and images from dusty objects (see *computing spectra and images* below). The output file of this computation is `dust_temperature.dat` which contains the dust temperature everywhere in the model.

2. Compute a spectrum or SED:

   With `radmc3d sed` you call RADMC-3D with the command of performing a ray-tracing computation to compute the spectral energy distribution (SED) for the model at hand. Typically you first need to have called `radmc3d mctherm` (see above) beforehand to compute dust temperatures (unless you have created the file `dust_temperature.dat` yourself because you have a special way of computing the dust temperature). With `radmc3d sed` the spectrum is computed for the wavelengths points given in the file `wavelength_micron.inp`, which is the same wavelength grid that is used for `radmc3d mctherm`. If you want to compute the spectrum at wavelength other than those used for the thermal Monte Carlo simulation, you should instead call `radmc3d spectrum`, and you have the full freedom to choose the spectral wavelengths points at will, and you can specify these in various ways described in Section *Specifying custom-made sets of wavelength points for the camera*. Most easily you can create a file called `camera_wavelength_micron.inp` (see Section *INPUT (optional): camera_wavelength_micron.inp*) and call RADMC-3D using `radmc3d spectrum loadlambda`. In all these cases the vantage point (where is the observer) can of course be set as well, see Section *Making SEDs, spectra, images for dust continuum* and Chapter *Making images and spectra*.

3. Compute an image:

   With `radmc3d image` you call RADMC-3D with the command of performing a ray-tracing computation to compute an image. You must specify the wavelength(s) at which you want the image by, for instance, calling RADMC-3D as `radmc3d image lambda 10`, which makes the image at $\lambda = 10\mu$m. But there are other ways by which the wavelength(s) can be set, see Section *Specifying custom-made sets of wavelength points for the camera*. In all these cases the vantage point (where is the observer) can of course be set as well, see Section *Making SEDs, spectra, images for dust continuum* and Chapter *Making images and spectra*.

4. Compute the local radiation field inside the model:

   With `radmc3d mcmono` you call RADMC-3D with the command of performing a wavelength-by-wavlength monochromatic Monte Carlo simulation (at the wavelengths that you specify in the file `mcmono_wavelength_micron.inp`). The output file of this computation is `mean_intensity.out` which contains the mean intensity J_ν as a function of the (x, y, z) (cartesian) or (r, θ, ϕ) (spherical) coordinates at the frequencies $\nu_i \equiv 10^4 c/\lambda_i$ where λ_i are the wavelengths (in μm) specified in the file `mcmono_wavelength_micron.inp`. The results of this computation can be interesting for, for instance, models of photochemistry. But if you use RADMC-3D only for computing spectra and images, then you will not use this.

In addition to the above main methods, you can ask RADMC-3D to do various minor things as well, which will be described throughout this manual.

5.6 How a model is set up and computed: a rough overview

A radiative transfer code such as RADMC-3D has the task of computing synthetic images and spectra of a model that you specify. You tell the code what the dust and/or gas density distribution in 3-D space is and where the star(s) are, and the code will then tell you what your cloud looks like in images and/or spectra. That's basically it. That's the main task of RADMC-3D.

First you have to tell RADMC-3D what 3-D distribution of dust and/or gas you want it to model. For that you must specify a coordinate system (cartesian or spherical) and a spatial grid. For cartesian coordinates this grid should be 3-D (although there are exceptions to this), while for spherical coordinates it can be 1-D (spherical symmetry), 2-D (axial symmetry) or 3-D (no symmetry). RADMC-3D is (for most part) a cell-based code, i.e. your grid devides space in cells and you have to tell RADMC-3D what the average densities of dust and/or gas are in these cells.

The structure of the grid is specified in a file `amr_grid.inp` (see Section *INPUT (required): amr_grid.inp or unstr_grid.inp*). All the other data, such as dust density and/or gas density are specified in other files, but all assume that the grid is given by `amr_grid.inp`.

We can also specify the locations and properties of one or more stars in the model. This is done in the `stars.inp` (see Section *INPUT (mostly required): stars.inp*) file.

Now suppose we want to compute the appearance of our model in dust continuum. We will describe this in detail in Chapter *Dust continuum radiative transfer*, but let us give a very rough idea here. We write, in addition to the `amr_grid.inp` and `stars.inp` files, a file `dust_density.inp` which specifies the density of dust in each cell (see Section *INPUT (required for dust transfer): dust_density.inp*). We also must write the main input file `radmc3d.inp` (see Section *INPUT: radmc3d.inp*), but we can leave it empty for now. We must give RADMC-3D a dust opacity table in the files `dustopac.inp` and for instance `dustkappa_silicate.inp` (see Section *INPUT (required for dust transfer): dustopac.inp and dustkappa_*.inp or dustkapscatmat_*.inp or dust_optnk_*.inp*). And finally, we have to give RADMC-3D a table of discrete wavelengths in the file `wavelength_micron.inp` that it will use to perform its calculations on. We then call the `radmc3d` code with the keyword `mctherm` (see Chapter *Dust continuum radiative transfer*) to tell it to perform a Monte Carlo simulation to compute dust temperatures everywhere. RADMC-3D will write this to the file `dust_temperature.dat`. If we now want to make a spectral energy distribution, for instance, we call `radmc3d sed` (see Section *Making spectra*) and it will write a file called `spectrum.out` which is a list of fluxes at the discrete wavelengths we specified in `wavelength_micron.inp`. Then we are done: we have computed the spectral energy distribution of our model. We could also make an image at wavelength 10 μm for instance with `radmc3d image lambda 10` (see Section *Basics of image making with RADMC-3D*). This will write out a file `image.out` containing the image data (see Section *OUTPUT: image.out or image_****.out*).

As you see, RADMC-3D reads all its information from tables in various files. Since you don't want to make large tables by hand, you will have to write a little computer program that generates these tables automatically. You can do this in any programming language you want. But in the example models (see Section *Running the example models*) we use the programming language Python (see Section *Requirements*) for this. It is easiest to indeed have a look at the example models to see how this is (or better: can be) done.

We will explain all these things in much more detail below, and we will discuss also many other radiative transfer problem types. The above example is really just meant to give an impression of how RADMC-3D works.

5.7 Organization of model directories

The general philosophy of the RADMC-3D code package is the following. The core of everything is the fortran code `radmc3d`. This is the main code which does the hard work for you: it makes the radiative transfer calculations, makes images, makes spectra etc. Normally you compile this code just once-and-for-all (see Chapter *Installation of RADMC-3D*), and then simply use the executable `radmc3d` for all models. There is an exception to this 'once-and-for-all' rule described in Section *Making special-purpose modified versions of RADMC-3D (optional)*, but in the present chapter we will not use this (see Chapter *Modifying RADMC-3D: Internal setup and user-specified radiative processes* for this instead). So we will stick here to the philosophy of compiling this code once and using it for all models.

So how to set up a model? The trick is to present `radmc3d` with a set of input files in which the model is described in all its details. The procedure to follow is this:

1. The best thing to do (to avoid a mess) is to make a directory for *each model*: one model, one directory. Since `radmc3d` reads multiple input files, and also outputs a number of files, this is a good way to keep organized and we recommend it strongly. So if we wish to make a new model, we make a new directory, or copy an old directory to a new name (if we merely want to make small changes to a prior model).
2. In this directory we generate the input files according to their required format (see Chapter *Main input and output files of RADMC-3D*). You can create these input files in any way you want. But since many of these input files will/must contain huge lists of numbers (for instance, giving the density at each location in your model), you will typically want to write some script or program in some language (be it either C, C++, Fortran, IDL, GDL, perl, python, you name it) that automatically creates these input files. *We recommend using Python, because we provide examples and standard subroutines in the programming language Python; see below for more details.* Section *Running the example models* describes how to use the example Python scripts to make these input files with Python.
3. When all the input files are created, and we make sure that we are inside the model directory, we call `radmc3d` with the desired command-line options (see Chapter *Command-line options*). This will do the work for us.
4. Once this is done, we can analyze the results by reading the output files (see Chapter *Main input and output files of RADMC-3D*). To help you reading and analyzing these output files you can use a set of Python routines that we created for the user (see Chapter *Python analysis tool set* and Section *Installing the simple Python analysis tools*). But here again, you are free to use any other plotting software and/or data postprocessing packages.

5.8 Running the example models

Often the fastest and easiest way to learn a code is simply to analyze and run a set of example models. They are listed in the `examples` directory. Each model occupies a separate directory. This is also the style we normally recommend: each model should have its own directory. Of course there are also exceptions to this rule, and the user is free to organize her/his data in any way he/she pleases. But in all the examples and throughout this manual each model has its own directory.

To run an example model, go into the directory of this model, and follow the directions that are written in the `README` file in each of these directories. *This is under the assumption that you have a full Python distribution installed on your system, including Numpy and Matplotlib.*

Let us do for instance `run_simple_1/`:

```
cd examples/run_simple_1
```

Now we must create all the input files for this model. These input files are all described in chapter *Main input and output files of RADMC-3D*, but let us here just 'blindly' follow the example. In this example most (all except one) of the input files are created using a Python script called `problem_setup.py`. To execute this script, this is what you do on the shell:

```
python problem_setup.py
```

This Python script has now created a whole series of input files, all ending with the extension `.inp`. To see which files are created, type the following in the shell:

```
ls -l *.inp
```

There is one file that this example does not create, and that is the file `dustkappa_silicate.inp`. This is a file that contains the dust opacity in tabulated form. This is a file that you as the user should provide to the RADMC-3D code package. The file `dustkappa_silicate.inp` is merely an example, which is an amorphous spherical silicate grain with radius 0.1 micron. But see Section *INPUT (required for dust transfer): dustopac.inp and dustkappa_*.inp or dustkapscatmat_*.inp or dust_optnk_*.inp* for more information about the opacities.

Now that the input files are created, we must run `radmc3d`:

```
radmc3d mctherm
```

This tells RADMC-3D to do the thermal Monte Carlo simulation. This may take some time. When the model is ready, the prompt of the shell returns. To see what files have been created by this run of the code, type:

```
ls -l *.dat
```

You will find the `dust_temperature.dat` containing the dust temperature everywhere in the model. See again chapter *Main input and output files of RADMC-3D* for details of these files. To create a spectral energy distribution (SED):

```
radmc3d sed incl 45.
```

This will create a file `spectrum.out`. To analyze these data you can use the Python routines delivered with the code (see Chapter *Python analysis tool set* and Section *Installing the simple Python analysis tools*).

There is a file `Makefile` in the directory. This is here only meant to make it easy to clean the directory. Type `make cleanmodel` to clean all the output from the radmc3d code. Type `make cleanall` to clean the directory back to basics.

Let us now do for instance model `run_simple_1_userdef/`:

```
cd examples/run_simple_1_userdef
```

This is the same model as above, but now the grid and the dust density are set up *inside* `radmc3d`, using the file `userdef_module.f90` which is present in this directory. See Chapter *Modifying RADMC-3D: Internal setup and user-specified radiative processes* for details and follow the directions in the `README` file. In short: first edit the variable `SRC` in the `Makefile` to point to the `src/` directory. Then type `make`. Then type `python problem_setup.py` on the shell command line (which now only sets up the frequency grid, the star and the `radmc3d.inp` file and some small stuff). Now you can run the model.

Please read the README file in each of the example model directories. Everything is explained there, including how to make the relevant plots.

CHAPTER

SIX

DUST CONTINUUM RADIATIVE TRANSFER

Many of the things related to dust continuum radiative transfer have already been said in the previous chapters. But here we combine these things, and expand with more in-depth information.

Most users simply want RADMC-3D to compute images and spectra from a model. This is done in a two-stage procedure:

1. First compute the dust temperature everywhere using the thermal Monte Carlo computation (Section *The thermal Monte Carlo simulation: computing the dust temperature*).
2. Then making the images and/or spectra (Section *Making SEDs, spectra, images for dust continuum*).

You can then view the output spectra and images with the Python tools or use your own plotting software.

Some expert users may wish to use RADMC-3D for something entirely different: to compute the local radiation field {em inside} a model, and use this for e.g. computing photochemistry rates of a chemical model or so. This is described in Section *Special-purpose feature: Computing the local radiation field*.

You may also use the thermal Monte Carlo computation of the dust temperature to help estimating the {em gas} temperature for the line radiative transfer. See Chapter *Line radiative transfer* for more on line transfer.

6.1 The thermal Monte Carlo simulation: computing the dust temperature

RADMC-3D can compute the dust temperature using the Monte Carlo method of Bjorkman & Wood (2001, ApJ 554, 615) with various improvements such as the continuous absorption method of Lucy (1999, A&A 344, 282). Once a model is entirely set up, you can ask `radmc3d` to do the Monte Carlo run for you by typing in a shell:

```
radmc3d mctherm
```

if you use the standard `radmc3d` code, or

```
./radmc3d mctherm
```

if you have created a local version of `radmc3d` (see Section *Making special-purpose modified versions of RADMC-3D (optional)*).

What the method does is the following: First all the netto sources of energy (or more accurately: sources of luminosity) are identified. The following net sources of energy can be included:

- *Stars:* You can specify any number of individual stars: their position, and their spectrum and luminosity (See Section *INPUT (mostly required): stars.inp*). This is the most commonly used source of luminosity, and as a beginning user we recommend to use only this for now.

- *Continuum stellar source:* For simulations of galaxies it would require by far too many individual stars to properly include the input of stellar light from the billions of stars in the galaxy. To overcome this problem you can specify a continuously spatially distributed source of stars. *NOTE: Still in testing phase.*
- *Viscous heating / internal heating:* Sometimes the dust grains acquire energy directly from the gas, for instance through viscous heating of the gas or adiabatic compression of the gas. This can be included as a spatially distributed source of energy. *NOTE: Still in progress... Not yet working.*

To compute the dust temperature we must have at least one source of luminosity, otherwise the equilibrium dust temperature would be everywhere 0.

The next step is that this total luminosity is divided into `nphot` packages, where `nphot` is 100000 by default, but can be set to any value by the user (see the file `radmc3d.inp` described in Section *INPUT: radmc3d.inp*). Then these photon packages are emitted by these sources one-by-one. As they move through the grid they may scatter off dust grains and thus change their direction. They may also get absorbed by the dust. If that happens, the photon package is immediately re-emitted in another direction and with another wavelength. The wavelength is chosen according to the recipe by Bjorkman & Wood (2001, ApJ 554, 615). The luminosity fraction that each photon package represents remains, however, the same. Each time a photon package enters a cell it increases the 'energy' of this cell and thus increases the temperature of the dust of this cell. The recipe for this is again described by Bjorkman & Wood (2001, ApJ 554, 615), but contrary to that paper we increase the temperature of the dust always when a photon package enters a cell, while Bjorkman & Wood only increase the dust temperature if a discrete absorption event has taken place. Each photon package will ping-pong through the model and never gets lost until it escapes the model through the outer edge of the grid (which, for cartesianl coordinates, is any of the grid edges in x, y or z, and for spherical coordinates is the outer edge of r). Once it escapes, a new photon package is launched, until also it escapes. After all photon packages have been launched and escaped, the dust temperature that remains is the final answer of the dust temperature.

One must keep in mind that the temperature thus computed is an *equilibrium* dust temperature. It assumes that each dust grain acquires as much energy as it radiates away. This is for most cases presumably a very good approximation, because the heating/cooling time scales for dust grains are typically very short compared to any time-dependent dynamics of the system. But there might be situations where this may not be true: in case of rapid compression of gas, near shock waves or in extremely optically thick regions.

NOTE: Monte Carlo simulations are based on pseudo-random numbers. The seed for the random number generator is by default set to -17933201. If you want to perform multiple identical simulations with a different random sequence you will need to set the seed by hand. This can be done by adding a line

```
iseed = -5415
```

(where -5415 is to be replaced by the value you want) to the `radmc3d.inp` file.

6.1.1 Modified Random Walk method for high optical depths

As you will soon find out: very optically thick models make the RADMC-3D thermal Monte Carlo simulations to be slow. This is because in the thermal Monte Carlo method a photon package is never destroyed unless it leaves the system. A photon package can thus 'get lost' deep inside an optically thick region, making millions (or even billions) of absorption+reemission or scattering events. Furthermore, you will notice that in order to get the temperatures in these very optically thick regions to be reliable (i.e. not too noisy) you may need a very large number of photon packages for your simulation, which slows down the simulation even more. It is hard to prevent such problems. Min, Dullemond, Dominik, de Koter & Hovenier (2009) A&A 497, 155 discuss two methods of dealing with this problem. One is a diffusion method, which we will not discuss here. The other is the 'Modified Random Walk' (MRW) method, based on the method by Fleck & Canfield (1984) J.Comput.Phys. 54, 508. Note that Robitaille (2010) A&A 520, 70 presented a simplification of this method. Min et al. first implemented this method into the MCMax code. It is also implemented in RADMC-3D, in Robitaille's simplified form.

The crucial idea of the method is that if a photon package 'gets lost' deep inside a single ultra-optically-thick cell, we can use the analytical solutions of the diffusion equation in a constant-density medium to predict where the photon

package will go next. This thus allows RADMC-3D to make a single large step of the photon package which actually corresponds to hundreds or thousands of absorption+reemission or scattering events.

The method works best if the optically thick cells are as large as possible. This is because the analytical solutions are only valid within a single cell, and thus the 'large step' can not be larger than a single cell size. Moreover, cell crossings will reduce the step length again to the physical mean free path, so the more cell crossings are made, the less effective the MRW becomes.

NOTE: The MRW is by default switched off. The reason is that it is, after all, an approximation. However, if RADMC-3D thinks that the MRW may help speed up the thermal Monte Carlo, it will make the suggestion to the user to switch on the MRW method.

NOTE: So far the MRW method is only implemented using the Planck mean opacity for estimating the 'large step'. This could, under certain conditions, be inaccurate. The reason why the (more accurate) Rosseland mean opacity is not used is that this precludes the precomputation and tabulation of the mean opacities if multiple independent dust species are used. Strictly speaking, even the Rosseland mean opacity is not entirely correct, but it is a good approximation (see Min et al. 2009). So far these simplifications do not seem to matter a lot. But if strong effects are seen, please report these. Conditions under which it is likely to make a difference (i.e. the present implementation becoming inaccurate) are when an internal heat source inside a super-optically thick region is introduced (e.g. viscous heating in a disk), and/or when the opacities are extremely wavelength-dependent (varying by orders of magnitude in small distances in wavelengths). So please use MRW with care. Upon request we may implement the true MRW: with the Rosseland mean, which, however, may make the code slower.

You can switch on the MRW by adding the following line to the `radmc3d.inp` file:

```
modified_random_walk = 1
```

6.2 Making SEDs, spectra, images for dust continuum

You can use RADMC-3D for computing spectra and images in dust continuum emission. This is described in detail in Chapter *Making images and spectra*. RADMC-3D needs to know not only the dust spatial distribution, given in the file `dust_density.inp`, but also the dust temperature, given in the file `dust_temperature.dat` (see Chapter *Binary I/O files* for the binary version of these files, which are more compact, and which you can use instead of the ascii versions). The `dust_temperature.dat` is normally computed by RADMC-3D itself through the thermal Monte Carlo computation (see Section *The thermal Monte Carlo simulation: computing the dust temperature*). But if you, the user, wants to specify the dust temperature at each location in the model youself, then you can simply create your own file `dust_temperature.dat` and skip the thermal Monte Carlo simulation and go straight to the creation of images or spectra.

The basic command to make a spectrum at the global grid of wavelength (specified in the file `wavelength_micron.inp`, see Section *INPUT (required): wavelength_micron.inp*) is:

```
radmc3d sed
```

You can specify the direction of the observer with `incl` and `phi`:

```
radmc3d sed incl 20 phi 80
```

which means: put the observer at inclination 20 degrees and ϕ-angle 80 degrees.

You can also make a spectrum for a given grid of wavelength (independent of the global wavelength grid). You first create a file `camera_wavelength_micron.inp`, which has the same format as `wavelength_micron.inp`. You can put any set of wavelengths in this file without modifying the global wavelength grid (which is used by the thermal Monte Carlo computation). Then you type

```
radmc3d spectrum loadlambda
```

and it will create the spectrum on this wavelength grid. More information about making spectra is given in Chapter *Making images and spectra*.

For creating an image you can type

```
radmc3d image lambda 10
```

which creates an image at wavelength λmu`m. More information about making images is given in Chapter *Making images and spectra*.

Important note: To handle scattering of light off dust grains, the ray-tracing is preceded by a quick Monte Carlo run that is specially designed to compute the 'scattering source function'. This Monte Carlo run is usually *much* faster than the thermal Monte Carlo run, but must be done at each wavelength. It can lead, however, to slight spectral noise, because the random photon paths are different for each wavelength. See Section *More about scattering of photons off dust grains* for details.

6.3 OpenMP parallelized Monte Carlo

Depending on the model properties and the number of photon packages used in the simulation the Monte Carlo calculation (in particular the thermal Monte Carlo, but under some conditions also the scattering Monte Carlo) can be a time-consuming computation when executed only in a serial mode. To improve this, these Monte Carlo calculations can be done in OpenMP parallel mode. The loop over photon packages is then distributed amongst the different threads, where each thread adopts a specific number of loop iterations following the order of the thread identification number. To this end the random number generator was modified. The important point for the parallel version is that different threads must not share the same random seed initially. To be certain that each thread is assigned a different seed at the beginning, the thread identity number is added to the initial seed.

The default value for the number of threads in the parallel version is set to one, so that the program is identical with the serial version, except for the random generator's initial seed. The user can change the value by either typing `setthreads <nr>`, where `<nr>` is the number of requested threads (integer value) in the command line or by adding a corresponding line to the `radmc3d.inp` file. If the chosen number of threads is larger than the available number of processor cores, the user is asked to reduce it.

For example, you can ask `radmc3d` to do the parallelized Monte Carlo run for you by typing in a shell:

```
radmc3d mctherm setthreads 4
```

or by adding the following keyword to the `radmc3d.inp` file:

```
setthreads = 4
```

which means that four threads are used for the thermal Monte Carlo computation.

For the image or spectrum you can do the same: just add `setthreads 4` or so on the command line or put `setthreads = 4` into the `radmc3d.inp` file.

Make sure that you have included the `-fopenmp` keyword in the `Makefile` and have compiled the whole `radmc3d` source code with this additional command before using the OpenMP parallelized thermal Monte Carlo version (cf. Section *Compiling the code with 'make'*).

6.4 Overview of input data for dust radiative transfer

In order to perform any of the actions described in Sections *The thermal Monte Carlo simulation: computing the dust temperature*, *Special-purpose feature: Computing the local radiation field* or *Making SEDs, spectra, images for dust continuum*, you must give RADMC-3D the following data:

- `amr_grid.inp`: The grid file (see Section *INPUT (required): amr_grid.inp or unstr_grid.inp*).
- `wavelength_micron.inp`: The global wavelength file (see Section *INPUT (required): wavelength_micron.inp*).
- `stars.inp`: The locations and properties of stars (see Section *INPUT (mostly required): stars.inp*).
- `dust_density.inp`: The spatial distribution of dust on the grid (see Section *INPUT (required for dust transfer): dust_density.inp*).
- `dustopac.inp`: A file with overall information about the various species of dust in the model (see Section *INPUT (required for dust transfer): dustopac.inp and dustkappa_*.inp or dustkapscatmat_*.inp or dust_optnk_*.inp*). One of the main pieces of information here is (a) how many dust species are included in the model and (b) the tag names of these dust species (see `dustkappa_XXX.inp` below). The file `dust_density.inp` must contain exactly this number of density distributions: one density distribution for each dust species.
- `dustkappa_XXX.inp`: One or more dust opacity files (where `XXX` should in fact be a tag name you define, for instance `dustkappa_silicate.inp`). The labels are listed in the `dustopac.inp` file. See Section *INPUT (required for dust transfer): dustopac.inp and dustkappa_*.inp or dustkapscatmat_*.inp or dust_optnk_*.inp* for more information.
- `camera_wavelength_micron.inp (optional)`: This file is only needed if you want to create a spectrum at a special set of wavelengths (otherwise use `radmc3d sed`).
- `mcmono_wavelength_micron.inp (optional)`: This file is only needed if you want to compute the radiation field inside the model by calling `radmc3d mcmono` (e.g. for photochemistry).

Other input files could be required in certain cases, but you will then be asked about it by RADMC-3D.

6.5 Special-purpose feature: Computing the local radiation field

If you wish to use RADMC-3D for computing the radiation field *inside* the model, for instance for computing photochemical rates in a chemical model, then RADMC-3D can do so by calling RADMC-3D in the following way:

```
radmc3d mcmono
```

This computes the mean intensity

$$J_\nu = \frac{1}{4\pi} \oint I_\nu(\Omega) d\Omega$$

(in units of $\mathrm{erg\,s^{-1}\,cm^{-2}\,Hz^{-1}\,ster^{-1}}$) as a function of the (x, y, z) (cartesian) or (r, θ, ϕ) (spherical) coordinates at frequencies $\nu_i \equiv 10^4 c/\lambda_i$ where λ_i are the wavelengths (in μm) specified in the file `mcmono_wavelength_micron.inp` (same format as the file `wavelength_micron.inp` which is described in Section *INPUT (required): wavelength_micron.inp*). The results of this computation can be interesting for, for instance, models of photochemistry.

The file that is produced by `radmc3d mcmono` is called `mean_intensity.out` and has the following form:

```
iformat                                  <=== Typically 2 at present
nrcells
nfreq                                    <=== Nr of frequencies
freq_1 freq_2 ... freq_nfreq             <=== List of frequencies in Hz
meanint[1,icell=1]
meanint[1,icell=2]
...
meanint[1,icell=nrcells]
meanint[2,icell=1]
meanint[2,icell=2]
...
meanint[2,icell=nrcells]
...
...
...
meanint[nfreq,icell=1]
meanint[nfreq,icell=2]
...
meanint[nfreq,icell=nrcells]
```

The list of frequencies will, in fact, be the same as those listed in the file `mcmono_wavelength_micron.inp`.

Note that if your model is very large, the computation of the radiation field on a large set of wavelength could easily overload the memory of the computer. However, often you are in the end not interested in the entire spectrum at each location, but just in integrals of this spectrum over some cross section. For instance, if you want to compute the degree to which dust shields molecular photodissociation lines in the UV, then you only need to compute the total photodissociation rate, which is an integral of the photodissociation cross section times the radiation field. In Section *Using the userdef module to compute integrals of J_\nu* it will be explained how you can create a userdef subroutine (see Chapter *Modifying RADMC-3D: Internal setup and user-specified radiative processes*) that will do this for you in a memory-saving way.

There is an important parameter for this Monochromatic Monte Carlo that you may wish to play with:

- `nphot_mono` The parameter `nphot_mono` sets the number of photon packages that are used for the Monochromatic Monte Carlo simulation. It has as default 100000, but that may be too little for 3-D models. You can set this value in two ways:
  - In the `radmc3d.inp` file as a line `nphot_mono = 1000000` for instance.
  - On the command-line by adding `nphot_mono 1000000`.

6.6 More about scattering of photons off dust grains

Photons can not only be absorbed and re-emitted by dust grains: They can also be scattered. Scattering does nothing else than change the direction of propagation of a photon, and in case polarization is included, its Stokes parameters. Strictly speaking it may also slightly change its wavelength, if the dust grains move with considerable speed they may Doppler-shift the wavelength of the outgoing photon (which may be relevant, if at all, when dust radiative transfer is combined with line radiative transfer, see chapter *Line radiative transfer*), but this subtle effect is not treated in RADMC-3D. For RADMC-3D scattering is just the changing of direction of a photon.

6.6.1 Five modes of treating scattering

RADMC-3D has five levels of realism of treatment of scattering, starting with `scattering_mode=1` (simplest) to `scattering_mode=5` (most realistic):

- *No scattering* (`scattering_mode=0`):

  If either the `dustkappa_XXX.inp` files do not contain a scattering opacity or scattering is switched off by setting `scattering_mode_max` to 0 in the `radmc3d.inp` file, then scattering is ignored. It is then assumed that the dust grains have zero albedo.

- *Isotropic scattering* (`scattering_mode=1`):

  If either the `dustkappa_XXX.inp` files do not contain information about the anisotropy of the scattering or anisotropic scattering is switched off by setting `scattering_mode_max` to 1 in the `radmc3d.inp` file, then scattering is treated as isotropic scattering. Note that this can be a bad approximation.

- *Anisotropic scattering using Henyey-Greenstein* (`scattering_mode=2`):

  If the `dustkappa_XXX.inp` files contain the scattering opacity and the g parameter of anisotropy (the Henyey-Greenstein g parameter which is equal, by definition, to $g = \langle \cos\theta \rangle$, where θ is the scattering deflection angle), and `scattering_mode_max` is set to 2 or higher in the `radmc3d.inp` file then anisotropic scattering is treated using the Henyey-Greenstein approximate formula.

- *Anisotropic scattering using tabulated phase function* (`scattering_mode=3`):

  To treat scattering using a tabulated phase function, you must specify the dust opacities using `dustkapscatmat_XXX.inp` files instead of the simpler `dustkappa_XXX.inp` files (see Section *The dustkapscatmat_*.inp files*). You must also set `scattering_mode_max` is set to 3 or higher.

- *Anisotropic scattering with polarization for last scattering* (`scattering_mode=4`):

  To treat scattering off randomly oriented particles with the full polarization you need to set `scattering_mode_max` is set to 4 or higher, and you must specify the full dust opacity and scattering matrix using the `dustkapscatmat_XXX.inp` files instead of the simpler `dustkappa_XXX.inp` files (see Section *The dustkapscatmat_*.inp files*). If `scattering_mode=4` the full polarization is only done upon the last scattering before light reaches the observer (i.e. it is only treated in the computation of the scattering source function that is used for the images, but it is not used for the movement of the photons in the Monte Carlo simulation). See Section *Polarization, Stokes vectors and full phase-functions* for more information about polarized scattering.

- *Anisotropic scattering with polarization, full treatment* (`scattering_mode=5`):

  For the full treatment of polarized scattering off randomly oriented particles, you need to set `scattering_mode_max` is set to 5, and you must specify the full dust opacity and scattering matrix using the `dustkapscatmat_XXX.inp` files instead of the simpler `dustkappa_XXX.inp` files (see Section *The dustkapscatmat_*.inp files*). See Section *Polarization, Stokes vectors and full phase-functions* for more information about polarized scattering. end{enumerate} Please refer to Sections *Scattering phase functions* and *Polarization, Stokes vectors and full phase-functions* for more information about these different scattering modes.

So in summary: the dust opacity files themselves tell how detailed the scattering is going to be included. If no scattering information is present in these files, RADMC-3D has no choice but to ignore scattering. If they only contain scattering opacities but no phase information (no g-factor), then RADMC-3D will treat scattering in the isotropic approximation. If the g-factor is also included, then RADMC-3D will use the Henyey-Greenstein formula for anisotropic scattering. If you specify the full scattering matrix (using the `dustkapscatmat_XXX.inp` files instead of the `dustkappa_XXX.inp` files) then you can use tabulated scattering phase functions, and even polarized scattering.

If `scattering_mode_max` is *not* set in the `radmc3d.inp` file, it is by default 9999, meaning: RADMC-3D will always use the maximally realistic scattering mode that the dust opacities allow.

BUT you can always limit the realism of scattering by setting the `scattering_mode_max` to 4, 3, 3, 1 or 0 in the file `radmc3d.inp`. This can be useful to speed up the calculations or be sure to avoid certain complexities of the full phase-function treatment of scattering.

At the moment there are some limitations to the full anisotropic scattering treatment:

- *Anisotropic scattering in 1-D and 2-D Spherical coordinates:*

  For 1-D spherical coordinates there is currently no possibility of treating anisotropic scattering in the image- and spectrum-making. The reason is that the scattering source function (see Section *Scattered light in images and spectra: The 'Scattering Monte Carlo' computation*) must be stored in an angle-dependent way. However, for 2-D spherical coordinates, this has been implemented, and for each grid 'cell' (actually an annulus) the scattering source function is now stored for an entire sequence of angles.

- *Full phase functions and polarization only for randomly-oriented particles:*

  Currently RADMC-3D cannot handle scattering off fixed-oriented non-spherical particles, because it requires a much more detailed handling of the angles. It would require at least 3 scattering angles (for axially-symmetric particles) or more (for completely asymmetric particles), which is currently beyond the scope of RADMC-3D.

6.6.2 Scattering phase functions

As mentioned above, for the different `scattering_mode` settings you have different levels of realism of treating scattering.

The transfer equation along each ray, ignoring polarization for now, is:

$$\frac{dI_\nu}{ds} = j_\nu^{\rm therm} + j_\nu^{\rm scat} - (\alpha_\nu^{\rm abs} + \alpha_\nu^{\rm scat}) I_\nu$$

where $\alpha_\nu^{\rm abs}$ and $\alpha_\nu^{\rm scat}$ are the extinction coefficients for absorption and scattering. Let us assume, for convenience of notation, that we have just one dust species with density dstribution ρ, absorption opacity $\kappa_\nu^{\rm abs}$ and scattering opacity $\kappa_\nu^{\rm scat}$. We then have

$$\begin{aligned}
\alpha_\nu^{\rm abs} &\equiv \rho \kappa_\nu^{\rm abs} \\
\alpha_\nu^{\rm scat} &\equiv \rho \kappa_\nu^{\rm scat} \\
j_\nu^{\rm therm} &= \alpha_\nu^{\rm abs} B_\nu(T)
\end{aligned}$$

where $B_\nu(T)$ is the Planck function. The last equation is an expression of Kirchhoff's law.

For *isotropic* scattering (`scattering_mode=1`) the scattering source function $j_\nu^{\rm scat}$ is given by

$$j_\nu^{\rm scat} = \alpha_\nu^{\rm scat} \frac{1}{4\pi} \oint I_\nu d\Omega$$

where the integral is the integral over solid angle. In this case $j_\nu^{\rm scat}$ does not depend on solid angle.

For *anisotropic* scattering (`scattering_mode>1`) we must introduce the scattering phase function $\Phi(\mathbf{n}_{\rm in}, \mathbf{n}_{\rm out})$, where $\mathbf{n}_{\rm in}$ is the unit direction vector for incoming radiation and $\mathbf{n}_{\rm out}$ is the unit direction vector for the scattered radiation. The scattering phase function is normalized to unity:

$$\frac{1}{4\pi} \oint \Phi(\mathbf{n}_{\rm in}, \mathbf{n}_{\rm out}) d\Omega_{\rm out} = \frac{1}{4\pi} \oint \Phi(\mathbf{n}_{\rm in}, \mathbf{n}_{\rm out}) d\Omega_{\rm in} = 1$$

where we integrated over all possible $\mathbf{n}_{\rm out}$ or $\mathbf{n}_{\rm in}$. Then the scattering source function becomes:

$$j_\nu^{\rm scat}(\mathbf{n}_{\rm out}) = \alpha_\nu^{\rm scat} \frac{1}{4\pi} \oint I_\nu(\mathbf{n}_{\rm in}) \Phi(\mathbf{n}_{\rm in}, \mathbf{n}_{\rm out}) d\Omega_{\rm in}$$

which is angle-dependent. The angular dependence means: a photon package has not completely forgotten from which direction it came before hitting the dust grain.

If we do not include the polarization of radiation and we have randomly oriented particles, then the scattering phase function will only depend on the scattering (deflection) angle θ defined by

$$\cos\theta \equiv \mu = \mathbf{n}_{\rm out} \cdot \mathbf{n}_{\rm in}$$

We will thus be able to write

$$\Phi(\mathbf{n}_{\rm in}, \mathbf{n}_{\rm out}) \equiv \Phi(\mu)$$

where $\Phi(\mu)$ is normalized as

$$\frac{1}{2}\int_{-1}^{+1} \Phi(\mu) d\mu = 1$$

If we have `scattering_mode=2` then the phase function is the Henyey-Greenstein phase function defined as

$$\Phi(\mu) = \frac{1-g^2}{(1+g^2-2g\mu)^{3/2}}$$

where the value of the anisotropy parameter g is taken from the dust opacity file. Note that for $g = 0$ you get $\Phi(\mu) = 1$ which is the phase function for isotropic scattering.

If we have `scattering_mode=3` then the phase function is tabulated by you. You have to provide the tabulated phase function as the $Z_{11}(\theta)$ scattering matrix element for a tabulated set of θ_i values, and this is done in a file `dustkapscatmat_xxx.inp` (see Section *The dustkapscatmat_*.inp files* and note that for `scattering_mode=3` the other Z_{ij} elements can be kept 0 as they are of no consequence). The relation between $Z_{11}(\theta)$ and $\Phi(\mu)$ is:

$$\Phi(\mu) \equiv \Phi(\cos(\theta)) = \frac{4\pi}{\kappa_{\rm scat}} Z_{11}(\theta)$$

(which holds at each wavelength individually).

If we have `scattering_mode=4` then the scattering in the Monte Carlo code is done according to the tabulated $\Phi(\mu)$ mode mentioned above, but for computing the scattering source function the full polarized scattering matrix is used. See Section *Polarization, Stokes vectors and full phase-functions*.

If we have `scattering_mode=5` then the scattering phase function is not only dependent on μ but also on the other angle. And it depends on the polarization state of the input radiation. See Section *Polarization, Stokes vectors and full phase-functions*.

6.7 Scattering of photons in the Thermal Monte Carlo run

So how is scattering treated in practice? In the thermal Monte Carlo model (Section *The thermal Monte Carlo simulation: computing the dust temperature*) the scattering has only one effect: it changes the direction of propagation of the photon packages whenever such a photon package experiences a scattering event. This may change the results for the dust temperatures subtly. In special cases it may even change the dust temperatures more strongly, for instance if scattering allows 'hot' photons to reach regions that would have otherwise been in the shadow. It may also increase the optical depth of an object and thus change the temperatures accordingly. But this is all there is to it.

If you include the full treatment of polarized scattering (`scattering_mode=5`), then a photon package also gets polarized when it undergoes a scattering event. This can affect the phase function for the next scattering event. This means that the inclusion of the full polarized scattering processes (as opposed to using non-polarized photon packages) can, at least in principle, have an effect on the dust temperatures that result from the thermal Monte Carlo computation. This effect is, however, rather small in practice.

6.8 Scattering of photons in the Monochromatic Monte Carlo run

For the monochromatic Monte Carlo calculation for computing the mean intensity radiation field (Section *Special-purpose feature: Computing the local radiation field*) the scattering has the same effect as for the thermal Monte Carlo model: it changes the direction of photon packages. In this way 'hot' radiation may enter regions which would otherwise have been in a shadow. And by increasing the optical depth of regions, it may increase the local radiation field by the greenhouse effect or decrease it by preventing photons from entering it. As in the thermal Monte Carlo model the effect of scattering in the monochromatic Monte Carlo model is simply to change the direction of motion of the radiation field, but for the rest nothing differs to the case without scattering. Also here the small effects caused by polarized scattering apply, like in the thermal Monte Carlo case.

6.8.1 Scattered light in images and spectra: The 'Scattering Monte Carlo' computation

For making images and spectra with the ray-tracing capabilities of RADMC-3D (see Section *Making SEDs, spectra, images for dust continuum* and Chapter *Making images and spectra*) the role of scattering is a much more complex one than in the thermal and monochromatic Monte Carlo runs. The reason is that the scattered radiation will eventually end up on your images and spectra.

If we want to make an image or a spectrum, then for each pixel we must integrate Eq. (`eq-ray-tracing-rt`) along the 1-D ray belonging to that pixel. If we performed the thermal Monte Carlo simulation beforehand (or if we specified the dust temperatures by hand) we know the thermal source function through Eq. (eq-thermal-source-function). But we have, at that point, no information yet about the scattering source function. The thermal Monte Carlo calculation {em could} have also stored this function at each spatial point and each wavelength and each observer direction, but that would require gigantic amounts of memory (for a typical 3-D model it might be many Gbytes, going into the Tbyte regime). So in RADMC-3D the scattering source function is {em not} computed during the thermal Monte Carlo run.

In RADMC-3D the scattering source function $j_\nu^{\rm scat}(\Omega')$ is computed {em just prior to} the ray-tracing through a brief 'Scattering Monte Carlo' run. This is done {em automatically} by RADMC-3D, so you don't have to worry about this. Whenever you ask RADMC-3D to make an image (and if the scattering is in fact included in the model, see Section *Five modes of treating scattering*), RADMC-3D will automatically realize that it requires knowledge of $j_\nu^{\rm scat}(\Omega')$, and it will start a brief single-wavelength Monte Carlo simulation for computing $j_\nu^{\rm scat}(\Omega')$. This single-wavelength 'Scattering Monte Carlo' simulation is relatively fast compared to the thermal Monte Carlo simulation, because photon packages can be destroyed by absorption. So photon packages do not bounce around for long, as they do in the thermal Monte Carlo simulation. This Scattering Monte Carlo simulation is in fact very similar to the monochromatic Monte Carlo model described in Section *Special-purpose feature: Computing the local radiation field*. While the monochromatic Monte Carlo model is called specifically by the user (by calling RADMC-3D with `radmc3d mcmono`), the Scattering Monte Carlo simulation is not something the user must specify him/her-self: it is automatically done by RADMC-3D if it is needed (which is typically before making an image or during the making of a spectrum). And while the monochromatic Monte Carlo model returns the mean intensity inside the model, the Scattering Monte Carlo simulation provides the raytracing routines with the scattering source function but does *not* store this function in a file.

You can see this happen if you have a model with scattering opacity included, and you make an image with RADMC-3D, you see that it prints `1000`, `2000`, `3000`, ... etc., in other words, it performs a little Monte Carlo simulation before making the image.

There is an important parameter for this Scattering Monte Carlo that you may wish to play with:

- `nphot_scat`

  The parameter `nphot_scat` sets the number of photon packages that are used for the Scattering Monte Carlo simulation. It has as default 100000, but that may be too little for 3-D models and/or cases where you wish

to reduce the 'streaky' features sometimes visible in scattered-light images when too few photon packages are used. You can set this value in two ways:

  - In the `radmc3d.inp` file as a line `nphot_scat = 1000000` for instance.
  - On the command-line by adding `nphot_scat 1000000`.

In Figure Fig. 6.1 you can see how the quality of an image in scattered light improves when increasing `nphot_scat`.

- `nphot_spec`

  The parameter `nphot_spec` is actually exactly the same as `nphot_scat`, but is used (and used only!) for the creation of spectra. The default is 10000, i.e. substantially smaller than `nphot_scat`. The reason for this separate parameter is that if you make spectra, you integrate over the image to obtain the flux (i.e. the value of the spectrum at that wavelength). Even if the scattered light image may look streaky, the integral may still be accurate. We can thus afford much fewer photon packages when we make spectra than when we make images, and can thus speed up the calculation of the spectrum. You can set this value in two ways:

  - In the `radmc3d.inp` file as a line `nphot_spec = 100000` for instance.
  - On the command-line by adding `nphot_spec 100000`.

  NOTE: It may be possible to get still very good results with even smaller values of `nphot_spec` than the default value of 10000. That might speed up the calculation of the spectrum even more in some cases. On the other hand, if you notice 'noise' on your spectrum, you may want to increase `nphot_spec`. If you are interested in an optimal balance between accuracy (high value of `nphot_spec`) and speed of calculation (low value of `nphot_spec`) then it is recommended to experiment with this value. If you want to be on the safe side, then set `nphot_spec` to a high value (i.e. set it to 100000, as `nphot_spec`).

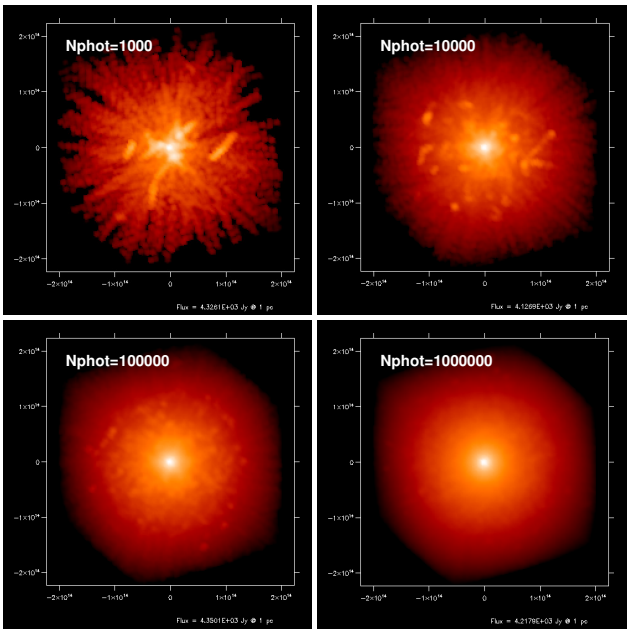

Fig. 6.1: The effect of `nphot_scat` on the image quality when the image is dominated by scattered light. The images show the result of model `examples/run_simple_2_scatmat` at $\lambda = 0.84\mu$m in which polarized scattering with the full scattering phase function and scattering matrix is used. See Section *Polarization, Stokes vectors and full phase-functions* about the scattering matrices for polarized scattering. See Section *Single-scattering vs. multiple-scattering* for a discussion about the 'scratches' seen in the top two panels.

WARNING: At wavelengths where the dominant source of photons is thermal dust emission but scattering is still important (high albedo), it cannot be excluded that the 'scattering monte carlo' method used by RADMC-3D produces

very large noise. Example: a very optically thick dust disk consisting of large grains (10 μm size), producing thermal dust emission in the near infrared in its inner disk regions. This thermal radiation can scatter off the large dust grains at large radii (where the disk is cold and where the only 'emission' in the near-infrared is thus the scattered light) and thus reveal the outer disk in scattered light emerging from the inner disk. However, unless `nphot_scat` is huge, most thermally emitted photons from the inner disk will be emitted so deeply in the disk interior (i.e. below the surface) that they will be immediately reabsorbed and lost. This means that that radiation that does escape is extremely noisy. The corresponding scattered light source function at large radii is therefore very noisy as well, unless `nphot_scat` is taken to be huge. Currently no elegant solution is found, but maybe there will in the future. Stay tuned...

NOTE: Monte Carlo simulations are based on pseudo-random numbers. The seed for the random number generator is by default set to -17933201. If you want to perform multiple identical simulations with a different random sequence you will need to set the seed by hand. This can be done by adding a line

```
iseed = -5415
```

(where -5415 is to be replaced by the value you want) to the `radmc3d.inp` file.

6.8.2 Single-scattering vs. multiple-scattering

If scattering is included in the images and spectra, the Monte Carlo run computes the full multiple-scattering problem. Photon packages are followed as they scatter and change their direction (possibly many times) until they escape to infinity or until they are extincted by many orders of magnitude (the exact extinction limit can be set by `mc_scat_maxtauabs`, which by default is set to 30, meaning a photon package is considered extincted when it has travelled an absorption optical depth of 30).

Important note: *In many (most?) cases this default value of* `mc_scat_maxtauabs=30` *is overly conservative. Especially when the scattering Monte Carlo is very time-consuming, you may want to experiment with a lower value. Try adding a line to the* `radmc3d.inp` *with*:

```
mc_scat_maxtauabs = 5
```

This may speed up the scattering Monte Carlo by up to a factor of 6, while still yielding reasonable results.

It can be useful to figure out how important the effect of multiple scattering in an image is compared to single scattering. For instance: a protoplanetary disk with a 'self-shadowed' geometry will show some scattering even in the shadowed region because some photon packages scatter {em into} the shadowed region and then scatter into the line of sight. To figure out if this is indeed what happens, you can make two images: one normal image with

```
radmc3d image lambda 1.0
cp image.out image_fullscat.out
```

and then another image which only treats single scattering:

```
radmc3d image lambda 1.0 maxnrscat 1
cp image.out image_singlescat.out
```

The command-line option `maxnrscat 1` tells RADMC-3D to stop following photon packages once they hit their first discrete scattering event. You can also check out the effect of single- and double-scattering (but excluding triple and higher order scattering) with: `maxnrscat 2`, etc.

Note that multiple scattering may require a very high number of photon packages (i.e. setting `nphot_scat` to a very high number). For single scattering with too low `nphot_scat` you typically see radial 'rays' in the image emanating from each stellar source of photons. For multiple scattering, when taking too low `nphot_scat` small you would see strange non-radial 'scratches' in the image (see Fig. Fig. 6.1, top two images). It looks as if someone has used a pen and randomly added some streaks. These streaks are the double-scattering events which, in that case, apparently are rare enough that they show up as individual streaks. To test whether these streaks are indeed such double scattering

events, you can use `maxnrscat 1`, and they should disappear. If the streaks are indeed very few, it may turn out that the single-scattering image (`maxnrscat 1`) is almost already the correct image. The double scattering is then only a minor addition to the image, but due to the finite Monte Carlo noise it would yield annoying streaks which ruin a nice image. If you are {em very sure} that the second scattering and higher-order scattering are only a very minor effect, then you might use the `maxnrscat 1` image as the final image. By comparing the flux in the images with full scattering and single scattering you can estimate how important the multiple-scattering contribution is compared to single scattering. But of course, it is always safer to simply increase `nphot_scat` and patiently wait until the Monte Carlo run is finished.

Tip: Analyzing the effect of multiple scattering using `selectscat`

If you are interested in analyzing the role that multiple scattering plays in your image (as opposed to single scattering), you can ask RADMC-3D to compute the scattering source function only for, for instance, the second scattering:

```
radmc3d image lambda 1.0 selectscat 2 2
```

Note that this should not be used for production runs, because the image that is produced is unphysical (it omits the first and third, fourth etc scatterings). But it can be useful to get a feeling for how important multiple scattering is, or to investigate if certain features in your image are due to multiple scattering or not. You can also select only the first scattering:

```
radmc3d image lambda 1.0 selectscat 1 1
```

or all scatterings except the first:

```
radmc3d image lambda 1.0 selectscat 2 100000
```

Note that if you make an image with `selectscat`, the thermal emission from the dust is still included (as is the case without `selectscat`). So if you want to see *only* the scattered light in the image, you need to manually set the dust temperature to zero everywhere (in the file `dust_temperature.dat`, see Chapter *Main input and output files of RADMC-3D*).

Also note that you can use `selectscat` also for the monochromatic Monte Carlo for computing the mean intensity field (see Section *Special-purpose feature: Computing the local radiation field*).

6.8.3 Simplified single-scattering mode (spherical coordinates)

If you are sure that multiple scattering is rare (low albedo and/or low optical depth), then you may be interested in using a simpler (non-Monte-Carlo) mode for including scattering in your images. But please first read Section *Single-scattering vs. multiple-scattering* and test if multiple scattering is indeed unimportant. If so, and if you are using spherical coordinates, a single star at the center which is point-like, and if you are confident that at the wavelength you are interested in the thermal dust emission is not strong enough to be a considerable source of light that can be scattered into the line-of-sight (i.e. all scattered light is scattered *star* light), then you can use the simplified single-scattering mode.

This mode does not use the Monte Carlo method to compute the scattering source function, but instead uses direct integration of the starlight through the grid. It is much faster than Monte Carlo, and it does not contain noise.

By adding `simplescat` to the command line when making an image or spectrum, you switch this mode on. Please compare first to the single-scattering Monte Carlo method (see Section *Single-scattering vs. multiple-scattering*; it should yield very similar result, but without noise) and then to the full multiple scattering Monte Carlo. The full multiple scattering case will likely produce more flux. If the difference is large, then you should not use the simple single scattering mode. However, if the difference is minor, then the single scattering approximation is reasonable.

6.8.4 Warning when using an-isotropic scattering

An important issue with anisotropic scattering is that if the phase function is very forward-peaked, then you may get problems with the *spatial* resolution of your model: it could then happen that one grid cell may be too much to the left to 'beam' the scattered light into your line of sight, while the next grid point will be too much to the right. A proper treatment of strongly anisotropic scattering therefore requires also a good check of the spatial resolution of your model. There are, however, also two possible tricks (approximations) to prevent problems. They both involve slight modifications of the dust opacity files:

- You can simply assure in the opacity files that the forward peaking of the phase function has some upper limit.
- Or you can simply treat extremely forward-peaked scattering as no scattering at all (simply setting the scattering opacity to zero at those wavelengths).

Both 'tricks' are presumably reasonable and will not affect your results, unless you concentrate in your modeling very much on the angular dependence of the scattering.

6.8.5 For experts: Some more background on scattering

The inclusion of the scattering source function in the images and spectra is a non-trivial task for RADMC-3D because of memory constraints. If we would have infinite random access memory, then the inclusion of scattering in the images and spectra would be relatively easy, as we could then store the entire scattering source function $j^{\rm scat}(x,y,z,\nu,\Omega)$ and use what we need at any time. But as you see, this function is a 6-dimensional function: three spatial dimensions, one frequency and one angular direction (which consists of two angles). For any respectable model this function is far too large to be stored. So nearly all the 'numerical logistic' complexity of the treatment of scattering comes from various ways to deal with this problem. In principle RADMC-3D makes the choices of which method to use itself, so the user is not bothered with it. But depending on which kind of model the user sets up, the performance of RADMC-3D may change as a result of this issue.

So here are a few hints as to the internal workings of RADMC-3D in this regard. You do not have to read this, but it may help understanding the performance of RADMC-3D in various cases.

- *Scattering in spectra and multi-wavelength images*

  If no scattering is present in the model (see Section *Five modes of treating scattering*), then RADMC-3D can save time when making spectra and/or multi-wavelength images. I will then do each integration of Eq. (`eq-ray-tracing-rt`) directly for all wavelengths at once before going to the next pixel. This saves some time because RADMC-3D then has to calculate the geometric stuff (how the ray moves through the model) just once for each ray. If, however, scattering is included, the scattering source function must be computed using the Scattering Monte Carlo computation. Since for large models it would be too memory consuming (in particular for 3-D models) to store this function for all positions *and* all wavelengths, it must do this calculation one-by-one for each wavelength, and calculate the image for that wavelength, and then go off to the next wavelength. This means that for each ray (pixel) the geometric computations (where the ray moves through the model) has to be redone for each new wavelength. This may slow down the code a bit.

- *Anisotropic scattering and multi-viewpoint images*

  Suppose we wish to look at an object at one single wavelength, but from a number of different vantage points. If we have {em isotropic} scattering, then we need to do the Scattering Monte Carlo calculation just once, and we can make multiple images at different vantage points with the same scattering source function. This saves time, if you use the 'movie' mode of RADMC-3D (Section *Multiple vantage points: the 'Movie' mode*). However, if the scattering is anisotropic, then the source function would differ for each vantage point. In that case the scattering source function must be recalculated for each vantage point. There is, deeply hidden in RADMC-3D, a way to compute scattering source functions for multiple vantage points within a single Scattering Monte Carlo run, but for the moment this is not yet activated. end{itemize}

6.9 Polarization, Stokes vectors and full phase-functions

The module in RADMC-3D that deals with polarization (`polarization_module.f90`) is based on code developed by Michiel Min for his MCMAX code, and has been used and modified for use in RADMC-3D with his permission.

Radiative transfer of polarized radiation is a relatively complex issue. A good and extensive review on the details of polarization is given in the book by Mishchenko, Travis & Lacis, 'Scattering, Absorption and Emission of Light by Small Particles', 2002, Cambridge University Press (also electronically available on-line). Another good book (and a classic!) is the book by Bohren & Huffman 'Absorption and scattering of light by small particles', Wiley-VCH. Finally, the ultimate classic is the book by van de Hulst 'Light scattering by small particles', 1981. For some discussions on how polarization can be built in into radiative transfer codes, see e.g. Wolf, Voshchinnikov & Henning (2002, A&A 385, 365).

When we wish to include polarization in our model we must follow not just the intensity I of light (or equivalently, the energy E of a photon package), but the full Stokes vector (I, Q, U, V) (see review above for definitions, or any textbook on radiation processes). If a photon scatters off a dust grain, then the scattering angular probability density function depends not only on the scattering angle μ, but also on the input state of polarization, i.e. the values of (I, Q, U, V). And the output polarization state will be modified. Moreover, even if we would not be interested in polarization at all, but we {em do} want to have a correct scattering phase function, we need to treat polarization, because a first scattering will polarize the photon, which will then have different angular scattering probability in the next scattering event. Normally these effects are very small, so if we are not particularly interested in polarization, one can usually ignore this effect without too high a penalty in reliability. But if one wants to be accurate, there is no way around a full treatment of the (I, Q, U, V).

Interaction between polarized radiation with matter happens through so-called Müller matrices, which are 4×4 matrices that can be multiplied by the (I, Q, U, V) vector. More on this later.

It is important to distinguish between two situations:

1. The simplest case (and fortunately applicable in many cases) is if all dust particles are *randomly oriented*, and there is *no preferential helicity* of the dust grains (i.e. for each particle shape there are equal numbers of particles with that shape and with its mirror copy shape). This is also automatically true if all grains are spherically symmetric. In this case the problem of polarized radiative transfer simplifies in several ways:

   - The scattering Müller matrix simplifies, and contains only 6 independent matrix elements (see later). Moreover, these matrix elements depend only on a single angle: the scattering angle θ, and of course on the wavelength. This means that the amount of information is small enough that these Müller matrix elements can be stored in computer memory in tabulated form, so that they do not have to be calculated real-time.
   - The total scattering cross section is independent of the input polarization state. Only the output radiation (i.e. in which direction the photon will scatter) depends on the input polarization state.
   - The absorption cross section is the same for all components of the (I, Q, U, V)-vector. In other words: the absorption Müller matrix is the usual scalar absorption coefficient times the unit matrix.

   The last two points assure that most of the structure of the RADMC-3D code for non-polarized radiation can remain untouched. Only for computing the new direction and polarization state of a photon after a scattering event in the Monte Carlo module, as well as for computing the scattering source function in the Monte Carlo module (for use in the camera module) we must do extra work. Thermal emission and thermal absorption remain the same, and computing optical depths remains also the same.

2. A (much!) more complex situation arises if dust grains are *non-spherical* and are somehow *aligned due to external forces*. For instance, particles tend to align themselves in the interstellar medium if strong enough magnetic fields are present. Or particles tend to align themselves due to the combination of gravity and friction if they are in a planetary/stellar atmosphere. Here are the ways in which things become more complex:

   - All the scattering Müller matrix components will become non-zero and independent. We will thus get 16 independent variables.

- The matrix elements will depend on four angles, of which one can, in some cases, be removed due to symmetry (e.g. if we have gravity, there is still a remaining rotational symmetry; same is true of particles are aligned by a $\vec{B}$-field; but if both gravity and a $\vec{B}$-field are present, this symmetry may get lost). It will in most practical circumstances not be possible to precalculate the scattering Müller matrix beforehand and tabulate it, because there are too many variables. The matrix must be computed on-the-fly.
- The total scattering cross section now *does* depend on the polarization state of the input photon, and on the incidence angle. This means that scattering extinction becomes anisotropic.
- Thermal emission and absorption extinction will also no longer be isotropic. Moreover, they are no longer scalar: they are described by a non-trivial Müller matrix.

The complexity of this case is rather large. As of version 0.41 we have included polarized thermal emission by aligned grains (see Section *Polarized emission and absorption by aligned grains*), and we will implement more of the above mentioned aspects of aligned grains step by step.

6.9.1 Definitions and conventions for Stokes vectors

There are different conventions for how to set up the coordinate system and define the Stokes vectors. Our definition follows the IAU 1974 definition as described in Hamaker & Bregman (1996) A&AS 117, pp.161.

In this convention the x' axis points to the north on the sky, while the y' axis points to the east on the sky (but see the 'important note' below). The z' axis points to the observer. This coordinate system is positively right-handed. The radiation moves toward positive z'. Angles in the (x', y') plane are measured counter-clockwise (angle=0 means positive x' direction, angle=$\pi/2$ means positive y' direction).

In the following we will (still completely consistent with the IAU definitions above, see the 'important note' below) define "up" to be positive y' and "right" to be positive x'. So, the (x', y') coordinates are in a plane perpendicular to the photon propagation, and oriented as seen by the observer of that photon. So the direction of propagation is toward you, while y' points up and x' points to the right, just as one would normally orient it.

Important Note: This is fully equivalent to adjusting the IAU 1974 definition to have x' pointing west and y' pointing north, which is perhaps more intuitive, since most images in the literature have this orientation. So for convenience of communication, let us simply adjust the IAU 1974 definition to have positive x' ('right') pointing west and positive y' ('up') pointing north. It will have no further consequences for the definitions and internal workings of RADMC-3D because RADMC-3D does not know what 'north' and 'east' are.

The (Q, U) definition (linear polarization) is such that a linearly polarized ray with $Q = +I$, $U = V = 0$ has the electric field in the $(x', y') = (1, 0)$ direction, while $Q = -I$, $U = V = 0$ has the electric field in the $(x', y') = (0, 1)$ direction. If we have $Q = 0$, $U = +I$, $V = 0$ then the E-field points in the $x' = y'$ direction, while $Q = 0$, $U = -I$, $V = 0$ the E-field points in the $x' = -y'$ direction (see Figure 1 of Hamaker & Bregman 1996).

The (V) definition (circular polarization) is such that (quoting directly from the Hamaker & Bregman paper): *For right-handed circularly polarized radiation, the position angle of the electric vector at any point increases with time; this implies that the y' component of the field lags the x' component. Also the electric vectors along the line of sight form a left-handed screw. The Stokes V is positive for right-handed circular polarization.*

We can put these definitions into the standard formulae:

$$
\begin{aligned}
Q &= I\cos(2\beta)\cos(2\chi)\\
U &= I\cos(2\beta)\sin(2\chi)\\
V &= I\sin(2\beta)
\end{aligned}
$$

The angle χ is the angle of the E-field in the (x', y') coordinates, measured counter-clockwise from x' (consistent with our definition of angles). Example: χ = 45 deg = $\pi/4$, then $\cos(2\chi) = 0$ and $\sin(2\chi) = 1$, meaning that $Q = 0$ and $U/I = +1$. Indeed this is consistent with the above definition that $U/I = +1$ is $E'_x = E'_y$.

The angle 2β is the phase difference between the y'-component of the E-field and the x'-component of the E-field such that for $0 < \beta < \pi/2$ the E-field rotates in a counter-clockwise sense. In other words: the y'-wave lags 2β behind the

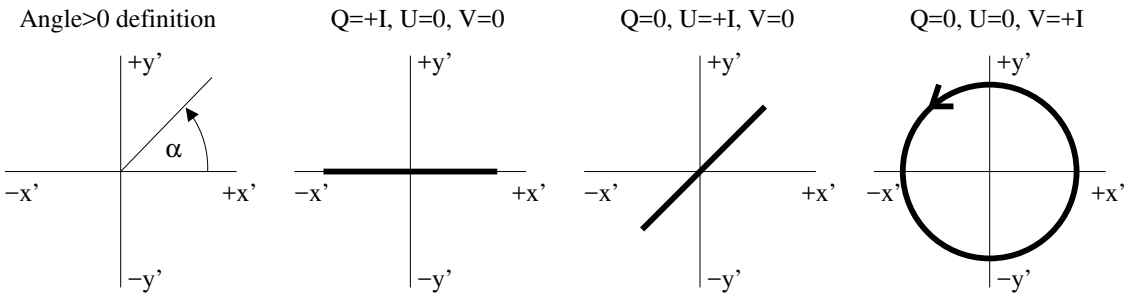

Fig. 6.2: The definition of the Stokes parameters used in RADMC-3D, which is consistent with the IAU 1974 definitions (see Hamaker & Bregman (1996) A&AS 117, pp.161). First panel shows that positive angle means counterclockwise. In the second to fourth panels the fat lines show how the tip of the real electric field vector goes as a function of time for an observer at a fixed location in space watching the radiation. The radiation moves toward the reader. We call the second panel ($Q = +I$) 'horizontally polarized', the third panel ($U = +I$) 'diagonally polarized by +45 degrees' and the fourth panel ($V = +I$) 'right-handed circularly polarized'. In the images produced by RADMC-3D (`image.out`, see Section *OUTPUT: image.out or image_****.out* and Fig. Fig. 8.1) the x' direction is the horizontal direction and the y' direction is the vertical direction.

x' wave. Example: if we have $\beta = \pi/4$, i.e. $2\beta = \pi/2$, then $\cos(2\beta) = 0$ and $\sin(2\beta) = 1$, so we have $Q = U = 0$ and $V/I = +1$. This corresponds to the y' wave being lagged $\pi/2$ behind the x' wave, meaning that we have a counterclockwise rotation. If we use the right-hand-rule and point the thumb into the direction of propagation (toward us) then the fingers indeed point in counter-rotating direction, meaning that $V/I = +1$ is righthanded polarized radiation.

In terms of the *real* electric fields of a plane monochromatic wave:

$$E'_x(t) = E_h \cos(\omega t - \Delta_h)$$
$$E'_y(t) = E_v \cos(\omega t - \Delta_v)$$

(with $E_h > 0$ and $E_v > 0$ and $\Delta_{h,v}$ are the phase lags of the components with respect to some arbitrary phase) we can write the Stokes components as:

$$I = E_h^2 + E_v^2$$
$$Q = E_h^2 - E_v^2$$
$$U = 2E_h E_v \cos(\Delta)$$
$$V = 2E_h E_v \sin(\Delta)$$

with $\Delta = \Delta_v - \Delta_h = 2\beta$.

In terms of the {em complex} electric fields of a plane monochromatic wave (the sign before the $i\omega t$ is important):

$$E'_x(t) = E_h e^{i(\Delta_h - \omega t)}$$
$$E'_y(t) = E_v e^{i(\Delta_v - \omega t)}$$

(with $E_h > 0$ and $E_v > 0$ real numbers and $\Delta_{h,v}$ are the phase lags of the components with respect to some arbitrary phase) we can write the Stokes components as:

$$I = \langle E_{x'} E^*_{x'} + E_{y'} E^*_{y'} \rangle$$
$$Q = \langle E_{x'} E^*_{x'} - E_{y'} E^*_{y'} \rangle$$
$$U = \langle E_{x'} E^*_{y'} + E_{y'} E^*_{x'} \rangle$$
$$V = i\langle E_{x'} E^*_{y'} - E_{y'} E^*_{x'} \rangle$$

6.9.2 Our conventions compared to other literature

The IAU 1974 definition is different from the definitions used in the Planck mission, for instance. So be careful. There is something said about this on the website of the healpix software http://healpix.jpl.nasa.gov/html/intronode12.htm .

Our definition is also different from the Mishchenko book and papers (see below). Compared to the books of Mishchenko and Bohren & Huffman, our definitions are:

$$
\begin{aligned}
I_{\rm ours} &= I_{\rm mishch} = I_{\rm bohrenhuffman} \\
Q_{\rm ours} &= Q_{\rm mishch} = Q_{\rm bohrenhuffman} \\
U_{\rm ours} &= -U_{\rm mishch} = -U_{\rm bohrenhuffman} \\
V_{\rm ours} &= -V_{\rm mishch} = -V_{\rm bohrenhuffman}
\end{aligned}
$$

As you see: only the U and V change sign. For a 4×4 Müller matrix M this means that the M_{II}, M_{IQ}, M_{QI}, M_{QQ}, as well as the M_{UU}, M_{UV}, M_{VU}, M_{VV} stay the same, while M_{IU}, M_{IV}, M_{QU}, M_{QV}, as well as M_{UI}, M_{UQ}, M_{VI}, M_{VQ} components would flip sign.

Compared to Mishchenko, Travis & Lacis book, what we call x' they call θ and what we call y' they call ϕ. In their Figure 1.3 (which describes the definition of the Stokes parameters) they have the θ direction pointing downward, rather than toward the right, i.e. rotated by 90 degrees clockwise compared to RADMC-3D. However, since RADMC-3D does not know what 'right' or 'down' are (only what x' and y' are) this rotation is merely a difference in how we plot things in a figure, and has no consequences for the results, as long as we define how x' and y' are oriented compared to our model (see Fig. Fig. 8.1 where $x_{\rm image}$ is our x' here and likewise for y').

Bohren & Huffman have the two unit vectors plotted in the following way: $\mathbf{e}_{\parallel}$ is plotted horizontally to the left and $\mathbf{e}_{\perp}$ is plotted vertically upward. Compared to us, our x' points toward {em minus} their $\mathbf{e}_{\parallel}$, while our y' points toward their $\mathbf{e}_{\perp}$, but since they plot their $\mathbf{e}_{\parallel}$ to the left, the orientation of our plot and their plots are consistent (i.e. if they say 'pointing to the right', they mean the same direction as we). But their definition of 'right-handed circular polarization' (clockwise when seen toward the source of the radiation) is our 'left handed'.

The book by Wendisch & Yang 'Theory of Atmospheric Radiative Transfer' uses the same conventions as Bohren & Huffman, but their basis vector $\mathbf{e}_{\parallel}$ is plotted vertically and $\mathbf{e}_{\perp}$ is plotted horizontally to the right. This only affects what they call 'horizontal' and 'vertical' but the math stays the same.

Our definition is identical to the one on the *English* Wikipedia page on Stokes parameters http://en.wikipedia.org/wiki/Stokes_parameters (on 2 January 2013), with the only exception that what they call 'righthanded' circularly polarized, we call 'lefthanded'. This is just a matter of nomenclature of what is right/left-handed, and since RADMC-3D does not know what 'right/lefthanded' is, this difference has no further consequences. *Note*, however, that the same Wikipedia page in different languages use different conventions! For instance, the German version of the page (on 2 January 2013) has the same Q and U definitions, but has the sign of V flipped.

Note that in RADMC-3D we have no global definition of the orientation of x' and y' (see e.g. Section *Defining orientation for non-observed radiation*). If we make an image with RADMC-3D, then the horizontal (x-) direction in the image corresponds to x' and the vertical (y-) direction corresponds to y', just as one would expect. So if you obtain an image from RADMC-3D and all the pixels in the image have $Q = I$ and $U = V = 0$, then the electric field points horizontally in the image.

6.9.3 Defining orientation for non-observed radiation

To complete our description of the Stokes parameters we still need to define in which direction we let x' and y' point if we do *not* have an obvious observer, i.e. for radiation moving through our object of interest which may never reach us. In the Monte Carlo modules of RADMC-3D, when polarization is switched on, any photon package does not only have a wavelength λ and a direction of propagation $\mathbf{n}$ associated with it, but also a second unit vector $\mathbf{S}$, which is always assured to obey:

$$|\mathbf{S}| = 1 \qquad \text{and} \qquad \mathbf{S} \cdot \mathbf{n} = 0$$

This leaves, for a given $\mathbf{n}$, one degree of freedom (any direction as long as it is perpendicular to $\mathbf{n}$). It is irrelevant which direction is chosen for this, but whatever choice is made, it sets the definitions of the x' and y' directions. The definitions are:

$$\begin{aligned} x' = &\quad \text{points in the direction} \quad \mathbf{S} \times \mathbf{n} \\ y' = &\quad \text{points in the direction} \quad \mathbf{S} \\ z' = &\quad \text{points in the direction} \quad \mathbf{n} \end{aligned}$$

So for $Q = -I$, $U = V = 0$ the electric field points in the direction of $\mathbf{S}$, while for $Q = +I$, $U = V = 0$ it is perpendicular to both $\mathbf{n}$ and $\mathbf{S}$.

However, if you are forced to change the direction of $\mathbf{S}$ for whatever reason, the Stokes components will also change. This coordinate transformation works as follows. We can transform from a '-basis to a ''-basis by rotating the $\mathbf{S}$-vector counter-clockwise (as seen by the observer watching the radiation) by an angle α. Any vector (x', y') in the '-basis will become a vector (x'', y'') in a ''-basis, given by the transformation:

$$\begin{pmatrix} x'' \\ y'' \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

NOTE: We choose (x', y') to be the usual counter-clockwise basis for the observer seeing the radiation. Rotating the basis in counter-clockwise direction means rotating the vector in that basis in clockwise direction, hence the sign convention in the matrix.

If we have (I, Q, U, V) in the '-basis (which we might have written as (I', Q', U', V') but by convention we drop the '), the (I'', Q'', U'', V'') in the ''-basis becomes

$$\begin{pmatrix} I'' \\ Q'' \\ U'' \\ V'' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(2\alpha) & \sin(2\alpha) & 0 \\ 0 & -\sin(2\alpha) & \cos(2\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} I \\ Q \\ U \\ V \end{pmatrix}$$

6.9.4 Polarized scattering off dust particles: general formalism

Suppose we have *one* dust particle of mass $m_{\rm grain}$ and we place it at location $\mathbf{x}$. Suppose this particle is exposed to a plane wave of electromagnetic radiation pointing in direction $\mathbf{n}_{\rm in}$ with a flux $\mathbf{F}_{\rm in} = F_{\rm in}\,\mathbf{n}_{\rm in}$. This radiation can be polarized, so that $F_{\rm in}$ actually is a Stokes vector:

$$F_{\rm in} = \begin{pmatrix} F_{I,\rm in} \\ F_{Q,\rm in} \\ F_{U,\rm in} \\ F_{V,\rm in} \end{pmatrix}$$

This particle will scatter some of this radiation into all directions. What will the flux of scattered radiation be, as observed at location $\mathbf{y} \neq \mathbf{x}$? Let us define the vector

$$\mathbf{r} = \mathbf{y} - \mathbf{x}$$

its length

$$r = |\mathbf{r}|$$

and the unit vector

$$\mathbf{e}_r = \frac{\mathbf{r}}{r}$$

We will assume that $r \gg a$ where a is the particle size. We define the {em scattering matrix elements} Z_{ij} (with i, j = $1, 2, 3, 4$) such that the measured outgoing flux from the particle at $\mathbf{y}$ is

$$\mathbf{F}_{\rm out} = F_{\rm out}\mathbf{e}_r$$

$$F_{\rm out} = \begin{pmatrix} F_{I,\rm out} \\ F_{Q,\rm out} \\ F_{U,\rm out} \\ F_{V,\rm out} \end{pmatrix} = \frac{m_{\rm grain}}{r^2} \begin{pmatrix} Z_{11} & Z_{12} & Z_{13} & Z_{14} \\ Z_{21} & Z_{22} & Z_{23} & Z_{24} \\ Z_{31} & Z_{32} & Z_{33} & Z_{34} \\ Z_{41} & Z_{42} & Z_{43} & Z_{44} \end{pmatrix} \begin{pmatrix} F_{I,\rm in} \\ F_{Q,\rm in} \\ F_{U,\rm in} \\ F_{V,\rm in} \end{pmatrix}$$

The values Z_{ij} depend on the direction into which the radiation is scattered (i.e. $\mathbf{e}_r$) and on the direction of the incoming flux (i.e. $\mathbf{n}$), but not on r: the radial dependence of the outgoing flux is taken care of through the $1/r^2$ factor in the above formula.

Some notes about our conventions are useful at this place. In many books the 'scattering matrix' is written as F_{ij} instead of Z_{ij}, and is defined as the Z_{ij} for the case when radiation comes from one particular direction: $\mathbf{n} = (0, 0, 1)$. In this manual and in the RADMC-3D code, however, we will always write Z_{ij}, because the symbol F can be confused with flux. The normalization of these matrix elements is also different in different books. In our case it has the dimension $\rm cm^2\ gram^{-1}\ ster^{-1}$. The conversion from the conventions of other books is (where $k = 2\pi/\lambda$ is the wave number in units of 1/cm):

$$Z_{ij,\rm RADMC-3D} = \frac{Z_{ij,\rm Mishchenko}}{m_{\rm grain}} = \frac{S_{ij,\rm BohrenH}}{k^2 m_{\rm grain}}$$

except that for the Z_{13}, Z_{14}, Z_{23}, Z_{24}, Z_{31}, Z_{41}, Z_{32}, Z_{42} elements (if non-zero) there must be a minus sign before the $Z_{ij,\rm RADMC-3D}$ because of the opposite U and V sign conventions (see Section *Our conventions compared to other literature*).

Note that the $S_{ij,\rm BohrenH}$ are the matrix elements obtained from the famous `BHMIE.F` code from the Bohren & Huffman book (see Chapter *Acquiring opacities from the WWW*).

6.9.5 Polarized scattering off dust particles: randomly oriented particles

In the special case in which we either have spherical particles or we average over a large number of randomly oriented particles, the Z_{ij} elements are no longer dependent on *both* $\mathbf{e}_r$ and $\mathbf{n}$ but only on the angle between them:

$$\cos\theta = \mathbf{n} \cdot \mathbf{e}_r$$

So we go from $Z_{ij}(\mathbf{n}, \mathbf{e}_r)$, i.e. a four-angle dependence, to $Z_{ij}(\theta)$, i.e. a one-angle dependence.

Now let us also assume that there is no netto helicity of the particles (they are either axisymmetric or there exist equal amounts of particles as their mirror symmetric counterparts). In that case (see e.g. Mishchenko book) of the 16 matrix elements only 6 are non-zero and independent:

$$F_{\rm out} = \begin{pmatrix} F_{I,\rm out} \\ F_{Q,\rm out} \\ F_{U,\rm out} \\ F_{V,\rm out} \end{pmatrix} = \frac{m_{\rm grain}}{r^2} \begin{pmatrix} Z_{11} & Z_{12} & 0 & 0 \\ Z_{12} & Z_{22} & 0 & 0 \\ 0 & 0 & Z_{33} & Z_{34} \\ 0 & 0 & -Z_{34} & Z_{44} \end{pmatrix} \begin{pmatrix} F_{I,\rm in} \\ F_{Q,\rm in} \\ F_{U,\rm in} \\ F_{V,\rm in} \end{pmatrix}$$

This is the case for scattering in RADMC-3D. Note that in Mie scattering the number of independent matrix elements reduces to just 4 because then $Z_{22} = Z_{11}$ and $Z_{44} = Z_{33}$. But RADMC-3D also allows for cases where $Z_{22} \neq Z_{11}$ and $Z_{44} \neq Z_{33}$, i.e. for opacities resulting from more detailed calculations such as DDA or T-matrix calculations.

Now, as described above, the Stokes vectors only have meaning if the directions of x' and y' are well-defined. For Eq. (eq-scatmat-for-randorient-nohelic) to be valid (and for the correct meaning of the Z_{ij} elements) the following definition is used: Before the scattering, the $\mathbf{S}$-vector of the photon package is rotated (and the Stokes vectors accordingly transformed) such that the new $\mathbf{S}$-vector is perpendicular to both $\mathbf{n}$ and $\mathbf{e}_r$. In other words, the scattering angle θ is a rotation of the photon propagation around the (new) $\mathbf{S}$-vector. The sign convention is such that

$$(\mathbf{n} \times \mathbf{e}_r) \cdot \mathbf{S} = \sin(\theta)$$

In other words, if we look into the incoming light (with z' pointing toward us), then for $\sin(\theta) > 0$ the photon is scattered into the $x' > 0$, $y' = 0$ direction (i.e. for us it is scattered to the right). The $\mathbf{S}$ vector for the outgoing photon remains unchanged, since the new $\mathbf{n}$ is also perpendicular to it.

So what does this all mean for the opacity? The scattering opacity tells us how much of the incident radiation is removed and converted into outgoing scattered radiation. The absorption opacity tells us how much of the incident radiation is removed and converted into heat. For randomly oriented particles without netto helicity both opacities are independent of the polarization state of the radiation. Moreover, the thermal emission is unpolarized in this case. This means that in the radiative transfer equation the extinction remains simple:

$$\frac{d}{ds}\begin{pmatrix} I_I \\ I_Q \\ I_U \\ I_V \end{pmatrix} = \begin{pmatrix} j_{\rm emis,I} \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} j_{\rm scat,I} \\ j_{\rm scat,Q} \\ j_{\rm scat,U} \\ j_{\rm scat,V} \end{pmatrix} - \rho(\kappa_{\rm abs} + \kappa_{\rm scat}) \begin{pmatrix} I_I \\ I_Q \\ I_U \\ I_V \end{pmatrix}$$

where I_I, I_Q, I_U, I_V are the intensities ($\mathrm{erg\,s^{-1}\,cm^{-2}\,Hz^{-1}\,ster^{-1}}$) for the four Stokes parameters, and likewise for $j_{\rm emis}$ and $j_{\rm scat}$, and finally, s the path length along the ray under consideration. Note that if we would allow for fixed-orientation dust particles (which we don't), Eq. (eq-radtrans-randomorient) would become considerably more complex, with extinction being matrix-valued and thermal emission being polarized.

Since $\kappa_{\rm scat}$ converts incoming radiation into outgoing scattered radiation, it should be possible to calculate $\kappa_{\rm scat}$ from angular integrals of the scattering matrix elements. For randomly oriented non-helical particles we indeed have:

$$\kappa_{\rm scat} = \oint Z_{11} d\Omega = 2\pi \int_{-1}^{+1} Z_{11}(\mu) d\mu$$

where $\mu = \cos\theta$. In a similar exercise we can calculate the anisotropy factor g from the scattering matrix elements:

$$g = \frac{2\pi}{\kappa_{\rm scat}} \int_{-1}^{+1} Z_{11}(\mu)\mu d\mu$$

This essentially completes the description of scattering as it is implemented in RADMC-3D.

We can precalculate the $Z_{ij}(\theta)$ for every wavelength and for a discrete set of values of θ, and store these in a table. This is indeed the philosophy of RADMC-3D: You have to precompute them using, for instance, the Mie code of Bohren and Huffman (see Chapter *Acquiring opacities from the WWW* for RADMC-3D compliant wrappers around that code), and then provide them to RADMC-3D through a file called `dustkapscatmat_xxx.inp` (where `xxx` is the name of the dust species) which is described in Section *The dustkapscatmat_*.inp files*. This file provides not only the matrix elements, but also the $\kappa_{\rm abs}$, $\kappa_{\rm scat}$ and g (the anisotropy factor). RADMC-3D will then internally check that Eqs.(eq-scatmat-selfconsist-kappa, eq-scatmat-selfconsist-g) are indeed fulfilled. If not, an error message will result.

One more note: As mentioned in Section *Definitions and conventions for Stokes vectors*, the sign conventions of the Stokes vector components we use (the IAU 1974 definition) are different from the Bohren & Huffman and Mishchenko books. For randomly oriented particles, however, the sign conventions of the Z-matrix elements are not affected, because those matrix elements that would be affected are those that are in the upper-right and lower-left quadrants of the matrix, and these elements are anyway zero. So we can use, for randomly oriented particles, the matrix elements from those books and their computer codes without having to adjust the signs.

6.9.6 Scattering and axially symmetric models

In spherical coordinates it is possible in RADMC-3D to set up axially symmetric models. The trick is simply to set the number of ϕ coordinate points `nphi` to 1 and to switch off the ϕ-dimension in the grid (see Section *INPUT (required): amr_grid.inp or unstr_grid.inp*). For isotropic scattering this mode has always been implemented. But for anisotropic scattering things become more complex. For such a model the scattering remains a fully 3-D problem: the scattering source function has to be stored not only as a function of r and θ, but also as a function of ϕ (for a given observer vantage point). The reason is that anisotropic scattering {em does} care about viewing angle (in contrast to isotropic scattering). So even though for an axisymmetric model the density and temperature functions only depend on r and θ (and are therefore mathematically 2-D), the scattering source function depends on r, θ and ϕ.

For this reason anisotropic scattering was, until version 0.40, not allowed for 2-D axisymmetric models. As of version 0.41 it is now possible to use the full polarized scattering mode (`scattering_mode=5`) also for 2-D axisymmetric models. The intermediate scattering modes (`scattering_mode=2, 3, 4`) remain incompatible with 2-D axisymmetry. Isotropic scattering remains, as before, fully compatible with 2-D axisymmetry.

One note of explanation: the way the full scattering is now implemented into the case of 2-D axisymmetry is the following: internally we compute not just the scattering source function for one angle, but for a whole set of ϕ angles (even though the grid has no ϕ-points). Each time a photon in the scattering Monte Carlo simulation enters a cell (which in 2-D axisymmetry is an annulus), a loop over 360 ϕ angles is performed, and the scattering source function is computed for all of these angles. {em This makes the code rather slow for each photon package!} But one needs fewer photon packages to get sufficiently high signal-to-noise ratio. You can experiment with fewer ϕ angles by adding, in `radmc3d.inp`, the following line (as an example):

```
dust_2daniso_nphi = 60
```

in which case instead of 360 the model will only use 60 ϕ points. That will speed up the code significantly, but of course will treat the ϕ-dependence of the scattering source function with lower precision.

For now the 2-D axisymmetric version of full scattering is only possible with first-order integration.

6.10 More about photon packages in the Monte Carlo simulations

In the 'standard' Monte Carlo approach, the input energy (e.g. starlight or, for the scattering Monte Carlo, the thermal emission of dust) is divided into N equal energy packages of photons, which then travel through the model and eventually either escape or get destroyed. This equal division scheme is, however, problematic for some model setups. For instance, if you have stars with vastly different luminosity in the model, then the brightest of these stars will dominate, by far, the number of output photon packages. This means that the material around low-brightness stars (which, by their proximity to these low-brightness stars, are still dominated by heating by these low-brightness stars) will experience very bad photon statistics.

To avoid this problem, RADMC-3D has, by default, its 'weighted photon package mode' switched on. This will make sure that each source of energy (i.e. each star, but also each other type of source) emits the same amount of photons. Only: bright stars will emit more energetic photon packages than dim stars.

The 'weighted photon package mode' will also solve another problem. Suppose a star lies far outside of the grid. It will emit most of its photons in directions that completely miss the grid. This means that RADMC-3D would waste

a lot of time drawing random numbers for photons that will anyway not affect the model. Also here the 'weighted photon package mode' solves the problem: It will focus the photon packages toward the model grid, and lower their energy to compensate for their favorable focusing toward the grid.

NOTE: You can switch the mode off by setting `mc_weighted_photons=0` in the `radmc3d.inp` file.

6.11 Polarized emission and absorption by aligned grains

NOTE: This mode is still in the testing phase

Grain alignment and its effects on radiative transfer is a complex topic. A review is e.g. Andersson, B.G., Lazarian, A., & Vaillancourt, J.E. (2015) 'Interstellar Dust Grain Alignment', Annual Review of Astronomy and Astrophysics, 53(1), 501–539. In RADMC-3D grain alignment is included only in a limited form. First and foremost: RADMC-3D does not know about the physics {em causing} the grain alignment. You, the user, will have to tell how the grain are aligned by giving the code a directional vector field and for each wavelength the degree to which the grain is aligned to that directional vector (more on this later). This is according to the RADMC-3D philosophy of doing {em only} the radiative transfer and leaving the physics of the material to the user.

6.11.1 Basics

Suppose we have flattened (oblate) ellipsoidal grains with one axis of symmetry and no helicity. (While helicity may be needed to radiatively spin up grains, we assume that on average the helicity of the grains is zero.). Let us assume that they are aligned with that symmetry axis along the y-axis. We view radiation from the point where the z-axis points toward us. Horizontally polarized light (which has E-field in horizontal direction, i.e. in x-direction) has $Q/I = +1$, vertically polarized light (with the $\vec{E}$ vector aligned with the symmetry axis of the grain) has $Q/I = -1$. We can then assume that the dust has different extinction coefficients for the horizontal and vertical axis. Let us call these:

$$\alpha_{\mathrm{abs},\nu,\mathrm{h}} \equiv \rho_d \kappa_{\mathrm{abs},\nu,\mathrm{h}}$$
$$\alpha_{\mathrm{abs},\nu,\mathrm{v}} \equiv \rho_d \kappa_{\mathrm{abs},\nu,\mathrm{v}}$$

We can define I, Q, U and V in terms of the electric field components E_x and E_y. The electric field components for a perfectly coherent wave can be written as

$$E_x = E_{x,0} \cos(\omega t - \Delta_x)$$
$$E_y = E_{y,0} \cos(\omega t - \Delta_y)$$

where Δ_x and Δ_y are phase lags. The phase lag between the y and x-fields is $\Delta = \Delta_y - \Delta_x$, meaning that for positive Δ the y-field lags behind the x-field. We then define the Stokes components as:

$$I = E_{x,0}^2 + E_{y,0}^2$$
$$Q = E_{x,0}^2 - E_{y,0}^2$$
$$U = 2E_{x,0}E_{y,0} \cos\Delta$$
$$V = 2E_{x,0}E_{y,0} \sin\Delta$$

Note that for $V = I$ ($\Delta = \pi/2$, i.e. the E_y lags $\pi/2$ behind E_x) we have *right-handed* circularly polarized light, meaning that the tip of the $\vec{E}$ field at a fixed point in space, when looking into the light (the propagation of light is toward the reader) rotates counter-clockwise (when the x-coordinate points right, and the y-coordinate points up). The 3-D helix of his field will be {em left-handed} (when the z-coordinate points into the propagation direction of the light, i.e. toward the reader, i.e. a right-handed coordinate system). For $Q = I$ we have linearly polarized light in which the $\vec{E}$-field lies in the x-direction. For $U = I$ we have linearly polarized light in which $\vec{E}$ lies along the $x = y$ line (when looking into the light). These definitions are consistent with the IAU 1974 definitions (Hamaker & Bregman 1996, A&AS 117, pp.161).

The E_x and E_y get absorbed in the following way:

$$E'_{x,0} = E_{x,0} e^{-\frac{1}{2}\alpha_{\mathrm{abs},\nu,\mathrm{h}} s}$$
$$E'_{y,0} = E_{y,0} e^{-\frac{1}{2}\alpha_{\mathrm{abs},\nu,\mathrm{v}} s}$$

where s is a length along the ray.

For this kind of problem it is convenient to introduce the so-called *modified Stokes parameters* I_h and I_v:

$$I_\mathrm{h} = \frac{1}{2}(I + Q)$$
$$I_\mathrm{v} = \frac{1}{2}(I - Q)$$

so that we have

$$I = I_\mathrm{h} + I_\mathrm{v}$$
$$Q = I_\mathrm{h} - I_\mathrm{v}$$

so that one can say, for perfectly coherent light,

$$I_\mathrm{h} = E^2_{x,0}$$
$$I_\mathrm{v} = E^2_{y,0}$$

With this we get the following extinction law:

$$I'_\mathrm{h} = I_\mathrm{h} e^{-\alpha_{\mathrm{abs},\nu,\mathrm{h}} s}$$
$$I'_\mathrm{v} = I_\mathrm{v} e^{-\alpha_{\mathrm{abs},\nu,\mathrm{v}} s}$$

How do U and V extinct? If we use Eqs. (eq-def-stokes-u, eq-def-stokes-v), and assume that the phase lag Δ will not change during the extinction, then

$$\begin{aligned} U' &= U e^{-\frac{1}{2}\alpha_{\mathrm{abs},\nu,\mathrm{h}} s} e^{-\frac{1}{2}\alpha_{\mathrm{abs},\nu,\mathrm{v}} s} \\ &= U e^{-\frac{1}{2}(\alpha_{\mathrm{abs},\nu,\mathrm{h}} + \alpha_{\mathrm{abs},\nu,\mathrm{v}}) s} \end{aligned}$$

This means that

$$\alpha_{\mathrm{abs},\nu,\mathrm{uv}} = \frac{1}{2}\left(\alpha_{\mathrm{abs},\nu,\mathrm{h}} + \alpha_{\mathrm{abs},\nu,\mathrm{v}}\right)$$

and

$$I'_\mathrm{u} = I_\mathrm{u} e^{-\alpha_{\mathrm{abs},\nu,\mathrm{uv}} s}$$
$$I'_\mathrm{v} = I_\mathrm{v} e^{-\alpha_{\mathrm{abs},\nu,\mathrm{uv}} s}$$

In matrix notation

$$\frac{d}{ds}\begin{pmatrix} I_\mathrm{h} \\ I_\mathrm{v} \\ U \\ V \end{pmatrix} = -\begin{pmatrix} \alpha_\mathrm{h} & 0 & 0 & 0 \\ 0 & \alpha_\mathrm{v} & 0 & 0 \\ 0 & 0 & \alpha_\mathrm{uv} & 0 \\ 0 & 0 & 0 & \alpha_\mathrm{uv} \end{pmatrix}\begin{pmatrix} I_\mathrm{h} \\ I_\mathrm{v} \\ U \\ V \end{pmatrix}$$

If we translate this to the usual Stokes components we get

$$\frac{d}{ds}\begin{pmatrix} I \\ Q \\ U \\ V \end{pmatrix} = -\begin{pmatrix} \alpha_1 & \alpha_2 & 0 & 0 \\ \alpha_2 & \alpha_1 & 0 & 0 \\ 0 & 0 & \alpha_1 & 0 \\ 0 & 0 & 0 & \alpha_1 \end{pmatrix}\begin{pmatrix} I \\ Q \\ U \\ V \end{pmatrix}$$

with

$$\alpha_1 = \frac{1}{2}\left(\alpha_{\mathrm{abs},\nu,\mathrm{h}} + \alpha_{\mathrm{abs},\nu,\mathrm{v}}\right) = \alpha_{\mathrm{abs},\nu,\mathrm{uv}}$$
$$\alpha_2 = \frac{1}{2}\left(\alpha_{\mathrm{abs},\nu,\mathrm{h}} - \alpha_{\mathrm{abs},\nu,\mathrm{v}}\right)$$

The emission will be also independently in horizontal and vertical direction. But nothing will be emitted in U or V direction. So it is most convenient to express the emission/absorption process in terms of the modified Stokes parameters:

$$\frac{dI_{\nu,\mathrm{h}}}{ds} = \alpha_{\mathrm{abs},\nu,\mathrm{h}}(\frac{1}{2}B_\nu(T) - I_{\nu,\mathrm{h}})$$
$$\frac{dI_{\nu,\mathrm{v}}}{ds} = \alpha_{\mathrm{abs},\nu,\mathrm{v}}(\frac{1}{2}B_\nu(T) - I_{\nu,\mathrm{v}})$$
$$\frac{dU_\nu}{ds} = -\alpha_{\mathrm{abs},\nu,\mathrm{uv}}U_\nu$$
$$\frac{dV_\nu}{ds} = -\alpha_{\mathrm{abs},\nu,\mathrm{uv}}V_\nu$$

In terms of matrix notation this becomes

$$\frac{d}{ds}\begin{pmatrix} I_\mathrm{h} \\ I_\mathrm{v} \\ U \\ V \end{pmatrix} = \begin{pmatrix} \frac{1}{2}\alpha_\mathrm{h}B_\nu(T) \\ \frac{1}{2}\alpha_\mathrm{v}B_\nu(T) \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} \alpha_\mathrm{h} & 0 & 0 & 0 \\ 0 & \alpha_\mathrm{v} & 0 & 0 \\ 0 & 0 & \alpha_\mathrm{uv} & 0 \\ 0 & 0 & 0 & \alpha_\mathrm{uv} \end{pmatrix} \begin{pmatrix} I_\mathrm{h} \\ I_\mathrm{v} \\ U \\ V \end{pmatrix}$$

In terms of the normal Stokes parameters this becomes

$$\frac{d}{ds}\begin{pmatrix} I \\ Q \\ U \\ V \end{pmatrix} = \begin{pmatrix} \alpha_1 B_\nu(T) \\ \alpha_2 B_\nu(T) \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} \alpha_1 & \alpha_2 & 0 & 0 \\ \alpha_2 & \alpha_1 & 0 & 0 \\ 0 & 0 & \alpha_1 & 0 \\ 0 & 0 & 0 & \alpha_1 \end{pmatrix} \begin{pmatrix} I \\ Q \\ U \\ V \end{pmatrix}$$

or written slightly differently:

$$\frac{d}{ds}\begin{pmatrix} I \\ Q \\ U \\ V \end{pmatrix} = \begin{pmatrix} \alpha_1 & \alpha_2 & 0 & 0 \\ \alpha_2 & \alpha_1 & 0 & 0 \\ 0 & 0 & \alpha_1 & 0 \\ 0 & 0 & 0 & \alpha_1 \end{pmatrix} \left[\begin{pmatrix} B_\nu(T) \\ 0 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} I \\ Q \\ U \\ V \end{pmatrix} \right]$$

So to sum things up: We need only the absorption opacity for light with $\vec{E}$ perpencidular to the symmetry axis ($\kappa_{\mathrm{abs},\nu,h}$) and the absorption opacity for light with $\vec{E}$ parallel to the symmetry axis ($\kappa_{\mathrm{abs},\nu,v}$).

6.11.2 Implementation in RADMC-3D

Polarized emission in the images and spectra

When creating images (and spectra) the `camera` module of RADMC-3D performs a ray-tracing calculation ('volume rendering') through the grid. Normally (for randomly oriented grains) the extinction along the line of sight is always unpolarized, i.e. each Stokes component is extincted equally much. The thermal emission along the line of sight is also unpolarized.

Now, however, we wish to include the effect of grain alignment in the ray-tracing. We assume that at position $\vec{x}$ in the grid our oblate grain is aligned such that the minor axis points in the direction of the orientation vector $\vec{n}_{\mathrm{align}}(\vec{x})$.

If the grain is prolate, we assume that it spins along one of its minor axes such that this spin axis is pointing along $\vec{n}_{\rm align}(\vec{x})$, so that, in effect, it acts as if it were an oblate grain again. In practice the alignment vector $\vec{n}_{\rm align}(\vec{x})$ does not lie always in the plane of the sky of the observer. Instead it will have an angle θ with the line-of-sight direction vector $\vec{n}_{\rm los}$ (note that this θ angle is different from the scattering angle θ), defined as

$$\cos\theta \equiv= |\vec{n}_{\rm align} \cdot \vec{n}_{\rm los}|$$

Here we assume that the grains have top/bottom symmetry so that we only have to concern ourselves with the positive values of $\cos\theta$, hence the $||$. If $\cos\theta = 1$ then we see the oblate grain from the top or the bottom, so that we do not expect any polarized emission. The strongest polarized emission is expected when $\cos\theta = 0$, which means that the oblate grain is seen edge-on.

We can now define the 'projected alignment vector' $\vec{n}_{\rm align,proj}$, which is the alignment vector projected into the image plane:

$$\vec{n}_{\rm align,proj} = \vec{n}_{\rm align} - (\vec{n}_{\rm align} \cdot \vec{n}_{\rm los})\, \vec{n}_{\rm los}$$

To use the equations from Section *Basics* we must first rotate our image plane coordinates (x, y) to new coordinates (x', y') such that the y' (vertical) direction points along the $\vec{n}_{\rm align,proj}$ vector while the x' (horizontal) direction points perpendicular to it. Let us write the Stokes vector of the radiation along the line of sight $(I_{\rm in}, Q_{\rm in}, U_{\rm in}, V_{\rm in})$, where we implicitly know that these are also a function of frequency ν. This Stokes vector is defined with respect to the vector $\vec{S}$ which is perpendicular to the line-of-sight direction vector $\vec{n}_{\rm los}$ and defines the direction in which the y-coordinate of the image plane points. We must now express this incoming radiation (at the start of the segment) in the new (x', y') image plane coordinates, i.e. with respect to the new vector $\vec{S}'$ that points along $\vec{n}_{\rm align,proj}$ (i.e. $\vec{S}'$ is the normalized version of $\vec{n}_{\rm align,proj}$). This rotation is performed using

$$\begin{pmatrix} I' \\ Q' \\ U' \\ V' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(2\alpha) & \sin(2\alpha) & 0 \\ 0 & -\sin(2\alpha) & \cos(2\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} I_{\rm in} \\ Q_{\rm in} \\ U_{\rm in} \\ V_{\rm in} \end{pmatrix}$$

where α is the angle between $\vec{S}'$ and $\vec{S}$ such that if (as seen by the observer) $\vec{S}'$ lies counter-clockwise from $\vec{S}$, α is positive (the usual definition). With this new Stokes vector (I', Q', U', V') we will now use the equations of Section *Basics*.

To be able to perform this rotation in a uniquely defined way, it is necessary that along each segment along the line of sight this new (x', y') orientation stays fixed (but can vary from segment to segment). As the line-of-sight ray enters a cell and leaves it again, this line element (segment) will have its image-plane coordinates rotated according to the alignment vector of that cell. As a result, the integration must be done first order (assuming all source terms to be constant along the segment). In principle second order integration would also be possible, but then the trick with the rotation of the image coordinate plane such that y' points along the orientation vector does no longer work, and the integration of the formal transfer equation would become much more complex, involving the full Müller matrix formulation. We will not do this, so we will stick to first order integration of Eq. eq-formal-rt-emisabs-in-rotated-system.

For convenience we will leave out the primes (') from here on, so while we write (I, Q, U, V) we mean in fact (I', Q', U', V'). We now compute $I_{\rm h}$ and $I_{\rm v}$ using Eqs. (eq-modif-stokes-h, eq-modif-stokes-v). Now, along this segment of the ray, we can write Eq. eq-formal-rt-emisabs-in-rotated-system in the following form:

$$\frac{d}{ds}\begin{pmatrix} I_{\rm h} \\ I_{\rm v} \\ U \\ V \end{pmatrix} = \begin{pmatrix} \alpha_{\rm h} & 0 & 0 & 0 \\ 0 & \alpha_{\rm v} & 0 & 0 \\ 0 & 0 & \alpha_{\rm uv} & 0 \\ 0 & 0 & 0 & \alpha_{\rm uv} \end{pmatrix} \left[\begin{pmatrix} \frac{1}{2}B_\nu(T) \\ \frac{1}{2}B_\nu(T) \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} I_{\rm h} \\ I_{\rm v} \\ U \\ V \end{pmatrix} \right]$$

It becomes clear that it is easy to perform the first order integration of this equation along this ray segment:

$$
\begin{aligned}
I_{\rm h,end} &= e^{-\tau_h} I_{\rm h,start} + \tfrac{1}{2} e^{-\tau_h} B_\nu(T) \\
I_{\rm v,end} &= e^{-\tau_v} I_{\rm v,start} + \tfrac{1}{2} e^{-\tau_v} B_\nu(T) \\
U_{\rm end} &= e^{-\tau_{uv}} U_{\rm start} \\
V_{\rm end} &= e^{-\tau_{uv}} V_{\rm start}
\end{aligned}
$$

where 'start' stands for the start of the ray segment, and 'end' the end of the ray segment (which becomes the start of the next ray segment), and $\tau_h = \alpha_{\rm h}\Delta s$, $\tau_v = \alpha_{\rm v}\Delta s$ and $\tau_{uv} = \alpha_{\rm uv}\Delta s$, with Δs being the length of the segment.

We now compute $I_{\rm end}$ and $Q_{\rm end}$, and rotate back to the (x, y) image plane coordinate system (i.e. using $\vec{S}$ instead of $\vec{S}'$ to define the Stokes parameters) by applying Eq. eq-rot-stokes-align but now with $\alpha \rightarrow -\alpha$, and we have the values of the Stokes parameter at the end of the ray segment. Now we repeat this whole procedure for the next ray segment.

Polarized emission as source term in the Monte Carlo simulation

The polarization effects and anisotropic emission by aligned grains will also affect the Monte Carlo simulations.

For the *thermal Monte Carlo* (see Section *The thermal Monte Carlo simulation: computing the dust temperature*) this effect is *not included*. In principle it should be included, but it would slow the code down, and it is unlikely to play a significant role for the dust temperature, in particular since the anisotropy of thermal emission is not expected to be so strong (and the polarization state is irrelevant for computing the dust temperature). It is clear that we make a small error here, but we believe that this is well within the much stronger uncertainties of the dust opacities.

For the *scattering Monte Carlo* (see Section *Scattered light in images and spectra: The 'Scattering Monte Carlo' computation*), however, this effect may be important! The polarization caused by scattering of light off dust grains yields of course different results if the incident light is unpolarized or if it is already strongly polarized through, for instance, polarized thermal emission. In RADMC-3D this is therefore built into the scattering Monte Carlo. This will not slow down the code much because (in contrast to the thermal Monte Carlo) the polarized thermal emission only has to be computed at the start of each photon path, if the photon is emitted by the dust.

The way this is included is that when a photon is emitted by the dust inside a cell, RADMC-3D first randomly chooses which of the dust species emits the photon (the probabilities are weighted by the contribution each dust species makes to the emissivity at the given wavelength). Then the emission direction is randomly chosen, based on the θ-dependent probability function (where θ is the angle with the alignment direction) given by the average of the orthogonal (horizontal) and parallel (vertical) absorption opacities. Once the emission direction is chosen, the polarization state of the photon package is computed based on the orthogonal and parallel absorption opacities. Then the photon package is sent on its way.

Note that if `alignment_mode = -1` then the polarized (and anisotropic) thermal emission by aligned grains is only included in the ray-tracing for images and spectra, while for `alignment_mode = 1` it is *also* included in the scattering Monte Carlo computation.

6.11.3 Consistency with other radiative processes

The above equations assume that the absorption/emission is the only radiative process included. However, in practice we also have other processes involved, such as line emission/absorption or the scattering source function. The way this can be treated here is to simply add these additional opacities to all four components of the extinction matrix of Eq. (eq-formal-rt-emisabs-in-rotated-system) and to add the additional emissivities to the vector with the Planck functions in Eq. (eq-formal-rt-emisabs-in-rotated-system). For the scattered light emissivity (which is a Stokes vector) we must also first perform a rotation from $\vec{S}$ to $\vec{S}'$ using the Stokes rotation formula of Eq. (eq-rot-stokes-align) before we add this emissivity to the equation. If we include the effect of alignment on the scattering (see Section *Effect of aligned grains on the scattering*) then also the scattering extinction will be different for the orthogonal (horizontal) and parallel (vertical) Stokes components. That is easy to include in this formalism.

6.11.4 Input files for RADMC-3D for aligned grains

In RADMC-3D we implement the functions $\kappa_{\mathrm{abs},\nu,h}$ and $\kappa_{\mathrm{abs},\nu,v}$ as a function of angle θ which the alignment axis makes with the light of sight. For θ we see the oblate grain from the top (face-on), so that there is no asymmetry between horizontal (orthogonal to the alignment orientation vector) and vertical (parallel to the alignment orientation vector). Then we will have $\kappa_{\mathrm{abs},\nu,h} = \kappa_{\mathrm{abs},\nu,v}$. For $\theta = 90^\circ$ we will have the maximum difference between $\kappa_{\mathrm{abs},\nu,h}$ and $\kappa_{\mathrm{abs},\nu,v}$. We write

$$\kappa_{\mathrm{abs},\nu,h}(\theta) = \kappa_{\mathrm{abs},\nu}\, k_{\nu,h}(\theta)$$
$$\kappa_{\mathrm{abs},\nu,v}(\theta) = \kappa_{\mathrm{abs},\nu}\, k_{\nu,v}(\theta)$$

where $k_{\nu,h}(\theta)$ and $k_{\nu,v}(\theta)$ are dimensionless functions, and where we take $\theta \in [0, 90]$ (in degrees), or equivalently $\cos(\theta) \in [0, 1]$. We impose the condition that if we randomly orient this grain, the average opacity becomes the one we computed for the randomly oriented grains:

$$\int_0^\infty \frac{1}{2}\left[\kappa_{\mathrm{abs},\nu,h}(\theta) + \kappa_{\mathrm{abs},\nu,v}(\theta)\right] d\mu = \kappa_{\mathrm{abs},\nu}$$

This yields the following integration condition on the dimensionless $k_{\nu,h}$ and $k_{\nu,v}$:

$$\int_0^\infty \frac{1}{2}\left[k_{\nu,h}(\theta) + k_{\nu,v}(\theta)\right] d\mu = 1$$

If we set, for all values of θ, $k_{\nu,h}(\theta) = k_{\nu,v}(\theta) = 1$ then we retrieve the result for spherical grains.

In RADMC-3D the functions $k_{\nu,h}(\theta)$ and $k_{\nu,v}(\theta)$ are read in via the file `dustkapalignfact_*.inp`. This file has the following structure:

```
# Any amount of arbitrary
# comment lines that tell which opacity this is.
# Each comment line must start with an # or ; or ! character
iformat                             <=== Typically 1 at present
nlam                                <=== Nr of wavelengths
nmu                                 <=== Nr of angles sampled
lambda[1]                           <=== Wavelength grid in micron
...
lambda[nlam]
theta[1]                            <=== Angle grid in degrees
...
theta[nmu]
k_orth[1,1]       k_para[1,1]       <=== The arrays k_orth and k_para
...
k_orth[nmu,1]     k_para[nmu,1]
k_orth[1,2]       k_para[1,2]
...
k_orth[nmu,2]     k_para[nmu,2]
...
...
...
k_orth[1,nlam]    k_para[1,nlam]
...
k_orth[nmu,nlam]  k_para[nmu,nlam]
```

The angles `theta` are in degrees and must start at 0 and end at 90, or vice versa. The `nmu` does not have to be the same (and the angles do not have to be the same) as those in the `dustkapscatmat_*.inp` file. But the wavelength grid must be identical to the one in the `dustkapscatmat_*.inp` file.

In order to make RADMC-3D read this file `dustkapalignfact_*.inp` the `dustopac.inp` file should, for this particular dust species, have '20' as the way in which this dust species is read (instead of 10 which is used for polarized scattering with the Z matrix).

In addition, RADMC-3D also needs to know the orientation direction of the grains. This is a vector field $\vec{p}_{\rm align}(\vec{x})$. The length of these vectors should be between 0 and 1, where 1 means that the grains are perfectly aligned and 0 means they are not aligned at all. The efficiency $\epsilon_{\rm align}$ is thus given by

$$\epsilon_{\rm align}(\vec{x}) = |\vec{p}_{\rm align}(\vec{x})|$$

The directional unit-vector of alignment $\vec{n}_{\rm align}(\vec{x})$ is thus

$$\vec{n}_{\rm align}(\vec{x}) = \left(\epsilon_{\rm align}(\vec{x})\right)^{-1} \vec{p}_{\rm align}(\vec{x})$$

The $\vec{p}_{\rm align}(\vec{x})$ vector field is in the file `grainalign_dir.inp` (or its binary formatted version `grainalign_dir.binp`). The format of this file is exactly the same as that of the gas velocity file `gas_velocity.inp`. The ascii format looks like:

```
iformat                                  <=== Typically 1 at present
nrcells
p_x[1]       p_y[1]       p_z[1]
..
p_x[nrcells] p_y[nrcells] p_z[nrcells]
```

Note that $|\vec{p}_{\rm align}(\vec{x})|$ should never be > 1. If it is found to be significantly > 1 at some point in the grid, then an error occurs. If it is only a tiny bit above 1, due to rounding errors, it will be normalized to 1.

The way in which partial alignment ($0 < \epsilon_{\rm align} < 1$) is treated in RADMC-3D is to treat the opacities and emissivities as simple linear sums of fully aligned and non aligned versions. For instance, Eqs. (eq-align-kappa-k-h, eq-align-kappa-k-v) then become

$$\kappa_{{\rm abs},\nu,h}(\theta) = \kappa_{{\rm abs},\nu} \left[\epsilon_{\rm align} k_{\nu,h}(\theta) + 1 - \epsilon_{\rm align}\right]$$
$$\kappa_{{\rm abs},\nu,v}(\theta) = \kappa_{{\rm abs},\nu} \left[\epsilon_{\rm align} k_{\nu,v}(\theta) + 1 - \epsilon_{\rm align}\right]$$

In order to tell RADMC-3D that it should include the effect of alignment on the thermal emission of dust grains one must add a line in the `radmc3d.inp` file with:

```
alignment_mode = 1
```

The example model in `examples/run_simple_1_align/` demonstrates how the input files have to be made to have RADMC-3D treat the aligned dust grains for thermal emission.

6.11.5 Effect of aligned grains on the scattering

This is, currently, not yet implemented.

6.12 Grain size distributions

6.12.1 Quick summary of how to implement grain sizes

A common application of RADMC-3D is continuum radiative transfer in media with a grain size distribution. RADMC-3D does not know the concept of "grain size distribution", and it does not care. You have to provide RADMC-3D will all the information it needs, such that it will handle the dust size distribution you want. All the responsibility lies with you, the user.

There are basically two ways by which you can make RADMC-3D treat a grain size distribution:

- Method 1: Mixing the dust opacities into a single opacity file, having RADMC-3D think that there is only one dust species. Fast and simple.
- Method 2: Computing N dust opacity files, having N independent dust species. Slower but more realistic and flexible.

In the following subsections we will discuss both methods.

6.12.2 Method 1: Size distribution in the opacity file (faster)

The simplest way is to compute a single dust opacity table (see Section *INPUT (required for dust transfer): dustopac.inp and dustkappa_*.inp or dustkapscatmat_*.inp or dust_optnk_*.inp*) for a single dust species. You compute the weighted dust opacity and put that into the file `dustkappa_XXX.inp` (for instance let's call it `dustkappa_sizedistrib.inp`) or `dustkapscatmat_XXX.inp` (for instance let's call it `dustkapscatmat_sizedistrib.inp`). All the information about the size distribution shape is then encoded in this opacity file, and RADMC-3D will never know that it is, in fact, a size distribution. The file `dust_density.inp` will then only contain the spatial distribution of a single grain species: that of the mixture of sizes. Advantage: it is the easiest. Disadvantage: the size distribution will be identical everywhere. Another disadvantage: each grain size will have the same temperature (because RADMC-3D does not know that these are different dust sizes).

This method be useful to save computer time. You essentially do all the work of computing the opacity of the size distribution beforehand (even before you start RADMC-3D), so that you get a single opacity file that already contains the size-distribution-weighted opacities. You must then be sure that you do the weighting such that the opacity is "cross section per gram of dust", where "dust" is already the entire grain size distribution. The dust density in the `dust_density.inp` file must then also be the density of the entire grain size distribution. See Section *The mathematics of grain size distributions* for more information about size distributions.

6.12.3 Method 2: Size distribution in the density file (better)

RADMC-3D can handle multiple dust species simultaneously and co-spatially. So you can have N grain sizes, each represented by its own opacity file and its own spatial density distribution. So if we, for example, have two sizes, 1 micron and 1 millimeter (i.e. $N = 2$) then we would have, for instance, two opacity files, `dustkappa_1micron.inp` and `dustkappa_1mm.inp` (don't forget to mark them both in `dustopac.inp` too), and within the `dust_density.inp` file we have two density fields. This allows you, for instance, to have the large grains near the midplane of a disk and the small grains vertically more extended, because you can determine the density ρ of each dust species completely independent from the others.

We have now two choices how to handle these species: (a) thermally coupled (set `itempdecoup = 0` in `radmc3d.inp`, see section *INPUT: radmc3d.inp*, or (b) thermally decoupled (default, but you can set `itempdecoup = 1` in `radmc3d.inp` to make sure). The default is thermally decoupled, because that is for most cases more realistic. If the grains are thermally decoupled, then, in the optically thin regions exposed to hot stellar radiation, the small grains tend to be hotter than the large ones. However, in optically thick regions the small and large grains will tend to automatically acquire similar or the same temperature(s).

Compared to the first method (with a single dust species), this method is more flexible (allowing different spatial distributions for different grain sizes) but also more costly (requiring the radiative transfer code to handle the interaction of the radiation with N independent dust species). You must then calculate each grain opacity file separately, and keep these normalized to "cross section per gram of this particular dust species or size". Here, the weighting is not done in the opacity files, but in the fact that each grain size (or species) i has its own density ρ_i. You would then need to make sure that these ρ_i are following the size distribution you wish. See Section *The mathematics of grain size distributions* for more information about size distributions.

6.12.4 The mathematics of grain size distributions

Grain size distributions can be confusing, so here is a small tutorial. Suppose we have the famous MRN (Mathis, Rumpl, Nordsieck) size distribution:

$$n(a)da \propto a^{-7/2}da$$

with a the radius of the grain, $n(a)da$ the number of grains between sizes a and $a+da$ per volume. We say that this powerlaw goes from $a = a_{\rm min}$ to $a = a_{\rm max}$, and we keep in mind that $a_{\rm max}$ can be (but does not have to) orders of magnitude larger than $a_{\rm min}$. The total dust density ρ is:

$$\rho = \int_{a_{\rm min}}^{a_{\rm max}} m(a)n(a)da$$

where $m(a)$ is the mass of the grain:

$$m(a) = \rho_s \frac{4\pi}{3} a^3$$

where ρ_s is the material density of the grain material (typically somewhere between 1 and 3.6 $\mathrm{gram/cm^3}$, dependent on the material).

For RADMC-3D we have to disretize this into N bins. Since we can have $a_{\rm max} \gg a_{\rm min}$, it is best to take a logarithmic grid in a, i.e. equal spacing in $\ln(a)$. So we divide the interval $[\ln(a_{\rm min}), \ln(a_{\rm max})]$ up into N equal size bins, numbering $i = 0$ to $i = N-1$, with cell centers denoted as $\ln(a_i)$ and the cell walls are denoted as $\ln(a_{i-1/2})$ for the left- and $\ln(a_{i+1/2})$ for the right-hand cell wall. We have $\ln(a_{-1/2}) = \ln(a_{\rm min})$ and $\ln(a_{N-1/2}) = \ln(a_{\rm max})$. For any i we have the same cell width in log-space: $\Delta\ln(a) = \Delta\ln(a_i) = \ln(a_{i+1/2}) - \ln(a_{i-1/2})$. Now the density for each bin is:

$$\rho_i = \int_{a_{i-1/2}}^{a_{i+1/2}} m(a)\, n(a)\, da = \int_{\ln(a_{i-1/2})}^{\ln(a_{i+1/2})} a\, m(a)\, n(a)\, d\ln(a)$$

If the bin width $\Delta\ln(a)$ is small enough, this can be approximated as

$$\rho_i \simeq a_i m(a_i) n(a_i) \Delta\ln(a)$$

The total dust density is then

$$\rho = \sum_{i=0}^{N-1} \rho_i$$

The opacity for bin i at some frequency ν is approximately $\kappa_\nu(a_i)$ if a small enough bin size is used. That means that the extinction coefficient

$$\alpha_\nu = \sum_{i=0}^{N-1} \rho_i \kappa_\nu(a_i)$$

In method 2 (Section *Method 2: Size distribution in the density file (better)*) this is exactly what happens: you specify N tables of $\kappa_\nu(a_i)$ (each table containing all frequencies for which you want to use the opacity), and the "mixing" happens in each grid cell on-the-fly depending on the local values of ρ_i. The values of ρ_i in the file `dust_density.inp` are exactly these ρ_i.

On the contrary, in method 1 (Section *Method 1: Size distribution in the opacity file (faster)*), you compute a nomalized $\hat{n}(a_i)$ such that

$$\sum_{i=0}^{N-1} a_i m(a_i) \hat{n}(a_i) \Delta\ln(a) = 1$$

so that with

$$\hat{\rho}_i \simeq a_i m(a_i) \hat{n}(a_i) \Delta \ln(a)$$

we get

$$\sum_{i=0}^{N-1} \hat{\rho}_i = 1$$

Now we can compute a grain-size-mean opacity:

$$\hat{\kappa}_\nu = \sum_{i=0}^{N-1} \hat{\rho}_i \kappa_\nu(a_i)$$

which is computed before running RADMC-3D, and will be valid at all locations in the spatial grid. At each cell we only have the total dust density ρ. The extinction coefficient is then

$$\alpha_\nu = \rho \hat{\kappa}_\nu$$

CHAPTER

SEVEN

LINE RADIATIVE TRANSFER

RADMC-3D is capable of modeling radiative transfer in molecular and/or atomic lines. Due to the complexity of line radiative transfer, and the huge computational and memory requirements of full-scale non-LTE line transfer, RADMC-3D has various different modes of line transfer. Some modes are very memory efficient, but slower, while others are faster, but less memory efficient, yet others are more accurate but much slower and memory demanding. The default mode (and certainly recommended initially) is LTE ray-tracing in the slow but memory efficient way: the *simple LTE mode* (see Section *Line transfer modes and how to activate the line transfer*). Since this is the default mode, you do not need to specify anything to have this selected.

7.1 Quick start for adding line transfer to images and spectra

Do properly model line transfer requires dedication and experimentation. This is *not* a simple task. See Section *What can go wrong with line transfer?* for an analysis of several pitfalls one may encounter. However, nothing is better than experimenting and thus gaining hands-on experience. So the easiest and quickest way to start is to start with one of the simple line transfer test models in the `examples/` directory.

So simply visit `examples/run_test_lines_1/`, `examples/run_test_lines_2/` or `examples/run_test_lines_3/` and follow the directions in the `README` file. The main features of adding line ray tracing to a model is to add the following files into any previously constructed model with dust radiative transfer:

- `lines.inp`: A control file for line transfer.
- `molecule_co.inp`: or any other molecular data file containing properties of the molecule or atom.
- `numberdens_co.inp` (or its binary version, see Chapter *Binary I/O files*) or that of another molecule: The number density of that molecule in units of cm^{-3}.
- `gas_temperature.inp` (or its binary version, see Chapter *Binary I/O files*): The gas temperature at each grid cell. You do not need to specify this file if you add the keyword `tgas_eq_tdust = 1` into the `radmc3d.inp` file.

7.2 Some definitions for line transfer

The formal transfer equation is:

$$\frac{dI_\nu(\omega)}{ds} = j_\nu(\omega) - \alpha_\nu(\omega) I_\nu(\omega)$$

which is true also for the lines. Here ω is the direction, ν the frequency, I the intensity. The emissivity j_ν and extinction α_ν for each line (given by i =upper level and j =lower level) is given by:

$$j_{ij}(\Omega,\nu) = \frac{h\nu}{4\pi} N n_i A_{ij} \varphi_{ij}(\omega,\nu)$$
$$\alpha_{ij}(\omega,\nu) = \frac{h\nu}{4\pi} N (n_j B_{ji} - n_i B_{ij}) \varphi_{ij}(\omega,\nu)$$

Here N is the number density of the molecule, n_i is the *fraction* of the molecules that are in level i, A_{ij} is the Einstein coefficient for spontaneous emission from level i to level j, and B_{ij} and B_{ji} are the Einstein-B-coefficients which obey:

$$A_{ij} = \frac{2h\nu_{ij}^3}{c^2} B_{ij}, B_{ji} g_j = B_{ij} g_i$$

where g are the statistical weights of the levels, h the Planck constant and c the light speed. The symbol $\varphi_{ij}(\omega,\nu)$ is the line profile function. For zero velocity field $\varphi_{ij}(\omega,\nu) = \tilde{\varphi}_{ij}(\nu)$, i.e. the line profile function is independent of direction. The tilde is to say that this is the comoving line profile. It is given by

$$\tilde{\varphi}_{ij}(\nu) = \frac{c}{a_{\rm tot}\nu_{ij}\sqrt{\pi}} \exp\left(-\frac{c^2(\nu-\nu_{ij})^2}{a_{\rm tot}^2\nu_{ij}^2}\right)$$

where ν_{ij} is the line-center frequency for the line and $a_{\rm tot}$ is the line width in units of cm/s. For pure thermal broadning we have

$$a_{\rm tot} = a_{\rm therm} = \sqrt{\frac{2kT_{\rm gas}}{m_{\rm mol}}}$$

where $m_{\rm mol}$ is the weight of the molecule in gram, k is the Boltzmann constant, $T_{\rm gas}$ the gas temperature in K. As we shall discuss in Section *INPUT: The local microturbulent broadening (optional)*: we can also add 'microturbulent line broadning' $a_{\rm turb}$, also in cm/s:

$$a_{\rm tot} = \sqrt{a_{\rm turb}^2 + a_{\rm therm}^2} = \sqrt{a_{\rm turb}^2 + \frac{2kT_{\rm gas}}{m_{\rm mol}}}$$

When we have macroscopic velocities in our model, then the line profile becomes angle-dependent (at a given lab-frame frequency):

$$\varphi_{ij}(\omega,\nu) = \tilde{\varphi}_{ij}\big(\nu(1-\vec{\omega}\cdot\vec{v}/c) - \nu_{ij}\big)$$

The radiative transfer equation for non overlapping lines is then

$$\frac{dI_{ij}(\omega,\nu)}{ds} = j_{ij}(\omega,\nu) - \alpha_{ij}(\omega,\nu) I_{ij}(\omega,\nu)\,.$$

But RADMC-3D naturally includes overlapping lines, at least in the ray-tracing (for spectra and images). For non-LTE modes the line overlapping is not yet (as of December 2011) included.

7.3 Line transfer modes and how to activate the line transfer

Line transfer can be done in various different ways. This is controlled by the global variable `lines_mode` (see below) and by the nature of the molecular/atomic data (see discussion in Section *INPUT: The line.inp file*).

7.3.1 Two different atomic/molecular data file types

Let us start with the latter: RADMC-3D does not have any atomic or molecular data hard-coded inside. It reads these data from data files that you provide. There are two fundamentally different ways to feed atomic/molecular data into RADMC-3D:

- Files containing the full level and line information (named `molecule_XXX.inp`, where `XXX` is the name of the molecule or atom). Atoms or molecules for which this data is provided can be treated in LTE as well as in non-LTE.
- Files containing only a line list (named `linelist_XXX.inp`, where `XXX` is the name of the molecule or atom). Atoms or molecules for which this data is provided can only be treated in LTE.

7.3.2 The different line modes (the `lines_mode parameter`)

For the atoms or molecules for which the full data are specified (the `molecule_XXX.inp` files) RADMC-3D has various different line transfer modes, including different treatments of LTE or non-LTE. Which of the modes you want RADMC-3D to use can be specified in the `radmc3d.inp` file by setting the variable `lines_mode`, for instance, by adding the following line to `radmc3d.inp`:

```
lines_mode = 3
```

for LVG + Escape Probability populations. If no option is given, then the *LTE mode* (`lines_mode=1`) is used.

The various line modes are:

- *LTE mode (=default mode):* [`lines_mode=1`]

  In this mode the line radiative transfer is done under LTE assumptions.

- *User-defined populations:* [`lines_mode=2`]

  This calls the routine `userdef_compute_levelpop()` to compute the level populations. This allows the user to specify the populations of the levels of the molecules freely.

- *Large Velocity Gradient (Sobolev) populations:* [`lines_mode=3`]

  This is one of the non-LTE modes of RADMC-3D. This mode calculates the angle-averaged velocity gradient, and uses this to compute the level populations according to the Large Velocity Gradient method (also often called Sobolev's method). This method is like an escape probability method, where the escape probability is calculated based on the velocity gradient. For this mode to work, the velocity field has to be read in, as well as at least one of the number densities of the collision partners of the molecule. See Section *Non-LTE Transfer: The Large Velocity Gradient (LVG) + Escape Probability (EscProb) method*.

- *Optically Thin non-LTE level populations method:* [`lines_mode=4`]

  This is one of the non-LTE modes of RADMC-3D. This mode calculates the non-LTE level populations under the assumption that all emitted line radiation escapes and is not reabsorbed. For this mode to work, at least one of the number densities of the collision partners of the molecule. See Section *Non-LTE Transfer: The optically thin line assumption method*.

- *User-defined populations:* [`lines_mode=-10`]

  This calls the routine `userdef_general_compute_levelpop()` on-the-fly during the ray-tracing. This is very much like `userdef_compute_levelpop()`, except that it leaves the entire line-related stuff to the user: It does not read the molecular data from a file. NOTE: This is a rather tricky mode, to be used only if you know very well what you are doing...

- *Full non-LTE modes:* {bf Not yet ready}

The default of the `lines_mode` variable is `lines_mode=1`.

NOTE 1: Line emission is automatically included in the images and spectra if RADMC-3D finds the file `lines.inp` in the model directory. You can switch off the lines with the command-line option `'noline'`.

NOTE 2: If you are very limited by memory, and if you use LTE, LVG+EscProb or optically thin populations, you can also ask RADMC-3D to *not* precalculate the level populations before the rendering, but instead compute them on-the-fly. This makes the code slower, but requires less memory. You can do this by choosing e.g. `lines_mode=-3` instead of `lines_mode=3` (for LVG+EscProb).

7.4 The various input files for line transfer

7.4.1 INPUT: The line transfer entries in the radmc3d.inp file

Like all other modules of `radmc3d`, also the line module can be steered through keywords in the `radmc3d.inp` file. Here is a list:

- `tgas_eq_tdust` (default: 0)

  Normally you must specify the gas temperature at each grid cell using the `gas_temperature.inp` file (or directly in the `userdef_module.f90`, see Chapter *Modifying RADMC-3D: Internal setup and user-specified radiative processes*). But sometimes you may want to compute first the dust temperature and then set the gas temperature equal to the dust temperature. You can do this obviously by hand: read the output dust temperature and create the equivalent gas temperature input file from it. But that is cumbersome. By setting `tgas_eq_tdust=1` you tell `radmc3d` to simply read the `dust_temperature.inp` file and then equate the gas temperature to the dust temperature. If multiple dust species are present, only the first species will be used.

7.4.2 INPUT: The line.inp file

Like with the dust (which has this `dustopac.inp` master file, also the line module has a master file: `lines.inp`. It specifies which molecules/atoms are to be modeled and in which file the molecular/atomic data (such as the energy levels and the Einstein A coefficients) are to be found

```
iformat                                  <=== Put this to 2
N                                        Nr of molecular or atomic species to be␣
↪modeled
molname1 inpstyle1 iduma1 idumb1 ncol1   Which molecule used as species 1 + other info
.
.
.
molnameN inpstyleN idumaN idumbN ncolN   Which molecule used as species N + other info
```

The `N` is the number of molecular or atomic species you wish to model. Typically this is 1. But if you want to *simultaneously* model for instance the ortho-H_2O and para-H_2O infrared lines, you would need to set this to 2.

The N lines following N (i.e. lines 3 to N+2) specify the molecule or atom, the kind of input file format (explained below), and two integers which, at least for now, can be simply set to 0 (see Section *For experts: Selecting a subset of lines and levels 'manually'* for the meaning of these integers - for experts only), plus finally third integer, which has to do with non-LTE transfer: the number of collision partners (set to 0 if you only intend to do LTE transfer).

The molecule name can be e.g. `co` for carbon monoxide. The file containing the data should then be called `molecule_co.inp` (even if it is an atom rather than a molecule; I could not find a good name which means both molecule or atom). This file should be either generated by the user, or (which is obviously the preferred option) taken from one of the databases of molecular/atomic radiative properties. Since there are a number of such databases and I

want the code to be able to read those files without the need of casting them into some special RADMC-3D format, `radmc3d` allows the user to select which *kind* of file the `molecule_co.inp` (for CO) file is. At present only one format is supported: the Leiden database. But more will follow. To specify to `radmc3d` to use the Leiden style, you put the `inpstyle` to 'leiden'. So here is a typical example of a `lines.inp` file:

```
2
1
co   leiden   0   0   0
```

This means: one molecule will be modeled, namely CO (and thus read from the file `molecule_co.inp`), and the data format is the Leiden database format.

NOTE: Since version 0.26 the file format number of this file `lines.inp` has increased. It is now 2, because in each line an extra integer is added.

NOTE: The files from the Leiden LAMDA database (see Section *INPUT: Molecular/atomic data: The molecule_XXX.inp file(s)*) are usually called something like `co.dat`. You will have to simply rename to `molecule_co.inp`.

Most molecular data files have, in addition to the levels and radiative rates, also the collision rates listed. See Section *INPUT: Molecular/atomic data: The molecule_XXX.inp file(s)*. For non-LTE radiative transfer this is essential information. The number densities of the collision partners (the particles with which the molecule can collide and which can collisionally excited or de-excite the molecule) are given in number density files with the same format as those of the molecule itself (see Section *INPUT: The number density of collision partners (for non-LTE transfer)*). However, we must tell RADMC-3D to which collision partner particle the rate tables listed in the `molecule_co.inp` are associated (see Section *INPUT: The number density of collision partners (for non-LTE transfer)* for a better explanation of the issue here). This can be done with the last of the integers in each line. Example: if the `lines.inp` file reads:

```
2
1
co   leiden   0   0   2
p-h2
o-h2
```

this means that the first collision rate table (starting with the number `3.2e-11` in the example of Section *INPUT: Molecular/atomic data: The molecule_XXX.inp file(s)*) is for collisions with particles for which the number density is given in the file `numberdens_p-h2.inp` and the second collision rate table (starting with the number `4.1e-11` in the example of Section *INPUT: Molecular/atomic data: The molecule_XXX.inp file(s)*) is for collisions with particles for which the number density is given in the file `numberdens_o-h2.inp`.

We could also decide to ignore the difference between para-H_2 and ortho-H_2, and simply use the first table (starting with the number `3.2e-11` in the example of Section *INPUT: Molecular/atomic data: The molecule_XXX.inp file(s)*), which is actually for para-H_2 only, as a proxy for the overall mixture of H_2 molecules. After all: The collision rate for para-H_2 and ortho-H_2 are not so very different. In that case we may simply ignore this difference and only provide a file `numberdens_h2.inp`, and link that to the first of the two collision rate tables:

```
2
1
co   leiden   0   0   1
h2
```

(Note: we cannot, in this way, link this to the second of the two tables, only to the first). But if we would do this:

```
2
1
co   leiden   0   0   3
p-h2
```

(continues on next page)

(continued from previous page)

```
o-h2
h
```

we would get an error, because only two collision rate tables are provided in `molecule_co.inp`.

Finally, as we will explain in Section *INPUT: Molecular/atomic data: The linelist_XXX.inp file(s)*, there is an alternative way to feed atomic/molecular data into RADMC-3D: By using linelists. To tell RADMC-3D to read a linelist file instead of a Leiden-style molecular/atomic data file, just write the following in the `lines.inp` file:

```
2
1
h2o  linelist 0   0   0
```

(example here is for water). This will make RADMC-3D read the `linelist_h2o.inp` file as a linelist file (see Section *INPUT: Molecular/atomic data: The linelist_XXX.inp file(s)*). Note that lines from a linelist will always be in LTE.

You can also have multiple species, for which some are of Leiden-style and some are linelist style. For instance:

```
2
2
co   leiden   0   0   2
p-h2
o-h2
h2o  linelist 0   0   0
```

Here the CO lines can be treated in a non-LTE manner (depending on what you put for `lines_mode`, see Section *Line transfer modes and how to activate the line transfer*), and the H_2O is treated in LTE.

7.4.3 INPUT: Molecular/atomic data: The molecule_XXX.inp file(s)

As mentioned in Section *INPUT: The line.inp file* the atomic or molecular fundamental data such as the level diagram and the radiative decay rates (Einstein A coefficients) are read from a file (or more than one files) named `molecule_XXX.inp`, where the `XXX` is to be replaced by the name of the molecule or atom in question. For these files RADMC-3D uses the Leiden LAMDA database format. Note that, instead of a `molecule_XXX.inp` file you can also give a linelist file, but this will be discussed in Section *INPUT: Molecular/atomic data: The linelist_XXX.inp file(s)*.

The precise format of the Leiden database data files is of course described in detail on their web page http://www.strw.leidenuniv.nl/~moldata/ . Here we only give a very brief overview, based on an example of CO in which only the first few levels are specified (taken from the LAMDA database):

```
!MOLECULE (Data from the LAMDA database)
CO
!MOLECULAR WEIGHT
28.0
!NUMBER OF ENERGY LEVELS
5
!LEVEL + ENERGIES(cm^-1) + WEIGHT + J
    1     0.000000000  1.0     0
    2     3.845033413  3.0     1
    3    11.534919938  5.0     2
    4    23.069512649  7.0     3
    5    38.448164669  9.0     4
!NUMBER OF RADIATIVE TRANSITIONS
```

(continues on next page)

(continued from previous page)

```
4
!TRANS + UP + LOW + EINSTEINA(s^-1) + FREQ(GHz) + E_u(K)
    1     2     1   7.203e-08     115.2712018      5.53
    2     3     2   6.910e-07     230.5380000     16.60
    3     4     3   2.497e-06     345.7959899     33.19
    4     5     4   6.126e-06     461.0407682     55.32
```

The first few lines are self-explanatory. The first of the two tables is about the levels. Column one is simply a numbering. Column 2 is the energy of the level E_k, specified in units of $1/\mathrm{cm}$. To get the energy in erg you multiply this number with hc/k where h is the Planck constant, c the light speed and k the Boltzmann constant. Column 3 is the degeneration number, i.e. the the g parameter of the level. Column 4 is redundant information, not used by the code.

The second table is the line list. Column 1 is again a simple counter. Column 2 and 3 specify which two levels the line connects. Column 4 is the radiative decay rate in units of $1/\mathrm{s}$, i.e. the Einstein A coefficient. The last two columns are redundant information that can be easily derived from the other information.

If you are interested in LTE line transfer, this is enough information. However, if you want to use one of the non-LTE modes of RADMC-3D, you must also have the collisional rate data. An example of a `molecule_XXX.inp` file that also contains these data is:

```
!MOLECULE (Data from the LAMDA database)
CO
!MOLECULAR WEIGHT
28.0
!NUMBER OF ENERGY LEVELS
10
!LEVEL + ENERGIES(cm^-1) + WEIGHT + J
    1     0.000000000  1.0     0
    2     3.845033413  3.0     1
    3    11.534919938  5.0     2
    4    23.069512649  7.0     3
    5    38.448164669  9.0     4
!NUMBER OF RADIATIVE TRANSITIONS
9
!TRANS + UP + LOW + EINSTEINA(s^-1) + FREQ(GHz) + E_u(K)
    1     2     1   7.203e-08     115.2712018      5.53
    2     3     2   6.910e-07     230.5380000     16.60
    3     4     3   2.497e-06     345.7959899     33.19
    4     5     4   6.126e-06     461.0407682     55.32
!NUMBER OF COLL PARTNERS
2
!COLLISIONS BETWEEN
2 CO-pH2 from Flower (2001) & Wernli et al. (2006) + extrapolation
!NUMBER OF COLL TRANS
10
!NUMBER OF COLL TEMPS
7
!COLL TEMPS
    5.0   10.0   20.0   30.0   50.0   70.0  100.0
!TRANS + UP + LOW + COLLRATES(cm^3 s^-1)
    1     2     1  3.2e-11 3.3e-11 3.3e-11 3.3e-11 3.4e-11 3.4e-11 3.4e-11
    2     3     1  2.9e-11 3.0e-11 3.1e-11 3.2e-11 3.2e-11 3.2e-11 3.2e-11
    3     3     2  7.9e-11 7.2e-11 6.5e-11 6.1e-11 5.9e-11 6.0e-11 6.5e-11
    4     4     1  4.8e-12 5.2e-12 5.6e-12 6.0e-12 7.1e-12 8.4e-12 1.2e-11
    5     4     2  4.7e-11 5.0e-11 5.1e-11 5.1e-11 5.1e-11 5.1e-11 5.1e-11
    6     4     3  9.0e-11 7.9e-11 7.1e-11 6.7e-11 6.5e-11 6.6e-11 7.2e-11
```

(continues on next page)

(continued from previous page)

```
    7     5     1  2.8e-12 3.1e-12 3.4e-12 3.7e-12 4.0e-12 4.4e-12 4.0e-12
    8     5     2  8.0e-12 9.6e-12 1.1e-11 1.2e-11 1.4e-11 1.6e-11 2.2e-11
    9     5     3  5.9e-11 6.2e-11 6.2e-11 6.1e-11 6.0e-11 5.9e-11 5.8e-11
   10     5     4  8.5e-11 8.2e-11 7.5e-11 7.1e-11 6.9e-11 6.9e-11 7.3e-11
!COLLISIONS BETWEEN
3 CO-oH2 from Flower (2001) & Wernli et al. (2006) + extrapolation
!NUMBER OF COLL TRANS
10
!NUMBER OF COLL TEMPS
7
!COLL TEMPS
    5.0   10.0   20.0   30.0   50.0   70.0  100.0
!TRANS + UP + LOW + COLLRATES(cm^3 s^-1)
    1     2     1  4.1e-11 3.8e-11 3.4e-11 3.3e-11 3.4e-11 3.5e-11 3.9e-11
    2     3     1  5.8e-11 5.6e-11 5.2e-11 5.0e-11 4.7e-11 4.7e-11 6.2e-11
    3     3     2  7.5e-11 7.1e-11 6.6e-11 6.2e-11 6.1e-11 6.2e-11 7.1e-11
    4     4     1  6.6e-12 7.1e-12 7.3e-12 7.5e-12 8.1e-12 9.0e-12 1.3e-11
    5     4     2  7.9e-11 8.3e-11 8.1e-11 7.8e-11 7.4e-11 7.3e-11 8.5e-11
    6     4     3  8.0e-11 7.5e-11 7.0e-11 6.8e-11 6.7e-11 6.9e-11 7.7e-11
    7     5     1  5.8e-12 6.1e-12 6.1e-12 6.1e-12 6.2e-12 6.3e-12 7.8e-12
    8     5     2  1.0e-11 1.2e-11 1.4e-11 1.4e-11 1.6e-11 1.8e-11 2.2e-11
    9     5     3  8.3e-11 8.9e-11 9.0e-11 8.8e-11 8.3e-11 8.1e-11 8.7e-11
   10     5     4  8.0e-11 7.9e-11 7.5e-11 7.2e-11 7.1e-11 7.1e-11 7.6e-11
```

As you see, the first part is the same. Now, however, there is extra information. First, the number of collision partners, for which these collisional rate data is specified, is given. Then follows the reference to the paper containing these data (this is not used by RADMC-3D; it is just for information). Then the number of collisional transitions that are tabulated (since collisions can relate any level to any other level, this number should ideally be `nlevels*(nlevels-1)/2`, but this is not strictly enforced). Then the number of temperature points at which these collisional rates are tabulated. Then follows this list of temperatures. Finally we have the table of collisional transitions. Each line consists of, first, the ID of the transition (dummy), then the upper level, then the lower level, and then the $K_{\rm up,low}$ collisional rates in units of $[{\rm cm}^3/s]$. The same is again repeated (because in this example we have two collision partners: the para-H_2 molecule and the ortho-H_2 molecule).

To get the collision rate $C_{\rm up,low}$ per molecule (in units of [1/s]) for the molecule of interest, we must multiply $K_{\rm up,low}$ with the number density of the collision partner (see Section *INPUT: The number density of collision partners (for non-LTE transfer)*). So in this example, the $C_{\rm up,low}$ becomes:

$$C_{\rm up,low} = N_{\rm p-H_2} K^{\rm p-H_2}_{\rm up,low} + N_{\rm o-H_2} K^{\rm o-H_2}_{\rm up,low}$$

The rates tabulated in this file are always the *downward* collision rate. The upward rate is internally computed by RADMC-3D using the following formula:

$$C_{\rm low,up} = C_{\rm up,low} \frac{g_{\rm up}}{g_{\rm low}} \exp\left(-\frac{\Delta E}{kT}\right)$$

where the g factors are the statistical weights of the levels, ΔE is the energy difference between the levels, k is the Boltzmann constant and T the gas temperature.

Some notes:

- When doing LTE transfer *and* you make RADMC-3D read a separate file with the partition function (Section *INPUT for LTE line transfer: The partition function (optional)*), you can limit the `molecule_XXX.inp` files to just the levels and lines you are interested in. But again: You *must* then read the partition function separately, and not let RADMC-3D compute it internally based on the `molecule_XXX.inp` file.
- When doing non-LTE transfer and/or when you let RADMC-3D compute the partition function internally you *must* make sure to include all possible levels that might get populated, otherwise you may overpredict the strength of the lines you are interested in.

- The association of each of the collision partners in this file to files that contain their spatial distribution is a bit complicated. See Section *INPUT: The number density of collision partners (for non-LTE transfer)*.

7.4.4 INPUT: Molecular/atomic data: The linelist_XXX.inp file(s)

In many cases molecular data are merely given as lists of lines (e.g. the HITRAN database, the Kurucz database, the Jorgensen et al. databases etc.). These line lists contain information about the line wavelength λ_0, the line strength $A_{\rm ud}$, the statistical weights of the lower and upper level and the energy of the lower or upper level. Sometimes also the name or set of quantum numbers of the levels, or additional information about the line profile shapes are specified. These line lists contain no *direct* information about the level diagram, although this information can be extracted from the line list (if it is complete). These lines lists also do not contain any information about collisional (de-)excitation, so they cannot be used for non-LTE line transfer of any kind. They only work for LTE line transfer. But such line lists are nevertheless used often (and thus LTE is then assumed).

RADMC-3D can read the molecular data in line-list-form (files named `linelist_XXX.inp`). RADMC-3D can in fact use both formats mixed (the line list one and the 'normal' one of Section *INPUT: Molecular/atomic data: The molecule_XXX.inp file(s)*). Some molecules may be specified as line lists (`linelist_XXX.inp`) while simultaneously others as full molecular files (`molecule_XXX.inp`, see Section *INPUT: Molecular/atomic data: The molecule_XXX.inp file(s)*). For the 'linelist molecules' RADMC-3D will then automatically use LTE, while for the other molecules RADMC-3D will use the mode according to the `lines_mode` value. This means that you can use this to have mixed LTE and non-LTE species of molecules/atoms within the same model, as long as the LTE ones have their molecular/atomic data given in a line list form. This can be useful to model situations where most of the lines are in LTE, but one (or a few) are non-LTE.

Now coming back to the linelist data. Here is an example of such a file (created from data from the HITRAN database):

```
! RADMC-3D Standard line list
! Format number:
1
! Molecule name:
h2o
! Reference: From the HITRAN Database (see below for more info)
! Molecular weight (in atomic units)
  18.010565
! Include table of partition sum? (0=no, 1=yes)
  1
! Include additional information? (0=no, 1=yes)
  0
! Nr of temperature points for the partition sum
    2931
!  Temp [K]      PartSum
 7.000000E+01  2.100000E+01
 7.100000E+01  2.143247E+01
 7.200000E+01  2.186765E+01
 7.300000E+01  2.230553E+01
....
....
....
 2.997000E+03  1.594216E+04
 2.998000E+03  1.595784E+04
 2.999000E+03  1.597353E+04
 3.000000E+03  1.598924E+04
! Nr of lines
  37432
! ID    Lambda [mic]  Aud [sec^-1]  E_lo [cm^-1]  E_up [cm^-1]  g_lo  g_up
     1  1.387752E+05  5.088000E-12  1.922829E+03  1.922901E+03   11.    9.
```

(continues on next page)

(continued from previous page)

```
     2  2.496430E+04  1.009000E-09  1.907616E+03  1.908016E+03   21.   27.
     3  1.348270E+04  1.991000E-09  4.465107E+02  4.472524E+02   33.   39.
     4  1.117204E+04  8.314000E-09  2.129599E+03  2.130494E+03   27.   33.
     5  4.421465E+03  1.953000E-07  1.819335E+03  1.821597E+03   21.   27.
....
....
....
 37429  3.965831E-01  3.427000E-05  7.949640E+01  2.529490E+04   15.   21.
 37430  3.965250E-01  1.508000E-04  2.121564E+02  2.543125E+04   21.   27.
 37431  3.964335E-01  5.341000E-05  2.854186E+02  2.551033E+04   21.   27.
 37432  3.963221E-01  1.036000E-04  3.825169E+02  2.561452E+04   27.   33.
```

The file is pretty self-explanatory. It contains a table for the partition function (necessary for LTE transfer) and a table with all the lines (or any subset you wish to select). The lines table columns are as follows: first column is just a dummy index. Second column is the wavelength in micron. Third is the Einstein-A-coefficient (spontaneous downward rate) in units of s^{-1}. Fourth and fifth are the energies above the ground state of the lower and upper levels belonging to this line in units of cm^{-1}. Sixth and seventh are the statistical weights (degenracies) of the lower and upper levels belonging to this line.

Note that you can tell RADMC-3D to read `linelist_h2o.inp` (instead of search for `molecule_h2o.inp`) by specifying `linelist` instead of `leiden` in the `lines.inp` file (see Section *INPUT: The line.inp file*).

7.4.5 INPUT: The number density of each molecular species

For the line radiative transfer we need to know how many molecules of each species are there per cubic centimeter. For molecular/atom species `XXX` this is given in the file `numberdens_XXX.inp` (see Chapter *Binary I/O files* for the binary version of this file, which is more compact, and which you can use instead of the ascii version). For each molecular/atomic species listed in the `lines.inp` file there must be a corresponding `numberdens_XXX.inp` file. The structure of the file is very similar (though not identical) to the structure of the dust density input file `dust_density.inp` (Section *INPUT (required for dust transfer): dust_density.inp*). For the precise way to address the various cells in the different AMR modes, we refer to Section *INPUT (required for dust transfer): dust_density.inp*, where this is described in detail.

For formatted style (`numberdens_XXX.inp`):

```
iformat                                  <=== Typically 1 at present
nrcells
numberdensity[1]
..
numberdensity[nrcells]
```

The number densities are to be specified in units of molecule per cubic centimeter.

7.4.6 INPUT: The gas temperature

For line transfer we need to know the gas temperature. You specify this in the file `gas_temperature.inp` (see Chapter *Binary I/O files* for the binary version of these files, which are more compact, and which you can use instead of the ascii versions). The structure of this file is identical to that described in Section *INPUT: The number density of each molecular species*, but of course with number density replaced by gas temperature in Kelvin. For the precise way to address the various cells in the different AMR modes, we refer to Section *INPUT (required for dust transfer): dust_density.inp*, where this is described in detail.

Note: Instead of literally specifying the gas temperature you can also tell `radmc3d` to copy the dust temperature (if it know it) into the gas temperature. See the keyword `tgas_eq_tdust` described in Section *INPUT: The line transfer*

entries in the radmc3d.inp file.

7.4.7 INPUT: The velocity field

Since gas motions are usually the main source of Doppler shift or broadening in astrophysical settings, it is obligatory to specify the gas velocity. This can be done with the file `gas_velocity.inp` (see Chapter *Binary I/O files* for the binary version of these files, which are more compact, and which you can use instead of the ascii versions). The structure is again similar to that described in Section *INPUT: The number density of each molecular species*, but now with three numbers at each grid point instead of just one. The three numbers are the velocity in x, y and z direction for Cartesian coordinates, or in r, θ and ϕ direction for spherical coordinates. Note that both in cartesian coordinates and in spherical coordinates *all* velocity components have the same dimension of cm/s. For spherical coordinates the conventions are: positive v_r points outwards, positive v_θ points downward (toward larger θ) for $0 < \theta < \pi$ (where 'downward' is toward smaller z), and positive v_ϕ means velocity in counter-clockwise direction in the x, y-plane.

For the precise way to address the various cells in the different AMR modes, we refer to Section *INPUT (required for dust transfer): dust_density.inp*, where this is described in detail.

7.4.8 INPUT: The local microturbulent broadening (optional)

The `radmc3d` code automatically includes thermal broadening of the line. But sometimes it is also useful to specify a local (spatially unresolved) turbulent width. This is not obligatory (if it is not specified, only the thermal broadening is used) but if you want to specify it, you can do so in the file `microturbulence.inp` (see Chapter *Binary I/O files* for the binary version of these files, which are more compact, and which you can use instead of the ascii versions). The file format is the same structure as described in Section *INPUT: The number density of each molecular species*. For the precise way to address the various cells in the different AMR modes, we refer to Section *INPUT (required for dust transfer): dust_density.inp*, where this is described in detail.

Here is the way it is included into the line profile:

$$a^2_{\rm linewidth} = a^2_{\rm turb} + \frac{2kT_{\rm gas}}{\mu}$$

where $T_{\rm gas}$ is the temperature of the gas, μ the molecular weight, k the Boltzmann constant and $a_{\rm turb}$ the microturbulent line width in units of cm/s. The $a_{\rm linewidth}$ is then the total (thermal plus microturbulent) line width.

7.4.9 INPUT for LTE line transfer: The partition function (optional)

If you use the LTE mode (either `lines_mode=-1` or `lines_mode=1`), then the partition function is required to calculate, for a given temperature the populations of the various levels. Since this involves a summation over *all* levels of all kinds that can possibly be populated, and since the molecular/atomic data file may not include all these possible levels, it may be useful to look the partition function up in some literature and give this to `radmc3d`. This can be done with the file `partitionfunction_XXX.inp`, where again `XXX` is here a placeholder for the actual name of the molecule at hand. If you do not have this file in the present model directory, then `radmc3d` will compute the partition function itself, but based on the (maybe limited) set of levels given in the molecular data file. The structure of the `partitionfunction_XXX.inp` file is:

```
iformat                    ; The usual format number, currently 1
ntemp                      ; The number of temperatures at which it is specified
temp(1)       pfunc(1)
temp(2)       pfunc(2)
  .             .
  .             .
  .             .
temp(ntemp)   pfunc(ntemp)
```

NOTE: RADMC-3D assumes the partition function to be defined in the following way:

$$Z(T) = \sum_{i=1} g_i e^{-(E_i - E_1)/kT}$$

In other words: the first level is assumed to be the ground state. This is done so that one can also use an energy definition in which the ground state energy is non-zero (example: Hydrogen $E_1 = -13.6$ eV). If you use molecular line datafiles that contain only a subset of levels (which is in principle no problem for LTE calculations) then it is essential that the ground state is included in this list, and that it is the first level (`ilevel=1`).

7.4.10 INPUT: The number density of collision partners (for non-LTE transfer)

For non-LTE line transfer (see e.g. Sections *Non-LTE Transfer: The Large Velocity Gradient (LVG) + Escape Probability (EscProb) method*, *Non-LTE Transfer: The optically thin line assumption method*) the molecules can be collisionally excited. The collision rates for each pair of molecule + collision partner are given in the molecular input data files (Section *INPUT: Molecular/atomic data: The molecule_XXX.inp file(s)*). To find how often a molecular level of a single molecule is collisionally excited to another level we also need to know the number density of the collision partner molecules. In the example in Section *INPUT: Molecular/atomic data: The molecule_XXX.inp file(s)* these were para-H_2 and ortho-H_2. We must therefore somehow tell RADMC-3D what the number densities of these molecules are. This is done by reading in the number densities for this(these) collision partner(s). The file for this has exactly the same format as that for the number density of any molecule (see Section *INPUT: The number density of each molecular species*). So for our example we would thus have two files, which could be named `numberdens_p-h2.inp` and `numberdens_o-h2.inp` respectively. See Section *INPUT: The number density of each molecular species* for details.

However, how does RADMC-3D know that the first collision partner of CO is called `p-h2` and the second `o-h2`? In principle the file `molecule_co.inp` give some information about the name of the collision partners. But this is often not machine-readable. Example, in `molecule_co.inp` of Section *INPUT: Molecular/atomic data: The molecule_XXX.inp file(s)* the line that should tell this reads:

```
2 CO-pH2 from Flower (2001) & Wernli et al. (2006) + extrapolation
```

for the first of the two (which is directly from the LAMDA database). This is hard to decipher for RADMC-3D. Therefore you have to tell this explicitly in the file `lines.inp`, and we refer to Section *INPUT: The line.inp file* for how to do this.

7.5 Making images and spectra with line transfer

Making images and spectra with/of lines works in the same way as for the continuum. RADMC-3D will check if the file `lines.inp` is present in your directory, and if so, it will automatically switch on the line transfer. If you insist on *not* having the lines switched on, in spite of the presence of the `lines.inp` file, you can add the option `noline` to `radmc3d` on the command line. If you don't, then lines are normally automatically switched on, except in situations where it is obviously not required.

You can just make an image at some wavelength and you'll get the image with any line emission included if it is there. For instance, if you have the molecular data of CO included, then:

```
radmc3d image lambda 2600.757
```

will give an image right at the CO 1-0 line center. The code will automatically check if (and if yes, which) line(s) are contributing to the wavelength of interest. Also it will include all the continuum emission (and absorption) that you would usually obtain.

There is, however, an exception to this automatic line inclusion: If you make a spectral energy distribution (with the command `sed`, see Section *Making spectra*), then lines are not included. The same is true if you use the `loadcolor`

command. But for normal spectra or images the line emission will automatically be included. So if you make a spectrum at wavelength around some line, you will get a spectrum including the line profile from the object, as well as the dust continuum.

It is not always convenient to have to know by heart the exact wavelengths of the lines you are interested in. So RADMC-3D allows you to specify the wavelength by specifying which line of which molecule, and at which velocity you want to render:

```
radmc3d image iline 2 vkms 2.4
```

If you have CO as your molecule, then iline 2 means CO 2-1 (the second line in the rotational ladder).

By default the first molecule is used (if you have more than one molecule), but you can also specify another one:

```
radmc3d image imolspec 2 iline 2 vkms 2.4
```

which would select the second molecule instead of the first one.

If you wish to make an entire spectrum of the line, you can do for instance:

```
radmc3d spectrum iline 1 widthkms 10
```

which produces a spectrum of the line with a passband going from -10 km/s to +10 km/s. By default 40 wavelength points are used, and they are evenly spaced. You can set this number of wavelengths:

```
radmc3d spectrum iline 1 widthkms 10 linenlam 100
```

which would make a spectrum with 100 wavelength points, evenly spaced around the line center. You can also shift the passband center:

```
radmc3d spectrum iline 1 widthkms 10 linenlam 100 vkms -10
```

which would make the wavelength grid 10 kms shifted in short direction.

Note that you can use the `widthkms` and `linenlam` keywords also for images:

```
radmc3d image iline 1 widthkms 10 linenlam 100
```

This will make a multi-color image, i.e. it will make images at 100 wavelenths points evenly spaced around the line center. In this way you can make channel maps.

For more details on how to specify the spectral sampling, please read Section *Specifying custom-made sets of wavelength points for the camera*. Note that keywords such as `incl`, `phi`, and any other keywords specifying the camera position, zooming factor etc, can all be used in addition to the above keywords.

7.5.1 Speed versus realism of rendering of line images/spectra

As usual with numerical modeling: including realism to the modeling goes at the cost of rendering speed. A 'fully realistic' rendering of a model spectrum or image of a gas line involves (assuming the level populations are already known):

1. Doppler-shifted emission and absorption.
2. Inclusion of dust thermal emission and dust extinction while rendering the lines.
3. Continuum emission scattered by dust into the line-of-sight
4. Line emission from (possibly obscured) other regions is allowed to scatter into the line-of-sight by dust grains (see Section *Line emission scattered off dust grains*).

RADMC-3D always includes the Doppler shifts. By default, RADMC-3D also includes dust thermal emission and extinction, as well as the scattered continuum radiation.

For many lines, however, dust continuum scattering is a negligible portion of the flux, so you can speed things up by not including dust scattering! This can be easily done by adding the `noscat` option on the command-line when you issue the command for a line spectrum or multi-frequency image. This way, the scattering source function is not computed (is assumed to be zero), and no scattering Monte Carlo runs are necessary. This means that the ray-tracer can now render all wavelength simultaneously (each ray doing all wavelength at the same time), and the local level populations along each ray can now be computed once, and be used for all wavelengths. *This may speed up things drastically, and for most purposes virtually perfectly correct*. Just beware that when you render short-wavelength lines (optical) or you use large grains, i.e. when the scattering albedo at the wavelength of the line is not negligible, this may result in a mis-estimation of the continuum around the line.

7.5.2 Line emission scattered off dust grains

NOTE: The contents of this subsection may not be 100% implemented yet.

Also any line emission from obscured regions that get scattered into the line of sight by the dust (if dust scattering is included) will be included. Note, however, that any possible Doppler shift *induced* by this scattering is *not* included. This means that if line emission is scattered by a dust cloud moving at a very large speed, then this line emission will be scattered by the dust, but no Doppler shift at the projected velocity of the dust will be added. Only the Doppler shift of the line-emitting region is accounted for. This is rarely a problem, because typically the dust that may scatter line emission is located far away from the source of line emission and moves at substantially lower speed.

7.6 Non-LTE Transfer: The Large Velocity Gradient (LVG) + Escape Probability (EscProb) method

The assumption that the energy levels of a molecule or atom are always populated according to a thermal distribution (the so-called 'local thermodynamic equilibrium', or LTE, assumption) is valid under certain circumstances. For instance for planetary atmospheres in most cases. But in the dilute interstellar medium this assumption is very often invalid. One must then compute the level populations consistent with the local density and temperature, and often also consistent with the local radiation field. Part of this radiation field might even be the emission from the lines themselves, meaning that the molecules radiatively influence their neighbors. Solving the level populations self-consistently is called 'non-LTE radiative transfer'. A full non-LTE radiative transfer calculation is, however, in most cases (a) too numerically demanding and sometimes (b) unnecessary. Sometimes a simple approximation of the non-LTE effects is sufficient.

One such approximation method is the 'Large Velocity Gradient' (LVG) method, also called the 'Sobolev approximation'. Please read for instance the paper by Ossenkopf (1997) 'The Sobolev approximation in molecular clouds', New Astronomy, 2, 365 for more explanation, and a study how it works in the context of molecular clouds. The LVG mode of RADMC-3D has been used for the first time by Shetty et al. (2011, MNRAS 412, 1686), and a description of the method is included in that paper. The nice aspect of this method is that it is, for most part, local. The only slightly non-local aspect is that a velocity gradient has to be computed by comparing the gas velocity in one cell with the gas velocity in neighboring cells.

As of RADMC-3D Version 0.33 the LVG method is combined with an escape probability (EscProb) method. In fact, LVG *is* a kind of escape probability method itself. It is just that for the classic EscProb method the photons can escape due to the finite size of the object, and thus the finite optical depth in the lines. In the LVG the object size is not the issue, but the gradient of the velocity. The line width combined with the velocity gradient give a length scale over which a photon can escape.

In the LVG + EscProb method the line-integrated mean intensity J_{ij} is given by

$$J_{ij} = (1-\beta_{ij})S_{ij} + \beta_{ij} J_{ij}^{\rm bg}$$

where $J_{ij}^{\rm bg}$ is the mean intensity of the background radiation field at frequency $\nu = \nu_{ij}$ (default is blackbody at 2.73 K, but this temperature can be varied with the `lines_tbg` variable in `radmc3d.inp`), while β_{ij} is the escape probability for line $i \rightarrow j$. This is given by

$$\beta_{ij} = \frac{1-\exp(-\tau_{ij})}{\tau_{ij}}$$

where τ_{ij} is the line-center optical depth in the line.

For the LVG method this optical depth is given by the velocity gradient:

$$\begin{aligned}\tau_{ij}^{\rm LVG} &= \frac{ch}{4\pi}\frac{N_{\rm molec}}{1.064\,|\nabla\vec{v}|}\left[n_j B_{ji} - n_i B_{ij}\right] \\ &= \frac{c^3}{8\pi\nu_{ij}^3}\frac{A_{ij} N_{\rm molec}}{1.064\,|\nabla\vec{v}|}\left[\frac{g_i}{g_j} n_j - n_i\right]\end{aligned}$$

(see e.g. van der Tak et al. 2007, A&A 468, 627), where n_i is the fractional level population of level i, $N_{\rm molec}$ the total number density of the molecule, $|\nabla\vec{v}|$ the absolute value of the velocity gradient, g_i the statistical weight of level i and ν_{ij} the line frequency for transition $i \rightarrow j$. In comparing to Eq. 21 of van der Tak's paper, note that their $N_{\rm mol}$ is a column density (cm^{-2}) and their ΔV is the line width (cm/s), while our $N_{\rm molec}$ is the number density (cm^{-3}) and $|\nabla\vec{v}|$ is the velocity gradient (s^{-1}). Their formula is thus in fact EscProb while ours is LVG.

For the EscProb method *without* velocity gradients, we need to be able to compute the total column depth $\Sigma_{\rm molec}$ in the direction where this $\Sigma_{\rm molec}$ is minimal. This is something that, at the moment, RADMC-3D cannot yet do. But this is something that can be estimated based on a 'typical length scale' L, such that

$$\Sigma_{\rm molec} \simeq N_{\rm molec}\, L$$

RADMC-3D allows you to specify L separately for each cell (in the file `escprob_lengthscale.inp` or its binary version). The simplest would be to set it to a global value equal to the typical size of the object we are interested in. Then the line-center optical depth, assuming a Gaussian line profile with width $a_{\rm linewidth}$, is

$$\tau_{ij}^{\rm EscProb} = \frac{hc\Sigma_{\rm molec}}{4\pi\sqrt{\pi}\,a_{\rm linewidth}}\left[n_j B_{ji} - n_i B_{ij}\right]$$

because $\phi(\nu=\nu_{ij}) = c/(a\nu_{ij}\sqrt{\pi})$.

The optical depth of the combined LVG + EscProb method is then:

$$\tau_{ij} = \min\left(\tau_{ij}^{\rm LVG}, \tau_{ij}^{\rm EscProb}\right)$$

This is then the τ_{ij} that needs to be inserted into Eq. (eq-escprob-beta-formula) for obtaining the escape probability β_{ij} (which includes escape due to LVG as well as the finite length scale L).

The LVG+EscProb method solves at each location the following statistical equilibrium equation:

$$\begin{aligned}&\sum_{j>i}\Big[n_j A_{ji} + (n_j B_{ji} - n_i B_{ij}) J_{ji}\Big] \\ &- \sum_{j<i}\Big[n_i A_{ij} + (n_i B_{ij} - n_j B_{ji}) J_{ij}\Big] \\ &+ \sum_{j\neq i}\Big[n_j C_{ji} - n_i C_{ij}\Big] = 0\end{aligned}$$

Replacing J_{ij} (and similarly J_{ji}) with the expression of Eq. (eq-linemeanint-escp) and subsequently replacing S_{ij} with the well-known expression for the line source function

$$S_{ij} = \frac{n_i A_{ij}}{n_j B_{ji} - n_i B_{ij}}$$

leads to

$$\sum_{j>i} \left[n_j A_{ji} \beta_{ji} + (n_j B_{ji} - n_i B_{ij}) \beta_{ji} J_{ji}^{\rm bg} \right]$$
$$- \sum_{j<i} \left[n_i A_{ij} \beta_{ij} + (n_i B_{ij} - n_j B_{ji}) \beta_{ij} J_{ij}^{\rm bg} \right]$$
$$+ \sum_{j \neq i} \left[n_j C_{ji} - n_i C_{ij} \right] = 0$$

A few iteration steps are necessary, because the β_{ij} depends on the optical depths, which depend on the populations. But since this is only a weak dependence, the iteration should converge rapidly.

To use the LVG+EscProb method, the following has to be done:

- Make sure that you use a molecular data file that contains collision rate tables (see Section *INPUT: Molecular/atomic data: The molecule_XXX.inp file(s)*).
- Make sure to provide file(s) containing the number densities of the collision partners, e.g. `numberdens_p-h2.inp` (see Section *INPUT: The number density of collision partners (for non-LTE transfer)*).
- Make sure to link the rate tables to the number density files in `lines.inp` (see Section *INPUT: The line.inp file*).
- Set the `lines_mode=3` in the `radmc3d.inp` file.
- You may want to also specify the maximum number of iterations for non-LTE iterations, by setting `lines_nonlte_maxiter` in the `radmc3d.inp` file. The default is 100 (as of version 0.36). If convergence is not reached within `lines_nonlte_maxiter` iterations, RADMC-3D stops.
- You may want to also specify the convergence criterion for non-LTE iterations, by setting `lines_nonlte_convcrit` in the `radmc3d.inp` file. The default is 1d-2 (which is not very strict! Smaller values may be necessary).
- Specify the gas velocity vector field in the file `gas_velocity.inp` (or `.binp`), see Section *INPUT: The velocity field*. If this file is not present, the gas velocity will be assumed to be 0 everywhere, meaning that you have pure escape probability.
- Specify the 'typical length scale' L at each cell in the file `escprob_lengthscale.inp` (or `.binp`). If this file is not present, then the length scale is assumed to be infinite, meaning that you are back at pure LVG. The format of this file is identical to that of the gas density.

Note that having no `escprob_lengthscale.inp` *nor* `gas_velocity.inp` file in your model directory means that the photons cannot escape at all, and you should find LTE populations (always a good test of the code).

Note that it is essential, when using the Large Velocity Gradient method without specifying a length scale, that the gradients in the velocity field (given in the file `gas_velocity.inp`, see Section *INPUT: The velocity field*) are indeed sufficiently large. If they are zero, then this effectively means that the optical depth in all the lines is assumed to be infinite, which means that the populations are LTE again. If you use LVG but *also* specify a length scale in the `escprob_lengthscale.inp` file, then this danger of unphysically LTE populations is avoided.

NOTE: Currently this method does not yet include radiative exchange with the dust continuum radiation field.

NOTE: Currently this method does not yet include radiative pumping by stellar radiation. Will be included soon.

7.7 Non-LTE Transfer: The optically thin line assumption method

An even simpler non-LTE method is applicable in *very* dilute media, in which the lines are all optically thin. This means that a photon that is emitted by the gas will never be reabsorbed. If this condition is satisfied, then the non-LTE level populations can be computed even easier than in the case of LVG (Section *Non-LTE Transfer: The Large Velocity Gradient (LVG) + Escape Probability (EscProb) method*). No iteration is then required. So to activate this, the following has to be done:

- Make sure that you use a molecular data file that contains collision rate tables (see Section *INPUT: Molecular/atomic data: The molecule_XXX.inp file(s)*).
- Make sure to provide file(s) containing the number densities of the collision partners, e.g. `numberdens_p-h2.inp` (see Section *INPUT: The number density of collision partners (for non-LTE transfer)*).
- Make sure to link the rate tables to the number density files in `lines.inp` (see Section *INPUT: The line.inp file*).
- Set the `lines_mode=4` in the `radmc3d.inp` file (see Section *INPUT: radmc3d.inp*).

NOTE: Currently this method does not yet include radiative pumping by stellar radiation.

*NOTE: This mode does not *make* a model optically thin. Only the populations of the levels are computed under the {bf assumption} that the lines are optically thin. If you subsequently make a spectrum or image of your model, all absorption effects are again included.*

7.8 Non-LTE Transfer: Full non-local modes (FUTURE)

In the near future RADMC-3D will hopefully also feature full non-LTE transfer, in which the level populations are coupled to the full non-local radiation field. Methods such as *lambda iteration* and *accelerated lambda iteration* will be implemented. For nomenclature we will call these 'non-local non-LTE modes'.

For these non-local non-LTE modes the level population calculation is done separately from the image/spectrum ray-tracing: You will run RADMC-3D first for computing the non-LTE populations. RADMC-3D will then write these to file. Then you will call RADMC-3D for making images/spectra. This is very similar to the dust transfer, in which you first call RADMC-3D for the Monte Carlo dust temperature computation, and after that for the ray-tracing. It is, however, different from the *local non-LTE* modes, where the populations are calculated automatically before any image/spectrum ray-tracing, and the populations do not have to be written to file (only if you want to inspect them: Section *Non-LTE Transfer: Inspecting the level populations*).

For now, however, RADMC-3D still does not have the non-local non-LTE modes.

7.9 Non-LTE Transfer: Inspecting the level populations

When doing line radiative transfer it is often useful to inspect the level populations. For instance, you may want to inspect how far from LTE your populations are, or just check if the results are reasonable. There are two ways to do this:

1. When making an image or spectrum, add the command-line option `writepop`, which will make RADMC-3D create output files containing the level population values. Example:

```
radmc3d image lambda 2300 writepop
```

2. Just calling `radmc3d` with the command-line option `calcpop`, which will ask RADMC-3D to compute the populations and write them to file, even without making any images or spectra. Example:

```
radmc3d calcpop
```

NOTE: For (future) non-local non-LTE modes (Section *Non-LTE Transfer: Full non-local modes (FUTURE)*) these level populations will anyway be written to a file, irrespective of the `writepop` command.

The resulting files will have names such as `levelpop_co.dat` (for the CO molecule). The structure is as follows:

```
iformat                                 <=== Typically 1 at present
nrcells
nrlevels_subset
level1  level2 .....                    <=== The level subset selection
popul[level1,1]  popul[level2,1] .....   <=== Populations (for subset) at cell 1
popul[level1,2]  popul[level2,2] .....   <=== Populations (for subset) at cell 2
.
.
popul[level1,nrcells]   popul[level2,nrcells] ....
```

The first number is the format number, which is simply for RADMC-3D to be backward compatible in the future, in case we decide to change/improve the file format. The nrcells is the number of cells.

Then follows the number of levels (written as `nrlevels_subset` above). Note that this is *not necessarily* equal to the number of levels found in the `molecule_co.inp` file (for our CO example). It will only be equal to that if the file has been produced by the command `radmc3d calcpop`. If, however, the file was produced after making an image or spectrum (e.g. through the command `radmc3d image lambda 2300 writepop`), then RADMC-3D will only write out those levels that have been used to make the image or spectrum. See Section *Background information: Calculation and storage of level populations* for more information about this. It is for this reason that the file in fact contains a list of levels that are included (the `level1 level 2 ...` in the above file format example).

After these header lines follows the actual data. Each line contains the populations at a spatial cell in units of cm^{-3}.

This file format is a generalization of the standard format which is described for the example of dust density in Section *INPUT (required for dust transfer): dust_density.inp*. Please read that section for more details, and also on how the format changes if you use 'layers'.

Also the unformatted style is described in Section *INPUT (required for dust transfer): dust_density.inp*. We have, however, here the extra complication that at each cell we have more than one number. Essentially this simply means that the length of the data per cell is larger, so that fewer cells fit into a single record.

7.10 Non-LTE Transfer: Reading the level populations from file

Sometimes you may want to make images and/or spectra of lines based on level populations that you calculated using another program (or calculated using RADMC-3D at some earlier time). You can ask RADMC-3D to read these populations from files with the same name and same format as, for example, `levelpop_co.dat` (for CO) as described in Section *Non-LTE Transfer: Inspecting the level populations*. The way to do this is to add a line:

```
lines_mode = 50
```

to the `radmc3d.inp` file.

You can test that it works by calculating the populations using another `lines_mode` and calling `radmc3d calcpop writepop` (which will produce the `levelpop_xxx.dat` file); then change `lines_mode` to 50, and call `radmc3d image iline 1`. You should see a message that RAMDC-3D is actually reading the populations (and it may, for 3-D models, take a bit of time to read the large file).

Because of the rather lage size of these files for 3-D models, it might be worthwhile to make sure to reduce the number of levels of the `molecule_xx.inp` files to only those you actually need.

7.11 What can go wrong with line transfer?

Even the simple task of performing a ray-tracing line transfer calculation with given level populations (i.e. the so-called *formal transfer equation*) is a non-trivial task in complex 3-D AMR models with possibly highly supersonic motions. I recommend the user to do extensive and critical experimentation with the code and make many simple tests to check if the results are as they are expected to be. In the end a result must be understandable in terms of simple argumentation. If weird effects show up, please do some detective work until you understand why they show up, i.e. that they are either a *real* effect or a numerical issue. There are many numerical artifacts that can show up that are *not* a bug in the code. The code simply does a numerical integration of the equations on some spatial- and wavelength-grid. If the user chooses these grids unwisely, the results may be completely wrong even if the code is formally OK. These possible pitfalls is what this section is about.

So here is a list of things to check:

1. Make sure that the line(s) you want to model are indeed in the molecular data file you use. Also make sure that it/they are included in the line selection (if you are using this option; by default all lines and levels from the molecular/atomic data files are included; see Section *Background information: Calculation and storage of level populations*).

2. If you do LTE line transfer, and you do not let `radmc3d` read in a special file for the partition function, then the partition function will be computed internally by `radmc3d`. The code will do so based on the levels specified in the `molecule_XXX.inp` file for molecule `XXX`. This requires of course that all levels that may be excited at the temperatures found in the model are in fact present in the `molecule_XXX.inp` file. If, for instance, you model 1.3 mm and 2.6 mm rotational lines of CO gas of up to 300 K, and your file `molecule_co.inp` only contains the first three levels because you think you only need those for your 1.3 and 2.6 mm lines, and you *don't* specify the partition function explicitly, then `radmc3d` will compute the partition function for all temperatures including 300 K based on only the first three levels. This is evidently wrong. The nasty thing is: the resulting lines won't be totally absurd. They will just be too bright. But this can easily go undetected by you as the user. So please keep this always in mind. Note that if you make a *selection* of the first three levels (see Section *For experts: Selecting a subset of lines and levels 'manually'*) but the file `molecule_XXX.inp` contains many more levels, then this problem will not appear, because the partition function will be calculated on the original data from the `molecule_XXX.inp` file, not from the selected levels. Of course it is safer to specify the true partition function directly through the file `partitionfunction_XXX.inp` (see Section *INPUT for LTE line transfer: The partition function (optional)*).

3. If you have a model with non-zero gas velocities, and if these gas velocities have cell-to-cell differences that are larger than or equal to the intrinsic (thermal+microturbulent) line width, then the ray-tracing will not be able to pick up signals from intermediate velocities. In other words, because of the discrete gridding of the model, only discrete velocities are present, which can cause numerical problems. See Fig. Fig. 7.2-Left for a pictographic representation of this problem. There are two possible solutions. One is the wavelength band method described in Section *Heads-up: In reality wavelength are actually wavelength bands*. But a more systematic method is the 'doppler catching' method described in Section *Preventing doppler jumps: The 'doppler catching method'* (which can be combined with the wavelength band method of Section *Heads-up: In reality wavelength are actually wavelength bands* to make it even more perfect).

7.12 Preventing doppler jumps: The 'doppler catching method'

If the local co-moving line width of a line (due to thermal/fundamental broadning and/or local subgrid 'microturbulence') is much smaller than the typical velocity fields in the model, then a dangerous situation can occur. This can happen if the co-moving line width is narrower than the doppler shift between two adjacent cells. When a ray is traced, in one cell the line can then have a doppler shift substantially to the blue of the wavelength-of-sight, while in the next cell the line suddenly shifted to the red side. If the intrinsic (= thermal + microturbulent) line width is smaller than these shifts, neither cell gives a contribution to the emission in the ray. See Fig. Fig. 7.1 for a pictographic representation of this problem. In reality the doppler shift between these two cells would be smooth, and thus the line would smoothly pass over the wavelength-of-sight, and thus make a contribution. Therefore the numerical integration may thus go wrong.

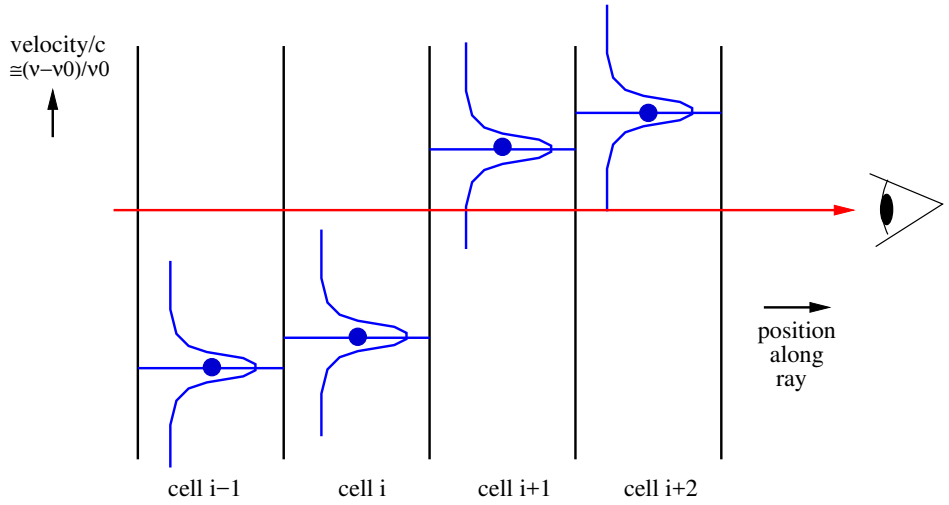

Fig. 7.1: Pictographic representation of the doppler jumping problem with ray-tracing through a model with strong cell-to-cell velocity differences.

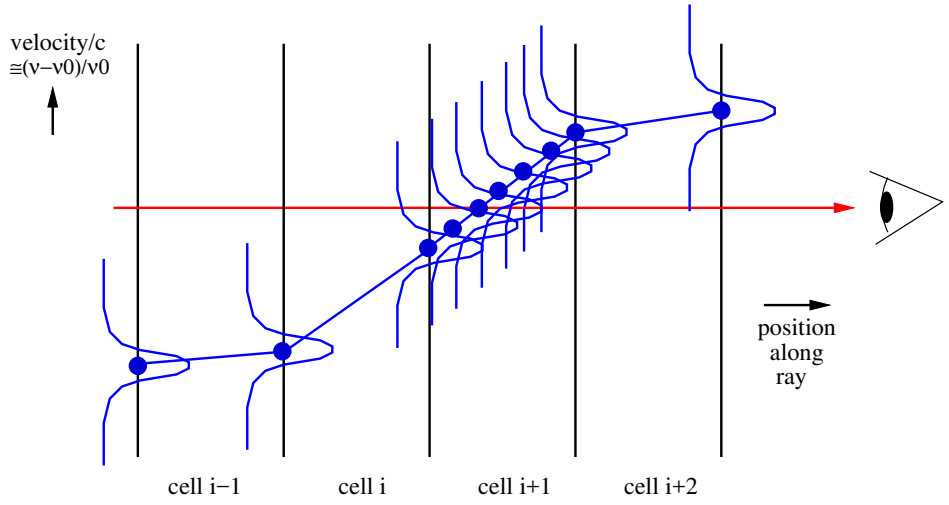

Fig. 7.2: Right: Pictographic representation of the doppler catching method to prevent this problem: First of all, second order integration is done instead of first order. Secondly, the method automatically detects a possibly dangerous doppler jump and makes sub-steps to neatly integrate over the line that shifts in- and out of the wavelength channel of interest.

The problem is described in more detail in Section *Heads-up: In reality wavelength are actually wavelength bands*,

and one possible solution is proposed there. But that solution does not always solve the problem.

RADMC-3D has a special method to catch situations like the above, and when it detects one, to make sub-steps in the integration of the formal transfer equation so that the smooth passing of the line through the wavelength-of-sight can be properly accounted for. Here this is called 'doppler catching', for lack of a better name. The technique was discussed in great detail in Pontoppidan et al. (2009, ApJ 704, 1482). The idea is that the method automatically tests if a line might 'doppler jump' over the current wavelength channel. If so, it will insert substeps in the integration at the location where this danger is present. See Fig. Fig. 7.2 for a pictographic representation of this method. Note that this method can only be used with the second order ray-tracing (see Section *Second order ray-tracing (Important information!)*); in fact, as soon as you switch the doppler catching on, RADMC-3D will automatically also switch on the second order ray-tracing.

To switch on doppler catching, you simply add the command-line option `doppcatch` to the image or spectrum command. For instance:

```
radmc3d spectrum iline 1 widthkms 10 doppcatch
```

(again: you do not need to add `secondorder`, because it is automatic when `doppcatch` is used).

The Doppler catching method will assure that the line is integrated over with small enough steps that it cannot accidently get jumped over. How fine these steps will be can be adjusted with the `catch_doppler_resolution` keyword in the `radmc3d.inp` file. The default value is 0.2, meaning that it will make the integration steps small enough that the doppler shift over each step is not more than 0.2 times the local intrinsic (thermal+microturbulent) line width. That is usually enough, but for some problems it might be important to ensure that smaller steps are taken. By adding a line:

```
catch_doppler_resolution = 0.05
```

to the `radmc3d.inp` file you will ensure that steps are small enough that the doppler shift is at most 0.05 times the local line width.

So why is doppler catching an *option*, i.e. why would this not be standard? The reason is that doppler catching requires second order integration, which requires RADMC-3D to first map all the cell-based quantities to the cell-corners. This requires extra memory, which for very large models can be problematic. It also requires more CPU time to calculate images/spectra with second order integration. So if you do not need it, i.e. if your velocity gradients are not very steep compared to the intrinsic line width, then it saves time and memory to not use doppler catching.

It is, however, important to realize that doppler catching is not the golden bullet. Even with doppler catching it might happen that some line flux is lost, but this time as a result of too low *image resolution*. This is less likely to happen in problems like ISM turbulence, but it is pretty likely to happen in models of rotating disks. Suppose we have a very thin local line width (i.e. low gas temperature and no microturbulence) in a rotating thin disk around a star. In a given velocity channel (i.e. at a given observer-frame frequency) a molecular line in the disk emits only in a very thin 'ear-shaped' ring or band in the image. The thinner the intrinsic line width, the thinner the band on the image. See Pontoppidan et al. (2009, ApJ 704, 1482) and Pavlyuchenkov et al. (2007, ApJ 669, 1262) for example. If the pixel-resolution of the image is smaller than that of this band, the image is simply underresolved. This has nothing to do with the doppler jumping problem, but can be equally devastating for the results if the user is unaware of this. There appears to be only one proper solution: assure that the pixel-resolution of the image is sufficiently fine for the problem at hand. This is easy to find out: The image would simply look terribly noisy if the resolution is insufficient. However, if you are not interested in the images, but only in the spectra, then some amount of noisiness in the image (i.e. marginally sufficient resolution) is OK, since the total flux is an integral over the entire image, smearing out much of the noise. It requires some experimentation, though.

Here are some additional issues to keep in mind:

- The doppler catching method uses second order integration (see Section *Second order ray-tracing (Important information!)*), and therefore all the relevant quantities first have to be interpolated from the cell centers to the cell corners. Well inside the computational domain this amounts to linear interpolation. But at the edges of the domain it would require *extra* polation. In 1-D this is more easily illustrated, because there the cell corners are in

fact cell interfaces. Cells i and $i+1$ share cell interface $i+1/2$. If we have N cells, i.e. cells $i = 1, \cdots, N$, then we have $N+1$ interfaces, i.e. interfaces $i = \frac{1}{2}, \cdots, N+\frac{1}{2}$. To get physical quantities from the cell centers to cell interfaces $i = \frac{3}{2}, \cdots, N-\frac{1}{2}$ requires just interpolation. But to find the physical quantities at cell interfaces $i = \frac{1}{2}$ and $i = N+\frac{1}{2}$ one has to extrapolate or simply take the values at the cell centers $i = 1$ and $i = N$. RADMC-3D does not do extrapolation but simply takes the average values of the nearest cells. Also the gas velocity is treated like this. This means that over the edge cells the gradient in the gas velocity tends to be (near) 0. Since for the doppler catching it is the gradient of the velocity that matters, this might yield some artifacts in the spectrum if the density in the border cells is high enough to produce substantial line emission. Avoiding this numerical artifact is relatively easy: One should then simply put the number density of the molecule in question to zero in the boundary cells.

- If you are using RADMC-3D on a 3-D (M)HD model which has strong shocks in its domain, then one must be careful that (magneto-)hydrodynamic codes tend to smear out the shock a bit. This means that there will be some cells that have intermediate density and velocity in the smeared out region of the shock. This is unphysical, but an intrinsic numerical artifact of numerical hydrodynamics codes. This might, under some conditions, lead to unphysical signal in the spectrum, because there would be cells at densities, temperatures and velocities that would be in between the values at both sides of the shock and would, in reality, not be there. It is very difficult to avoid this problem, and even to find out if this problem is occurring and by how much. One must simply be very careful of models containing strong shocks and do lots of testing. One way to test is to use the doppler catching method and vary the doppler catching resolution (using the `catch_doppler_resolution` keyword in `radmc3d.inp`).
- If using line transfer in spherical coordinates using doppler catching, the linear interpolation of the line shift between the beginning and the end of a segment may not always be enough to accurately prevent doppler jumps. This is because in addition to the physical gradient of gas velocity, the projected gas velocity along a ray changes also along the ray due to the geometry (the use of spherical coordinates). Example: a spherically symmetric radially outflowing wind with constant outward velocity $v_r =$ is constant, the 3-D *vector* $\vec{v}$ is not constant, since it always points outward. A ray through this wind will thus have a varying $\vec{n} \cdot \vec{v}$ along the ray. In the cell where the ray reaches its closest approach to the origin of the coordinate system the $\vec{n} \cdot \vec{v}$ will vary the strongest. This may be such a strong effect that it could affect the reliability of the code. *As of version 0.41 of this code a method is in place to prevent this*. It is switched on by default, but it can be switched off manually for testing purposes. See Section *Second order integration in spherical coordinates: a subtle issue* for details.

7.13 Background information: Calculation and storage of level populations

If RADMC-3D makes an image or a spectrum with molecular (or atomic) lines included, then the level populations of the molecules/atoms have to be computed. In the standard method of ray-tracing of images or spectra, these level populations are first calculated in each grid cell and stored in a global array. Then the raytracer will render the image or spectrum.

The storage of the level populations is a tricky matter, because if this is done in the obvious manner, it might require a huge amount of memory. This would then prevent us from making large scale models. For instance: if you have a molecule with 100 levels in a model with 256x256x256 $\simeq 1.7 \times 10^7$ cells, the global storage for the populations alone (with each number in double precision) would be roughly 100x8x256x256x256 $\simeq$ 13 Gigabyte.

However, if you intend to make a spectrum in just 1 line, you do not need all these level populations. To stick to the above example, let us take the CO 1-0 line, which is then line 1 and which connects levels $J = 1$ and $J = 0$, which are levels 2 and 1 in the code (if you use the Leiden database CO data file). Once the populations have been computed, we only need to store the levels 1 and 2. This would then require 2x8x256x256x256 $\simeq$ 0.26 Gigabyte, which would be *much* less memory-costly.

As of version 0.29 RADMC-3D automatically figures out which levels have to be stored in a global array, in order to be able to render the images or the spectrum properly. RADMC-3D will go through all the lines of all molecules and

checks if they contribute to the wavelength(s), of the image(s) or the spectrum. Once it has assembled a list of 'active' lines, it will make a list of 'active' levels that belong to these lines. It will then declare this to be the 'subset' of levels for which the populations will be stored globally.

In other words: RADMC-3D now takes care of the memory-saving storage of the populations automatically.

How does RADMC-3D decide whether a line contributes to some wavelength λ? A line i with line center λ_i is considered to contribute to an image at wavelength λ if

$$|\lambda_i - \lambda| \leq C_{\rm margin} \Delta\lambda_i$$

where $\Delta\lambda_i$ is the line width (including all contributions) and $C_{\rm margin}$ is a constant. By default

$$C_{\rm margin} = 12$$

But you can change this to another value, say 24, by adding in the `radmc3d.inp` file a line containing, e.g. `lines_widthmargin = 24`.

You can in fact get a dump of the level populations that have been computed and used for the image(s)/spectrum you created, by adding `writepop` on the command line. Example:

```
radmc3d spectrum iline 1 widthkms 10 writepop
```

This then creates (in addition to the spectrum) a file called (for our example of the CO molecule) `levelpop_co.dat`. Here is how you can read this data in Python:

```
from radmc3d_tools import simpleread
data = simpleread.read_levelpop()
```

The `data` object then contains `data.pop` and `data.relpop`, which are the level populations in $1/cm^3$ and in normalized form.

If, for some reason, you want always *all* levels to be stored (and you can afford to do so with the size of your computer's memory), you can make RADMC-3D do so by adding `noautosubset` as a keyword to the command line, or by adding `lines_autosubset = 0` to the `radmc3d.inp` file. However, for other than code testing purposes, it seems unlikely you will wish to do this.

7.14 In case it is necessary: On-the-fly calculation of populations

There might be rare circumstances in which you do not want to have to store the level populations in a global array. For example: you are making a spectrum of the CO bandhead, in which case you have many tens of lines in a single spectrum. If your model contains 256x256x256 cells (see example in Section *Background information: Calculation and storage of level populations*) then this might easily require many Gigabytes of memory just to store the populations.

For the LTE, LVG and optically thin level population modes there is a way out: You can force RADMC-3D to compute the populations *on-the-fly* during the ray-tracing, which does not require a global storage of the level populations.

The way to do this is simple: Just make the `lines_mode` negative. So for on-the-fly LTE mode use `lines_mode=-1`, for on-the-fly user-defined populations mode use `lines_mode=-2`, for on-the-fly LVG mode use `lines_mode=-3` and for on-the-fly optically thin populations use `lines_mode=-4`.

NOTE: The drawback of this method is that, under certain circumstances, it can slow down the code dramatically. This slow-down happens if you use e.g. second-order integration (Section *Second order ray-tracing (Important information!)*) and/or doppler catching (Section *Preventing doppler jumps: The 'doppler catching method'*) together with non-trivial population solving methods like LVG. So please use the on-the-fly method only when you are forced to do so (for memory reasons).

7.15 For experts: Selecting a subset of lines and levels 'manually'

As explained in Section *Background information: Calculation and storage of level populations*, RADMC-3D automatically makes a selection of levels for which it will allocate memory for the global level population storage.

If, for some reason, you wish to make this selection yourself 'by hand', this can also be done. However, please be informed that there are very few circumstances under which you may want to do this. The automatic subset selection of RADMC-3D is usually sufficient!

If you decided to really want to do this, here is how:

1. Switch off the automatic subset selection by adding `noautosubset` as a keyword to the command line, or by adding `lines_autosubset = 0` to the `radmc3d.inp` file.
2. In the `lines.inp` file, for each molecule, modify the '0 0' (the first two zeroes after 'leiden') in the way described below.

In Section *INPUT: The line.inp file* you can see that each molecule has a line like:

```
co   leiden   0  0  0
```

or so (here for the example of CO). In Section *INPUT: The line.inp file* we explained the meaning of the third number, but we did not explain the meaning of the first and second ones. These are meant for this subset selection. If we want to store only the first 10 levels of the CO molecule, then replace the above line with:

```
co   leiden   0  10  0
```

If you want to select specific levels (let us choose the `ilevel=3` and `ilevel=4` levels of the above example), then write:

```
co   leiden   1  2  0
3 4
```

The '1' says that a list of levels follows, the '2' says that two levels will be selected and the next line with '3' and '4' say that levels 3 and 4 should be selected.

CHAPTER

EIGHT

MAKING IMAGES AND SPECTRA

Much has already been said about images and spectra in the chapters on dust radiative transfer and line radiative transfer. But here we will combine all this and go deeper into this material. So presumably you do not need to read this chapter if you are a beginning user. But for more sophisticated users (or as a reference manual) this chapter may be useful and presents many new features and more in-depth insight.

8.1 Basics of image making with RADMC-3D

Images and spectra are typically made after the dust temperature has been determined using the thermal Monte Carlo run (see Chapter *Dust continuum radiative transfer*). An image can now be made with a simple call to `radmc3d`:

```
radmc3d image lambda 10
```

This makes an image of the model at wavelength $\lambda = 10\mu$. We refer to Section *OUTPUT: image.out or image_****.out* for details of this file and how to interpret the content. See Chapter *Python analysis tool set* for an extensive Python tools that make it easy to read and handle these files. The vantage point is at infinity at a default inclination of 0, i.e. pole-on view. You can change the vantage point:

```
radmc3d image lambda 10 incl 80 phi 30
```

which now makes the image at inclination 80 degrees away from the z-axis (i.e. almost edge-on with respect to the x-y plane), and rotates the location of the observer by 30 degrees clockwise around the z-axis (Here clockwise is defined with the z-axis pointing toward you, i.e. with respect to the observer the model is rotated counter-clockwise around the z-axis by 30 degrees).

You can also rotate the camera in the image plane with

```
radmc3d image lambda 10 incl 45 phi 30 posang 20
```

which rotates the camera by 20 degrees clockwise (i.e. the image rotates counter-clockwise). Figures Fig. 8.1 and Fig. 8.2 show the definitions of all three angles. Up to now the camera always pointed to one single point in space: the point (0,0,0). You can change this:

```
radmc3d image lambda 10 incl 45 phi 30 posang 20 pointau 3.2 0.1 0.4
```

which now points the camera at the point (3.2,0.1,0.4), where the numbers are in units of AU. The same can be done in units of parsec:

```
radmc3d image lambda 10 incl 45 phi 30 posang 20 pointpc 3.2 0.1 0.4
```

Note that `pointau` and `pointpc` are always 3-D positions specified in cartesian coordinates. This remains also true when the model-grid is in spherical coordinates and/or when the model is 2-D (axisymmetric) or 1-D (spherically symmetric): 3-D positions are always specified in x,y,z.

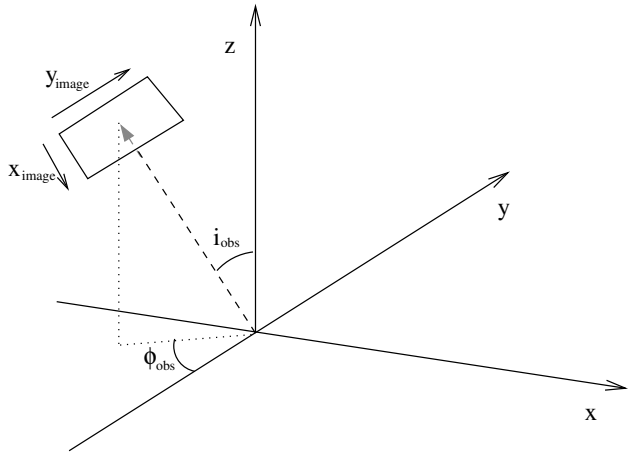

Fig. 8.1: Figure depicting how the angles 'incl' and 'phi' place the camera for images and spectra made with RADMC-3D. The code uses a right-handed coordinate system. The figure shows from which direction the observer is looking at the system, where $i_{\rm obs}$ is the 'incl' keyword and $\phi_{\rm obs}$ is the 'phi' keyword. The $x_{\rm image}$ and $y_{\rm image}$ are the horizontal (left-to-right) and vertical (bottom-to-top) coordinates of the image. For $i_{\rm obs} = 0$ and $\phi_{\rm obs} = 0$ the $x_{\rm image}$ aligns with the 3-D x-coordinate and $y_{\rm image}$ aligns with the 3-D y-coordinate.

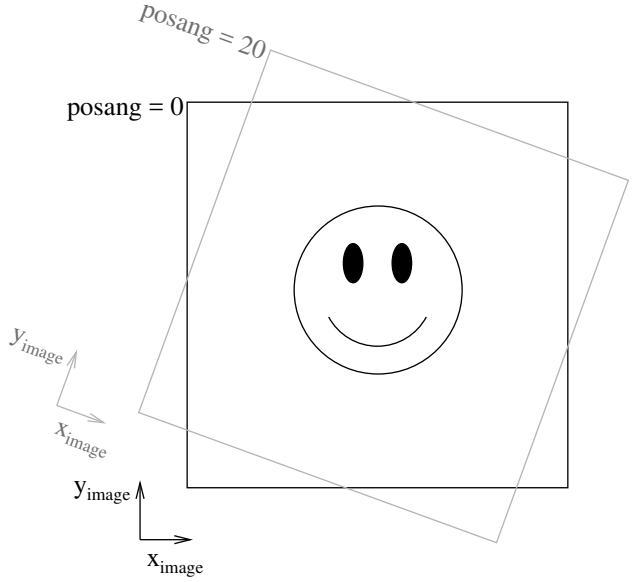

Fig. 8.2: This figure shows the way the camera can be rotated in the image plane using 'posang'. Positive 'posang' means that the camera is rotated clockwise, so the object shown is rotated counter-clockwise with respect to the image coordinates.

Let's now drop the pointing again, and also forget about the `posang`, and try to change the number of pixels used:

```
radmc3d image lambda 10 incl 45 phi 30 npix 100
```

This will make an image of 100x100. You can also specify the x- and y- direction number of pixels separately:

```
radmc3d image lambda 10 incl 45 phi 30 npixx 100 npixy 30
```

Now let's forget again about the number of pixels and change the size of the image, i.e. which zooming factor we have:

```
radmc3d image lambda 10 incl 45 phi 30 sizeau 30
```

This makes an image which has 30 AU width and 30 AU height (i.e. 15 AU from the center in both directions). Same can be done in units of parsec

```
radmc3d image lambda 10 incl 45 phi 30 sizepc 30
```

Although strictly speaking redundant is the possibility to zoom-in right into a selected box in this image:

```
radmc3d image lambda 10 incl 45 phi 30 zoomau -10 -4. 0 6
```

which means that we zoom in to the box given by $-10 \leq x \leq -4$ AU and $0 \leq y \leq 6$ AU on the original image (note that `zoomau -15 15 -15 15` gives the identical result as `sizeau 30`). This possibility is strictly speaking redundant, because you could also change the `pointau` and `sizeau` to achieve the same effect (unless you want to make a non-square image, in which case this is the only way). But it is just more convenient to do any zooming-in this way. Please note that when you make non-square images with `zoomau` or `zoompc`, the code will automatically try to keep the pixels square in shape by adapting the number of pixels in x- or y- direction in the image and adjusting one of the sizes a tiny bit to assure that both x- and y- size are an integer times the pixel size. These are very small adjustments (and only take place for non-square zoom-ins). If you want to force the code to take *exactly* the zoom area, and you don't care that the pixels then become slightly non-square, you can force it with `truezoom`:

```
radmc3d image lambda 10 incl 45 phi 30 sizeau 30 zoomau -10 -4. 0 3.1415 truezoom
```

If you do not want the code to adjust the number of pixels in x- and y- direction in its attempt to keep the pixels square:

```
radmc3d image lambda 10 incl 45 phi 30 sizeau 30 zoomau -10 -4. 0 3.1415 npixx 100␣
↪npixy 4 truepix
```

Now here are some special things. Sometimes you would like to see an image of just the dust, not including stars (for stars in the image: see Section *Stars in the images and spectra*). So blend out the stars in the image, you use the `nostar` option:

```
radmc3d image lambda 10 incl 45 phi 30 nostar
```

Another special option is to get a 'quick image', in which the code does not attempt assure flux conservation in the image (see Section *The issue of flux conservation: recursive sub-pixeling* for the issue of flux conservation). Doing the image with flux conservation is slower than if you make it without flux conservation. Making an image without flux conservation can be useful if you want to have a 'quick look', but is strongly discouraged for actual scientific use. But for a quick look you can do:

```
radmc3d image lambda 10 incl 45 phi 30 nofluxcons
```

If you want to produce images with a smoother look (and which also are more accurate), you can ask RADMC-3D to use second order integration for the images:

```
radmc3d image lambda 10 incl 45 phi 30 secondorder
```

NOTE: The resulting intensities may be slightly different from the case when first order integration (default) is used, in particular if the grid is somewhat course and the objects of interest are optically thick. Please consult Section *Second order ray-tracing (Important information!)* for more information.

Important for polarized radiative transfer: If you use polarized scattering, then you may want to creat images with polarization information in them. You have to tell RADMC-3D to do this by adding `stokes` to the command line:

```
radmc3d image lambda 10 incl 45 phi 30 stokes
```

The definitions of the Stokes parameters (orientation etc) can be found in Section *Definitions and conventions for Stokes vectors* and the format of `image.out` in this case can be found in Section *OUTPUT: image.out or image_****.out*.

Note: All the above commands call `radmc3d` separately. If it needs to load a large model (i.e. a model with many cells), then the loading may take a long time. If you want to make many images in a row, this may take too much time. Then it is better to call `radmc3d` as a child process and pass the above commands through the biway pipe (see Chapter chap-child-mode).

8.2 Making multi-wavelength images

Sometimes you want to have an image of an object at multiple wavelength simultaneously. Rather than calling RADMC-3D separately to make an image for each wavelength, you can make all images in one command. The only thing you have to do is to tell RADMC-3D which wavelengths it should take. There are various different ways you can tell RADMC-3D what wavelengths to take. This is described in detail in Section *Specifying custom-made sets of wavelength points for the camera*. Here we will focus as an example on just one of these methods. Type, for instance,

```
radmc3d image incl 45 phi 30 lambdarange 5. 20. nlam 10
```

This will create 10 images at once, all with the same viewing perspective, but at 10 wavelengths regularly distributed between 5 μmu`m. All images are written into a single file, `image.out` (See Section *OUTPUT: image.out or image_****.out* for its format).

In Python you simply type:

```
from radmc3dPy import image
a=image.readImage()
```

and you will get all images at once. To plot one of them:

```
image.plotImage(image=a,ifreq=3)
```

which will plot image number 3 (out of images number 0 to 9). To find out which wavelength this image is at:

```
print(a.wav[3])
```

which will return 7.9370053 in this example.

Note that all of the commands in Section *Basics of image making with RADMC-3D* are of course also applicable to multi-wavelength images, except for the `lambda` keyword, as this conflicts with the other method(s) of specifying the wavlengths of the images. Now please turn to Section *Specifying custom-made sets of wavelength points for the camera* for more information on how to specify the wavelengths for the multiple wavelength images.

8.3 Making spectra

The standard way of making a spectrum with `radmc3d` is in fact identical to making 100x100 pixel images with flux conservation (i.e. recursive sub-pixeling, see Section *The issue of flux conservation: recursive sub-pixeling*) at multiple frequencies. You can ask `radmc3d` to make a *spectral energy distribution (SED)* with the command

```
radmc3d sed incl 45 phi 30
```

This will put the observer at inclination 45 degrees and angle phi 30 degrees, and make a spectrum with wavelength points equal to those listed in the `wavelength_micron.inp` file.

The output will be a file called `spectrum.out` (see Section *OUTPUT: spectrum.out*).

You can also make a spectrum on a set of wavelength points of your own choice. There are multiple ways by which you can specify the set of frequencies/wavelength points for which to make the spectrum: they are described in Section *Specifying custom-made sets of wavelength points for the camera*. If you have made your selection in such a way, you can make the spectrum at this wavelength grid by

```
radmc3d spectrum incl 45 phi 30 <COMMANDS FOR WAVELENGTH SELECTION>
```

where the last stuff is telling `radmc3d` how to select the wavelengths (Section *Specifying custom-made sets of wavelength points for the camera*). An example:

```
radmc3d spectrum incl 45 phi 30 lambdarange 5. 20. nlam 100
```

will make a spectrum with a regular wavelength grid between 5 and 20 μm and 100 wavelength points. But see Section *Specifying custom-made sets of wavelength points for the camera* for more details and options.

The output file `spectrum.out` will have the same format as for the `sed` command.

Making a spectrum can take RADMC-3D some time, especially in the default mode, because it will do its best to shoot its rays to pick up all cells of the model (see Section *The solution: recursive sub-pixeling*). In particularly in spherical coordinates RADMC-3D can be perhaps *too* conservative (and thus slow). For spherical coordinates there are ways to tell RADMC-3D to be somewhat less careful (and thereby faster): see Section *Recursive sub-pixeling in spherical coordinates*.

Note that you can adjust the fine-ness of the images from which the spectrum is calculated using `npix`:

```
radmc3d sed incl 45 phi 30 npix 2
```

What this does is use a 2x2 pixel image instead of a 100x100 pixel image as the starting resolution. Of course, if it would really be just a 2x2 pixel image, the flux would be entirely unreliable and useless. However, using the above mentioned 'sub-pixeling' (see Section *The solution: recursive sub-pixeling*) it will automatically try to recursively refine these pixels until the required level of refinement is reached. So under normal circumstances even npix=2 is enough, and in earlier versions of RADMC-3D this 2x2 top-level image resolution was in fact used as a starting point. But for safety reasons this has now been changed to the standard 100x100 resolution which is also the default for normal images. If 100x100 is not enough, try e.g.:

```
radmc3d sed incl 45 phi 30 npix 400
```

which may require some patience.

8.3.1 What is 'in the beam' when the spectrum is made?

As mentioned above, a spectrum is simply made by making a rectangular image at all the wavelengths points, and integrating over these images. The resulting fluxes at each wavelength point is then the spectral flux at that wavelength point. This means that the integration area of flux for the spectrum is (a) rectangular and (b) of the same size at all wavelengths.

So, what *is* the size of the image that is integrated over? The answer is: it is the same size as the default size of an image. In fact, if you make a spectrum with

```
radmc3d spectrum incl 45 phi 30 lambdarange 5. 20. nlam 10
```

then this is the same as if you would type

```
radmc3d image incl 45 phi 30 lambdarange 5. 20. nlam 10
```

and read in the file `image.out` in into Python (see Section *Making multi-wavelength images*) or your favorite other data language, and integrate the images to obtain fluxes. In other words: the command `spectrum` is effectively the same as the command `image` but then instead of writing out an `image.out` file, it will integrate over all images and write a `spectrum.out` file.

If you want to have a quick look at the area over which the spectrum is to be computed, but you don't want to compute all the images, just type e.g.:

```
radmc3d image lambda 10 incl 45 phi 30
```

then you see an image of your source at $\lambda = 10\mu$m, and the integration area is precisely this area - at all wavelengths. Like with the images, you can specify your viewing area, and thus your integration area. For instance, by typing

```
radmc3d image lambda 10 incl 45 phi 30 zoomau -2 -1 -0.5 0.5
```

makes an image of your source at $\lambda = 10\mu$m at inclination 45 degrees, and orientation 30 degrees, and zooms in at an are from -2 AU to -1 AU in x-direction (in the image) and from -0.5 AU to 0.5 AU in y-direction (in the image). To make an SED within the same integration area:

```
radmc3d sed incl 45 phi 30 zoomau -2 -1 -0.5 0.5
```

In this case we have an SED with a 'beam size' of 1 AU diameter, but keep in mind that the 'beam' is square, not circular.

8.3.2 Can one specify more realistic 'beams'?

Clearly, a wavelength-independent beam size is unrealistic, and also the square beam is unrealistic. So is there a way to do this better? In reality one should really know exactly how the object is observed and how the flux is measured. If you use an interferometer, for instance, maybe your flux is meant to be the flux in a single synthesized beam. For a spectrum obtained with a slit, the precise flux is dependent on the slit width: the wider the slit, the more signal you pick up, but it is a signal from a larger area.

So if you really want to be sure that you know exactly what you are doing, then the best method is to do this youself by hand. You make multi-wavelength images:

```
radmc3d image incl 45 phi 30 lambdarange 5. 20. nlam 10
```

and integrate over the images in the way you think best mimics the actual observing procedure. You can do so, for instance, in Python. See Section *Making multi-wavelength images* for more information about multi-wavelength images.

But to get some reasonable estimate of the effect of the wavelength-dependent size and circular geometry of a 'beam', RADMC-3D allows you to make spectra with a simplistic circular mask, the radius of which can be specified as a function of wavelength in the file `aperture_info.inp` (see Section *INPUT: aperture_info.inp*). This file should contain a table of mask radii at various wavelengths, and when making a spectrum with the command-line keyword `useapert` the mask radii will be found from this table by interpolation. In other words: the wavelength points of the `aperture_info.inp` file do not have to be the same as those used for the spectrum. But their range *must* be larger or equal than the range of the wavelengths used for the spectrum, because otherwise interpolation does not work. In the most extreme simplistic case the `aperture_info.inp` file contains merely two values: one for a very short wavelength (shorter than used in the spectrum) and one for a very long wavelength (longer than used in the spectrum). The interpolation is then done double-logarithmically, so that a powerlaw is used between sampling points. So if you use a telescope with a given diameter for the entire range of the spectrum, two sampling points would indeed suffice.

You can now make the spectrum with the aperture in the following way:

```
radmc3d sed useapert dpc 100
```

The keyword `dpc 100` is the distance of the observer in units of parsec, here assumed to be 100. This distance is necessary because the aperture information is given in arcseconds, and the distance is used to convert this is image size.

Important note: Although you specify the distance of the observer here, the `spectrum.out` file that is produced is still normalized to a distance of 1 parsec.

Note also that in the above example you can add any other keywords as shown in the examples before, as long as you add the `useapert` keyword and specify `dpc`.

A final note: the default behavior of RADMC-3D is to use the square field approach described before. You can explicitly turn off the use of apertures (which may be useful in the child mode of RADMC-3D) with the keyword `noapert`, but normally this is not necessary as it is the default.

8.4 Specifying custom-made sets of wavelength points for the camera

If you want to make a spectrum at a special grid of wavelengths/frequencies, with the `spectrum` command (see Section *Making spectra*), you must tell `radmc3d` which wavelengths you want to use. Here is described how to do this in various ways.

8.4.1 Using `lambdarange` and (optionally) `nlam`

The simplest way to choose a set of wavelength for a spectrum is with the `lambdarange` and (optionally) `nlam` command line options. Here is how to do this:

```
radmc3d spectrum incl 45 phi 30 lambdarange 5. 20.
```

This will make a spectrum between 5 and 20 μmu`m. You can change the number of wavelength points as well:

```
radmc3d spectrum incl 45 phi 30 lambdarange 5. 20. nlam 1000
```

This will do the same, but creates a spectrum of 1000 wavelength points.

You can use the `lambdarange` and `nlam` options also for multi-wavelength images:

```
radmc3d image incl 45 phi 30 lambdarange 5. 20. nlam 10
```

but it is wise to choose `nlam` small, because otherwise the output file, containing all the images, would become too large.

8.4.2 Using `allwl`

You can also tell RADMC-3D to simply make an image at all of the wavelengths in the `wavelength_micron.inp` file:

```
radmc3d image incl 45 phi 30 allwl
```

The keyword `allwl` stands for 'all wavelengths'.

8.4.3 Using `loadcolor`

By giving the command `loadcolor` on the command line, `radmc3d` will search for the file `color_inus.inp`. This file contains integers selecting the wavelengths from the file `wavelength_micron.inp`. The file is described in Section *The color_inus.inp file (required with comm-line option 'loadcolor')*.

8.4.4 Using `loadlambda`

By giving the command `loadlambda` on the command line, `radmc3d` will search for the file `camera_wavelength_micron.inp`. This file contains a list of wavelengths in micron which constitute the grid in wavelength. This file is described in Section *INPUT (optional): camera_wavelength_micron.inp*.

8.4.5 Using `iline`, `imolspec` etc (for when lines are included)

By adding for instance `iline 3` to the command line you specify a window around line number 3 (by default of molecule 1). By also specifying for instance `imolspec 2` you select line 3 of molecule 2. By adding `widthkms 3` you specify how wide the window around the line should be (3 km/s in this example). With `vkms 2` you set the window offset from line center by 2 km/s in this example. By adding `linenlam 30` you set the number of wavelength points for this spectrum to be 30 in this example. So a complete (though different) example is:

```
radmc3d spectrum incl 45 phi 30 iline 2 imolspec 1 widthkms 6.0 vkms 0.0 linenlam 40
```

8.5 Heads-up: In reality wavelength are actually wavelength bands

In a radiative transfer program like `RADMC-3D` the images or spectral fluxes are calculated at *exact* wavelengths. This would correspond to making observations with infinitely narrow filters, i.e. filters with $\Delta\lambda = 0$. This is not how real observations work. In reality each wavelength channel has a finite width $\Delta\lambda$ and the measured flux (or image intensity) is an average over this range. To be even more precise, each wavelength channel i has some profile $\Phi_i(\lambda)$ defined such that

$$\int_0^\infty \Phi_i(\lambda) d\lambda = 1$$

For wide filters such as the standard photometric systems (e.g. UVBRI in the optical and JHK in the near infrared) these profiles span ranges with a width of the order of λ itself. Many instruments have their own set of filters. Usually one can download these profiles as digital tables. It can, under some circumstances, be important to include a treatment of

these profiles in the model predictions. As an example take the N band. This is a band that includes the 10 μlambda_i` for $1 \leq i \leq n$, where n is the number of wavelength samples, and then compute the filter-averaged flux with:

$$F_{band} = \int_0^\infty \Phi_i(\lambda) F(\lambda) d\lambda = \sum_{i=1}^{n} \Phi_i F_i \delta\lambda$$

where $\delta\lambda$ is the wavelength sampling spacing used. The same is true for image intensities. `RADMC-3D` will not do this automatically. You have to tell it the λ_i sampling points, let it make the images or fluxes, and you will then have to perform this sum yourself. *Note that this will not always be necessary!* In many (most?) cases the dust continuum is not expected to change so dramatically over the width of the filter that such degree of accuracy is required. So you are advised to think carefully: 'do I need to take care of this or can I make do with a single wavelength sample for each filter?'. If the former, then do the hard work. If the latter: then you can save time.

8.5.1 Using channel-integrated intensities to improve line channel map quality

When you make line channel maps you may face a problem that is somehow related to the above issue of single-λ-sampling versus filter-integrated fluxes/intensities. If the model contains gas motion, then doppler shift will shift the line profile around. In your channel map you may see regions devoid of emission because the lines have doppler shifted out of the channel you are looking at. However, as described in Section *What can go wrong with line transfer?*, if the intrinsic line width of the gas is smaller than the cell-to-cell velocity differences, then the channel images may look very distorted (they will look 'blocky', as if there is a bug in the code). Please refer to Section *What can go wrong with line transfer?* for more details and updates on this important, but difficult issue. It is not a bug, but a general problem with ray-tracing of gas lines in models with large velocity gradients.

As one of the β-testers of `RADMC-3D`, Rahul Shetty, has found out, this problem can often be alleviated a lot if you treat the finite width of a channel. By taking multiple λ_i points in each wavelength channel (i.e. multiple v_i points in each velocity channel) and simply averaging the intensities (i.e. assuming a perfectly square Φ function) and taking the width of the channels to be not smaller (preferably substantially wider) than the cell-to-cell velocity differences, this 'blocky noise' sometimes smoothes out well. However, it is always safer to use the 'doppler catching' mode (see Section *Preventing doppler jumps: The 'doppler catching method'*) to automatically prevent such problems (though this mode requires more computer memory).

8.6 The issue of flux conservation: recursive sub-pixeling

8.6.1 The problem of flux conservation in images

If an image of nx×ny pixels is made simply by ray-tracing one single ray for each pixel, then there is the grave danger that certain regions with high refinement (for instance with AMR in cartesian coordinates, or near the center of the coordinate system for spherical coordinates) are not properly 'picked up'. An example: suppose we start with a circumstellar disk ranging from 0.1 AU out to 1000 AU. Most of the near infrared flux comes from the very inner regions near 0.1 AU. If an image of the disk is made with 100x100 pixels and image half-size of 1000 AU, then none of the pixels in fact pass through these very bright inner regions, for lack of spatial resolution. The problem is then that the image, when integrated over the entire image, does not have the correct flux. What *should* be is that the centermost pixels contain the flux from this innermost region, even if these pixels are much larger than the entire bright region. In other words, the intensity of these pixels must represent the average intensity, averaged over the entire pixel. Strictly speaking one should trace an infinite continuous 2-D series of rays covering the entire pixel and then average over all these rays; but this is of course not possible. In practice we should find a way to estimate the average intensity with only a finite number of rays.

8.6.2 The solution: recursive sub-pixeling

In RADMC-3D what we do is to use some kind of 'adaptive grid refinement' of the pixels of the image. For each pixel in the image the intensity is computed through a call to a subroutine called `camera_compute_one_pixel()`. In this subroutine a ray-tracing is performed for a ray that ends right in the middle of our pixel. During the ray-tracing, however, we check if we pass regions in the model grid that have grid cells with sizes S that are smaller than the pixel size divided by some factor $f_{\rm ref}$ (where pixel size is, like the model grid size S itself, measured in centimeters. If this is found *not* to be true, then the pixel size was apparently ok, and the intensity resulting from the ray-tracing is now returned as the final intensity of this pixel. If, however, this condition *is* found to be true, then the result of this ray is rejected, and instead 2x2 sub-pixels are computed by calling the `camera_compute_one_pixel()` subroutine recursively. We thus receive the intensity of each of these four sub-pixels, and we return the average of these 4 intensities.

Note, by the way, that each of these 2x2 subpixels may be split even further into 2x2 sub-pixels etc until the desired resolution is reached, i.e. until the condition that S is larger or equal to the pixel size divided by $f_{\rm ref}$ is met. This is illustrated in Fig. Fig. 8.3. By this recursive calling, we always end up at the top level with the average intesity of the entire top-level pixel. This method is very similar to quad-tree mesh refinement, but instead of retaining and returning the entire complex mesh structure to the user, this method only returns the final average intensity of each (by definition top level) pixel in the image. So the recursive sub-pixeling technique described here is all done internally in the RADMC-3D code, and the user will not really notice anything except that this sub-pixeling can of course be computationally more expensive than if such a method is not used.

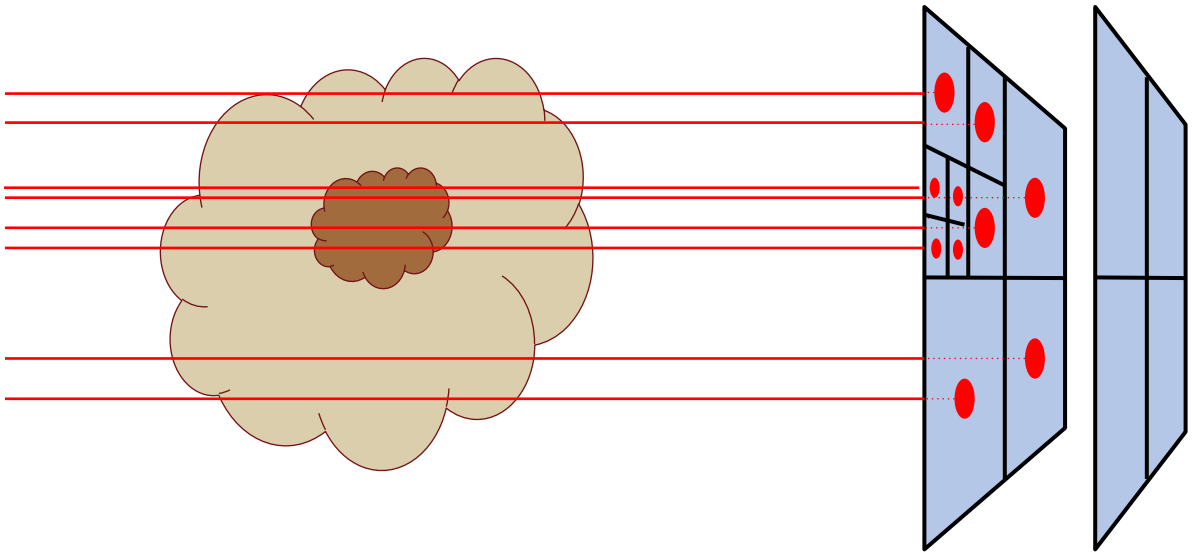

Fig. 8.3: Pictographic representation of how the recursive sub-pixeling for images works. Pixels are recursively split in 2x2 subpixels as far as needed to resolve the 3-D grid structure of the model. But at the end, the fluxes of all subpixels are summed up such that the resulting image has a regular grid again.

Note that the smaller we choose $f_{\rm ref}$ the more accurate our image becomes. In the `radmc3d.inp` file the value of $f_{\rm ref}$ can be set by setting the variable `camera_refine_criterion` to the value you want $f_{\rm ref}$ to be. Not setting this variable means RADMC-3D will use the default value which is reasonable as a choice (default is 1.0). The smaller you set `camera_refine_criterion`, the more accurate and reliable the results become (but the heavier the calculation becomes, too).

NOTE: The issue of recursive sub-pixeling becomes tricky when stars are treated as spheres, i.e. non-point-like (see Section *Stars in the images and spectra* and Chapter *More information about the treatment of stars*).

8.6.3 A danger with recursive sub-pixeling

It is useful to keep in mind that for each pixel the recursive sub-pixeling is triggered if the ray belonging to that pixel encounters a cell that is smaller than the pixel size. This *normally* works well if $f_{\rm ref}$ is chosen small enough. But if there exist regions in the model where one big non-refined cell lies adjacent to a cell that is refined, say, 4 times (meaning the big cell has neighbors that are 16 times smaller!), then if the ray of the pixel just happens to miss the small cells and only passes the big cell, it won't 'notice' that it may need to refine to correctly capture the tiny neighboring cells accurarely.

Such a problem only happens if refinement levels jump by more than 1 between adjacent cells. If so, then it may be important to make $f_{\rm ref}$ correspondingly smaller (by setting `camera_refine_criterion` in `radmc3d.inp` to the desired value). A bit of experimentation may be needed here.

8.6.4 Recursive sub-pixeling in spherical coordinates

In spherical coordinates the recursive sub-pixeling has a few issues that you may want to be aware of. First of all, in 1-D spherical coordinates each cell is in fact a shell of a certain thickness. In 2-D spherical coordinates cells are rings. In both cases the cells are not just local boxes, but have 2 or 1 (respectively) extended dimensions. RADMC-3D takes care to still calculate properly how to define the recursive sub-pixeling scale. But for rays that go through the central cavity of the coordinate system there is no uniquely defined pixel resolution to take. The global variable `camera_spher_cavity_relres` (with default value 0.05) defines such a relative scale. You can change this value in the `radmc3d.inp` file.

A second issue is when the user introduces extreme 'separable refinement' (see Section *Separable grid refinement in spherical coordinates (important!)* and Figure Fig. 9.3) in the R, Θ or Φ coordinate. This may, for instance, be necessary near the inner edge of a dusty disk model in order to keep the first cell optically thin. This may lead, however, to extremely deep sub-pixeling for rays that skim the inner edge of the grid. This leads to a huge slow-down of the ray-tracing process although it is likely not to give much a different result. By default RADMC-3D plays it safe. If you wish to prevent this excessive sub-pixeling (at your own risk) then you can set the following variables in the `radmc3d.inp` file:

- `camera_min_drr` which sets a lower limit to the $\Delta R/R$ taken into account for the sub-pixeling (region 'B' in Figure Fig. 9.2). The default is 0.003. By setting this to e.g. 0.03 you can already get a strong speed-up for models with strong R-refinement.
- `camera_min_dangle` which sets a lower limit to $\Delta\Theta$ (region 'C' in Figure Fig. 9.2) and/or $\Delta\Phi$. The default is 0.05. By setting this to e.g. 0.1 you can already get some speed-up for models with e.g. strong Θ-refinement.

It is important to keep in mind that the smaller you make this number, the more accurate and reliable the results. It may be prudent to experiment with smaller values of `camera_min_drr` for models with extremely optically thick inner edges, e.g. a protoplanetary disk with an abrupt inner edge and a high dust surface density. For a disk model with a very thin vertical extent it will be important to choose small values of `camera_min_dangle`, perhaps even smaller than the default value.

For your convenience: Because it can be sometimes annoying to always have to play with the `camera_min_drr`, `camera_min_dangle` and `camera_spher_cavity_relres` values, and since it is usually (!) not really necessary to have such extremely careful subpixeling, RADMC-3D now has a new command line option called `sloppy`. This command-line option will set: `camera_min_drr=0.1`, `camera_min_dangle=0.1` and `camera_spher_cavity_relres=0.1`. So if you have an image like this:

```
radmc3d image lambda 10 incl 45 phi 30 sloppy
```

then it will make the image with moderate, but not excessive subpixeling. This may, under some circumstances, speed up the image-making in spherical coordinates by a large factor. Similar for making spectra. For instance:

```
radmc3d sed incl 45 phi 30 sloppy
```

can be, under some circumstances, very much faster than without the sloppy option.

Note,however, that using the `sloppy` option and/or setting the values of `camera_min_drr`, `camera_min_dangle` and `camera_spher_cavity_relres` in the `radmc3d.inp` file by hand, {bf is all at your own risk!} It is always prudent to check your results, now and then, against a non-sloppy calculation.

8.6.5 How can I find out which pixels RADMC-3D is recursively refining?

Sometimes you notice that the rendering of an image or spectrum takes much more time than you expected. When recursive sub-pixeling is used for imaging, RADMC-3D will give diagnostic information about how many more pixels it has rendered than the original image resolution. This factor can give some insight if extreme amount of sub-pixeling refinement has been used. But it does not say where in the image this occurs. If you want to see exactly which pixels and subpixels RADMC-3D has rendered for some image, you can use the following command-line option:

```
radmc3d image lambda 10 diag_subpix
```

This `diag_subpix` option will tell RADMC-3D to write a file called `subpixeling_diagnostics.out` which contains four columns: One for the x-coordinate of the (sub-)pixel, one for the y-coordinate of the (sub-)pixel, one for the x-width of the (sub-)pixel and a final one for the y-width of the (sub-)pixel. In Python you can then use, for instance, the Numpy `loadtxt` method to read these columns.

If this diagnostic shows that the subpixeling is excessive (which can particularly happen in spherical coordinates) then you might want to read Section *Recursive sub-pixeling in spherical coordinates*.

8.6.6 Alternative to recursive sub-pixeling

As an alternative to using this recursive sub-pixeling technique to ensure flux conservation for images, one can simply enhance the spatial resolution of the image, for instance

```
radmc3d image lambda 10 npix 400
```

Or even 800 or so. This has the clear advantage that the user gets the complete information of the details in the image (while in the recursive sub-pixeling technique only the averages are retained). The clear disadvantages are that one may need rediculously high-resolution images (i.e. large data sets) to resolve all the details and one may waste a lot of time rendering parts of the image which do not need that resolution. The latter is typically an issue when images are rendered from models that use AMR techniques.

8.7 Stars in the images and spectra

Per default, stars are still treated as point sources. That means that none of the rays of an image can be intercepted by a star. Starlight is included in each image as a post-processing step. First the image is rendered without the stars (though with of course all the emission of dust, lines etc *induced* by the stars) and then for each star a ray tracing is done from the star to the observer (where only extinction is taken into account, because the emission is already taken care of) and the flux is then added to the image at the correct position. You can switch off the inclusion of the stars in the images or spectra with the `nostar` command line option.

However, as of version 0.17, stars can also be treated as the finite-size spheres they are. This is done with setting `istar_sphere = 1` in `radmc3d.inp`. However, this mode can slow down the code a bit or even substantially. And it may still be partly under development, so the code may stop if it is required to handle a situation it cannot handle yet. See Chapter *More information about the treatment of stars* for details.

8.8 Second order ray-tracing (Important information!)

Ideally we would like to assure that the model grid is sufficiently finely spaced everywhere. But in many cases of interest one does not have this luxury. One must live with the fact that, for memory and/or computing time reasons, the grid is perhaps a bit coarser than would be ideal. In such a case it becomes important to consider the 'order' of integration of the transfer equation. By default, for images and spectra, RADMC-3D uses first order integration: The source term and the opacity in each cell are assumed to be constant over the cell. This is illustrated in Fig. Fig. 8.4.

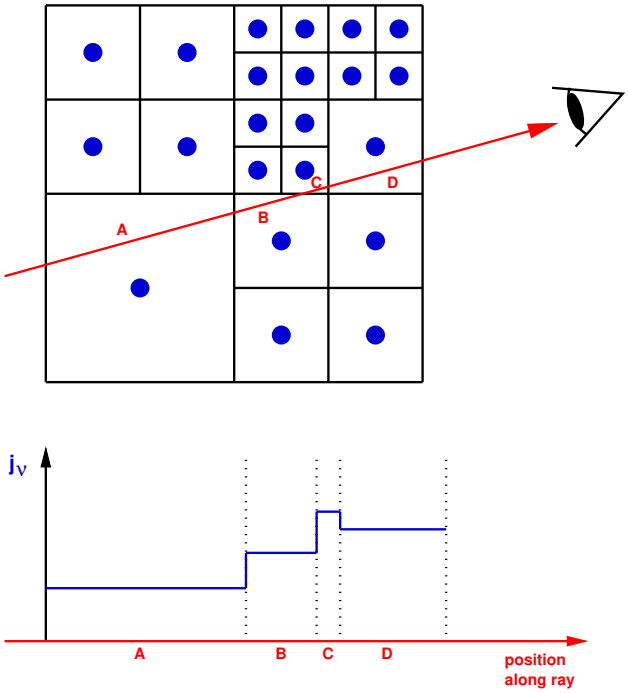

Fig. 8.4: Pictographic representation of the *first order* integration of the transfer equation along a ray (red line with arrow head) through an AMR grid (black lines). The grid cuts the ray into ray segments A, B, C and D. At the bottom it is shown how the integrands are assumed to be along these four segments. The emissivity function j_ν and extinction function α_ν are constant within each cell and thus constant along each ray segment.

The integration over each cell proceeds according to the following formula:

$$I_{\rm result} = I_{\rm start} e^{-\tau} + (1 - e^{-\tau}) S$$

where $S = j/\alpha$ is the source function, assumed constant throughout the cell, $\tau = \alpha\,\Delta s$ is the optical depth along the path that the ray makes through the cell, and $I_{\rm start}$ is the intensity upon entering the cell. This is the default used by RADMC-3D because the Monte Carlo methods also treat cells as having constant properties over each cell. This type of simple integration is therefore the closest to how the Monte Carlo methods (thermal MC, scattering MC and mono MC) 'see' the grid. However, with first order integration the images look somewhat 'blocky': you can literally see the block structure of the grid cells in the image, especially if you make images at angles aligned with the grid. For objects with high optical depths you may even see grid patterns in the images.

RADMC-3D can also use second order integration for its images and spectra. This is illustrated in Fig. Fig. 8.5.

This is done with a simple `secondorder` option added on the command line, for instance:

```
radmc3d image lambda 10 secondorder
```

The integration now follows the formula (Olson et al. 1986):

$$I_{\rm result} = I_{\rm start} e^{-\tau} + (1 - e^{-\tau} - \beta) S_{\rm start} + \beta S_{\rm end}$$

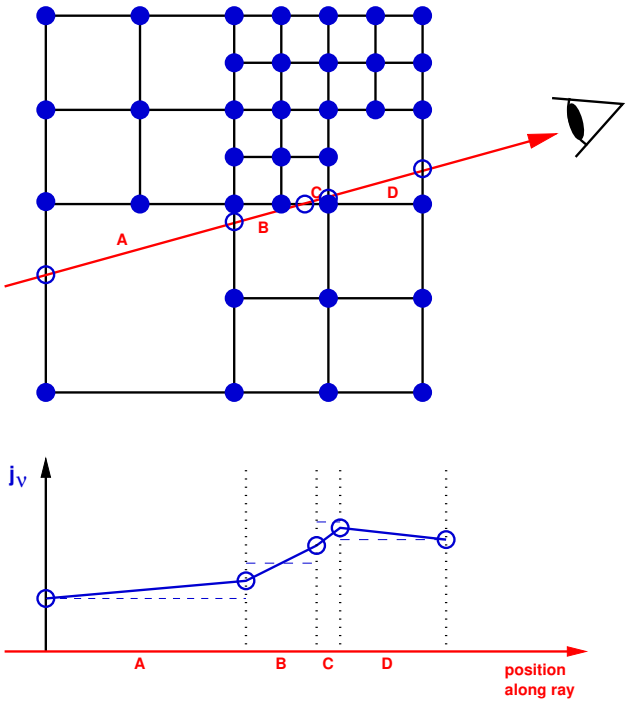

Fig. 8.5: Pictographic representation of the *second order* integration of the transfer equation along a ray (red line with arrow head) through an AMR grid (black lines). The grid cuts the ray into ray segments A, B, C and D. At the bottom it is shown how the integrands are assumed to be along these four segments. The emissivity function j_ν and extinction function α_ν are given at the cell corners (solid blue circles), and linearly interpolated from the cell corners to the locations where the ray crosses the cell walls (open blue circles). Then, along each ray segment the emissivity and extinction functions are assumed to be linear functions, so that the integration result is quadratic.

with

$$\beta = \frac{\tau - 1 + e^{-\tau}}{\tau}$$

and

$$\tau = \frac{\alpha_{\rm start} + \alpha_{\rm end}}{2} \Delta s$$

For $\tau \to 0$ we have the limit $\beta \to \tau/2$, while for $\tau \to \infty$ we have the limit $\beta \to 1$.

The values of α, S etc., at the 'start' position are obtained at the cell interface where the ray enters the cell. The values at the 'end' position are obtained at the cell interface where the ray leaves the cell. The above formulas represent the exact solution of the transfer equation along this ray-section if we assume that all variables are linear functions between the 'start' and 'end' positions.

The next question is: How do we determine the physical variables at the cell interfaces ('start' and 'end')? After all, initially all variables are stored for each cell, not for each cell interface or cell corner. The way that RADMC-3D does this is:

- First create a 'grid of cell corners', which we call the *vertex grid* (see the solid blue dots in Fig. Fig. 8.5). The cell grid already implicitly defines the locations of all the cell corners, but these corners are, by default, not explicitly listed in computer memory. When the `secondorder` option is given, however, RADMC-3D will explicitly find all cell corners and assign an identity (a unique integer number) to each one of them. NOTE: Setting up this vertex grid costs computer memory!
- At each vertex (cell corner) the physical variables of the (up to) 8 cells touching the vertex are averaged with equal weight for each cell. This now maps the physical variables from the cells to the vertices.
- Whenever a ray passes through a cell wall, the physical variables of the 4 vertices of the cell wall are interpolated bilinearly onto the point where the ray passes through the cell wall (see the open blue circles in Fig. Fig. 8.5). This gives the values at the 'start' or 'end' points.
- Since the current 'end' point will be the 'start' point for the next ray segment, the physical variables need only be obtained once per cell wall, as they can be recycled for the next ray segment. Each set of physical variables will thus be used twice: once for the 'end' and once for the 'start' of a ray segment (except of course at the very beginning and very end of the ray).

If you compare the images or spectra obtained with first order integration (default, see Figs. Fig. 8.8 and Fig. 8.9) or second order integration (see Figs. Fig. 8.8 and Fig. 8.9) you see that with the first order method you still see the cell structure of the grid very much. Also numerical noise in the temperature due to the Monte Carlo statistics is much more prominent in the first order method. The second order method makes much smoother results.

For line transfer the second order mode can be even improved with the 'doppler catching method', see Section *Preventing doppler jumps: The 'doppler catching method'*.

WARNING: Second order integration for the images and spectra from dust continuum emission can in some cases lead to overestimation of the fluxes. This is because the dust temperature calculated using the thermal Monte Carlo algorithm assumes the temperature to be constant over each cell. The second order integration for the images and spectra will, however, smear the sources a bit out. This then leads to 'leaking' of emissivity from optically thick cells into optically thin cells. These optically thin cells can then become too bright.

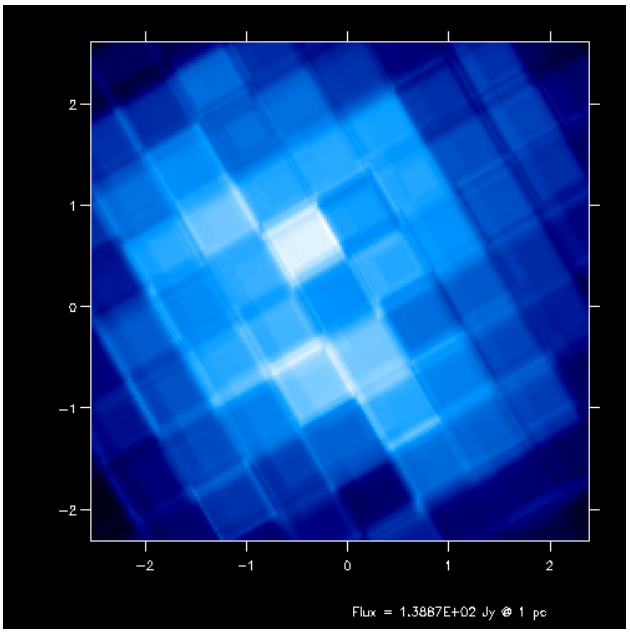

Fig. 8.6: First-order integration of transfer equation in ray-tracing seen at inclination 4 degrees.

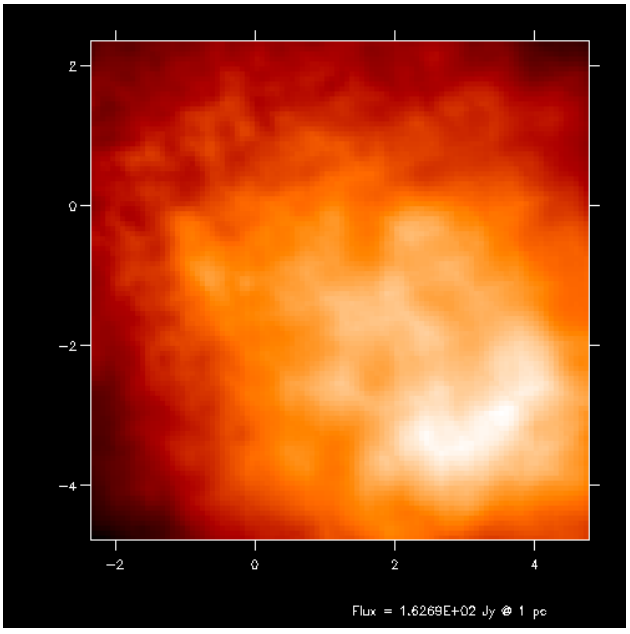

Fig. 8.7: First-order integration of transfer equation in ray-tracing seen at inclination 60 degrees.

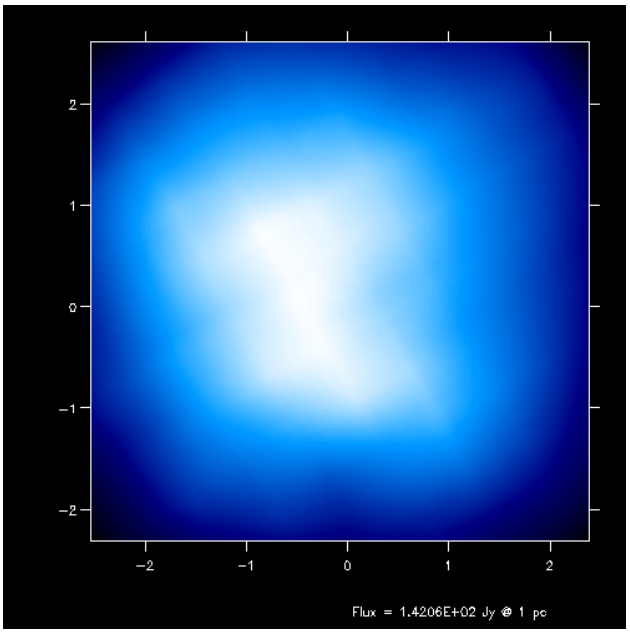

Fig. 8.8: Second-order integration of transfer equation in ray-tracing seen at inclination 4 degrees.

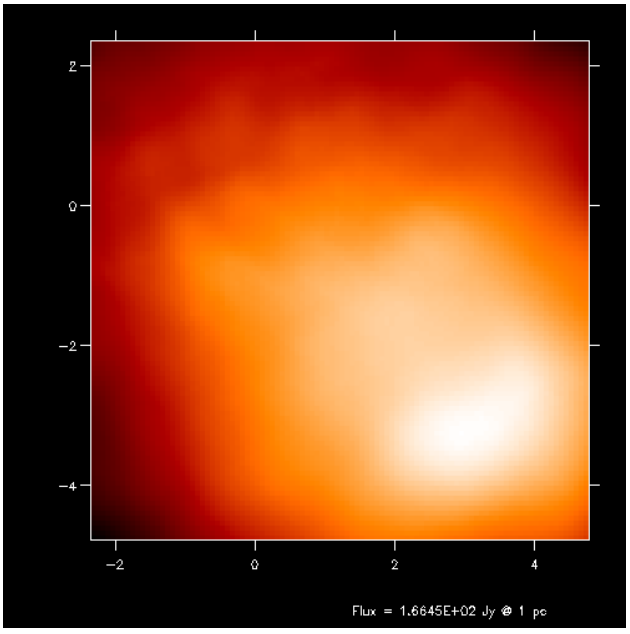

Fig. 8.9: Second-order integration of transfer equation in ray-tracing seen at inclination 60 degrees.

8.8.1 Second order integration in spherical coordinates: a subtle issue

The second order integration (as well as the doppler-catching method, see Section *Preventing doppler jumps: The 'doppler catching method'*) work in cartesian coordinates as well as in spherical coordinates. In spherical coordinates in 1-D (spherical symmetry) or 2-D (axial symmetry) there is, however, a very subtle issue that can lead to inaccuracies, in particular with line transfer. The problem arises in the cell where a ray reaches its closest approach to the origin of the coordinate system (or closest approach to the symmetry axis). There the ray segment can become fairly long, and its angle with respect to the symmetry axis and/or the origin can drastically change within this single ray-segment. This can sometimes lead to inaccuracies.

As of version 0.41 of `RADMC-3D` a new global variable is introduced, `camera_maxdphi`, which has as default the value 0.1, but which can be set to another value in the `radmc3d.inp` file. It sets the maximum angle (in radian) which a ray segment in spherical coordinates is allowed to span with respect to the origin of the coordinate system. If a ray segment spans an angle larger than that, the ray-segment is cut into smaller segments. This means that in that cell the ray will consist of more than one segment.

If `camera_maxdphi=0` this segment cutting is switched off (for backward compatibility to earlier versions of `RADMC-3d`).

8.9 Circular images

RADMC-3D offers (optionally!) an alternative to the usual x-y rectangular pixel arrangement of images: *circular images*. Here the pixels are not arranged in rows that are vertically stacked (x, y), but in concentric circles (r, ϕ). Such a pixel arrangement is, of course, radically different from what we usually consider "an image", and it is therefore not possible to view such an image with the usual image viewing methods (such as Python's `plt.imshow()`). Or more precisely: if you would use `plt.imshow()` on a circular image you would see something that you would not recognize as the image it should represent.

So what is the purpose? It is useful for models created on a spherical coordinate system. Such models can have structure at a huge range of scales, from very tiny (at the small-end side of the radius coordinate r) to very large (at the large-end side of the radius coordinate r). If you make a normal image, you have to pick the right "zoom factor": are you interested to see the outer regions or more interested in the inner regions? If you choose a "zoomed out" image, you will under-resolve the inner regions. If you choose a "zoomed in" image, you will not see the outer regions (they are beyond the edge of the image). One solution could be to choose a huge number of pixels, but that would create huge image files.

Circular images solve this dilemma. By arranging the pixels not in (x, y) but instead of (r, ϕ), the r coordinate grid of the image will automatically be adapted to the r coordinate grid of the spherical coordinate system. If the latter is logarithmically spaced, so will the circular image.

Here is how it works: Assuming you have a model in spherical coordinates, you can create a circular image as follows:

```
radmc3d image circ lambda 10
```

which creates a circular image at wavelength $\lambda = 10\mu m$.

Using `radmc3dPy` you can read this image as follows:

```
from radmc3dPy import image
im = image.readcircimage()
```

The data is now in `im.image`. A radial plot of the intensity at a given angle ϕ could be made as follows:

```
import matplotlib.pyplot as plt
plt.loglog(im.rc,im.image[:,0,0,0])
```

(continues on next page)

(continued from previous page)

```
plt.xlabel('r [cm]')
plt.ylabel(r'$I_\nu [erg\, cm^{-2}\, s^{-1}\, Hz^{-1}\, ster^{-1}]$')
```

The result will look like shown in Fig. Fig. 8.10 .

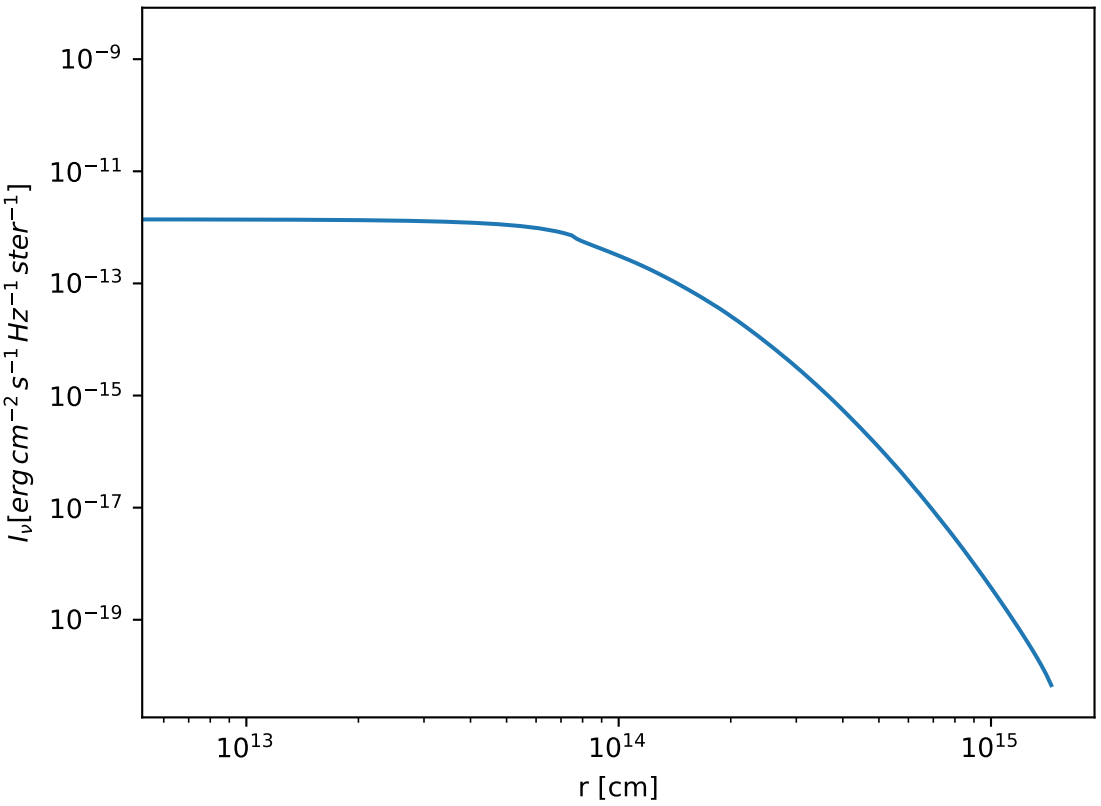

Fig. 8.10: Example of a circular image of a 1-D spherical model (the model in the `examples/run_spher1d_1/` directory).

If you have 2-D or 3-D models in spherical coordinates, the circular images (should) have not only a grid in r, but also ϕ grid points. A simple plot such as Fig. Fig. 8.10 will only show the intensity for a single phi choice. There is no "right" or "wrong" way of displaying such an image. It depends on your taste. You could, of course, remap onto a "normal" image, but that would defeat the purpose of circular images. You could also display the (r, ϕ) image directly with e.g. `plt.imshow()`, which simply puts the r axis horizontally on the screen, and the ϕ axis vertically, essentially creating a 'heat map' of the intensity as a function of r and ϕ.

This is illustrated in the model `examples/run_spher2d_1/`. Fig. Fig. 8.11 shows the circular image (as a 'heat map') at a wavelength of $\lambda = 10\ \mu m$. For comparison, the same image is shown as a 'normal' image in Fig. Fig. 8.12.

With a bit of "getting used to" one will find that the circular images will reveal a lot of information.

Note: Fig. **:numfig:`fig-circ-image-2d`** shows an effect similar to what is shown in Fig. Fig. 15.1. This indicates that near the inner radius of the model, the radial grid is under-resolved in example model `examples/run_spher2d_1/`: see Section *Careful: Things that might go wrong*, point 'Too optically thick cells at the surface or inner edge'. So, to improve the reliability of model `examples/run_spher2d_1/`, one would need to refine the radial grid near the inner edge and/or smooth the density there.

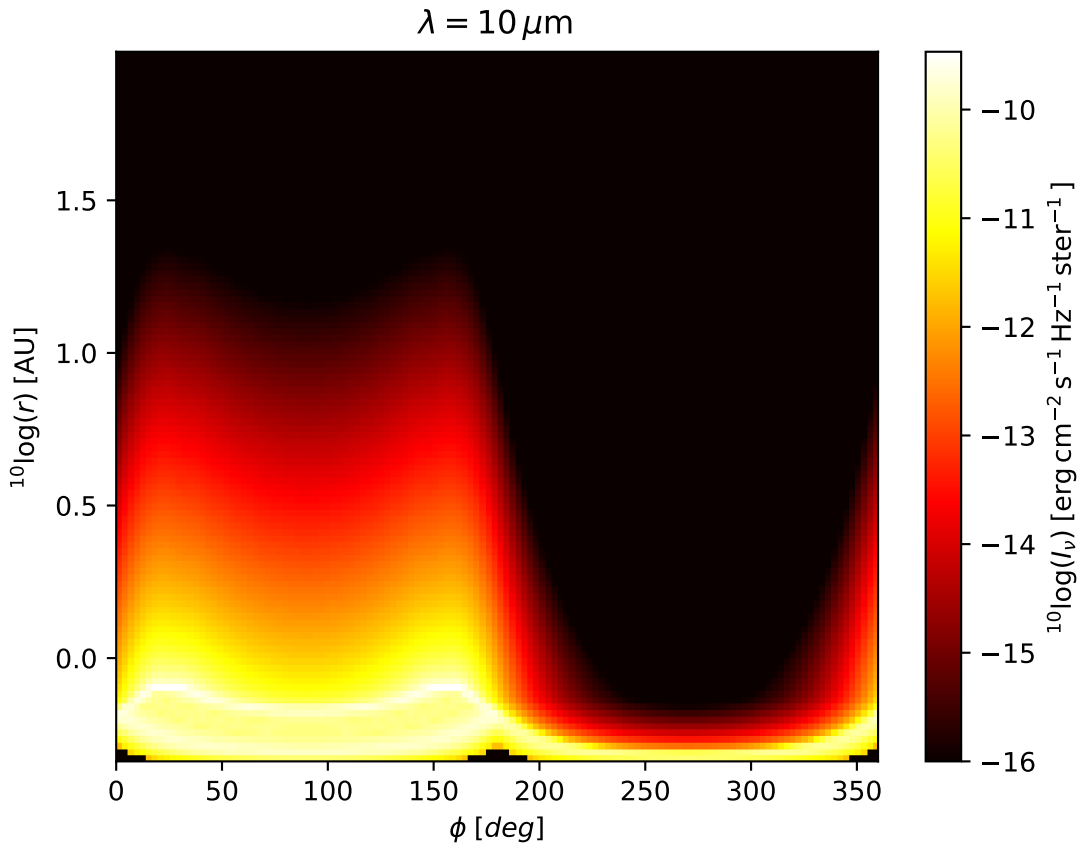

Fig. 8.11: Example of a circular image of a 2-D spherical model (the model in the `examples/run_spher2d_1/` directory).

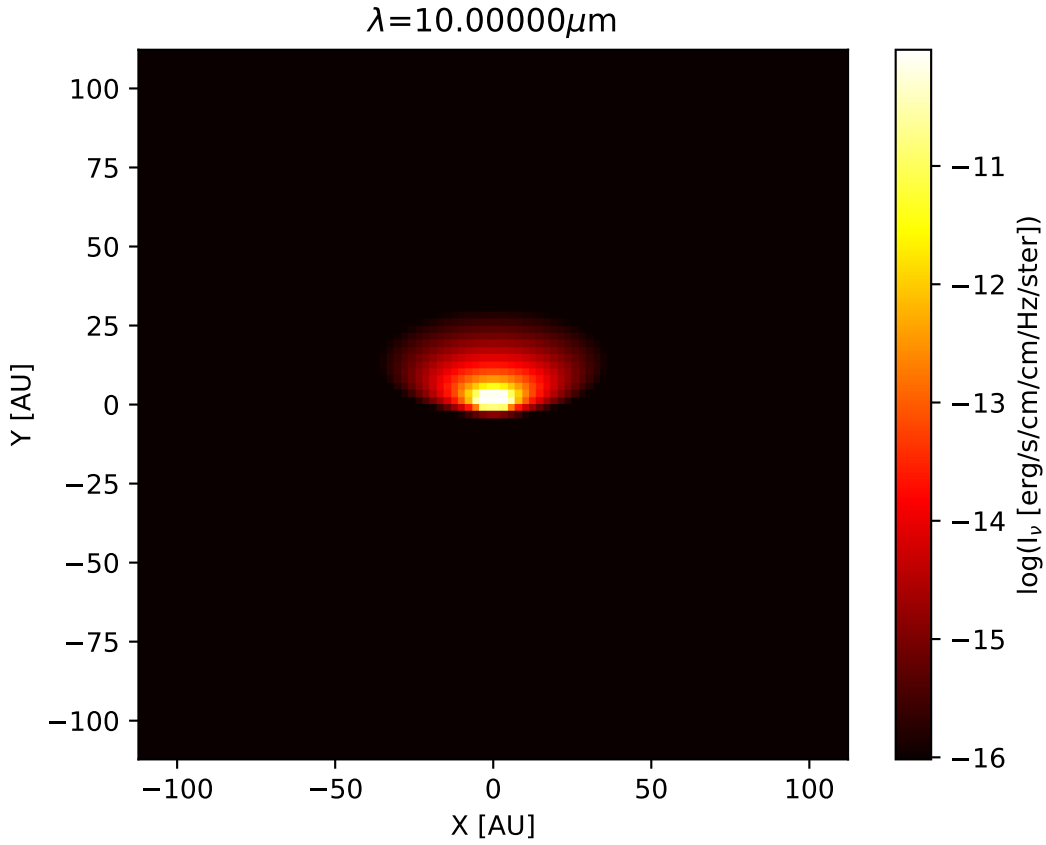

Fig. 8.12: The rectangular (‘normal’) version of the image of Fig. Fig. 8.11. As one can see: the inner regions of this image are not well-resolved.

8.10 Visualizing the $\tau = 1$ surface

To be able to interpret the outcome of the radiative transfer calculations it is often useful to find the spatial location of the $\tau = 1$ surface (or, for that matter, the $\tau = 0.1$ surface or any $\tau = \tau_s$ surface) as seen from the vantage point of the observer. This makes it easier to understand where the emission comes from that you are seeing. RADMC-3D makes this possible. Thanks to Peter Schilke and his team, for suggesting this useful option.

The idea is to simply replace the command-line keyword `image` with `tausurf 1.0`. The 1.0 stands for $\tau_s = 1.0$, meaning we will find the $\tau = 1.0$ surface. Example: Normally you might make an image with e.g. the following command:

```
radmc3d image lambda 10 incl 45 phi 30
```

Now you make a $\tau = 1$ surface with the command:

```
radmc3d tausurf 1.0 lambda 10 incl 45 phi 30
```

or a $\tau = 0.2$ surface with

```
radmc3d tausurf 0.2 lambda 10 incl 45 phi 30
```

The image output file `image.out` will now contain, for each pixel, the position along the ray in centimeters where $\tau = \tau_s$. The zero point is the surface perpendicular to the direction of observation, going through the pointing position (which is, by default $(0, 0, 0)$, but see the description of `pointau` in Section *Basics of image making with RADMC-3D*). Positive values mean that the surface is closer to the observer than the plane, while negative values mean that the surface is behind the plane.

If, for some pixel, there exists no $\tau = \tau_s$ point because the total optical depth of the object for the ray belonging to that pixel is less than τ_s, then the value will be -1e91.

You can also get the 3-D (i.e. x, y, z) positions of each of these points on the $\tau = \tau_s$ surface. They are stored in the file `tausurface_3d.out`.

Note that if you make multi-frequency images, you will also get multi-frequency $\tau = \tau_s$ surfaces. This can be particularly useful if you want to understand the sometimes complex origins of the shapes of molecular/atomic lines.

You can also use this option in the local observer mode, though I am not sure how useful it is. Note, however, that in that mode the value stored in the `image.out` file will describe the distance in centimeter to the local observer. The larger the value, the farther away from the observer (contrary to the case of observer-at-infinity).

Example usage:

```
radmc3d tausurf 1 lambda 10 incl 45 phi 30
```

8.11 For public outreach work: local observers inside the model

While it may not be very useful for scientific purposes (though there may be exceptions), it is very nice for public outreach to be able to view a model from the inside, as if you, as the observer, were standing right in the middle of the model cloud or object. One can then use physical or semi-physical or even completely ad-hoc opacities to create the right 'visual effects'. RADMC-3D has a viewing mode for this purpose. You can use different projections:

- *Projection onto flat screen:*

  The simplest one is a projection onto a screen in front (or behind) the point-location of the observer. This gives an image that is good for viewing in a normal screen. This is the default (`camera_localobs_projection=1`).

- *Projection onto a sphere:*

  Another projection is a projection onto a sphere, which allow fields of view that are equal or larger than 2π of the sky. It may be useful for projection onto an OMNIMAX dome. This is projection mode `camera_localobs_projection=2`.

You can set the variable `camera_localobs_projection` to 1 or 2 by adding on the command line `projection 2` (or 1), or by setting it in the `radmc3d.inp` as a line `camera_localobs_projection = 2` (or 1).

To use the local projection mode you must specify the following variables on the command line:

- `sizeradian`: This sets the size of the image in radian (i.e. the entire width of the image). Setting this will make the image width and height the same (like setting `sizeau` in the observer-at-infinity mode, see Section *Basics of image making with RADMC-3D*).
- `zoomradian`: *Instead* of `sizeradian` you can also specify `zoomradian`, which is the local-observer version of `zoomau` or``zoompc`` (see Section *Basics of image making with RADMC-3D*).
- `posang`: The position angle of the camera. Has the same meaning as in the observer-at-infinity mode.
- `locobsau` or `locobspc`: Specify the 3-D location of the local observer inside the model in units of AU or parsec. This requires 3 numbers which are the x, y and z positions (also when using spherical coordinates for the model setup: these are still the cartesian coordinates).
- `pointau` or `pointpc`: These have the same meaning as in the observer-at-infinity model. They specify the 3-D location of the point of focus for the camera (to which point in space is the camera pointing) in units of AU or parsec. This requires 3 numbers which are the x, y and z positions (also when using spherical coordinates for the model setup: these are still the cartesian coordinates).
- `zenith` (optional): For Planetarium Dome projection (`camera_localobs_projection=2`) it is useful to make the pointing direction not at the zenith (because then the audience will always have to look straight up) but at, say, 45 degrees. You can facilitate this (optionally) by adding the command line option `zenith 45` for a 45 degrees offset. This means that if you are sitting under the OMNIMAX dome, then the camera pointing (see `pointau` above) is 45 degrees in front of you rather than at the zenith. This option is highly recommended for dome projections, but you may need to play with the angle to see which gives the best effect.

Setting `sizeradian`, `zoomradian`, `locobsau` or `locobspc` on the command line automatically switches to the local observer mode (i.e. there is no need for an extra keyword setting the local observer mode on). To switch back to observer-at-infinity mode, you specify e.g. `incl` or `phi` (the direction toward which the observer is located in the observer-at-infinity mode). Note that if you accidently specify both e.g. `sizeradian` and `incl`, you might end up with the wrong mode, because the mode is set by the last relevant entry on the command line.

The images that are produced using the local observer mode will have the x- and y- pixel size specifications in radian instead of cm. The first line of an image (the format number of the file) contains then the value 2 (indicating local observer image with pixel sizes in radian) instead of 1 (which indicates observer-at-infinity image with pixel sizes in cm).

NOTE: For technical reasons dust scattering is (at least for now) not included in the local observer mode! It is discouraged to use the local observer mode for scientific purposes.

8.12 Multiple vantage points: the 'Movie' mode

It can be useful, both scientifically and for public outreach, to make movies of your model, for instance by showing your model from different vantage points or by 'travelling' through the model using the local observer mode (Section *For public outreach work: local observers inside the model*). For a movie one must make many frames, each frame being an image created by RADMC-3D's image capabilities. If you call `radmc3d` separately for each image, then often the reading of all the large input files takes up most of the time. One way to solve this is to call `radmc3d` in 'child mode' (see Chapter chap-child-mode). But this is somewhat complicated and cumbersome. A better way is to use RADMC-3D's 'movie mode'. This allows you to ask RADMC-3D to make a sequence of images in a single call. The way to do this is to call `radmc3d` with the `movie` keyword:

```
radmc3d movie
```

This will make `radmc3d` to look for a file called `movie.inp` which contains the information about each image it should make. The structure of the `movie.inp` file is:

```
iformat
nframes
<<information for frame 1>>
<<information for frame 2>>
<<information for frame 3>>
...
<<information for frame nframes>>
```

The `iformat` is an integer that is described below. The `nframes` is the number of frames. The `<<information for frame xx>>` are lines containing the information of how the camera should be positioned for each frame of the movie (i.e. for each imag). It is also described below.

There are multiple ways to tell RADMC-3D how to make this sequence of images. Which if these ways RADMC-3D should use is specified by the `iformat` number. Currently there are 2, but later we may add further possibilities. Here are the current possibilities

- `iformat=1`: The observer is at infinity (as usual) and the `<<information for frame xx>>` consists of the following numbers (separated by spaces):

  ```
  pntx pnty pntz hsx hsy pa incl phi
  ```

  These 8 numbers have the following meaning:

  - `pntx,pnty,pntz`: These are the x, y and z coordinates (in units of cm) of the point toward which the camera is pointing.
  - `hsx,hsy`: These are the image half-size in horizontal and vertical direction on the image (in units of cm).
  - `pa`: This is the position angle of the camera in degrees. This has the same meaning as for a single image.
  - `incl,phi`: These are the inclination and phi angle toward the observer in degrees. These have the same meaning as for a single image.

- `iformat=-1`: The observer is local (see Section *For public outreach work: local observers inside the model*) and the `<<information for frame xx>>` consists of the following numbers (separated by spaces):

  ```
  pntx pnty pntz hsx hsy pa obsx obsy obsz
  ```

  These 9 numbers have the following meaning:

  - `pntx,pnty,pntz,hsx,hsy,pa`: Same meaning as for `iformat=1`.
  - `obsx,obsy,obsz`: These are the x, y and z position of the local observer (in units of cm).

Apart from the quantities that are thus set for each image separately, all other command-line options still remain valid.

Example, let us make a movie of 360 frames of a model seen at infinity while rotating the object 360 degrees, and as seen at a wavelength of $\lambda = 10\mu$ file:

```
1
360
0. 0. 0. 1e15 1e15 0. 60.  1.
0. 0. 0. 1e15 1e15 0. 60.  2.
0. 0. 0. 1e15 1e15 0. 60.  3.
.
.
.
0. 0. 0. 1e15 1e15 0. 60.  358.
0. 0. 0. 1e15 1e15 0. 60.  359.
0. 0. 0. 1e15 1e15 0. 60.  360.
```

We now call RADMC-3D in the following way:

```
radmc3d movie lambda 10. npix 200
```

This will create image files `image_0001.out`, `image_0002.out`, all the way to `image_0360.out`. The images will have a full width and height of 2×10^{15}phi`-angle.

Another example: let us move through the object (local observer mode), approaching the center very closely, but not precisely:

```
-1
101
0. 0. 0. 0.8 0.8 0. 6.e13 -1.0000e15 0.
0. 0. 0. 0.8 0.8 0. 6.e13 -0.9800e15 0.
0. 0. 0. 0.8 0.8 0. 6.e13 -0.9600e15 0.
.
.
0. 0. 0. 0.8 0.8 0. 6.e13 -0.0200e15 0.
0. 0. 0. 0.8 0.8 0. 6.e13  0.0000e15 0.
0. 0. 0. 0.8 0.8 0. 6.e13  0.0200e15 0.
.
.
0. 0. 0. 0.8 0.8 0. 6.e13  0.9600e15 0.
0. 0. 0. 0.8 0.8 0. 6.e13  0.9800e15 0.
0. 0. 0. 0.8 0.8 0. 6.e13  1.0000e15 0.
```

Here the camera automatically rotates such that the focus remains on the center, as the camera flies by the center of the object at a closest-approach to the center of 6×10^{13} cm. The half-width of the image is 0.8 radian.

Important note: If you have scattering switched on, then every rendering of an image makes a new scattering Monte Carlo run. Since Monte Carlo produces noise, this would lead to a movie that is very jittery (every frame has a new noise set). It is of course best to avoid this by using so many photon packages that this is not a concern. But in practice this may be very CPU-time consuming. You can also fix the noise in the following way: add `resetseed` to the command-line call:

```
radmc3d movie resetseed
```

and it will force each new scattering Monte Carlo computation to start with the same seed, so that the photons will exactly move along the same trajectories. Now only the scattering phase function will change because of the different vantage points, but not the Monte Carlo noise. You can in fact set the actual value of the initial seed in the `radmc3d.inp` file by adding a line

```
iseed = -5415
```

(where -5415 is to be replaced by the value you want) to the `radmc3d.inp` file. Note also that if your movie goes through different wavelengths, the resetseed will likely not help fixing the noisiness, because the paths of photons will change for different wavelengths, even with the same initial seed.

CHAPTER

NINE

MORE INFORMATION ABOUT THE GRIDDING

We already discussed the various types of grids in Section *The spatial grid*, and the grid input file structure is described in Section *INPUT (required): amr_grid.inp or unstr_grid.inp*. In this chapter let us take a closer look at the gridding possibilities and things to take special care of.

9.1 Regular grids

A regular grid is called 'grid style 0' in RADMC-3D. It can be used in Cartesian coordinates as well as in spherical coordinates (Section *Coordinate systems*).

A regular grid, in our definition, is a multi-dimensional grid which is separable in x, y and z (or in spherical coordinates in r, θ and ϕ). You specify a 1-D monotonically increasing array of values $x_1, x_2, \cdots, x_{\mathrm{nx}+1}$ which represent the cell walls in $x - direction$. You do the same for the other directions: $y_1, y_2, \cdots, y_{\mathrm{ny}+1}$ and $z_1, z_2, \cdots, z_{\mathrm{nz}+1}$. The value of, say, x_2 is the same for every position in y and z: this is what we mean with 'separable'.

In Cartesian coordinates RADMC-3D enforces perfectly cubic grid cells (i.e. linear grids). But that is only to make the image sub-pixeling easier (see Section *The solution: recursive sub-pixeling*). For spherical grids this is not enforced, and in fact it is strongly encouraged to use non-linear grids in spherical coordinates. Please read Section *Separable grid refinement in spherical coordinates (important!)* if you use spherical coordinates!

In a regular grid you specify the grids in each direction separately. For instance, the x-grid is given by specifying the cell walls in x-direction. If we have, say, 10 cells in x-direction, we must specify 11 cell wall positions. For instance: $x_i = \{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$. For the y-direction and z-direction likewise. Fig. Fig. 9.1 shows an example of a 2-D regular grid of 4x3 cells.

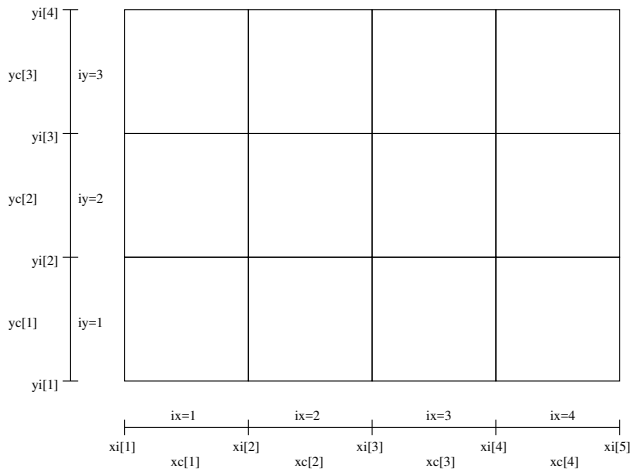

Fig. 9.1: Example of a regular 2-D grid with `nx=4` and `ny=3`.

In Cartesian coordinates we typically define our model in full 3-D. However, if your problem has translational symmetries, you might also want to consider the 1-D plane-parallel mode (see Section *1-D Plane-parallel models*).

In full 3-D Cartesian coordinates the cell sizes *must* be perfectly cubical, i.e. the spacing in each direction must be the same. If you need a finer grid in some location, you can use the AMR capabilities discussed below.

In spherical coordinates you can choose between 1-D spherically symmetric models, 2-D axisymmetric models or fully 3-D models. In spherical coordinates you do *not* have restrictions to the cell geometry or grid spacing. You can choose any set of numbers $r_1, \cdots, r_{\rm nr}$ as radial grid, as long as this set of numbers is larger than 0 and monotonically increasing. The same is true for the θ-grid and the ϕ-grid.

The precise way how to set up a regular grid using the `amr_grid.inp` file is described in Section *Regular grid*. The input of any spatial variables (such as e.g. the dust density) uses the sequence of grid cells in the same order as the cells are specified in that `amr_grid.inp` file.

For input and output data to file, for stuff on a regular grid, the order of nested loops over coordinates would be:

```
do iz=1,amr_grid_nz
   do iy=1,amr_grid_ny
      do ix=1,amr_grid_nx
         << read or write your data >>
      enddo
   enddo
enddo
```

For spherical coordinates we have the following association: $x \rightarrow r$, $y \rightarrow \theta$, $z \rightarrow \phi$.

9.2 Separable grid refinement in spherical coordinates (important!)

Spherical coordinates are a very powerful way of dealing with centrally-concentrated problems. For instance, collapsing protostellar cores, protoplanetary disks, disk galaxies, dust tori around active galactic nuclei, accretion disks around compact objects, etc. In other words: problems in which a single central body dominates the problem, and material at all distances from the central body matters. For example a disk around a young star goes all the way from 0.01 AU out to 1000 AU, covering 5 orders of magnitude in radius. Spherical coordinates are the easiest way of dealing with such a huge radial dynamic range: you simply make a radial grid, where the grid spacing $r_{i+1} - r_i$ scales roughly with r_i.

This is called a *logarithmic radial grid*. This is a grid whith a spacing in which $(r_{i+1} - r_i)/r_i$ is constant with r. In this way you assure that you have always the right spatial resolution in r at each radius. In spherical coordinates it is highly recomended to use such a log spacing. But you can also refine the r grid even more (in addition to the log-spacing). This is also strongly recommended near the inner edge of a circumstellar shell, for instance. Or at the inner dust rim of a disk. There you must refine the r grid (by simply making the spacing smaller as you approach the inner edge from the outside) to assure that the first few cells are optically thin and that there is a gradual transition from optically thin to optically thick as you go outward. This is particularly important for, for instance, the inner rim of a dusty disk.

In spherical coordinates you can vary the spacing in r, θ and ϕ completely freely. That means: you could have for instance r to be spaced as $1.00, 1.01, 1.03, 1.05, 1.1, 1.2, 1.35, \cdots$. There is no restriction, as long as the coordinate points are monotonically increasing. In Figs Fig. 9.2 and Fig. 9.3 this is illustrated.

Note that in addition to separable refinement, also AMR refinement is possible in spherical coordinates. See Section Oct-tree Adaptive Mesh Refinement.

For models of accretion disks it can, for instance, be useful to make sure that there are more grid points of θ near the equatorial plane $\theta = \pi/2$. So the grid spacing between $\theta = 0.0$ and $\theta = 1.0$ may be very coarse while between $\theta = 1.0$ and $\theta = \pi/2$ you may put a finer grid. All of this 'grid refinement' can be done without the 'AMR' refinement technique: this is the 'separable' grid refinement, because you can do this separately for r, for θ and for ϕ.

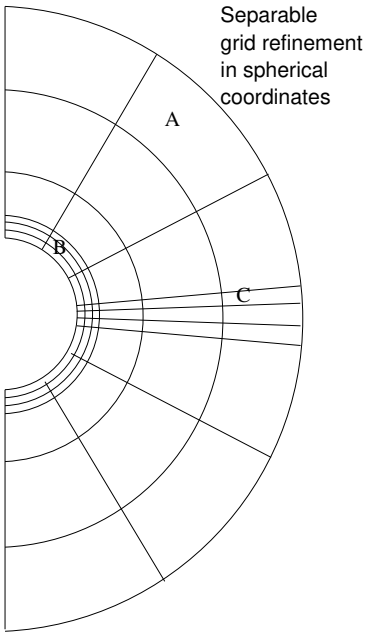

Fig. 9.2: Example of a spherical 2-D grid in which the radial and θ grids are refined in a 'separable' way. In radial direction the inner cells are refined ('B' in the right figure) and in θ direction the cells near the equatorial plane are refined ('C' in the right figure). This kind of grid refinement does not require oct-tree AMR: the grid remains separable. For models in which the inner grid edge is also the inner model edge (e.g. a simple model of a protoplanetary disk with a sharp inner cut-off) this kind of separable grid refinement in R-direction may be essential to avoid problems with optically thick inner cells (see e.g. Fig. Fig. 15.1 for an example of what could go wrong if you do not do this). Separable grid refinement in Θ-direction is typically important for protoplanetary disk models, where the midplane and surface layers of the disk need to have sufficient resolution, but any possible surrounding spherical nebula may not.

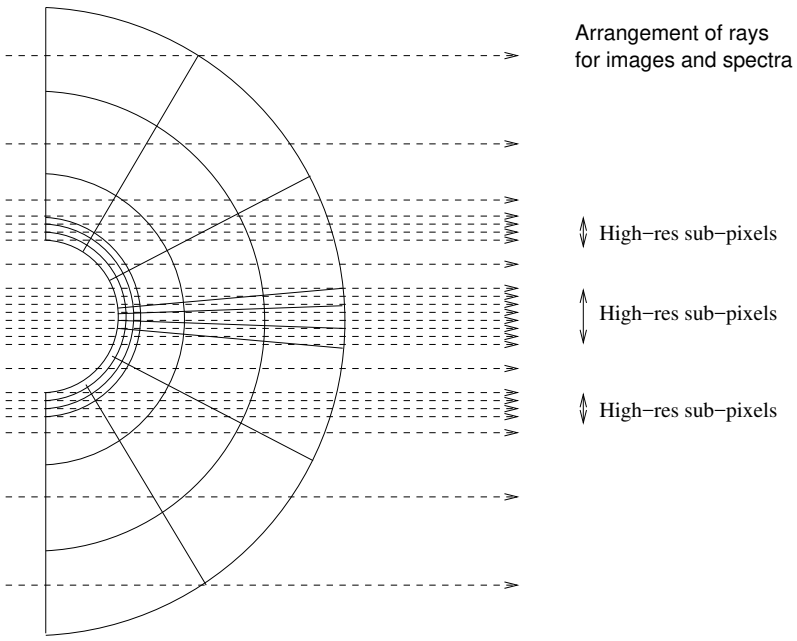

Fig. 9.3: When making an image, RADMC-3D will automatically make 'sub-pixels' to ensure that all structure of the model as projected on the sky of the observer are spatially resolved. Extreme grid refinement leads thus to extreme sub-pixeling. See Section *Recursive sub-pixeling in spherical coordinates* for details, and ways to prevent excessive sub-pixeling when this is not necessary.

Sometimes, however, separable refinement may not help you to refine the grid where necessary. For instance: if you model a disk with a planet in the disk, then you may need to refine the grid around the planet. You could refine the grid in principle in a separable way, but you would then have a large redundancy in cells that are refined by far away from the planet. Or if you have a disk with an inner rim that is not exactly at $r = r_{\rm rim}$, but is a rounded-off rim. In these cases you need refinement exactly located at the region of interest. For that you need the 'AMR' refinement (Sections *Oct-tree Adaptive Mesh Refinement* and *Layered Adaptive Mesh Refinement*).

Important note: When using strong refinement in one of the coordinates r, θ or ϕ, image-rendering and spectrum-rendering can become very slow, because of the excessive sub-pixeling this causes. There are ways to limit the sub-pixeling for those cases. See the Section on sub-pixeling in spherical coordinate: Section *Recursive sub-pixeling in spherical coordinates*.

9.3 Oct-tree Adaptive Mesh Refinement

An oct-tree refinened grid is called 'grid style 1' in RADMC-3D. It can be used in Cartesian coordinates as well as in spherical coordinates (Section *Coordinate systems*).

You start from a normal regular base grid (see Section *Regular grids*), possibly even with 'separable refinement' (see Section *Separable grid refinement in spherical coordinates (important!)*). You can then split some of the cells into 2x2x2 subcells (or more precisely: in 1-D 2 subcells, in 2-D 2x2 subcells and in 3-D 2x2x2 subcells). If necessary, each of these 2x2x2 subcells can also be split into further subcells. This can be repeated as many times as you wish until the desired grid refinement level is reached. Each refinement step refines the grid by a factor of 2 in linear dimension, which means in 3-D a factor of 8 in volume. In this way you get, for each refined cell of the base grid, a tree of refinement. The base grid can have any size, as long as the number of cells in each direction is an even number. For instance, you can have a 6x4 base grid in 2-D, and refine cell (1,2) by one level, so that this cell splits into 2x2 subcells.

Note that it is important to set which dimensions are 'active' and which are 'non-active'. For instance, if you have a 1-D model with 100 cells and you tell RADMC-3D (see Section *Oct-tree-style AMR grid*) to make a base grid of 100x1x1 cells, but you still keep all three dimensions 'active' (see Section *Oct-tree-style AMR grid*), then a refinement of cell 1 (which is actually cell (1,1,1)) will split that cell into 2x2x2 subcells, i.e. it will also refine in y and z direction. Only if you explicitly switch the y and z dimensions off the AMR will split it into just 2 subcells.

Oct-tree mesh refinement is very powerful, because it allows you to refine the grid exactly there where you need it. And because we start from a regular base grid like the grid specified in Section *Regular grids*, we can start designing our model on a regular base grid, and then refine where needed. See Fig. Fig. 9.4.

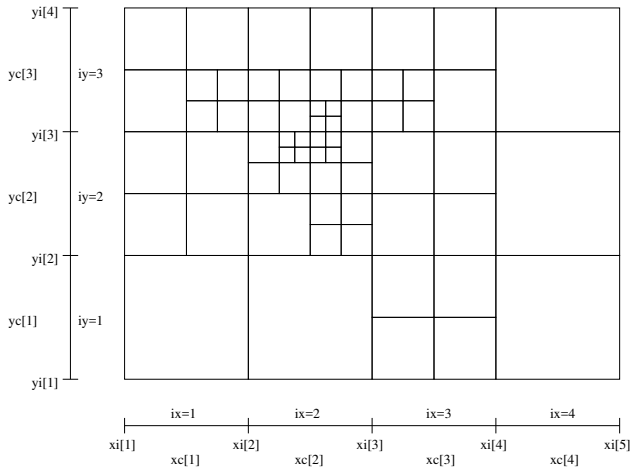

Fig. 9.4: Example of a 2-D grid with oct-tree refinement. The base grid has `nx=4` and `ny=3`. Three levels of refinement are added to this base grid.

The AMR stand for 'Adaptive Mesh Refinement', which may suggest that RADMC-3D will refine internally. At the moment this is not yet the case. The 'adaptive' aspect is left to the user: he/she will have to 'adapt' the grid such that it is sufficiently refinened where it is needed. In the future we may allow on-the-fly adaption of the grid, but that is not yet possible now.

One problem with oct-tree AMR is that it is difficult to handle such grids in external plotting programs, or even in programs that set up the grid. While it is highly flexible, it is not very user-friendly. Typically you may use this oct-tree refinement either because you import data from a hydrodynamics code that works with oct-tree refinement (e.g. FLASH, RAMSES), or when you internally refine the grid using the `userdef_module.f90` (see Chapter *Modifying RADMC-3D: Internal setup and user-specified radiative processes*). In the former case you are anyway forced to manage the complexities of AMR, while in the latter case you can make use of the AMR modules of RADMC-3D internally to handle them. But if you do not need to full flexibility of oct-tree refinement and want to use a simpler kind of refinement, then you can use RADMC-3D's alternative refinement mode: the layer-style AMR described in Section *Layered Adaptive Mesh Refinement* below.

The precise way how to set up such an oct-tree grid using the `amr_grid.inp` file is described in Section *Oct-tree-style AMR grid*. The input of any spatial variables (such as e.g. the dust density) uses the sequence of grid cells in the same order as the cells are specified in that `amr_grid.inp` file.

9.4 Layered Adaptive Mesh Refinement

A layer-style refinened grid is called 'grid style 10' in RADMC-3D. It can be used in Cartesian coordinates as well as in spherical coordinates (Section *Coordinate systems*).

This is an alternative to the full-fledged oct-tree refinement of Section *Oct-tree Adaptive Mesh Refinement*. The main advantage of the layer-style refinement is that it is far easier to handle by the human brain, and thus easier for model setup and the analysis of the results.

The idea here is that you start again with a regular grid (like that of Section *Regular grids*), but you can now specify a rectangular region which you want to refine by a factor of 2. The way you do this is by choosing the starting indices of the rectangle and specifying the size of the rectangle by setting the number of cells in each direction from that starting point onward. For instance, setting the starting point at (2,3,1) and the size at (1,1,1) will simply refine just cell (2,3,1) of the base grid into a set of 2x2x2 sub-cells. But setting the starting point at (2,3,1) and the size at (2,2,2) will split cells (2,3,1), (3,3,1), (2,4,1), (3,4,1), (2,3,2), (3,3,2), (2,4,2) and (3,4,2) each into 2x2x2 subcells. This in fact is handled as a 4x4x4 regular sub-grid patch. And setting the starting point at (2,3,1) and the size at (4,6,8) will make an entire regular sub-grid patch of 8x12x16 cells. Such a sub-grid patch is called a *layer*.

The nice thing of these layers is that each layer (i.e. subgrid patch) is handled as a regular sub-grid. The base grid is layer number 0, and the first layer is layer number 1, etc. Each layer (including the base grid) can contain multiple sub-layers. The only restriction is that each layer fits entirely inside its parent layer, and layers with the same parent layer should not overlap. Each layer can thus have one or more sub-layers, each of which can again be divided into sub-layers. This builds a tree structure, with the base layer as the trunk of the tree (this is contrary to the oct-tree structure, where each base grid *cell* forms the trunk of its own tree). In Fig. Fig. 9.5 an example is shown of two layers with the same parent (= layer 0 = base grid), while in Fig. Fig. 9.6 an example is shown of two nested layers.

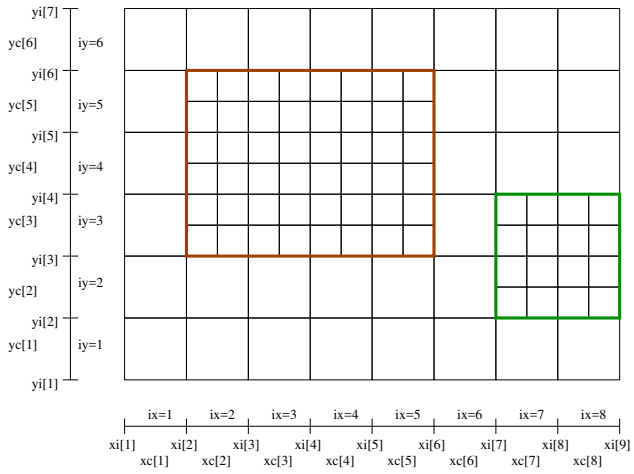

Fig. 9.5: Example of a 2-D base grid with `nx`=4 and `ny`=3, with two AMR-layers added to it. This example has just one level of refinement, as the two layers (brown and green) are on the same level (they have the same parent layer = layer 0).

If you now want to specify data on this grid, then you simply specify it on each layer separately, as if each layer is a separate entity. Each layer is treated as a regular grid, irrespective of whether it contains sub-layers or not. So if we have a base grid of 4x4x4 grid cells containing two layers: one starting at (1,1,1) and having (2,2,2) size and another starting at (3,3,3) and having (1,1,2) size, then we first specify the data on the $4^3 = 64$ base grid, then on the $(2 \times 2)^3 = 64$ grid cells of the first layer and then on the 2x2x4=16 cells of the second layer. Each of these three layers are regular grids, and the data is inputted/outputted in the same way as if these are normal regular grids (see Section *Regular grids*). But instead of just one such regular grid, now the data file (e.g. `dust_density.inp`) will contain three successive lists of numbers, the first for the base grid, the second for the first layer and the last for the second layer. You may realize at this point that this will introduce a redundancy. See Subsection *On the 'successively regular'*

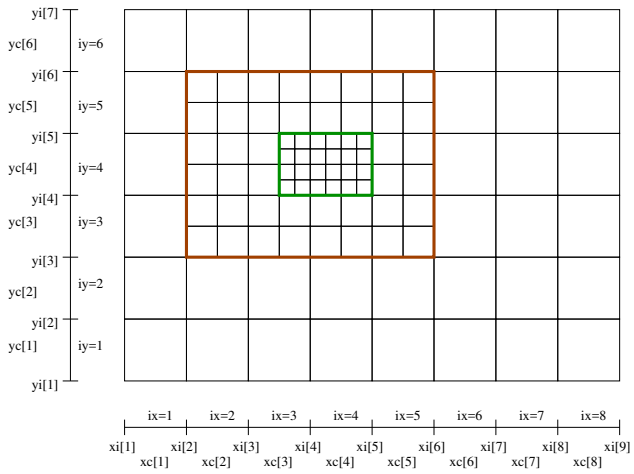

Fig. 9.6: Example of a 2-D base grid with `nx`=4 and `ny`=3, with two nested AMR-layers added to it. This example has two levels of refinement, as layer 1 (brown) is the parent of layer 2 (green).

kind of data storage, and its slight redundancy for a discussion of this redundancy.

The precise way how to set up such an oct-tree grid using the `amr_grid.inp` file is described in Section *Layer-style AMR grid*. The input of any spatial variables (such as e.g. the dust density) uses the sequence of grid cells in the same order as the cells are specified in that `amr_grid.inp` file.

9.4.1 On the 'successively regular' kind of data storage, and its slight redundancy

With the layered grid refinement style there will be *redundant* data in the data files (such as e.g. the `dust_density.inp` file. Each layer is a regular (sub-)grid and the data will be specified in each of these grid cells of that regular (sub-)grid. If then some of these cells are overwritten by a higher-level layer, these data are then redundant. We could of course have insistent that only the data in those cells that are not refined by a layer should be written to (or read from) the data files. But this would require quite some clever programming on the part of the user to a-priori find out where the layers are and therefore which cells should be skipped. We have decided that it is far easier to just insist that each layer (including the base grid, which is layer number 0) is simply written to the data file as a regular block of data. The fact that some of this data will be not used (because they reside in cells that are refined) means that we write more data to file than really exists in the model. This makes the files larger than strictly necessary, but it makes the data structure by far easier. Example: suppose you have a base grid of 8x8x8 cells and you replace the inner 4x4x4 cells with a layer of 8x8x8 cells (each cell being half the size of the original cells). Then you will have for instance a `dust_density.inp` file containing 1024 values of the density: 8^3=512 values for the base grid and again 8^3=512 values for the refinement layer. Of the first 8^3=512 values 4^3=64 values are ignored (they could have any value as they will not be used). The file is thus 64 values larger than strictly necessary, which is a redundancy of 64/1024=0.0625. If you would have used the oct-tree refinement style for making exactly the same grid, you would have only 1024-64=960 values in your file, making the file 6.25% smaller. But since 6.25% is just a very small difference, we decided that this is not a major problem and the simplicity of our 'successively regular' kind of data format is more of an advantage than the 6.25% redundance is a disadvantage.

9.5 Unstructured grids (Delaunay, Voronoi, or more general)

RADMC-3D can handle unstructures grids of a variety of types. This works (so far) only for 3-D cartesian coordinates. The two most well-known are Delaunay grids and Voronoi grids. But these are just special cases of a more general unstructured grid capability. RADMC-3D only needs to know, for each grid cell:

1. The volume of the cell
2. The cell walls of the cell

and, conversely, for each cell wall which two cells that the wall separates. A cell wall is only defined by two vectors that define the 2D plane in 3D space: a support vector $\mathbf{s}$ and a normal vector $\mathbf{n}$. By giving to RADMC-3D a file `unstr_grid.inp` containing all this information, the user determines the shape of each cell. Each of these cells has an integer index, starting with 1, and increasing by steps of 1 until the number of cells. The density and other variables in the usual input files such as `dust_density.inp` are then associated to these cells in that order (first value belongs to cell 1, second to cell 2 etc).

This information is enough for RADMC-3D to compute where a ray passes from one cell to its neighbor, and how the passage through the cell affects the cell's temperature and scattering source function.

Some of this grid information is redundant, and depends on which information is contained in `unstr_grid.inp`. Typically the user provides only the information about the grid he/she wants to provide, and RADMC-3D will try to complete the rest if it can (otherwise it will give an error).

For example: For a Voronoi grid you only need to specify, for each cell wall, which two cells the wall separates. If RADMC-3D lacks further information, it will by default assume that the cell walls lie exactly in between the two cell center points and perpendicular to the line connecting them, thus naturally giving you a Voronoi grid. The support vectors and normal vectors of the walls are then computed internally in RADMC-3D. Unfortunately, RADMC-3D lacks the capability to calculate the cell volumes for Voronoi cells, so you will have to provide the cell volume.

As another example: For a Delaunay grid you only need to specify the vertices (corner points of the tetraheders), which vertices span a wall, and which two cells the wall separates (or one cell and the vacuum). The support vectors and normal vectors of the walls are then computed internally in RADMC-3D. For these simple shaped cells (just simplices) RADMC-3D can calculate the volume itself.

Computing this information for a Delaunay or Voronoi grid can be done using external software packages such as `qhull`. In fact, the `qhull` package is built-in into the `scipy.spatial` library of Python, and is thefore easily usable from within Python. To create the necessary `unstr_grid.inp` files using these libraries you can use the following tools provided with the RADMC-3D package:

1. `python/radmc3d_tools/radmc3d_delaunay_grid.py` : A tool to generate a Delaunay grid from a given set of vertex points. Note that the cells are then created by this tool, and they may not be the same (and almost certainly not in the same order) as the cells you may have assumed beforehand. So if you obtain a model from someone else that is on a Delaunay grid, and the physical values are in the grid cells, you may need to figure out how to associate the cells from that other model with the cells generated by this package. However, usually the physical values would, in the case of a Delaunay grid, be specified at the cell vertices. Then you would need to interpolate into the (new) cells, because RADMC-3D needs the physical values inside the cells, not at the vertices.
2. `python/radmc3d_tools/radmc3d_voronoi_grid.py` : A tool to generate a Voronoi grid from a given set of cell center points. The physical variables are also specified on these points. The vertex points are not required as input. Only needed is information about each cell wall: which two cells are divided by the wall, and information about each cell: it's volume. In voronoi grids there will be 'open cells' which go out to infinity. The physical variables in these cells are ignored: These cells are assumed to be empty.

Examples of these two types of grids are shown in Fig. 9.7 and Fig. 9.8.

However, you can also generate your own grid designs by specifying the support vectors and normal vectors and two cells belonging to each wall. However, this is not an entirely trivial task to do correctly, because the result has to obey

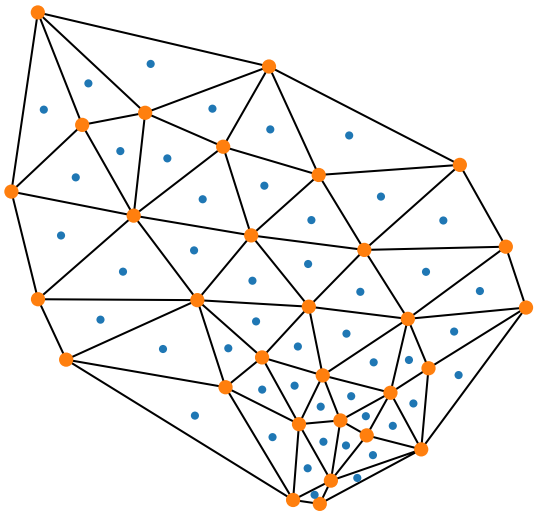

Fig. 9.7: Example of unstructured grid: a Delaunay grid. Figure is only symbolic, since the actual grid is 3D. Blue points are cell centers, orange points are vertices.

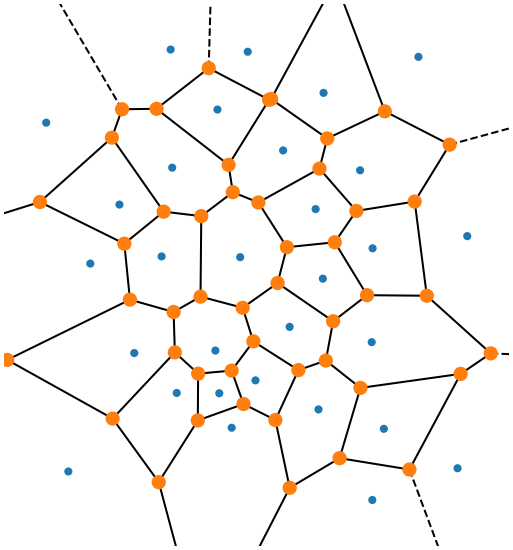

Fig. 9.8: Example of unstructured grid: a Voronoi grid. Figure is only symbolic, since the actual grid is 3D. Blue points are cell centers, orange points are vertices.

the following conditions:

1. Each cell must be convex in shape.
2. Beware not to forget any necessary wall. A missing wall will likely cause very unexpected behavior.
3. There should be no "empty holes" between the cells: inside the grid cells should fill space perfectly. This is not a trivial condition to fulfill: ill defined grid walls can lead to photons finding themselves outside a grid cell, but within the grid. This leads to RADMC-3D to crash.
4. If you specify the cell walls with vertices instead of support- and normal vectors, the vertices belonging to a cell must lie perfectly in same plane.
5. If you have walls that face the vacuum, they are part of what is called the hull of the grid (Note: A Delaunay grid has a hull, but a Voronoi grid does not: it has instead open cells). It is best (though not strictly necessary) to assure that the hull is convex, since that makes the code faster.

The way to make your own (non-Delaunay, non-Voronoi) grid is to generate a file `unstr_grid.inp` in the appropriate way. The technical details how this file is formatted are given in Section *Unstructured grid*.

9.6 1-D Plane-parallel models

Sometimes it can be useful to make simple 1-D plane parallel models, for instance if you want to make a simple 1-D model of a stellar atmosphere. RADMC-3D is, however, by nature a 3-D code. But as of version 0.31 it features a genuine 1-D plane-parallel mode as well. This coordinate type has the number 10. In this mode the x- and y-coordinates are the in-plane coordinates, while the z-coordinate is the 1-D coordinate. We thus have a 1-D grid in the z-coordinate, but no grid in x- or y-directions.

You can make a 1-D plane-parallel model by setting some settings in the `amr_grid.inp` file. Please consult Section *INPUT (required): amr_grid.inp or unstr_grid.inp* for the format of this file. The changes/settings you have to do are (see example below): (1) set the coordinate type number to 10, (2) set the x and y dimensions to non-active and (3) setting the cell interfaces in x to -1d90, +1d90, and likewise for y. Here is then how it looks:

```
1                                       <=== Format number = 1
0                                       <=== Grid style (0=regular grid)
10                                      <=== Coordinate type (10=plane-parallel)
0                                       <=== (obsolete)
0  0  1                                 <=== x and y are non-active, z is active
1  1  100                               <=== x and y are 1 cell, in z we have 100 cells
-1e90 1e90                              <=== cell walls in x are at "infinity"
-1e90 1e90                              <=== cell walls in y are at "infinity"
zi[1]        zi[2]        zi[3]        ........  zi[nz+1]
```

The other input files are for the rest as usual, as in the 3-D case.

You can now make your 1-D model as usual. For 1-D plane-parallel problems it is often useful to put a thermal boundary at the bottom of the model. For instance, if the model is a stellar atmosphere, you may want to cap the grid from below with some given temperature. See Section *Thermal boundaries in Cartesian coordinates* for details on how to set up thermal boundaries.

In the 1-D plane-parallel mode some things work a bit different than in the "normal" 3-D mode:

- Images are by default 1x1 pixels, because in a plane-parallel case it is useless to have multiple pixels.
- Spectra cannot be made, because "spectrum" is (in RADMC-3D 'language') the flux as a function of frequency as seen at a very large distance of the object, so that the object is in the "far field". Since the concept of "far-field" is no longer meaningful in a plane-parallel case, it is better to make frequency-dependent 1x1 pixel images. This gives you the frequence-dependent intensity, which is all you should need.

- Stars are not allowed, as they have truly 3-D positions, which is inconsistent with the plane-parallel assumption.

But for the rest, most stuff works similarly to the 3-D version. For instance, you can compute dust temperatures with

```
radmc3d mctherm
```

as usual.

9.6.1 Making a spectrum of the 1-D plane-parallel atmosphere

As mentioned above, the 'normal' 3-D way of making a spectrum of the 1-D plane-parallel atmosphere is not possible, because formally the atmosphere is infinitely extended. Instead you can obtain a spectrum in the form of an intensity ($\mathrm{erg\,s^{-1}\,cm^{-2}\,Hz^{-1}\,ster^{-1}}$) as a function of wavelength. To do this you ask RADMC-3D to make a multi-wavelength image of the atmosphere under a certain inclination (inclination 0 meaning face-on), e.g.:

```
radmc3d image allwl incl 70
```

This make an SED at $\lambda = 10\,\mu$m for the observer seeing the atmosphere at an inclination of 70 degrees. This produces a file image.out, described in Section *OUTPUT: image.out or image_****.out*. The image is, in fact, a 1x1 pixel multi-wavelength image. The `allwl` (which stands for 'all wavelengths') means that the spectral points are the same as those in the `wavelength_micron.inp` file (see Section *INPUT (required): wavelength_micron.inp*). You can also specify the wavelengths in a different way, e.g.:

```
radmc3d image lambdarange 5 20 nlam 10
```

In fact, see Section *Making multi-wavelength images* and Section *Specifying custom-made sets of wavelength points for the camera* for details.

9.6.2 In 1-D plane-parallel: no star, but incident parallel flux beams

In 1-D plane-parallel geometry it is impossible to include meaningful stars as sources of photons. This is not a technical issue, but a mathematical truth: a point in 1-D is in reality a plane in 3-D. As a replacement RADMC-3D offers (only in 1-D plane-parallel geometry) the possibility of illuminating the 1-D atmosphere from above with a flux, incident onto the atmosphere in a prescribed angle. This allows you to model, e.g., the Earth's atmosphere being illuminated by the sun at a given time of the day. This is done by providing an ascii file called `illum.inp` which has the following form (similar, but not identical, to the `stars.inp` file, see Section *INPUT (mostly required): stars.inp*):

```
iformat                          <=== Put this to 2 !
nillum     nlam
theta[1]      phi[1]
  .             .
  .             .
theta[nillum] phi[nillum]
lambda[1]
  .
  .
lambda[nlam]
flux[1,illum=1]
  .
  .
flux[nlam,illum=1]
flux[1,illum=2]
  .
  .
```

(continues on next page)

(continued from previous page)

```
flux[nlam,illum=2]
  .
  .
  .
  .
flux[nlam,illum=nstar]
```

Here `nillum` is the number of illuminating beams you want to specify. Normally this is 1, unless you have, e.g., a planet around a double star. The `theta` is the angle (in degrees) under which the beam impinges onto the atmosphere. If you have `theta`=0, then the flux points vertically downward (sun at zenith). If you have `theta`=89, then the flux points almost parallel to the atmosphere (sunset). It is not allowed to put `theta`=90.

You can, if you wish, also put the source behind the slab, i.e. `theta`>90. Please note, however, that if you compute the spectrum of the plane-parallel atmosphere the direct flux from these illumination beams does not get picked up in the spectrum.

9.6.3 Similarity and difference between 1-D spherical and 1-D plane-parallel

Note that this 1-D plane-parallel mode is only available in z-direction, and only for cartesian coordinates! For spherical coordinates, a simple switch to 1-D yields spherically symmetric 1-D radiative transfer, which is, however, geometrically distinct from 1-D plane-parallel radiative transfer. However, you can also use a 1-D spherically symmetric setup to 'emulate' 1-D plane parallel problems: You can make, for instance, a radial grid in which $r_{\rm nr}/r_1 - 1 \ll 1$. An example: $r = \{10000.0, 10000.1, 10000.2, \cdots, 10001.0\}$. This is not perfectly plane-parallel, but sufficiently much so that the difference is presumably indiscernable. The spectrum is then automatically that of the entire large sphere, but by dividing it by the surface area, you can recalculate the local flux. In fact, since a plane-parallel model usually is meant to approximate a tiny part of a large sphere, this mode is presumably even more realistic than a truly 1-D plane-parallel model.

9.7 Thermal boundaries in Cartesian coordinates

By default all boundaries of the computational domain are open, in the sense that photons can move out freely. The only photons that move into the domain from the outside are those from the interstellar radiation field (see Section *The interstellar radiation field: external source of energy*) and from any stars that are located outside of the computational domain (see Section *INPUT (mostly required): stars.inp*). For some purposes it might, however, be useful to have one or more of the six boundaries in 3-D to be closed. RADMC-3D offers the possibility, in cartesian coordinates, to convert the boundaries (each of the six separately) to a thermal boundary, i.e. a blackbody emitter at some user-secified temperature. If you want that the left X-boundary is a thermal wall at T=100 Kelvin, then you add the following line to the `radmc3d.inp` file:

```
thermal_boundary_xl = 100
```

and similarly for xr (right X-boundary), yl, yr, zl and/or zr. You can set this for each boundary separately, and particularly you can choose to set just one or just two of the boundaries to thermal boundaries. Note that setting `thermal_boundary_xl=0` is equivalent to switching off the thermal boundary.

Note that if you now make an image of the box, the ray-tracer will show you still the inside of the box, through any possible thermal boundary. In other words: for the imaging or spectra these thermal boundaries are opaque for radiation entering the grid, while they are transparent for radiation exiting the grid. In other words, we see the blackbody emission from the backside walls, but not of the frontside walls. In this way we can have a look inside the box in spite of the thermal walls.

CHAPTER

TEN

MORE INFORMATION ABOUT THE TREATMENT OF STARS

How stars are treated in RADMC-3D is perhaps something that needs some more background information. This is the structure:

1. *Stars as individual objects:*

   The most standard way of injecting stellar light into the model is by putting one or more individual stars in the model. A star can be placed anywhere, both inside the grid and outside. The main input file specifying their location and properties is: `stars.inp`. The stars can be treated in two different ways, depending on the setting of the variable `istar_sphere` that can be set to 0 or 1 in the file `radmc3d.inp` file.

   - The default is to treat stars as zero-size point sources. This is the way it is done if (as is the default) `istar_sphere=0`. The stars are then treated as point sources in spite of the fact that their radius is specified as non-zero in the `stars.inp` file. This default mode is the easiest and quickest. For most purposes it is perfectly fine. Only if you have material very close to a stellar surface it may be important to treat the finite size(s) of the star(s).
   - If `istar_sphere=1` in the `radmc3d.inp` file, then all stars are treated as spheres, their radii being the radii specified in the `stars.inp` file. This mode can be tricky, so please read Section *Stars treated as spheres*.

2. *Smooth distributions of zillions of stars:*

   For modeling galaxies or objects of that size scale, it is of course impossible and unnecessary to treat each star individually. So *in addition to the individual stars* you can specify spatial distributions of stars, assuming that the number of stars is so large that there will always be a very large number of them in each cell. Please note that using this possibility does *not* exclude the use of individual stars as well. For instance, for a galaxy you may want to have distributions of unresolved stars, but one single 'star' for the active nucleus and perhaps a few individual 'stars' for bright star formation regions or O-star clusters or so. The distribution of stars is described in Section *Distributions of zillions of stars*.

3. *An external 'interstellar radiation field':*

   Often an object is affected not only by the stellar radiation from the stars inside the object itself, but also by the diffuse radiation from the many near and far stars surrounding the object. This 'Interstellar Radiation Field' can be treated by RADMC-3D as well. This is called the 'external source' in RADMC-3D. It is described in Section *The interstellar radiation field: external source of energy*.

10.1 Stars treated as point sources

By default the stars are treated as point-sources. Even if the radius is specified as non-zero in the `stars.inp` file, they are still treated as points. The reason for this is that it is much easier and faster for the code to treat them as point-sources. Point sources cannot occult anything in the background, and nothing can partly occult them (they are only fully or not occulted, of course modulo optical depth of the occulting object). This approximation is, however, not valid if the spatial scales you are interested in are not much larger (or even the same or smaller) than the size of the star. For instance, if we are interested in modeling the radiative transfer in a disk around a Brown Dwarf, where dust can survive perhaps even all the way down to the stellar surface, we must take the non-point-like geometry of the star into account. This is because due to its size, the star can shine *down* onto the disk, which would not be possible if the star is treated as a point source. However, for a dust disk arounda Herbig Ae star, where the dust evaporation radius is at about 0.5 AU, the star can be treated as a point-source without problems.

So if you just use RADMC-3D as-is, or if you explicitly set `istar_sphere=0` in the file `radmc3d.inp`, then the stars are all treated as point sources.

10.2 Stars treated as spheres

For problems in which the finite geometrical size of the star (or stars) is/are important, RADMC-3D has a mode by which the stars are treated as spheres. This can be necessary for instance if you model a disk around a Brown Dwarf, where the dusty disk goes all the way down to the stellar surface. The finite size of the star can thus shine *down* onto the disk, but only if its finite size is treated as such. In the default point-source approximation the surface layers of such a disk would be too cold, because this 'shining down onto the disk' phenomenon is not treated.

You can switch this mode on by setting `istar_sphere=1` in the file `radmc3d.inp`. Note that no limb darkening or brightening is included in this mode, and currently RADMC-3D does not have such a mode available.

This mode is, however, somewhat complex. A sphere can partly overlap the grid, while being partly outside the grid. A sphere can also overlap multiple cells at the same time, engulfing some cells entirely, while only partly overlapping others. The correct and fast treatment of this makes the code a bit slower, and required some complex programming. So the user is at the moment advised to use this mode only if necessary and remain aware of possible errors for now (as of version 0.17).

For the Monte Carlo simulations the finite star size means that photon packages are emitted from the surface of the sphere of the star. It also means that any photon that re-enters the star during the Monte Carlo simulation is assumed to be lost.

10.3 Distributions of zillions of stars

For models of galaxies it is important to be able to have distributed stellar sources instead of individual stars. The way to implement this in a model for RADMC-3D is to

1. Prepare one or more *template stellar spectra*, for instance, one for each stellar type you wish to include. These must be specified in the file `stellarsrc_templates.inp` (see Section *INPUT (optional): stellarsrc_templates.inp*). Of course the more templates you have, the more memory consuming it becomes, which is of particular concern for models on large grids. You can of course also take a sum of various stellar types as a template. For instance, if we wish to include a 'typical' bulge stellar component, then you do not need to treat each stellar type of bulge stars separately. You can take the 'average spectrum per gram of average star' as the template and thus save memory.
2. For each template you must specify the *spatial distribution*, i.e. how many stars of each template star are there per unit volume in each cell. The stellar density is, in fact, given as gram-of-star/cm^3 (i.e. not as number density

of stars). The stellar spatial densities are specified in the file `stellarsrc_density.inp` (see Section *INPUT (optional): stellarsrc_density.inp*).

Note that if you have a file `stellarsrc_templates.inp` in your model directly, then the stellar sources are automatically switched on. If you do not want to use them, then you must delete this file.

The smooth stellar source distributions are nothing else than source functions for the radiative transfer with the spectral shape of the template stellar spectra from the `stellarsrc_templates.inp`. You will see that if you make a spectrum of your object, then even if the dust temperature etc is zero everywhere, you still see a spectrum: that of the stellar template(s). In the Monte Carlo simulations these stellar templates act as net sources of photons, that subsequently move through the grid in a Monte Carlo way.

Note that the smooth stellar source distributions assume that the zillions of stars that they represent are so small that they do not absorb any appreciable amount of radiation. They are therefore pure sources, not sinks.

10.4 The interstellar radiation field: external source of energy

You can include an *isotropic* interstellar radiation field in RADMC-3D. This will take effect both in the making of spectra and images, as well as in the Monte Carlo module.

The way to activate this is to make a file `external_source.inp` and fill it with the information needed (see Section *INPUT (optional): external_source.inp*).

10.4.1 Role of the external radiation field in Monte Carlo simulations

For the Monte Carlo simulations this means that photons may be launched from outside inward. The way that this is done is that RADMC-3D will make a sphere around the entire grid, just large enough to fit in the entire grid but not larger. Photon packages can freely leave this sphere. But if necessary, photon packages can be launched from this sphere inward. RADMC-3D will then calculate the total luminosity of this sphere, which is $L = 4\pi^2 I r^2_{\rm sphere}$ where I is the intensity. For monochromatic Monte Carlo it is simply $I = I_\nu$, while for the thermal Monte Carlo it is $I = \int_0^\infty I_\nu d\nu$, where I_ν is the intensity as specified in the file `external_source.inp`. Note that if the sphere would have been taken larger, then the luminosity of the external radiation field would increase. This may seem anti-intuitive. The trick, however, is that if the sphere is larger, then also more of these interstellar photons never enter the grid and are lost immediately. That is why it is so important that RADMC-3D makes the sphere as small as possible, so that it limits the number of lost photon packages. It also means that you, the user, would make the grid much larger than the object you are interested in, then RADMC-3D is forced to make a large sphere, and thus potentially many photons will get lost: they may enter the outer parts of the grid, but there they will not get absorbed, nor will they do much.

In fact, this is a potential difficulty of the use of the external sources: since the photon packages are lauchned from outside-inward, it may happen that only few of them will enter in the regions of the model that you, the user, are interested in. For instance, you are modeling a 3-D molecular cloud complex with a few dense cold starless cores. Suppose that no stellar sources exist in this model, only the interstellar radiation field. The temperature in the centers of these starless cores will be determined by the interstellar radiation field. But since the cores are very small compared to the total model (e.g. you have used AMR to refine the grid around/in these cores), the chance of each external photon package to 'hit' the starless core is small. It means that the larger the grid or the smaller the starless core, the more photon packages (`nphot`, see Section *The thermal Monte Carlo simulation: computing the dust temperature*) one must use to make sure that at least some of them enter the starless cores. If you choose `phot` too small in this case, then the temperature in these cores would remain undetermined (i.e. they will be zero in the results).

10.4.2 Role of the external radiation field in images and spectra

The interstellar radiation field also affects the images and spectra that you make. Every ray will start at minus-infinity with the intensity given by the external radiation field, instead of 0 as it would be if no external radiation field is specified. If you make an image, the background of your object will then therefore not be black. You can even make silhouette images like those of the famous silhouette disks in Orion.

But there is a danger: if you make spectra, then also the background radiation is inside the beam, and will thus contribute to the spectrum. In fact, the larger you make the beam the more you will pick up of the background. This could thus lead to the spectrum of your source to be swamped by the background if you do not specify a beam in the spectrum.

10.5 Internal heat source

Sometimes the gas and dust inside the object of interest gets heated up by some internal process such as friction, magnetic reconnection, chemical reactions, etc. A nice example is the 'viscous heating' inside an accretion disk. This net heat source can be included in RADMC-3D by creating a file `heatsource.inp`. The format of the file is described in Section *INPUT (optional): heatsource.inp*. It is the same as for other scalar fields.

With this input file you have to specify in each cell how much energy per second per cubic centimeter is released in the form of heat. This energy will then be emitted as radiation by the dust. The way the code does this in the Bjorkman & Wood algorithm is that it will launch photon packages from these cells. The difference with the stellar energy input (see Section *Distributions of zillions of stars*) is that the energy is first injected into the dust of the cell, and then emitted as thermal dust emission. The launching of the photon package is therefore always a thermal dust emission. In contrast, in the stellar energy input method of Section *Distributions of zillions of stars* the photon package is launched directly, with a wavelength randomly drawn from the local stellar spectrum shape. The difference between these two methods will be most apparent for optically thin models. For very optically thick cases, where the heat source is released deep inside an optically thick object, both methods will presumably yield the same result. Nevertheless, it is recommended to use the heat source method for cases such as chemical or viscous heating of the gas and dust, even for optically thick cases.

A note of caution: in spite of the fact that this heat source method allows you to add additional energy sources, the object of study must still be in local thermodynamic equilibrium (LTE). If the gas+dust mixture is flowing and experiences significant adiabatic heating and cooling events, then the LTE condition is no longer met and RADMC-3D will not be able to give reliable answers. Sometimes one may be able to fudge this in some clever way, but one should always be aware that strictly speaking the Bjorkman & Wood Monte Carlo method only works if in each cell all energy input (be it radiative absorption or an internal heat source) is balanced exactly by the same amount of radiative energy output. The algorithm computes the dust temperature on that assumption: it computes how much energy the cell gains (by the heat source or by absorbing photons) and then it requires that the temperature of the dust is such that precisely the same amount of radiative energy is emitted.

10.5.1 Slow performance of RADMC-3D with heat source

For very optically thick models, such as the inner regions of actively accreting dusty protoplanetary disks, the use of this heat source can lead to extremely slow performance. The reason is that all photons originating from this heat source will start their journey right in the middle of the most optically thick regions, requiring these photons to make gazillions of absorption/re-emission events before finally diffusing out. It should in principle work if the code runs long enough. But one must have some patience. The use of the Modified Random Walk method (see Section *Modified Random Walk method for high optical depths*) would then be useful to speed things up, but still it can take time.

A few things might be useful to consider. One is that protoplanetary disks only have such insane optical depths ($\tau \gtrsim 10^5$) if none of the dust has coagulated to bigger grains. This might be the correct assumption, especially in the very early phases of protoplanetary disk evolution. But dust coagulation is known to be quick, so it might equally

well be that, say, 90% of the small grain dust has already grown to larger grains, which have less opacity. This is of course just a pure guess. Another thing is that many MHD models of disk turbulence show that most of the energy is not released near the midplane, but instead at one or two scale heights above the midplane. Both considerations would lower the optical depth for the energy to get out of the disk, speeding up the calculation. And the outcoming spectrum or image it will presumably not be affected that much, because at the end of the day the effective temperature of the disk surface must anyway be such that it radiates away the internal heat, independent of how deep inside the disk this heat is released.

CHAPTER

ELEVEN

MODIFYING RADMC-3D: INTERNAL SETUP AND USER-SPECIFIED RADIATIVE PROCESSES

It has been mentioned several times before that as an alternative to the standard *compile once-and-for-all' philosophy, one can also use RADMC-3D by modifying the code directly so that ``radmc3d`* will have new functionality that might be of use for you. We refer to Section *Making special-purpose modified versions of RADMC-3D (optional)* for an in-depth description of how to modify the code in a way that is *non-invasive* to the main code. We urge the reader to read Section *Making special-purpose modified versions of RADMC-3D (optional)* first before continuing to read this chapter. In all of the following we assume that the editings to the fortran files are done in the local way described in Section *Making special-purpose modified versions of RADMC-3D (optional)* so that the original source files in the `src/` directory stay unaffected, and only local copies are edited.

11.1 Setting up a model *inside* of RADMC-3D

The most common reason for editing the code itself is for setting up the model *internally* rather than reading in all data via input files. For a list of advantages and disadvantages of setting models up internally as opposed to the standard way, see Section *Some caveats and advantages of internal model setup* below. Setting up a model within RADMC-3D is done by making a local copy of the file `userdef_module.f90` and editing it (see Section *Making special-purpose modified versions of RADMC-3D (optional)*). This file contains a set of standard subroutines that are called by the main program at special points in the code. Each subroutine has a special purpose which will be described below. By keeping a subroutine empty, nothing is done. By filling it with your own code lines, you can set up the density, temperature or whatever needs to be set up for the model. In addition to this you can do the following as well:

- Add new variables or arrays in the module header (above the `contains` command), which you can use in the subroutines of the `userdef_module.f90` module. You are completely free to add any new variables you like. A small tip: it may be useful (though not required) to start all their names with e.g. `userdef_` to make sure that no name conflicts with other variables in the code happen.
- Add new subroutines at will (below the `contains` command) which you can call from within the standard subroutines.
- Introduce your own `radmc3d` command-line options (see Section *The pre-defined subroutines of the userdef_module.f90*).
- Introduce your own `radmc3d.inp` namelist variables (see Section *The pre-defined subroutines of the userdef_module.f90*).

Often you still want some of the input data to be still read in in the usual way, using input files. For instance, you may want to still read the `dustopac.inp` and the opacities using the `dustkappa_xxx.inp` files. This is all possible. Typically, you simply keep the files you still want RADMC-3D to read, and omit the files that contain data that you allocate and set in the `userdef_module.f90`. This is all a bit complicated, so the best way to learn how to do this is to start from the example directories in which a model is set up with the `userdef_module.f90` method.

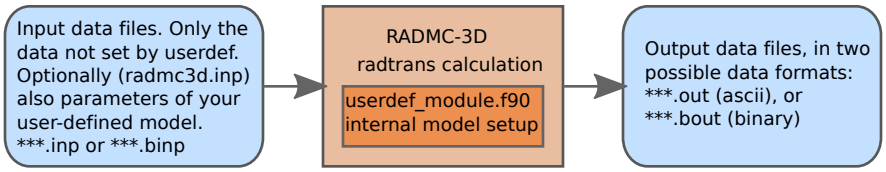

Fig. 11.1: Pictographic representation of the dataflow for the case when you define your model *internally* using the `userdef_module.f90`.

In Fig. Fig. 11.1 the dataflow for the user-defined model setup is graphically depicted.

11.2 The pre-defined subroutines of the userdef_module.f90

The idea of the `userdef_module.f90` is that it contains a number of standard pre-defined subroutines that are called from the `main.f90` code (and *only* from there). Just browse through the `main.f90` file and search for the sequence `calluserdef_` and you will find all the points where these standard routines are called. It means that at these points you as the user have influence on the process of model setup. Here is the list of standard routines and how they are used. They are ordered roughly in chronological order in which they are called.

- `userdef_defaults()`

  This subroutine allows you to set the default value of any new parameters you may have introduced. If neither on the command line nor in the `radmc3d.inp` file the values of these parameters are set, then they will simply retain this default value.

- `userdef_commandline(buffer,numarg,iarg,fromstdi,gotit)`

  This subroutine allows you to add your own command-line options for `radmc3d`. The routine has a series of standard arguments which you are not allowed to change. The `buffer` is a string containing the current command line option that is parsed. You will check here if it is an option of your module, and if yes, activate it. An example is listed in the code. You an also require a second argument, for which also an example is listed in the original code.

- `userdef_commandline_postprocessing()`

  After the command line options have been read, it can be useful to check if the user has not asked for conflicting things. Here you can do such checks.

- `userdef_parse_main_namelist()`

  Here you can add your own namelist parameters that read from the `radmc3d.inp` file. An example is provided in the original code.

- `userdef_main_namelist_postprocessing()`

  Also here, after the entire `radmc3d.inp` file has been read and interpreted, you can do some consistency checks and postprocessing here.

- `userdef_prep_model()`

  This routine can be used if you wish to set up the grid not from input files but internally. You will have to know how to deal with the `amr_module.f90` module. You can also set your own global frequency grid here. And finally, you can set your own stellar sources here. In all cases, if you set these things here (which requires you to make the proper memory allocations, or in case of the gridding, let the `amr_module.f90` do the memory allocations for you) the further course of `radmc3d` will skip any of its own settings (it will simply detect if these arrays are allocated already, and if yes, it will simply not read or allocate them anymore).

- `userdef_setup_model()`

  This is the place where you can actually make your own model setup. By the time this subroutine is called, all your parameters have been read in, as well as all of the other parameters from the original `radmc3d` code. So you can now set up the dust density, or the gas velocity or you name it. For all of these things you will have to allocate the arrays youself (!!!). Once you did this, the rest of the `radmc3d` code won't read those data anymore, because it detects that the corresponding arrays have already been allocated (by you). This allows you to completely circumvent the reading of any of the following files by making these data yourself here at this location:

  - `amr_grid.inp` or in the future the input files for any of the other griding types.
  - `dust_density.inp`
  - `dust_temperature.dat`
  - `gas_density.inp`
  - `gas_temperature.inp`
  - `gas_velocity.inp`
  - `microturbulence.inp`
  - `levelpop_XXX.dat`
  - `numberdens_XXX.inp`

  To learn how to set up a model in this way, we refer you for now to the `ioput_module.f90` or `lines_module.f90` and search for the above file names to see how the arrays are allocated and how the data are inserted. I apologise for not explaining this in more detail at this point. But examples are or will be given in the `examples/` directory.

- `userdef_dostuff()`

  This routine will be called by the main routine to allow you to do any kind of calculation after the main calculation (for instance after the monte carlo simulation). This is done within the execution-loop.

- `userdef_compute_levelpop()`

  This is a subroutine that can be called by the camera module for on-the-fly calculation of level populations according to your own recipe. This may be a bit tricky to use, but I hope to be able to provide some example(s) in the near future.

- `userdef_srcalp()`

  This subroutine allows you to add any emission/absorption process you want, even fake ones. For instance, you could use this to create nicely volume-rendered images of your 3-D models with fake opacities, which are chosen to make the image look nice and/or insight-giving. You can also use this to add physical processes that are not yet implemented in RADMC-3D. This subroutine allows you full freedom and flexibility to add emissivity and extinction whereever/however you like. To activate it you must set `incl_userdef_srcalp=1` in the `radmc3d.inp` file.

- `userdef_writemodel()`

  This allows the user to dump any stuff to file that the user computed in this module. You can also use this routine to write out files that would have been used normally as input file (like `amr_grid.inp` or `dust_density.inp`) so that the Python routines can read them if they need. In particular the grid information may be needed by these external analysis tools. Here is a list of standard subroutines you can call for writing such files:

  - `write_grid_file()`
  - `write_dust_density()`
  - ...more to come...

For now this is it, more routines will be included in the future.

Note that the `userdef_compute_levelpop()` subroutine, in contrast to all the others, is called not from the `main.f90` program but from the `camera_module.f90` module. This is why the camera module is the only module that is higher in compilation ranking than the userdef module (i.e. the userdef module will be compiled before the camera module). For this reason the userdef module has no access to the variables of the camera module. For the rest, the userdef module has access to the variables in all other modules.

Note also that not all input data is meant to be generated in this way. The following types of data are still supposed to be read from file:

- Dust opacity data
- Molecular fundamental data

Please have a look in the `examples/` directory for models which are set up in this internal way.

11.3 Some caveats and advantages of internal model setup

Setting up the models internally has several advantages as well as disadvantages compared to the standard way of feeding the models into `radmc3d` via files. The advantages are, among others:

- You can modify the model parameters in `radmc3d.inp` and/or in the command line options (depending on how you allow the user to set these parameters, i.e. in the `userdef_parse_main_namelist()` routine and/or in the `userdef_commandline()` routine. You then do not need to run Python anymore (except for setting up the basic files; see examples). Some advantages of this:
  - It allows you, for instance, to create a version of the `radmc3d` code that acts as if it is a special-purpose model. You can specify model parameters on the command line (rather than going through the cumbersome Python stuff).
  - It is faster: even a large model is built up quickly and does not require a long read from large input files.
- You can make use of the AMR module routines such as the `amr_branch_refine()` routine, so you can adaptively refine the grid while you are setting up the model.

Some of the disadvantages are:

- The model needs to be explicitly written out to file and read into Python or any other data plotting package before you can analyze the density structure to test if you've done it right. You can explicitly ask `./radmc3d` to call the `userdef_writemodel()` subroutine (which is supposed to be writing out all essential data; but that is the user's responsibility) by typing `./radmc3dwritemodel`.
- Same is true for the grid, and this is potentially even more dangerous if not done. You can explicitly ask `./radmc3d` to write out the grid file by typing `./radmc3dwritegridfile`. Note that if you call the `write_grid_file()` subroutine from within `userdef_writemodel()`, then you do not have to explicitly type `./radmc3dwritegridfile` as well. Note also that `radmc3d` will automatically call the `write_grid_file()` subroutine when it writes the results of the thermal Monte Carlo computation, if it has its grid from inside (i.e. it has not read the grid from the file `amr_grid.inp`.
- It requires a bit more knowledge of the internal workings of the `radmc3d` code, as you will need to directly insert code lines in the `userdef_module.f90` file.

11.4 Using the userdef module to compute integrals of J_ν

With the monochromatic Monte Carlo computation (see Section *Special-purpose feature: Computing the local radiation field*) we can calculate the mean intensity J_ν at every location in the model at a user-defined set of wavelengths. However, as mentioned before, for large models and large numbers of wavelengths this could easily lead to a data volume that is larger than what the computer can handle. Since typically the main motivation for computing J_ν is to compute some integral of the the form:

$$Q = \int_0^\infty J_\nu K_\nu d\nu$$

where K_ν is some cross section function or so, it may not be necessary to store the entire function J as a function of nu. Instead we would then only by interested in the result of this integral at each spatial location.

So it would be useful to allow the user to do this computation internally. We should start by initializing $Q(x, y, z) = 0$ (or $Q(r, \theta, \phi) = 0$ if you use spherical coordinates). Then we call the monochromatic Monte Carlo routine for the first wavelength we want to include, and multiply the resulting mean intensities with an appropriate $\Delta\nu$ and add this to $Q(x, y, z)$. Then we do the monochromatic Monte Carlo for the next wavelength and again add to Q everywhere. We repeat this until our integral (at every spatial location on the grid) is finished, and we are done. This saves a huge amount of memory.

Since this is somewhat hard to explain in this PDF document, we refer to the example model `run_example_jnu_integral/`.

STILL IN PROGRESS.

11.5 Some tips and tricks for programming user-defined subroutines

Apart from the standard subroutines that *must* be present in the `userdef_module.f90` file (see Section *The pre-defined subroutines of the userdef_module.f90*), you are free to add any subroutines or functions that you want, which you can call from within the predefined subroutines of Section *The pre-defined subroutines of the userdef_module.f90*. You are completely free to expand this module as you wish. You can add your own variables, your own arrays, allocate arrays, etc.

Sometimes you may need to know 'where you are' in the grid. For instance, the subroutine `userdef_compute_levelpop()` is called with an argument `index`. This is the index of the current cell from within which the subroutine has been called. You can now address, for instance, the dust temperature at this location:

```
temp = dusttemp(1,index)
```

(for the case of a single dust species). You may also want to know the coordinates of the center of the cell. For this, you must first get a pointer to the AMR-tree structure of this cell. The pointer `b` is declared as

```
type(amr_branch), pointer :: b
```

Then you can point the pointer to that cell structure

```
b => amr_index_to_leaf(index)%link
```

And now you can get the x,y,z-coordinates of the center of the cell:

```
xc = amr_finegrid_xc(b%ixyzf(1),1,b%level)
yc = amr_finegrid_xc(b%ixyzf(2),2,b%level)
zc = amr_finegrid_xc(b%ixyzf(3),3,b%level)
```

Or the left and right cell walls:

```
xi_l = amr_finegrid_xi(b%ixyzf(1),1,b%level)
yi_l = amr_finegrid_xi(b%ixyzf(2),2,b%level)
zi_l = amr_finegrid_xi(b%ixyzf(3),3,b%level)
xi_r = amr_finegrid_xi(b%ixyzf(1)+1,1,b%level)
yi_r = amr_finegrid_xi(b%ixyzf(2)+1,2,b%level)
zi_r = amr_finegrid_xi(b%ixyzf(3)+1,3,b%level)
```

11.6 Creating your own emission and absorption processes

RADMC-3D Allows you to add your own physics to the ray-tracing images and spectra. At every point during the ray-tracing process, when it computes the emissivity and extinction coefficients j_ν and α_ν it calls the `userdef_srcalp()` subroutine, giving it the `index` in which cell we are, the frequencies of the different image channels and the `src` and `alp` arrays which are for resp.j_ν and α_ν. You can *add* any process by

```
src(:) = src(:) + .....
alp(:) = alp(:) + .....
```

where ..... is your formula. You can find the local variables like density and temperature using the `index`, e.g.:

```
rho_g = gasdens(index)
```

You can be completely free in your choices. If you need some information that is not usually read into RADMC-3D, you can add read commands in the `userdef_setup_model()` subroutine, e.g.:

```
call read_gas_density(1)
```

See the example directory `examples/run_simple_userdefsrc` for more ideas.

CHAPTER

TWELVE

PYTHON ANALYSIS TOOL SET

While the code RADMC-3D is written in fortran-90, there is an extensive set of tools written in Python that make it easier for the user to set up models and interpret results. See Section *Installing the simple Python analysis tools* for where they are and how they can be properly installed so that they are easy to use.

The RADMC-3D package has two support-libraries:

1. `python/tools/simpleread.py`

   The `python/tools/simpleread.py` is a set of functions to read the most important data files used by RADMC-3D. However, the `simpleread.py` module is very simple, and does not read all RADMC-3D files in all formats. It can therefore only be used for certain (simple) models, and is primarily useful as a didactical tool.

2. `python/radmc3dPy`

   The `radmc3dPy` package is a stand-alone Python package, written by Attila Juhasz, meant for the pre- and post-processing of RADMC-3D files. It has its own manual, and has to be installed using e.g.~Python's `pipinstall` method. This is described in the README file in that package.

12.1 The simpleread.py library

For the most rudimentary analysis of the output (or input) files of RADMC-3D you can use the `simpleread.py` file, which you can find in the `python/tools/` directory. If everything has been installed correctly, you should be able to use it within Python like this:

```
from radmc3d_tools.simpleread import *
```

Examples of data files you can read:

```
d = read_dustdens()
d = read_dusttemp()
d = read_image()
d = read_spectrum()
d = read_dustkappa()
d = read_gastemp()
d = read_gasvelocity()
d = read_molnumdens('co')
d = read_mollevelpop('co')
d = read_subbox(name='dust_temperature')
d = read_subbox(name='dust_density')
```

Of course each one only if the corresponding file is present. Note that 'co' is just an example molecule. In all these reading functions, except the ones for images and spectra, the reading function automatically calls:

```
grid = read_grid()
```

which reads the information about the spatial grid. This is then put inside the `d` object like this: `d.grid`.

Here is an example of how you can plot the data (let us take the `examples/run_simple_1/` model, after we ran `radmc3d mctherm` and `radmc3d image incl 60 phi 30 lambda 1000`):

```
import matplotlib.pyplot as plt
from radmc3d_tools.simpleread import *
import radmc3d_tools.natconst as nc
tm = read_dusttemp()
plt.figure()
plt.plot(tm.grid.x/nc.au,tm.dusttemp[:,16,16])
plt.xlabel('x [au]')
plt.ylabel('T [K]')
im = read_image()
plt.figure()
plt.imshow(im.image[:,:,0],vmax=3e-14)
plt.show()
```

Important: These reading functions are rather basic. At the moment, no binary file support is included (though this may change), no AMR octree grids can be read, and several other limitations. For more sophisticated Python tools, use the radmc3dPy library.

Note: For the `read_subbox()` function, you need to read the section on the creation of regular-gridded datacubes of your 3D model, which is Section *Making a regularly-spaced datacube ('subbox') of AMR-based models*.

12.2 The radmc3dPy library

The `radmc3dPy` library is a sophisticated Python library that you can use for the in-depth analysis of the output (or input) files of RADMC-3D. It supports most in/output formats of RADMC-3D, including octree grids, binary file formats etc.

The package is stand-alone, and has its own bitbucket repository:

https://bitbucket.org/at_juhasz/radmc3dpy/

But you can find a copy of this package also inside the RADMC-3D package, in the directory `python/radmc3dPy/` .

The `radmc3dPy` package has its own manual, so we will not reiterate it here. Instead, please simply open the html manual in that package with a browser. The entry file of that manual is the `doc/html/index.html`. On a Mac you can simply type `opendoc/html/index.html` on the command line when you are in the `radmc3dPy` directory. To install `radmc3dPy` please consult the `README` file in the `radmc3dPy` directory.

Once it is installed, you can use `radmc3dPy` in Python in the following way:

1. Make sure to start Python 3 using {small ipython –matplotlib} if you start Python from the command line. If you instead use a Jupyter notebook, make sure that as a first line you use `%matplotlib inline` to get the plots inside the notebook. These are standard Python things, so if you have trouble, ask your python friends or system manager.
2. Once you are inside Python you can include `radmc3dPy` using a simple `from radmc3dPy import *`. This loads a series of radmc3dPy sub-libraries, including `analyze`, `image` and several others.

We give here a very concise overview of the `radmc3dPy` package. Please refer to the above mentioned stand-alone documentation for more details.

12.3 Model creation from within radmc3dPy

Several of the example models of the RADMC-3D `examples/` directory have been implemented as part of the `radmc3dPy` package. This allows you to launch these models straight from within `radmc3dPy`. But this is merely optional. You can equally well use the models in the `examples/` directory in the RADMC-3D package, and post-process the results with `radmc3dPy`.

To use one of the `radmc3dPy`-internal models, create a directory (e.g. `mymodel`), go into it, and go into iPython. Then type `from radmc3dPy import *`. By typing `models.getModelNames()` you get a list of available models. Suppose we choose the model 'ppdisk', then we would go about like this (for example):

```
from radmc3dPy import *
analyze.writeDefaultParfile('ppdisk')
setup.problemSetupDust('ppdisk', mdisk='1e-5*ms', gap_rin='[10.0*au]', gap_rout='[40.
↪*au]', gap_drfact='[1e-5]', nz='0')
```

This example will set up a protoplanetary disk model in 2-D (r, θ), with a gap between 10 and 40 au. You can now run RADMC-3D to compute the dust temperature structure, by calling (on the Linux shell):

```
radmc3d mctherm
```

An image can be created with (again on the Lunix shell):

```
radmc3d image lambda 1000 incl 60
```

And the image can be displayed (in Python) by

```
import matplotlib.pyplot as plt
from matplotlib import cm
from radmc3dPy import *
im=image.readImage()
image.plotImage(im,vmax=3e-3,au=True,cmap=cm.gist_heat)
```

12.4 Diagnostic tools in radmc3dPy

No matter whether you use the `radmc3dPy`-internal model set, or you create your own model setup, you can use the extensive tool set inside `radmc3dPy` to analyze the model itself, and the results of RADMC-3D calculations. In everything below, we assume that you use `from radmc3dPy import *` beforehand.

12.4.1 Read the `amr_grid.inp` file

Use `grid=analyze.readGrid()` to read the information about the spatial and wavelength grid.

12.4.2 Read all the spatial data

Using `data=analyze.readData()` you read the entire spatial structure of the model: The dust density, dust temperature, velocity etc.

12.4.3 Read the `image.out` file

Using `im=image.readImage()` you read the `image.out` file created by RADMC-3D (if you call radmc3d for creating an image). You can use the `image.plotImage()` function to display the image with the proper axes and color bar.

12.4.4 Read the `spectrum.out` file

Any spectrum you create (a file called `spectrum.out` can be read using `s=analyze.readSpectrum()`.

CHAPTER

THIRTEEN

ANALYSIS TOOLS INSIDE OF RADMC3D

There are also some special purpose features in the Fortran-90 `radmc3d` code that can be useful for analyzing complex AMR-gridded models.

13.1 Making a regularly-spaced datacube ('subbox') of AMR-based models

Because handling AMR-based models in Python or other data analysis packages can be rather cumbersome, we decided that it would be useful to create the possibility in `radmc3d` to generate 1-D, 2-D or 3-D regularly spaced 'cut-outs' or 'sub-boxes' (whatever you want to call them) of any variable of the model.

13.1.1 Creating a subbox

You can call `radmc3d` directly from the shell asking it to make the subbox. Here is an example:

```
./radmc3d subbox_dust_density subbox_nxyz 64 64 64 subbox_xyz01 -2.e15 2.e15 -2.e15 2.
↪e15 -2.e15 2.e15
```

which creates a regularly sampled 64x64x64 datacube of the dust density, with x grid between -2×10^{15} cm and $+2 \times 10^{15}$ cm and likewise for y and z (note that these box boundaries are the walls of the regularly spaced cells of the subbox). The file that this creates is called `dust_density_subbox.out` (see section *Format of the subbox output files* for the format of this file). For the dust temperature the command is `./radmc3d subbox_dust_temperature`, in which case the file is called `dust_temperature_subbox.out`.

You can also rotate the box along three angles: ϕ_1, θ, and ϕ_2, for example:

```
./radmc3d subbox_dust_temperature subbox_nxyz 64 64 64 subbox_xyz01 -2.e15 2.e15 -2.
↪e15 2.e15 -2.e15 2.e15 subbox_phi1 30 subbox_theta 60  subbox_phi2 45
```

(Note that as of version 2.0 of RADMC-3D these angles are in degrees instead of radian). An example for the level populations would be:

```
./radmc3d subbox_levelpop subbox_nxyz 64 64 64 subbox_xyz01 -2.e15 2.e15 -2.e15 2.e15␣
↪-2.e15 2.e15
```

Note about subbox for level populations: By default all level populations will be written out. However, if you would add the `subbox_levelpop` keyword in a call to RADMC-3D for making an image or spectrum, then it will only write out the level populations that have been used for that image. Example:

```
./radmc3d image lambda 2600 subbox_levelpop subbox_nxyz 64 64 64 subbox_xyz01 -2.e15␣
↪2.e15 -2.e15 2.e15 -2.e15 2.e15
```

would give a much smaller `'levelpop_co_subbox.out'` file, because only the first two levels are included (remember that $\lambda = 2600\,\mu$m is the J1-0 line of CO). See Section *Background information: Calculation and storage of level populations* for more information on how RADMC-3D automatically selects a subset of levels for storage in the global array (and thus also for writing out to file).

13.1.2 Format of the subbox output files

All the files produced by the subbox method have the following format:

```
iformat                                 <=== Typically 2 at present
nx ny nz nv                             <=== Box of nx*ny*nz cells, each with nv␣
↪values
x0 x1 y0 y1 z0 z1                       <=== The x, y and z boundaries of the box
phi1 theta phi2                         <=== Three rotation angles of the box
                                        <=== Empty line
1 2 3 4 ....                            <=== Identifications of the nv values
                                        <=== Empty line
data[ix=1,iy=1,iz=1,iv=1]
data[ix=2,iy=1,iz=1,iv=1]
.
.
data[ix=nx,iy=1,iz=1,iv=1]
data[ix=1,iy=2,iz=1,iv=1]
.
.
.
data[ix=nx,iy=ny,iz=nz,iv=1]
                                        <=== Empty line between components
data[ix=1,iy=1,iz=1,iv=2]
.
.
data[ix=nx,iy=ny,iz=nz,iv=2]
                                        <=== Empty line between components
.
.
.
                                        <=== Empty line between components
data[ix=1,iy=1,iz=1,iv=nv]
.
.
data[ix=nx,iy=ny,iz=nz,iv=nv]
```

and they are always in ascii format. For a subbox of the level populations the identification numbers are the levels. For instance, if only the populations of levels 4 and 8 are in this file, then `nv=2` and the line with the identification numbers will be `48`. For all other quantities (dust density, dust temperature) this line of identification numbers is simply `123` etc.

13.1.3 Using the `radmc3d_tools` to read the subbox data

In Section *The simpleread.py library* a set of simple Python tools are discussed to read a variety of output files from RADMC-3D (as well as input files to RADMC-3D) for further analysis.

Also for the subbox output there is now a Python function to read those. Example: First run RADMC-3D:

```
radmc3d mctherm
radmc3d subbox_dust_density subbox_nxyz 64 64 64 subbox_xyz01 -2.e14 2.e14 -2.e14 2.
↪e14 -2.e14 2.e14
radmc3d subbox_dust_temperature subbox_nxyz 64 64 64 subbox_xyz01 -2.e14 2.e14 -2.e14␣
↪2.e14 -2.e14 2.e14
```

Then go into Python and do:

```
from radmc3d_tools.simpleread import *
dustdens = read_subbox(name='dust_density')
dusttemp = read_subbox(name='dust_temperature')
grid     = dustdens.grid
import matplotlib.pyplot as plt
rhodustmin = 1e-18
plt.figure()
plt.imshow(np.log10(dustdens.data[:,:,32]+rhodustmin),extent=[grid.x[0],grid.x[-1],
↪grid.y[0],grid.y[-1]])
plt.figure()
plt.imshow(dusttemp.data[:,:,32])
plt.show()
```

13.2 Alternative to subbox: arbitrary sampling of AMR-based models

For some purposes it is useful to sample values of various quantities at arbitrary positions in the grid. The idea is very much like the subbox method of Section *Making a regularly-spaced datacube ('subbox') of AMR-based models*, but instead of a regular subbox grid the user provides a list of 3-D points where he/she wants to sample the variables of the model. Here is how to do this. First you must produce a file containing the list of 3-D positions. The file is called `sample_points.inp` and is an ascii file that looks as follows:

```
iformat                                  <=== Typically 1 at present
npt                                      <=== Nr of 3-D sampling points
xpt[1]  ypt[1]  zpt[1]                   <=== 3-D coordinates of point 1
xpt[2]  ypt[2]  zpt[2]                   <=== 3-D coordinates of point 2
xpt[3]  ypt[3]  zpt[3]                   <=== 3-D coordinates of point 3
...
...
```

An example for the case in which you want to sample at just one point:

```
1
1
1.49d13   4.02d14   1.03d12
```

If you want to let RADMC-3D do the sampling of the dust density and temperature, type (after you have calculated the temperature using `radmc3dmctherm`):

```
radmc3d sample-dustdens sample-dusttemp
```

You can also do the dust temperature calculation and the sampling in one go:

```
radmc3d mctherm sample-dustdens sample-dusttemp
```

You can also do only `sample-dusttemp` or only `sample-dustdens`. The output is written to files `dust_density_sample.out` resp.`dust_temperature_sample.out`. The format of these files is (take dust density as example):

```
iformat                                  <=== Typically 2 at present
npt  nv                                  <=== Nr of point and size of datavector
                                         <=== Empty line
1 2 3 4 ....                             <=== Identifications of the nv values
                                         <=== Empty line
dustdensity[ipt=1,iv=1]
dustdensity[ipt=2,iv=1]
...
dustdensity[ipt=npt,iv=1]
                                         <=== Empty line between components
dustdensity[ipt=1,iv=2]
...
dustdensity[ipt=npt,iv=2]
                                         <=== Empty line between components
...
                                         <=== Empty line between components
dustdensity[ipt=npt,iv=nv]
```

where `nv` is in this case the nr of species of dust and `iv=``ispecies``.

For a sample of the level populations the identification numbers are the levels. For instance, if only the populations of levels 4 and 8 are in this file, then `nv=2` and the line with the identification numbers will be `48`. For all other quantities (dust density, dust temperature) this line of identification numbers is simply `123` etc.

Later we will add other possible arrays to sample (at the moment it is only dust density, dust temperature and level populations). But you can also implement this yourself. Search in the following files for the following parts to add your own sampling:

- In `rtglobal_module.f90`: Search for `do_sample_dustdens` and add your own variable, e.g. `o_sample_myvariable`.
- In `main.f90`: Search for `do_sample_dustdens` and you will find all places where you have to add your own stuff, i.e. where you will have to add statements like `if(do_sample_myvariable)` or where you have to set `do_sample_myvariable=.true.` or reset `do_sample_myvariable=.false.` etc.

That should do it.

CHAPTER

FOURTEEN

VISUALIZATION WITH VTK TOOLS (E.G. PARAVIEW OR VISIT)

Since 3-D models can be very hard to visualize, and since RADMC-3D is not made for quick rendering, it can be very useful to make use of a number of freely available 3-D rendering tools, for example:

- Paraview www.paraview.org
- VisIt visit.llnl.gov

RADMC-3D can create data files for use with these tools. The file format is VTK (Visual Tool Kit), which is a simple ascii file format which is used by various programs. Those tools are not only useful for visualizing the 3-D structure of the model, but also for visualizing the structure of the grid which can be, when using AMR, rather complex.

The file that RADMC-3D writes is called `model.vtk`. You should be able to open it directly from within e.g. paraview. Figures Fig. 14.1 and Fig. 14.2 gives an example of how you can analyze a complex geometry with AMR refinement with Paraview. The file {em always} includes the information about the grid. In addition you can also make RADMC-3D add scalar fields or vector fields.

To create a VTK file for viewing the grid only, type:

```
radmc3d vtk_grid
```

To create a VTK file for viewing the gas density (this file then also includes the grid of course) type:

```
radmc3d vtk_gas_density
```

Since density can span a huge range, the 10-log of the density (in units of gram/cm^3) is written instead. For the gas temperature:

```
radmc3d vtk_gas_temperature
```

which is written in Kelvin (and linearly, not log). For the dust density of dust species 1:

```
radmc3d vtk_dust_density 1
```

and for dust species 2:

```
radmc3d vtk_dust_density 2
```

Also these densities are 10-log. RADMC-3D typically computes the dust temperature using a Monte Carlo approach. By typing

```
radmc3d vtk_dust_temperature 1
```

RADMC-3D will try to read the dust temperature from the file `dust_temperature.dat` (if this file has been created earlier by a `radmc3d mctherm` call) and then create the VTK file. You can also let RADMC-3D compute the temperature directly and write it out to VTK right afterward:

```
radmc3d mctherm vtk_dust_temperature 1
```

If you are doing line transfer you may wish to visualize the number density of the molecules (or atoms):

```
radmc3d vtk_molspec 1
```

(for molecular species 1). This number density (in cm^{-3}) is also written in 10-log form. You may also wish to visualize the polulations of level 1 (ground state) of molecule 2:

```
radmc3d vtk_levelpop 2 1
```

The gas velocity field can be written to VTK file by

```
radmc3d vtk_velocity
```

This is a vector field.

Note: The VTK mode works for 3-D cartesian and 3-D spherical coordinates (thanks, Attila Juhasz, for the 3-D spherical mode!).

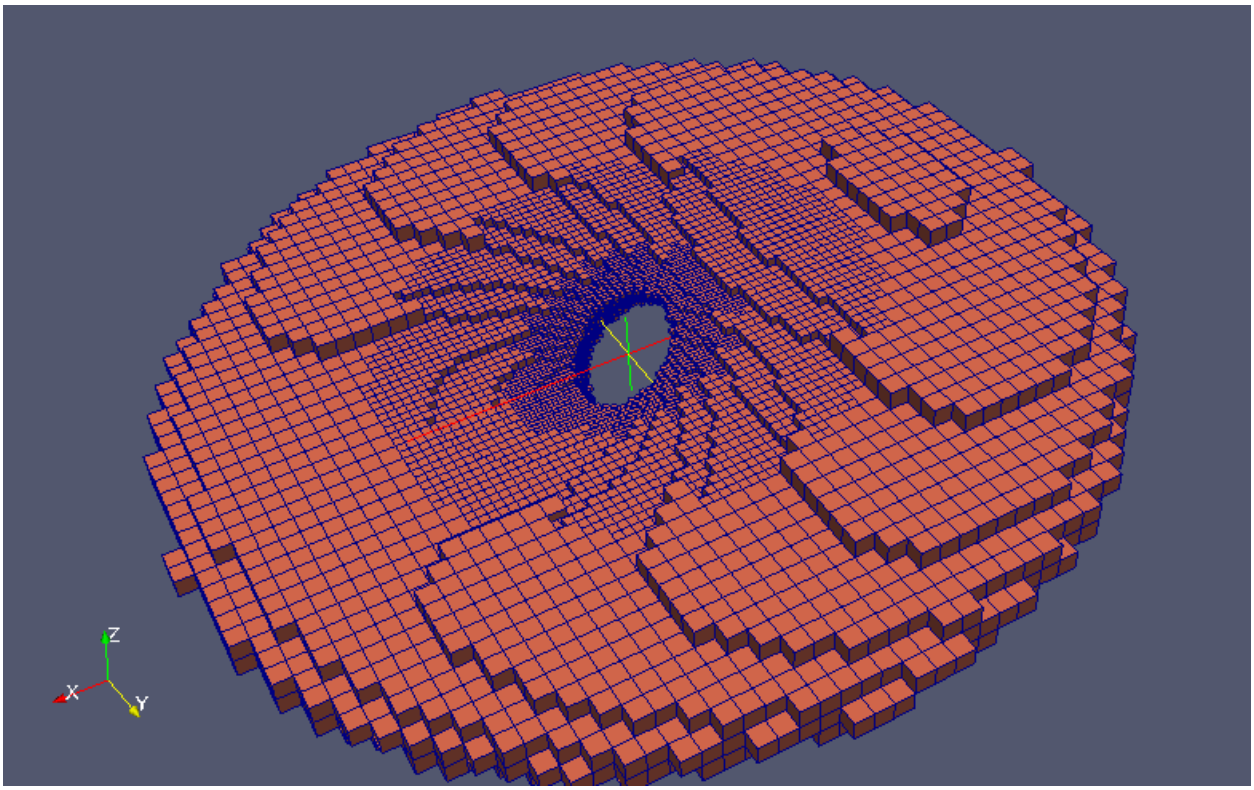

Fig. 14.1: Example of image created with Paraview, using the VTK output of RADMC-3D. The model shown here is a warped disk model by Katherine Rosenfeld, in 3-D cartesian coordinates with oct-tree AMR refinement.

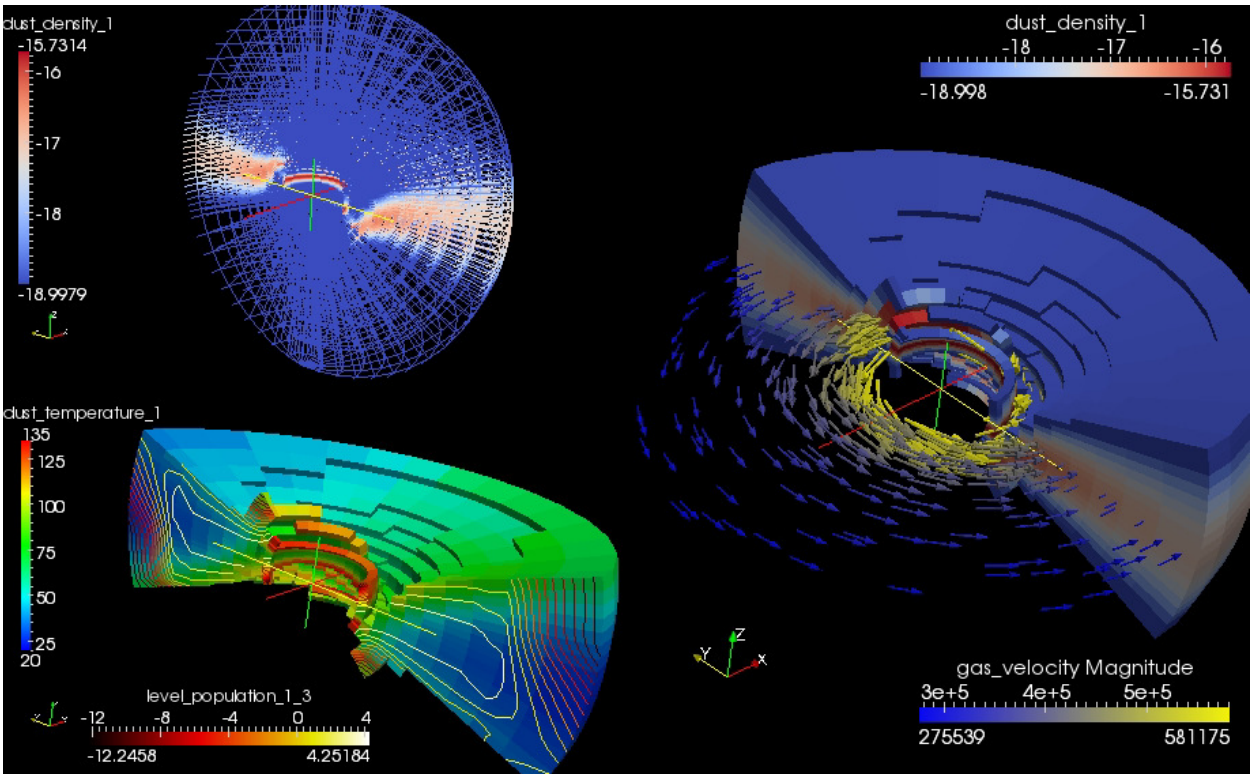

Fig. 14.2: Example of image created with Paraview, using the VTK output of RADMC-3D. The model shown here is a warped disk model by Attila Juhasz, in 3-D spherical coordinates with separable refinement, but without AMR refinement. The model is kept low-resolution on purpose, to show the grid structure better.

CHAPTER

FIFTEEN

TIPS, TRICKS AND PROBLEM HUNTING

15.1 Tips and tricks

RADMC-3D is a large software package, and the user will in all likelihood not understand all its internal workings. In this section we will discuss some issues that might be useful to know when you do modeling.

- *Things that can drastically slow down ray-tracing:*

  When you create images or spectra, `radmc3d` will perform a ray-tracing calculation. You may notice that sometimes this can be very fast, but for other problems it can be very slow. This is because, depending on which physics is switched on, different ray-tracing strategies must be followed. For instance, if you use a dust opacity without scattering opacity (or if you switch dust scattering off by setting `scattering_mode_max` to 0 in the `radmc3d.inp` file), and you make dust continuum images, or make SEDs, this may go very rapidly: less than a minute on a modern computer for grids of 256x256x256. However, when you include scattering, it may go slower. Why is that? That is because at each wavelength `radmc3d` will now have to make a quick Monte Carlo scattering model to compute the dust scattering source function. This costs time. And it will cost more time if you have `nphot_scat` set to a high value in the `radmc3d.inp` file, although it will create better images. Furthermore, if you *also* include gas lines using the simple LTE or simple LVG methods, then things become even slower, because each wavelength channel image is done after each other, and each time all the populations of the molecular levels have to be re-computed. If dust scattering would be switched off (which is for some wavelength domains presumably not a bad approximation; in particular for the millimeter domain), then no scattering Monte Carlo runs have to be done for each wavelength. Then the code can ray-trace all wavelength simultaneously: each ray is traced only once, for all wavelength simultaneously. Then the LTE/LVG level populations have to be computed only once at each location along the ray. So if you use dust and lines simultaneously, it can be advantageous for speed if you can afford to switch off the dust scattering, for instance, if you model sub-millimeter lines in regions with dust grains smaller than 10 micron or so. If you must include scattering, but your model is not so big that you may get memory limitation problems, then you may also try the fast LTE or fast LVG modes: in those modes the level populations are pre-computed before the ray-tracing starts, which saves time. But that may require much memory.

15.2 Bug hunting

Although we of course hope that `radmc3d` will not run into troubles or crash, it is nevertheless possible that it will. There are several ways by which one can hunt for bugs, and we list here a few obvious ones:

- In principle the `Makefile` should make sure that all dependencies of all modules are correct, so that the most dependent modules are compiled last. But during the further development of the code perhaps this may be not 100% guaranteed. So try do `makeclean` followed by `make` (or `makeinstall`) to assure a clean make.
- In the `Makefile` you can add (or uncomment) the line `BCHECK=-fbounds-check`, if you use `gfortran`. Find the array boundary check switch on your own compiler if it is not `gfortran`.

- Make sure that in the `main.f90` code the variable `debug_check_all` is set to 1. This will do some on-the-fly checks in the code.

15.3 Some tips for avoiding troubles and for making good models

Here is a set of tips that we recommend you to follow, in order to avoid troubles with the code and to make sure that the models you make are OK. This list is far from complete! It will be updated as we continue to develop the code.

1. Make a separate directory for each model. This avoids confusion with the many input and output files from the models.
2. When experimenting: regularly keep models that work, and continue experimenting with a fresh model directory. If things go wrong later, you can always fall back on an older model that *did* work well.
3. Keep model directories within a parent directory of the code, just like it is currently organized. This ensures that each model is always associated to the version of the code for which it was developed. If you update to a new version of the code, it is recommended to simply copy the models you want to continue with to the new code directory (and edit the `SRC` variable in the `Makefile` if you use the techniques described in Section *Making special-purpose modified versions of RADMC-3D (optional)* and Chapter *Modifying RADMC-3D: Internal setup and user-specified radiative processes*).
4. If you make a new model, try to start with as clean a directory as possible. This avoids that you accidently have a old files hanging around, their presence of which may cause troubles in your new model. So if you make a model update, make a new directory and then copy only the files that are necesary (for instance, `problem_setup.py`, `dustkappa_silicate.inp`, `Makefile` and other necessary files). One way of doing this easily is to write a little perl script or csh script that does this for you.
5. In the example model directories there is always a `Makefile` present, even if no local `*.f90` files are present. The idea is that by typing {small make cleanall} you can safely clean up the model directory and restore it to pre-model status. This can be useful for safely cleaning model directories so that only the model setup files remain there. It may save enormous amounts of disk space. But of course, it means that if you revisit the model later, you would need to redo the Monte Carlo simulations again, for instance. It is a matter of choice between speed of access to results on the one hand and disk space on the other hand.
6. If you use LVG or escape probability to compute the level populations of molecules, please be aware that you must include all levels that could be populated, not only the levels belonging to the line you are interested in.

15.4 Careful: Things that might go wrong

In principle RADMC-3D should be fine-tuned such that it produced reliable results in most circumstances. But radiative transfer modeling, like all kinds of modeling, is not an entirely trivial issue. Extreme circumstances can lead to wrong results, if the user is not careful in doing various sanity checks. This section gives some tips that you, the user, may wish to do to check that the results are ok. This is not an exhaustive list! So please remain creative yourself in coming up with good tests and checks.

1. *Too low number of photon packages for thermal Monte Carlo*

   If the number of photon packages for the thermal Monte Carlo simulation (Section *The thermal Monte Carlo simulation: computing the dust temperature*) is too low, the dust temperatures are going to be very noisy. Some cells may even have temperature zero. This may not only lead to noisy images and spectra, but also simply wrong results. However, deep inside optically thick clouds (or protoplanetary disks) it will be hard to avoid this problem. Since those regions are very deep below the $\tau = 1$ surface, it might not be always too critical in that case. A bit of experimenting might be necessary.

2. *Too low number of photon packages for scattering*

   When making images or spectra in which dust scattering is important, the scattered light emissivity is computed by a quick Monte Carlo simulation before the ray-tracing (see Section *Scattered light in images and spectra: The 'Scattering Monte Carlo' computation*). This requires the setting of the number of photon packages used for this (the variable `nphot_scat` for images and equivalently `nphot_spec` for spectra, both can be set in the `radmc3d.inp` file). If you see too much 'noise' in your scattering image, you can improve this by setting `nphot_scat` to a larger value (default = 100000). If your spectrum contains too much noise, try setting `nphot_spec` to a larger value (default = 10000).

3. *Too optically thick cells at the surface or inner edge*

   You may want to experiment with grid resolution and refinement. Strictly speaking the transition from optically thin to optically thick, as seen both by the radiation entering the object and by the observer, has to occur over more than one cell. That is for very optically thick models, one may need to introduce grid refinement in various regions. As an example: an optically thick protoplanetary disk may have an extremely sharp thin-thick transition near the inner edge. To get the spectra and images right, it is important that these regions are resolved by the grid (note: once well inside the optically thick interior, it is no longer necessary to resolve individual optical mean free paths, thankfully). It should be said that in practice it is often impossible to do this in full strictness. But you may want to at least experiment a bit with refining the grid (using either 'separable refinement', see Section *Separable grid refinement in spherical coordinates (important!)*, or AMR refinement, see Section *Oct-tree-style AMR grid*). An example how wrong things can go at the inner edge of a protoplanetary disk, if the inner cells are not assured to be optically thin through grid refinement (and possibly additionally a bit of smoothing of the density profile too) is given in Fig. Fig. 15.1.

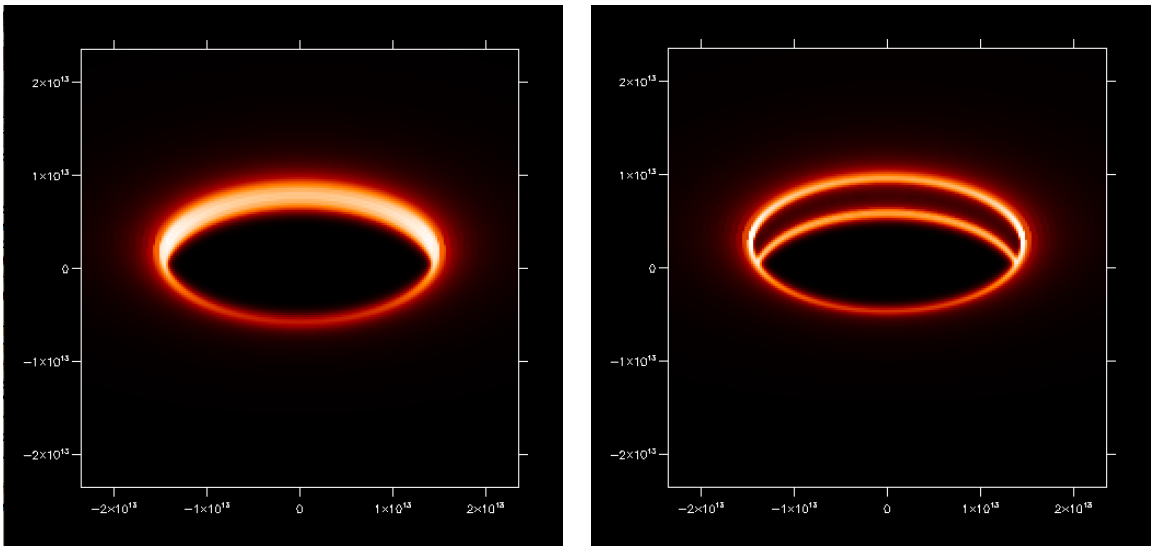

Fig. 15.1: Example of what can go wrong with radiative transfer if the inner cells of a model are optically thick (i.e.if no grid refinement is used, see Section *Separable grid refinement in spherical coordinates (important!)*). Shown here are scattered light images at $\lambda = 0.7\mu$Delta R/R=0.04`. Left image: the inner cells are marginally optically thin $\Delta\tau \simeq 1$, creating a bright inner ring, as is expected. Right image: ten times higher optical depth, making the inner cells optically thick with roughly $\Delta\tau \simeq 10$, resulting in a wrong image in which the emission near the midplane is strongly reduced. The reason for that is that the scattering source function, due to photons scattering at the inner 10% of the inner cell, is diluted over the entire cell, making the scattered light brighness 10x lower than it should be.

4. *Model does not fit onto the grid (or onto the refined part of the grid)*

   The grid must be large enough to contain the entire $\tau_\lambda = 1$ surface of a model at all relevant wavelengths. If you use grid refinement, the same is true for the $\tau_\lambda = 1$ surface being within the refinened part of the grid. This is not trivial! If you, for instance, import a 3-D hydrodynamic model into RADMC-3D, then it is a common problem

that the $\tau_\lambda = 1$ surface 'wants' to be outside of the grid (or outside of the higher-resolution part of the θ-grid if you use separable grid refinement: see Fig. fig-spher-sep-ref). For example: if you make a * hydrodynamic* model of a protoplanetary disk in R, Θ and Φ coordinates, you typically want to model only the lower 2 pressure scale heights of the disk, since that contains 99.5% of the mass of the disk. However, for *radiative transfer* this may not be enough, since if the disk has an optical depth of $\tau = 10^3$, the optically thin surface layer is less than 0.1% of the disk mass, meaning that you need to model the lower 3 (not 2!) pressure scale heights. Simply inserting the hydrodynamics model with the first 2 scale heights would lead to an artifical cut-off of the disk. In other words, the real $\tau_\lambda = 1$ surface 'wants' to be outside of the grid (or outside of the refined part of the grid). This leads to wrong results.

15.5 Common technical problems and how to fix them

When using a complex code such as RADMC-3D there are many ways you might encounter a problem. Here is a list of common issues and tips how to fix them.

1. *After updating RADMC-3D to a new version, some setups don't work anymore.*

   This problem can be due to several things:

   - When your model makes a local `radmc3d` executable (see Section *Making special-purpose modified versions of RADMC-3D (optional)*), for instance when you use the `userdef_module.f90` to set up the model, then you may need to edit the `SRC` variable in the `Makefile` again to point to the new code directory, and type `makeclean` followed by `make`.
   - Are you sure to have recompiled `radmc3d` again *and* installed it (by going in `src/` and typing `makeinstall`)?
   - Try going back to the old version and recheck that the model works well there. If that works, and the above tricks don't fix the problem, then it may be a bug. Please contact the author.

2. *After updating RADMC-3D to a new version: the new features are not present/working.*

   Maybe again the `Makefile` issue above.

3. *After updating RADMC-3D to a new version: model based on userdef_module fails to compile*

   If you switch to a new version of the code and try to 'make' an earlier model that uses the userdef_module.f90, it might sometimes happen that the compilation fails because some subroutine `userdef_***` is not known (here `***` is some name). Presumably what happened is that a new user-defined functionality has been added to the code, and the corresponding subroutine `userdef_***` has been added to the `userdef_module.f90`. If, however, in your own `userdef_module.f90` this subroutine is not yet built in, then the compiler can't find this subroutine and complains. Solution: just add a dummy subroutine to your `userdef_module.f90` with that name (have a look at the `userdef_module.f90` in the `src/` directory). Then recompile and it should now work.

4. *While reading an input file, RADMC-3D says 'Fortran runtime error: End of file'*

   This can of course have many reasons. Some common mistakes are:

   - In `amr_grid.inp` you may have specified the coordinates of the nx*ny*nz grid centers instead of (nx+1)*(ny+1)*(nz+1) grid cell interfaces.
   - You may have no line feed at the end of one of the ascii input files. Some fortran compilers can read only lines that are officially ended with a return or line feed. Solution: Just write an empty line at the end of such a file.

5. *My changes to the main code do not take effect*

   Did you type, in the `src/` directory, the full `makeinstall`? If you type just `make`, then the code is compiled but not installed as the default code.

6. *My userdef_module.f90 stuff does not work*

   If you run `radmc3d` with own userdefined stuff, then you must make sure to run the right executable. Just typing `radmc3d` in the shell might cause you to run the standard compilation instead of your special-purpose one. Try typing `./radmc3d` instead, which forces the shell to use the local executable.

7. *When I make images from the command line, they take very long*

   If you make images with `radmc3dimage` (plus some keywords) from the command line, the default is that a flux-conserving method of ray-tracing is used, which is called recursive sub-pixeling (see Section *The issue of flux conservation: recursive sub-pixeling*). You can make an image without sub-pixeling with the command-line option `nofluxcons`. That goes much faster, and also gives nice images, but the flux (the integral over the entire image) may not be accurate.

8. *My line channel maps (images) look bad*

   If you have a model with non-zero gas velocities, and if these gas velocities have cell-to-cell differences that are larger than or equal to the intrinsic (thermal+microturbulent) line width, then the ray-tracing will not be able to pick up signals from intermediate velocities. In other words, because of the discrete gridding of the model, only discrete velocities are present, which can cause numerical problems. There are two possible solutions to this problem. One is the wavelength band method described in Section *Heads-up: In reality wavelength are actually wavelength bands*. But a more systematic method is the 'doppler catching' method described in Section *Preventing doppler jumps: The 'doppler catching method'* (which can be combined with the wavelength band method of Section *Heads-up: In reality wavelength are actually wavelength bands* to make it even more perfect).

9. *My line spectra look somewhat noisy*

   If you include dust continuum scattering (Section *More about scattering of photons off dust grains*) then the ray-tracer will perform a scattering Monte Carlo simulation at each wavelength. If you look at lines where dust scattering is still a strong source of emission, and if `nphot_scat` (Section *Scattered light in images and spectra: The 'Scattering Monte Carlo' computation*) is set to a low value, then the different random walks of the photon packages in each wavelength channel may cause slightly different resulting fluxes, hence the noise.

10. *My dust continuum images look very noisy/streaky: many 'lines' in the image*

    There are two possible reasons:

    1. *Photon noise in the thermal Monte Carlo run:* If you have too few photon packages for the thermal Monte Carlo computation (see Chapter *Dust continuum radiative transfer*), then the dust temperatures are simply not well computed. This may give these effects. You must then increase `nphot` in the `radmc3d.inp` file to increase the photon statistics for the thermal Monte Carlo run.

    2. *Photon noise in the scattering Monte Carlo run:* If you are making an image at a wavelength at which the disk is not emitting much thermal radiation, then what you will see in the image is scattered light. `RADMC-3D` makes a special Monte Carlo run for scattered light before each image. This Monte Carlo run has its own variable for setting the number of photon packages: `nphot_scat`. If this value is set too low, then you can see individual 'photon'-trajectories in the image, making the image look bad. It is important to note that this can only be remedied by increasing `nphot_scat` (in the `radmc3d.inp` file, see Section *Scattered light in images and spectra: The 'Scattering Monte Carlo' computation*), not by setting `nphot` (which is the number of photon packages for the thermal Monte Carlo computation). Please also read Section *Single-scattering vs. multiple-scattering* for a detailed discussion about the effects of multiple scattering and the possibility of it leading to streaks in the images.

    However, it might also mean that something is wrong with the setup. A few common setup-errors that could cause these issues are:

    - Accidently created a way too massive object. Let us discuss this with an example of a protoplanetary disk: suppose you created, in spherical coordinates, not a protoplanetary disk with $M_{\rm disk} = 0.01\,M_\odot$ but accidently one with $M_{\rm disk} = 10\,M_\odot$. In such a case a lot of things will go wrong. First of all the

inner edge of the disk will almost certainly behave strangely (see Fig. *Example of what can go wrong with radiative transfer if the inner cells of a model are optically thick (i.e.if no grid refinement is used, see Section sec-separable-refinement). Shown here are scattered light images at \lambda=0.7\muDelta R/R=0.04`. Left image: the inner cells are marginally optically thin \Delta\tau\simeq 1, creating a bright inner ring, as is expected. Right image: ten times higher optical depth, making the inner cells optically thick with roughly \Delta\tau\simeq 10, resulting in a wrong image in which the emission near the midplane is strongly reduced. The reason for that is that the scattering source function, due to photons scattering at the inner 10% of the inner cell, is diluted over the entire cell, making the scattered light brighness 10x lower than it should be.*). Secondly, the surface of the disk will almost certainly be cut-off in the way decribed in Section *Careful: Things that might go wrong*, in which case the surface of the disk will be hardly illuminated by the star, because the disk surface is then exactly conical (i.e.starlight will not be able to impinge on the surface). This will lead to very low photon statistics at the surface.

CHAPTER

SIXTEEN

MAIN INPUT AND OUTPUT FILES OF RADMC-3D

RADMC-3D is written in fortran-90. It is written in such a way that the user prepares input files (ending in `.inp`) for the program and then calls `radmc3d` with some command-line options. The program then reads the input files, and based on the command-line options will perform a certain calculation, and finally outputs the results to output files (ending in `.out`) or intermediate files (ending in `.dat`) which need further processing. In principle the user therefore needs to compile the program only once, and can then use the executable from that point onward. In this chapter we will describe the various input/output and intermediate files and their formats. Just for clarity: the Python routines in the `python/` directory are only meant to make it easier for the user to prepare the `.inp` files, and to make sense of the `.out` and `.dat` files. They are not part of the main code `radmc3d`.

A few comments on RADMC-3D input and output files:

- Most (though not all) files start with a *format number*. This number simply keeps track of the version of the way the information is stored the file. The idea is that if new versions of RADMC-3D come out in the future, it would be good to have the possibility that new information is added to the files. The format number is there to tell RADMC-3D whether a file is the new version or still an older version.
- RADMC-3D has four types of I/O files:
  1. Files ending with `.inp` or `.binp` are input files that allow the user to specify to RADMC-3D which problem to solve.
  2. Files ending with `.dat` or `.bdat`are intermediate files that are typically created by RADMC-3D itself, but can also be read by RADMC-3D for further processing. For instance, the dust temperature is computed by the Monte Carlo method, but can also be read in later for ray-tracing.
  3. Files ending with `.out` or `.bout` are final products of RADMC-3D, such as an image or spectrum.
  4. File ending with `.info` are small files containing some numbers that are useful to better interpret the output files of RADMC-3D. They are typically not very important for every-day use.
- For many of the I/O files RADMC-3D can read and write formatted (i.e. text style: ascii) files, or binary files (i.e. C-style unformatted). This is specified by the file extension. See Chapter *Binary I/O files* for more details.

16.1 INPUT: radmc3d.inp

The `radmc3d.inp` file is a namelist file with the main settings for RADMC-3D. The namelist is not a standard Fortran namelist style, but a simple *name* = *value* list. If a name is not specified, the default values are taken. So if the `radmc3d.inp` file is empty, then all settings are standard. Note that some of these settings can be overwritten by command-line options! Here is a non-exhaustive list of the variables that can be set.

- `incl_dust` (default: depends on which input files are present)

Normally RADMC-3D will recognize automatically whether dust continuum emission, absorption and scattering must be included: if e.g. a file called `dustopac.inp` is present, it assumes that the dust must be included. But with this flag you can explicitly tell RADMC-3D whether it must be included (1) or not (0).

- `incl_lines` (default: depends on which input files are present)

  Normally RADMC-3D will recognize automatically whether line emission and absorption must be included: if e.g. a file called `lines.inp` is present, it assumes that molecular/atomic lines must be included. But with this flag you can explicitly tell RADMC-3D whether it must be included (1) or not (0).

- `nphot` or `nphot_therm` (default: 100000)

  The number of photon packages used for the thermal Monte Carlo simulation.

- `nphot_scat` (default: 100000)

  The number of photon packages for the scattering Monte Carlo simulations, done before image-rendering.

- `nphot_spec` (default: 10000)

  The number of photon packages for the scattering Monte Carlo simulations, done during spectrum-calculation. This is actually the same functionality as for `nphot_scat`, but it is used (and only used) for the spectrum and SED calculations. The reason to have a separate value for this is that for spectra you may not need as many photon packages as for imaging, because you anyway integrate over the images. Many of the annoying 'stripe noise' in images when using insufficiently large `nphot_scat` will cancel each other out in the flux calculation. So `nphot_spec` is usually taken smaller than `nphot_scat`.

- `nphot_mono` (default: 100000)

  The number of photon packages for the Monte Carlo simulations for the `mcmono` calculation (see Section *Special-purpose feature: Computing the local radiation field*).

- `iseed` (default: -17933201) *[Fine-tuning only]*

  A starting value of the random seed for the Monte Carlo simulation.

- `ifast` (default: 0) *[Fine-tuning only]*

  By setting this to 1 or 2 you will get a faster Monte Carlo simulation, at the cost of being less accurate.

- `enthres` (default: 0.01) *[Fine-tuning only]*

  This is the fraction by which the energy in each cell may increase before the temperature is recalculated in the Monte Carlo simulation. The smaller this value, the more accurate the thermal Monte Carlo simulation, but the more computationally costly. 0.01 has proven to be fine.

- `itempdecoup` (default: 1)

  If set to 0, then the temperatures of all coexisting dust species are always forced to be the same. If 1, then each dust species is thermally independent of the other.

- `istar_sphere` (default: 0)

  If 0 (=default), then all stars are treated as point-sources. If 1, then all stars are treated as finite-size spheres. This mode is more accurate and more realistic, but the applications are a bit more restricted. Such finite-size stars are (for technical reasons) not always allowed anywhere in the model. But for problems of circumstellar disks and envelopes in spherical coordinates, it is recommended to set this to 1. Typically, if a star is outside the grid (in spherical coordinates this can also be at the origin of the coordinate system, as long as the inner radius of the coordinate system is larger than the stellar radius!) the use of the finite-size star mode is always possible. But if the star is on the grid, there are technical limitations.

- `ntemp` (default: 1000) *[Fine-tuning only]*

The temperatures are determined in the Monte Carlo method using tabulated pre-computed integrals. This saves time. This is the number of temperatures for which this is precalculated. The temperatures are sampled in a logarithmic way, i.e. log(temp) is linearly equally spaced between log(temp0) and log(temp1), see below.

- `temp0` (default: 0.01) *[Fine-tuning only]*

  The lowest pre-calculated temperature.

- `temp1` (default: 1e5) *[Fine-tuning only]*

  The highest pre-calculated temperature.

- `scattering_mode_max`

  When `radmc3d` reads the dust opacity files it checks if one or more of the opacity files has scattering opacity included. If yes, the `scattering_mode` will automatically be set to 1. It will also check if one or more includes *anisotropic* scattering. If yes, the `scattering_mode` will automatically be set to 2. But the user *may* nevertheless want to exclude anisotropic scattering or exclude scattering altogether (for instance for testing purposes, or if the user knows from experience that the scattering or anisotropic nature of scattering is not important for the problem at hand). Rather than editing the opacity files to remove the scattering and/or Henyey-Greenstein g-factors, you can limit the value that `radmc3d` is allowed to make `scattering_mode` by setting the variable `scattering_mode_max`. If you set `scattering_mode_max=0` then no matter what opacity files you have, scattering will not be treated. If you set `scattering_mode_max=1`, then no matter what opacity files you have, scattering will be treated in an isotropic way.

- `unformatted` (Obsolete)

- `rto_style` (default=1)

  This determines whether the output of space-dependent data will be in ASCII form (`rto_style=1`), f77-unformatted form (`rto_style=2`, obsolete) or binary form (`rto_style=3`). See Chapter *Binary I/O files* for details.

- `camera_tracemode` (default: 1)

  If `camera_tracemode`=-1, the images that are rendered by RADMC-3D will instead by the column depth traced along each ray. If `camera_tracemode`=-2, the images that are rendered by RADMC-3D will instead by the continuum optical depth traced along each ray. By default `camera_tracemode`=1, which is the normal mode, where real images are being created.

- `camera_nrrefine` (default: 100)

  For images: to assure that flux is correctly sampled, the image pixels will not just be rendered one ray per pixel. Instead, if necessary, a pixel will spawn 2x2 sub-pixels recursively (each of which can split again into 2x2 until the required resolution is obtained) so as to assure that the flux in each pixel is correct. `camera_nrrefine` tells how deep RADMC-3D is allowed to recursively refine. 100 is therefore effectively infinite. Putting this to 0 means that you go back to 1 ray per pixel, which is fast, but may seriously misrepresent the flux in each pixel. See Section *The issue of flux conservation: recursive sub-pixeling* for more details.

- `camera_refine_criterion` (default: 1.0) *[Fine-tuning only]*

  Setting this value to smaller than 1 means that you refine the recursive pixeling until a tighter criterion is met. The smaller this value, the more accurate the fluxes in each pixel, but the longer it takes to render. See Section *The issue of flux conservation: recursive sub-pixeling* for more details.

- `camera_incl_stars` (default: 1)

  If 0, then only the interstellar/circumstellar material is rendered for the images and spectra. If 1, then also the stellar flux is included in the spectra and images.

- `camera_starsphere_nrpix` (default: 20) *[Fine-tuning only]*

For rectangular images and for the spectra/SEDs (but not for spectra/SEDs created with circular pixel arrangements, see Section *Circular images*), this number tells RADMC-3D how much it should do sub-pixeling over the stellar surface. That is: 20 means that at least 20 sub-pixels are assured over the stellar surface. This is important for flux conservation (see Section *The issue of flux conservation: recursive sub-pixeling*).

- `camera_spher_cavity_relres` (default: 0.05) *[Fine-tuning only]*

  Determines the size of sub-pixels inside the inner grid radius of spherical coordinates.

- `camera_localobs_projection` (default: 1)

  (Only for local observer mode) The type of projection on the sphere of observation.

- `camera_min_dangle` (default 0.05) *[Fine-tuning only]*

  Fine-tuning parameter for recursive subpixeling (see Section *The solution: recursive sub-pixeling*), for spherical coordinates, assuring that not too fine subpixeling would slow down the rendering of images or spectra too much.

- `camera_max_dangle` (default 0.3) *[Fine-tuning only]*

  Fine-tuning parameter for recursive subpixeling (see Section *The solution: recursive sub-pixeling*), for spherical coordinates, preventing that too coarse subpixeling would reduce the accuracy.

- `camera_min_dr` (default 0.003) *[Fine-tuning only]*

  Fine-tuning parameter for recursive subpixeling, for spherical coordinates, assuring that not too fine subpixeling would slow down the rendering of images or spectra too much.

- `camera_diagnostics_subpix` (default: 0)

  Setting this to 1 forces RADMC-3D to write out a file called `subpixeling_diagnostics.out` which contains four columns, for respectivly: `px,py,pdx,pdy`, i.e. the pixel position and its size. This is for all pixels, including the sub-pixels created during the recursive subpixeling procedure (Section *The solution: recursive sub-pixeling*). This allows the user to find out if the recursive subpixeling went well or if certain areas were over/under-resolved. This is really only meant as a diagnostic.

- `camera_secondorder` (default: 0)

  If set to 1, RADMC-3D will interpolate all emission/absorption quantities to the cell corners, and then use a second order integration routine with bilinear interpolation of the source terms to integrate the ray-tracing formal transfer equations. See Section *Second order ray-tracing (Important information!)* for more information about the second order integration: It is recommended to read it!

- `camera_interpol_jnu` (default: 0) *[Fine-tuning only]*

  Fine-tuning parameter for ray-tracing, only used for when second order integration is done (i.e. if `camera_secondorder=1`). If 0 (default), then the source function S_ν is the one that is interpolated on the grid, while if 1, then the emissivity j_ν is the one that is interpolated on the grid. The differences are minimal, but if strange results appear (when using second order integration) then you may want to experiment a bit with this parameter.

- `mc_weighted_photons` (default: 1) *[Fine-tuning only]*

  If `mc_weighted_photons=1` (default) then in Monte Carlo simulations not all photon packages will have the same energy (see Section *More about photon packages in the Monte Carlo simulations*). The energy will be weighted such that each star or emission mechanism will emit, on average, the same number of photon packages. As an example: If you have a stellar binary consisting of an O-star surrounded by a Brown Dwarf, but the Brown Dwarf is surrounded by a disk, then although the O star is much brighter than the O-star, the very inner regions of the Brown Dwarf disk is still predominantly heated by the Brown Dwarf stellar surface, because it is much closer to that material. If you do not have weighted photon packages, then statistically the Brown Dwarf would emit perhaps 1 or 2 photon packages, which makes the statistics of the energy balance in the inner disk very bad. By `mc_weighted_photons=1` both the Brown Dwarf and the O-star will each emit the same number

of photon packages; just the energy of the photon packages emitted by the Brown Dwarf are much less energetic than those from the O-star. This now assures a good photon statistics everywhere.

- `optimized_motion` (default: 0) *[Fine-tuning only]*

  If `optimized_motion` is set to 1, then RADMC-3D will try to calculate the photon motion inside cells more efficiently. This may save computational time, but since it is still not very well tested, please use this mode with great care! It is always safer not to use this mode.

- `lines_mode` (default: 1)

  This mode determines how the level populations for line transfer are computed. The default is 1, which means: Local Thermodynamic Equilibrium (LTE). For other modes, please consult Chapter *Line radiative transfer*.

- `lines_maxdoppler` (default: 0.3) *[Fine-tuning only]*

  If the doppler catching mode is used (see Section *Preventing doppler jumps: The 'doppler catching method'*), this parameter tells how fine RADMC-3D must sample along the ray, in units of the doppler width, when a line is doppler-shifting along the wavelength-of-sight.

- `lines_partition_ntempint` (default 1000) *[Fine-tuning only]*

  Number of temperature sampling points for the internally calculated partition function for molecular/atomic lines.

- `lines_partition_temp0` (default 0.1) *[Fine-tuning only]*

  Smallest temperature sampling point for the internally calculated partition function for molecular/atomic lines.

- `lines_partition_temp1` (default 1E5) *[Fine-tuning only]*

  Largest temperature sampling point for the internally calculated partition function for molecular/atomic lines.

- `lines_show_pictograms` (default 0)

  If 1, then print a pictogram of the levels of the molecules/atoms.

- `tgas_eq_tdust` (default: 0)

  By setting `tgas_eq_tdust=1` you tell `radmc3d` to simply read the `dust_temperature.inp` file and then equate the gas temperature to the dust temperature. If multiple dust species are present, only the first species will be used.

- `subbox_nx,subbox_ny,subbox_nz,subbox_x0,subbox_x1,subbox_y0,subbox_y1, subbox_z0,subbox_z1`

  Parameters specifying the subbox size for the subbox extraction. See Section *Making a regularly-spaced datacube ('subbox') of AMR-based models* for details.

16.2 INPUT (required): amr_grid.inp or unstr_grid.inp

This is the file that specifies what the spatial grid of the model looks like. See Chapter *More information about the gridding*. This file is essential, because most other `.inp` and `.dat` files are simple lists of numbers which do not contain any information about the grid. All information about the grid is contained in the `amr_grid.inp`, also for non-AMR regular grids, or alternatively in the `unstr_grid.inp` file for unstructured grids.

There are three possible AMR grid styles:

- Regular grid: No mesh refinement. This is grid style 0.
- Oct-tree-style AMR ('Adaptive Mesh Refinement', although for now it is not really 'adaptive'). This is grid style 1.

- Layer-style AMR. This is grid style 10.

Alternatively, there are a variety of possible unstructured grids:

- Delaunay
- Voronoi
- Self-designed

16.2.1 Regular grid

For a regular grid, without grid refinement, the `amr_grid.inp` looks like:

```
iformat                                    <=== Typically 1 at present
0                                          <=== Grid style (regular = 0)
coordsystem
gridinfo
incl_x       incl_y       incl_z
nx           ny           nz
xi[1]        xi[2]        xi[3]       ........  xi[nx+1]
yi[1]        yi[2]        yi[3]       ........  yi[ny+1]
zi[1]        zi[2]        zi[3]       ........  zi[nz+1]
```

The meaning of the entries are:

- `iformat`: The format number, at present 1. For unformatted files this must be 4-byte integer.
- `coordsystem`: If `coordsystem < 100` the coordinate system is cartesian. If `100 <= coordsystem < 200` the coordinate system is spherical (polar). Cylindrical coordinates have not yet been built in in this version. For unformatted files this must be 4-byte integer.
- `gridinfo`: If `gridinfo==1` there will be abundant grid information written into this file, possibly useful for post-processing routines. Typically this is redundant information, so it is advised to set `gridinfo=0` to save disk space. In the following we will assume that `gridinfo=0`. For unformatted files this must be 4-byte integer.
- `incl_x,incl_y,incl_z`: These are either 0 or 1. If 0 then this dimension is not active (so upon grid refinement no refinement in this dimension is done). If 1 this dimension is fully active, even if the number of base grid cells in this direction is just 1. Upon refinement the cell will also be splitted in this dimension. For unformatted files these numbers must be 4-byte integer.
- `nx,ny,nz`: These are the number of grid cells on the base grid in each of these dimensions. For unformatted files these numbers must be 4-byte integer.
- `xi[1]...xi[nx+1]`: The edges of the cells of the base grid in x-direction. For `nx` grid cells we have `nx+1` cell walls, hence `nx+1` cell wall positions. For unformatted files these numbers must be 8-byte reals (=double precision).
- `yi[1]...yi[ny+1]`: Same as above, but now for y-direction.
- `zi[1]...zi[nz+1]`: Same as above, but now for z-direction.

Example of a simple 2x2x2 regular grid in cartesian coordinates:

```
1
0
1
0
1  1  1
```

(continues on next page)

(continued from previous page)

```
2  2  2
-1.  0. 1.
-1.  0. 1.
-1.  0. 1.
```

16.2.2 Oct-tree-style AMR grid

For a grid with oct-tree style grid refinement (see Section *Oct-tree Adaptive Mesh Refinement*), the `amr_grid.inp` looks like:

```
iformat                                   <=== Typically 1 at present
1                                         <=== Grid style (1 = Oct-tree)
coordsystem
gridinfo
incl_x       incl_y       incl_z
nx           ny           nz
levelmax     nleafsmax    nbranchmax      <=== This line only if grid style == 1
xi[1]        xi[2]        xi[3]       ........  xi[nx+1]
yi[1]        yi[2]        yi[3]       ........  yi[ny+1]
zi[1]        zi[2]        zi[3]       ........  zi[nz+1]
(0/1)                  <=== 0=leaf, 1=branch (only if amrstyle==1)
(0/1)                  <=== 0=leaf, 1=branch (only if amrstyle==1)
(0/1)                  <=== 0=leaf, 1=branch (only if amrstyle==1)
(0/1)                  <=== 0=leaf, 1=branch (only if amrstyle==1)
(0/1)                  <=== 0=leaf, 1=branch (only if amrstyle==1)
(0/1)                  <=== 0=leaf, 1=branch (only if amrstyle==1)
(0/1)                  <=== 0=leaf, 1=branch (only if amrstyle==1)
(0/1)                  <=== 0=leaf, 1=branch (only if amrstyle==1)
(0/1)                  <=== 0=leaf, 1=branch (only if amrstyle==1)
...
...
```

The keywords have the same meaning as before, but in addition we have:

- `(0/1)`: *NOTE: Only for* `amrstyle==1`. These are numbers that are either 0 or 1. If 0, this means the current cell is a leaf (= a cell that is not refined and is therefore a 'true' cell). If 1, the current cell is a branch with 2 (in 1-D), 4 (in 2-D) or 8 (in 3-D) daughter cells. In that case the next (0/1) numbers are for these daughter cells. In other words, we immediately recursively follow the tree. The order in which this happens is logical. In 3-D the first daughter cell is (1,1,1), then (2,1,1), then (1,2,1), then (2,2,1), then (1,1,2), then (2,1,2), then (1,2,2) and finally (2,2,2), where the first entry represents the x-direction, the second the y-direction and the third the z-direction. If one or more of the daughter cells is also refined (i.e. has a value 1), then first this sub-tree is followed before continuing with the rest of the daughter cells. If we finally return to the base grid at some point, the next (0/1) number is for the next base grid cell (again possibly going into this tree if the value is 1). The order in which the base grid is scanned in this way is from `1` to `nx` in the innermost loop, from `1` to `ny` in the middle loop and from `1` to `nz` in the outermost loop. For unformatted files these numbers must be 4-byte integers, one record per number.

Example of a simple 1x1x1 grid which is refined into 2x2x2 and for which the (1,2,1) cell is refined again in 2x2x2:

```
1
1
1
0
1  1  1
```

(continues on next page)

(continued from previous page)

```
1  1  1
10 100 100
-1.  1.
-1.  1.
-1.  1.
1
0
0
1
0
0
0
0
0
0
0
0
0
0
0
0
```

16.2.3 Layer-style AMR grid

For a grid with layer-style grid refinement (see Section *Layered Adaptive Mesh Refinement*), the `amr_grid.inp` looks like:

```
iformat                                  <=== Typically 1 at present
10                                       <=== Grid style (10 = layer-style)
coordsystem
gridinfo
incl_x       incl_y       incl_z
nx           ny           nz
nrlevels     nrlayers                    <=== This line only if grid style == 10
xi[1]        xi[2]        xi[3]       ........  xi[nx+1]
yi[1]        yi[2]        yi[3]       ........  yi[ny+1]
zi[1]        zi[2]        zi[3]       ........  zi[nz+1]
parentid     ix  iy  iz   nx  ny  nz
parentid     ix  iy  iz   nx  ny  nz
parentid     ix  iy  iz   nx  ny  nz
parentid     ix  iy  iz   nx  ny  nz
.
.
.
```

The keywords have the same meaning as before, but in addition we have:

- `nrlevels`: How many levels you plan to go, where `nrlevels==0` means no refinement, `nrlevels==1` means one level of refinement (factor of 2 in resolution), etc.
- `nrlayers`: How many layers do you have, with `nrlayers==0` means no refinement, `nrlayers==1` means one layer of refinement (factor of 2 in resolution), etc.
- `parentid`: (For each layer) The parent layer for this layer. `parentid==0` means parent is base grid. First layer has `id==1`.

- `ix,iy,iz`: (For each layer) The location in the parent layer where the current layer starts.
- `nx,ny,nz`: (For each layer) The size of the layer as measured in units of the the parent layer. So the actual size of the current layer will be (in 3-D): `2*nx,2*ny,2*nz`. In 2-D, with only the x- and y- dimensions active, we have a size of `2*nx,2*ny` with of course size 1 in z-direction.

As you can see, this is a much easier and more compact way to specify mesh refinement. But it is also less 'adaptive', as it is always organized in square/cubic patches. But it is much easier to handle for the user than full oct-tree refinement.

Note that this layer-style refinement is in fact, internally, translated into the oct-tree refinement. But you, as the user, will not notice any of that. The code will input and output entirely in layer style.

NOTE: The layers must be specify in increasing refinement level! So the first layer (layer 1) must have the base grid (layer 0) as its parent. The second layer can have either the base grid (layer 0) or the first layer (layer 1) as parent, etc. In other words: the parent layer must always already have been specified before.

Example of a simple 2-D 4x4 grid which has a refinement patch in the middle of again 4x4 cells (=2x2 on the parent grid), and a patch of 2x2 (=1x1 on the parent grid) starting in the upper left corner:

```
1
100
1
0
1  1  0
4  4  1
1  2
-2. -1. 0. 1. 2.
-2. -1. 0. 1. 2.
-0.5 0.5
0  2  2  1  2  2  1
0  1  1  1  1  1  1
```

This has just one level of refinement, but two patches at level 1.

Anothe example: two recursive layers. Again start with a 2-D 4x4 grid, now refine it in the middle with again a 4x4 sub-grid (=2x2 on the parent grid = layer 0) and then again a deeper layer of 4x4 (=2x2 on the parent grid = layer 1) this time starting in the corner:

```
1
100
1
0
1  1  0
4  4  1
2  2
-2. -1. 0. 1. 2.
-2. -1. 0. 1. 2.
-0.5 0.5
0  2  2  1  2  2  1
1  1  1  1  2  2  1
```

Note that with this layer-style grid, the input data will have to be speficied layer-by-layer: first the base grid, then the first layer, then the second etc. This is worked out in detail for `dust_density.inp` in Section *INPUT (required for dust transfer): dust_density.inp*. This will include redundant data, because you specify the data on the entire base grid, also the cells that later will be replaced by a layer. Same is true for any layer that has sub-layers. The data that is specified in these regions will be simply ignored. But for simplicity we do still require it to be present, so that irrespective of the deeper layers, the data in any layer (including the base grid, which is layer number 0) is simply organized as a simple data cube. This redundancy makes the input and output files larger than strictly necessary, but it is much easier to handle as each layer is a datacube. For memory/hardisk-friendly storage you must use the oct-tree

refinement instead. The layers are meant to make the AMR much more accessible, but are somewhat more memory consuming.

16.2.4 Unstructured grid

An unstructured grid (no matter whether Delaunay, Voronoi or general type) is specified in the file called `unstr_grid.inp`:

```
iformat                          <=== Typically 2 at present
ncells                           <=== Nr of cells (both "closed" and "open" ones)
nwalls                           <=== Nr of walls
nverts                           <=== Nr of vertices
cell_max_nr_walls                <=== Max number of walls per cell
cell_max_nr_verts                <=== Max number of vertices per cell (0 means: not␣
↪relevant)
wall_max_nr_verts                <=== Max number of vertices per wall (required if␣
↪saving vertices)
vert_max_nr_cells                <=== Max number of cells per vertex (0 means: not␣
↪relevant)
ncells_open                      <=== Nr of "open" cells
hull_nwalls                      <=== Nr of cell walls at the surface
isave_volumes                    <=== Include a list of cell volumes?
isave_sn_vectors                 <=== Include a list of s and n vectors?
isave_cellcenters                <=== Include a list of cell center positions?
isave_vertices                   <=== Include the list of vertices?
isave_size                       <=== Include the list of cell sizes?
hull_convex                      <=== Are the surface cell walls convex?
Volume of cell 1                 \  [If isave_volumes==1]
...                              |- Cell volumes
Volume of cell ncells            /
s and n of wall 1                \  [If isave_sn_vectors==1]
...                              |- Support and normal vectors of walls: s_x s_y s_z␣
↪n_x n_y n_z
s and n of wall nwalls           /
idx_left idx_right wall 1        \
...                              |- Left and right cell indices of each cell wall
idx_left idx_right wall nwalls   /
cell center cell 1               \  [If isave_cellcenters==1]
...                              |- Center point of cells: p_x p_y p_z
cell center cell ncells          /
idx_vertices wall 1              \  [If isave_vertices==1]
...                              |- Indices of vertices spanning wall (wall_max_nr_
↪verts for each
idx_vertices wall nwalls         |  wall, if too many: pad with 0)
vertex 1                         |
...                              |- Vertices: v_x v_y v_z
vertex nverts                    /
```

The `ncells` always has to give the number of cells. The order of the cells is always the same as the order of the physical variables in files such as `dust_density.inp` and the like. The `nwalls` always has to give the number of cell walls separating the cells. Each cell wall can only separate two cells. Even if more cells lie on the same plane, each pair of adjacent cells must share one and only one wall. It is not a problem if more than one wall lie in the same plane. The `nverts` is only required if you specify the vertices (i.e. if `isave_vertices==1`). Set to 0 if you don't need it.

Since each cell can have a multitude of walls, for interal bookkeeping it is useful to know in advance the maximum number of walls per cell, which is given by `cell_max_nr_walls` (if it is chosen too small, RADMC-3D will try

to internally correct it). Likewise for the maximum number of vertices per cell given by `cell_max_nr_verts`.

If (and only if) you provide the vertices of the grid (setting `isave_vertices=1`) you also need to specify the maximum number of vertices per wall `wall_max_nr_verts` (this you must specify correctly, as it determined how many vertex indices per wall are provided further down) and the maximum number of cells per vertex `vert_max_nr_cells`, which can be an estimate (RADMC-3D will try to correct it).

At the surface of the grid there are either 'hull walls' or 'open cells'. For Delaunay grids, and presumably for your own-designed grids, you will have 'hull walls' which have only 1 cell on one side (the first of the two cell indices specified) and the other side facing the vacuum. The hull walls should close the grid, separating the grid from the outside. For Voronoi grids the cells on the edge of the domain are, instead, open: They go off to infinity. You have to specify how many hull walls or open cells you have (RADMC-3D has only been tested for either hull walls or open cells, not a combination): `hull_nwalls` or `ncells_open`.

The way you specify the grid is flexible. By setting `isave_volumes=1` you give the cell volume for each cell (see below). Setting instead `isave_volumes=0` asks RADMC-3D to compute the cell volumes internally. So far RADMC-3D can only compute cell volumes if the cells are tetrahedral (i.e.~simplices), but maybe in the future this may change/improve. By setting `isave_sn_vectors=1` you give the support- and normal-vector of each wall. This makes the gridding very flexible, but it also costs a lot of disk space: 6 floats per wall. If you instead set `isave_sn_vectors=0` then you ask RADMC-3D to compute these internally from whatever other information is has (either from the vertices or by assuming Voronoi-type walls). By setting `isave_cellcenters=1` you give the exact location of each cell center point. For Voronoi grids this is a must, as the Voronoi grid is defined by its cell centers. For other grids (including Delaunay) this can be skipped (set `isave_cellcenters=0`) if the vertices are specified. If you set `isave_vertices=1` you specify the vertices. For Delaunay grids this is a must, as the Delaunay grid is defined by its vertices. For Voronoi grids you can omit these (by setting `isave_vertices=0`) as RADMC-3D will then construct the cell walls automatically assuming them to lie exactly in between the two points. Finally, for the radiative transfer calculations as well as for handling precision errors, it is useful for RADMC-3D to know what the 'size' is of a cell (even though with non-cubic cells a 'size' may not be uniquely defined). If not specified (`isave_size==0`) then RADMC-3D will estimate it as the cubic root of the volume. But if you set `isave_size=1` then you can specify, for each cell, this 'size'.

Depending on the settings of `isave_volumes`, `isave_sn_vectors`, `isave_cellcenters`, `isave_vertices`, and `isave_size`, the corresponding data are listed in the order shown above.

Here is an example of a `unstr_grid.inp` file for a Delaunay grid:

```
2                       <=== Format number 2
199462                  <=== Nr of cells (for Delaunay grids usually >> nr of vertices)
399691                  <=== Nr of walls (for Delaunay grids usually >> nr of vertices)
30000                   <=== Nr of vertices
4                       <=== For Delaunay grid: always exactly 4 cell walls per cell
4                       <=== For Delaunay grid: always exactly 4 vertices per cell
3                       <=== For Delaunay grid: always exactly 3 vertices per wall
0                       <=== Not necessary to specify
0                       <=== For Delaunay grid: no open cells
1534                    <=== Nr of walls at the surface (the hull)
0                       <=== For Delaunay grid: RADMC-3D computes the volume
0                       <=== For Delaunay grid: RADMC-3D computes the s and n
0                       <=== For Delaunay grid: RADMC-3D computes the cell centers
1                       <=== For Delaunay grid: Must specify the vertices
0                       <=== Compute the size from the volume**0.33333
1                       <=== For Delaunay grid: Yes, the hull is convex
1 0                     <=== First wall, connecting to cell 1 to the vacuum (i.e. part␣
↪of hull)
5 0
7 0
9 0
...                     <=== Many, many lines...
```

(continues on next page)

(continued from previous page)

```
154309 0
154310 0              <=== Wall connecting cell 154310 to the vacuum (last of the hull
↪walls)
1 45                  <=== Wall connecting cells 1 and 45 (first of the internal walls)
1 160
1 2
...                   <=== Many, many lines...
199460 199462
199461 199462         <=== Last cell wall, connecting cells 199461 and 199462
22156 15798 12402     <=== First cell wall: indices of the three vertices of this wall
19848 2591 5486
...                   <=== Many, many lines...
26358 1240 11951
26358 19577 11951     <=== Last cell wall: indices of the three vertices of this wall
-9.158735473036884375e+13 -7.011778907885211719e+13 6.039256438409164844e+13   <===
↪First vertex
-6.263232133764155469e+13 1.286041325985346719e+14 6.671766425868976562e+12
...                                                                             <===
↪Many, many lines...
-8.918647910718467188e+13 -7.056527384485303125e+13 -5.088009729690025000e+13
-6.440888241873650000e+13 -1.108250266699581299e+12 -2.545886642466098438e+13  <===
↪Last vertex
```

And here is an example of a `unstr_grid.inp` file for a Voronoi grid:

```
2                     <=== Format number 2
30000                 <=== Nr of cells (= number of points)
230058                <=== Nr of walls
0                     <=== Can be kept 0
33                    <=== Maximum nr of walls per cell (can be many for Voronoi cells)
0                     <=== Can be kept 0
0                     <=== Can be kept 0 because we don't list the vertices
0                     <=== Can be kept 0 because we don't list the vertices
769                   <=== Nr of open cells
0                     <=== Nr of hull walls (for Voronoi: must be 0)
1                     <=== For Voronoi grid: Must specify cell volumes
0                     <=== For Voronoi grid: RADMC-3D computes the s and n
1                     <=== For Voronoi grid: Cell centers must be specified
0                     <=== For Voronoi grid: Vertices not explicitly necessary
0                     <=== Compute the size from the volume**0.33333
0                     <=== Since we have no hull walls, this is irrelevant
2.772699983914995096e+38   <=== Volume of the first cell
3.855434192315507956e+38
0.000000000000000000e+00   <=== Zero volume = open cell
4.187946751113876048e+38
...                        <=== Many, many lines...
4.765148695599201280e+38
1.699867329158221159e+39   <=== Volume of the last cell
13738 5006                 <=== First cell wall connects cell 13738 with cell 5006
13738 29031
...                        <=== Many, many lines...
28953 29538
29177 29815                <=== Last cell wall connects cell 29177 with 29815
3.224145067583187109e+13 1.185761258774503594e+14 6.399869810360167969e+13    <===
↪First cell center
1.024309911261237656e+14 9.824700277721357812e+13 -2.709320785406102344e+13
...                                                                            <===
↪Many, many lines...
```

(continues on next page)

(continued from previous page)

```
-4.27852264356717109e+13 -1.51907701847051269e+13 1.09802990185917015e+14
-8.08782268184614062e+13 -1.15879422515319156e+14 4.05916591162385625e+13  <===␣
↪Last cell center
```

RADMC-3D does not need to explicitly know if a grid is Voronoi or Delaunay, because the input file, in particular which data are given (see the `isave_xxx` flags), automatically makes it correct. In the actual internal workings of RADMC-3D it does not care about whether it is Voronoi or Delaunay: it only cares about cell volumes and cell walls.

16.3 INPUT (required for dust transfer): dust_density.inp

This is the file that contains the dust densities. It is merely a list of numbers. Their association to grid cells is via the file `amr_grid.inp` (see Chapter *Binary I/O files* for the binary version of this file, which is more compact). Each dust species will have its own density distribution, completely independently of the others. That means that at each position in space several dust species can exist, and the density of these can be fully freely specified. The structure of this file is as follows. For formatted style (`dust_density.inp`):

```
iformat                                  <=== Typically 1 at present
nrcells
nrspec
density[1,ispec=1]
..
density[nrcells,ispec=1]
density[1,ispec=2]
..
..
..
density[nrcells,ispec=nrspec]
```

Here `nrspec` is the number of independent dust species densities that will be given here. It can be 1 or larger. If it is 1, then of course the `density[1,ispec=2]` and following lines are not present in the file. The `nrcells` is the number of cells. For different kinds of grids this can have different meaning. Moreover, for different kinds of grids the order in which the density values are given is also different. So let us now immediately make the following distinction (See Chapter *More information about the gridding* on the different kinds of grids):

- *For regular grid and oct-tree AMR grids:*

  The value of `nrcells` denotes the number of *true* cells, excluding the cells that are in fact the parents of 2x2x2 subcells; i.e. the sum of the volumes of all true cells (=leafs) adds up to the volume of the total grid). The order of these numbers is always the same 'immediate recursive subtree entry' as in the `amr_grid.inp` (Section *INPUT (required): amr_grid.inp or unstr_grid.inp*).

- *For layer-style AMR grids:*

  The value of `nrcells` denotes the number of values that are specified. This is generally a bit more than the true number of cells specified in the oct-tree style AMR (see above). In the layer-style AMR mode you specify the dust density (or any other value) first at all cells of the base grid (whether a cell is refined or not does not matter), the at all cells of the first layer, then the second layer etc. Each layer is a regular (sub-)grid, so the order of the values is simply the standard order (same as for regular grids). This means, however, that the values of the density in the regular grid cells that are replaced by a layer are therefore redundant. See Section *On the 'successively regular' kind of data storage, and its slight redundancy* for a discussion of this redundancy. The main advantage of this layer-style grid refinement is that the input and output always takes place on *regular* grids and subgrids (=layers). This is much easier to handle than the complexities of the oct-tree AMR.

16.3.1 Example: `dust_density.inp` for a regular grid

Now let us look at an example of a `dust_density.inp` file, starting with one for the simplified case of a regular 3-D grid (see Sections *Regular grid* and *Regular grids*):

```
iformat                                    <=== Typically 1 at present
nrcells
nrspec
density[1,1,1,ispec=1]
density[2,1,1,ispec=1]
..
density[nx,1,1,ispec=1]
density[1,2,1,ispec=1]
..
..
density[nz,ny,nz,ispec=1]
density[1,1,1,ispec=2]
..
..
..
density[nz,ny,nz,ispec=nrspec]
```

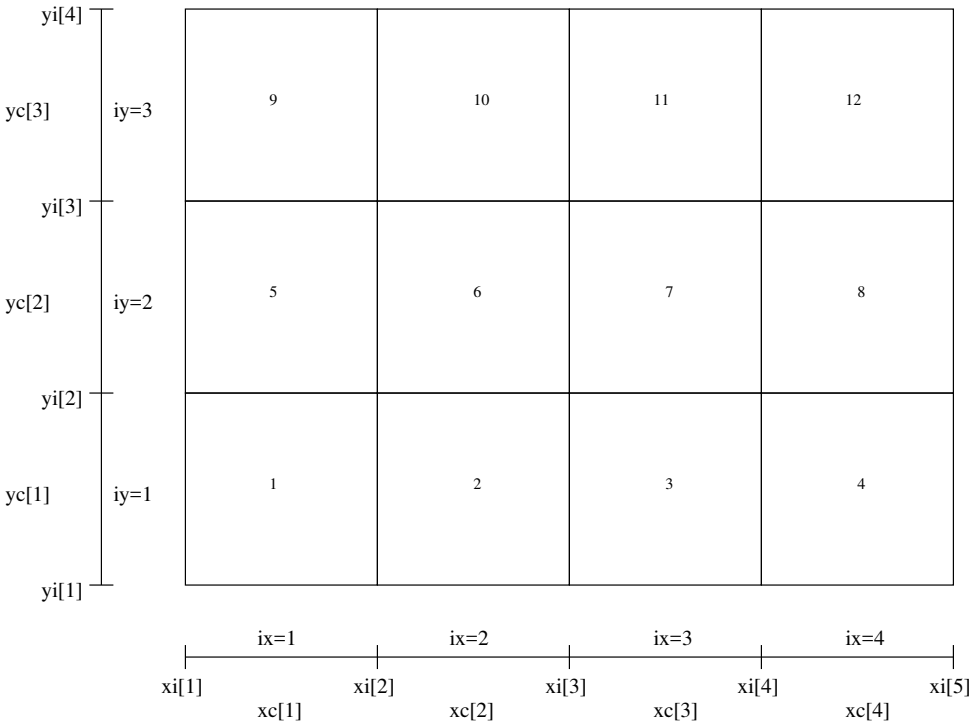

Fig. 16.1: Example of a regular 2-D grid with `nx`=4 and `ny`=3 (as Fig. Fig. 9.1), with the order of the cells shown as numbers in the cells.

16.3.2 Example: `dust_density.inp` for an oct-tree refined grid

For the case when you have an oct-tree refined grid (see Sections *Oct-tree-style AMR grid* and *Oct-tree Adaptive Mesh Refinement*), the order of the numbers is the same as the order of the cells as specified in the `amr_grid.(u)inp` file (Section *INPUT (required): amr_grid.inp or unstr_grid.inp*). Let us take the example of a simple 1x1x1 grid which is refined into 2x2x2 and for which the (1,2,1) cell is refined again in 2x2x2 (this is exactly the same example as shown in Section *Oct-tree-style AMR grid*, and for which the `amr_grid.inp` is given in that section). Let us also assume that we have only one dust species. Then the `dust_density.inp` file would be:

```
iformat                                 <=== Typically 1 at present
15                                      <=== 2x2x2 - 1 + 2x2x2 = 15
1                                       <=== Let us take just one dust spec
density[1,1,1]                          <=== This is the first base grid cell
density[2,1,1]
density[1,2,1;1,1,1]                    <=== This is the first refined cell
density[1,2,1;2,1,1]
density[1,2,1;1,2,1]
density[1,2,1;1,2,1]
density[1,2,1;1,1,2]
density[1,2,1;2,1,2]
density[1,2,1;1,2,2]
density[1,2,1;1,2,2]                    <=== This is the last refined cell
density[2,2,1]
density[1,1,2]
density[2,1,2]
density[1,2,2]
density[2,2,2]                          <=== This is the last base grid cell
```

A more complex example is shown in Fig. *Example of a 2-D grid with oct-tree refinement (as Fig. fig-oct-tree-amr) with the order of the cells shown as numbers in the cells.*. An unformatted version is also available, in the standard way (see above).

16.3.3 Example: `dust_density.inp` for a layer-style refined grid

For the case when you have an layer-style refined grid (see Sections *Layer-style AMR grid* and *Layered Adaptive Mesh Refinement*) you specify the density in a series of regular boxes (=layers). The first box is the base grid, the second the first layer, the third the second layer etc. The value `nrcells` now tells the combined sizes of the all the boxes. If we take the second example of Section *Layer-style AMR grid*: a simple 2-D 4x4 grid which has a refinement patch (=layer) in the middle of again 4x4 cells, and again one patch of 4x4 this time, however, starting in the upper left corner (see the `amr_grid.inp` file given in Section *Layer-style AMR grid*), then the `dust_density.inp` file has the following form:

```
iformat                                 <=== Typically 1 at present
48                                      <=== 4x4 + 4x4 + 4x4 = 48
1                                       <=== Let us take just one dust spec
density[1,1,1,layer=0]
density[2,1,1,layer=0]
density[3,1,1,layer=0]
density[4,1,1,layer=0]
density[1,2,1,layer=0]
density[2,2,1,layer=0]                  <=== This a redundant value
density[3,2,1,layer=0]                  <=== This a redundant value
density[4,2,1,layer=0]
density[1,3,1,layer=0]
density[2,3,1,layer=0]                  <=== This a redundant value
```

(continues on next page)

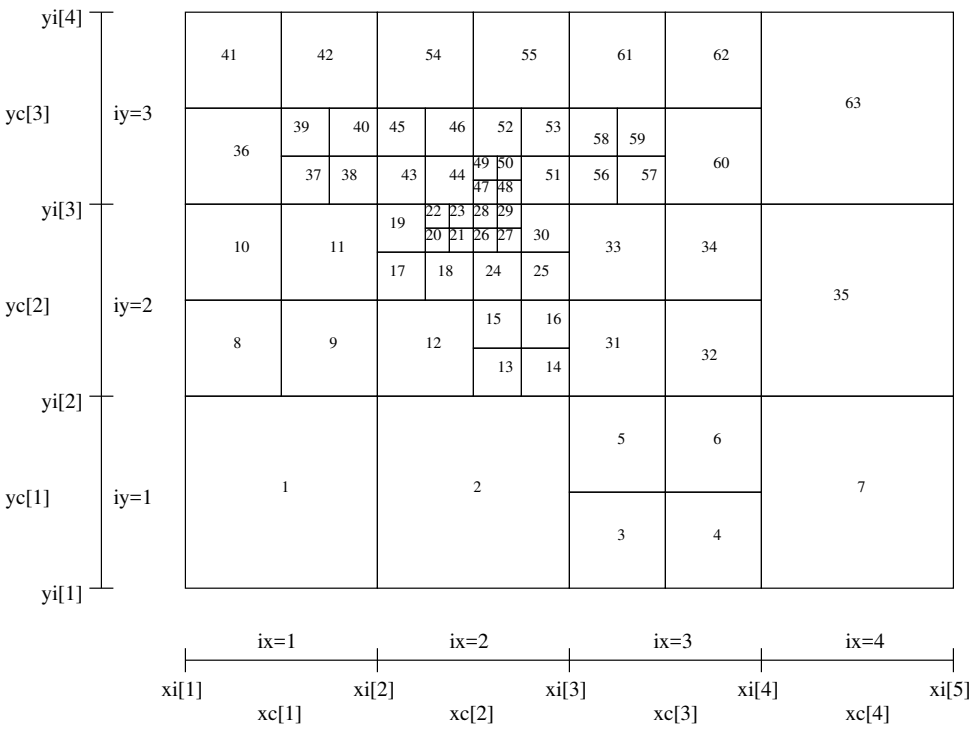

Fig. 16.2: Example of a 2-D grid with oct-tree refinement (as Fig. *Example of a 2-D grid with oct-tree refinement. The base grid has nx=4 and ny=3. Three levels of refinement are added to this base grid.*) with the order of the cells shown as numbers in the cells.

(continued from previous page)

```
density[3,3,1,layer=0]                    <=== This a redundant value
density[4,3,1,layer=0]
density[1,4,1,layer=0]
density[2,4,1,layer=0]
density[3,4,1,layer=0]
density[4,4,1,layer=0]
density[1,1,1,layer=1]                    <=== This a redundant value
density[2,1,1,layer=1]                    <=== This a redundant value
density[3,1,1,layer=1]
density[4,1,1,layer=1]
density[1,2,1,layer=1]                    <=== This a redundant value
density[2,2,1,layer=1]                    <=== This a redundant value
density[3,2,1,layer=1]
density[4,2,1,layer=1]
density[1,3,1,layer=1]
density[2,3,1,layer=1]
density[3,3,1,layer=1]
density[4,3,1,layer=1]
density[1,4,1,layer=1]
density[2,4,1,layer=1]
density[3,4,1,layer=1]
density[4,4,1,layer=1]
density[1,1,1,layer=2]
density[2,1,1,layer=2]
density[3,1,1,layer=2]
density[4,1,1,layer=2]
```

(continues on next page)

(continued from previous page)

```
density[1,2,1,layer=2]
density[2,2,1,layer=2]
density[3,2,1,layer=2]
density[4,2,1,layer=2]
density[1,3,1,layer=2]
density[2,3,1,layer=2]
density[3,3,1,layer=2]
density[4,3,1,layer=2]
density[1,4,1,layer=2]
density[2,4,1,layer=2]
density[3,4,1,layer=2]
density[4,4,1,layer=2]
```

An unformatted version is also available, in the standard way (see above).

It is clear that 48 is now the total number of values to be read, which is 16 values for layer 0 (= base grid), 16 values for layer 1 and 16 values for layer 2. It is also clear that some values are redundant (they can have any value, does not matter). But it at least assures that each data block is a simple regular data block, which is easier to handle. Note that these values (marked as redundant in the above example) *must* be present in the file, but they can have any value you like (typically 0).

Note that if you have multiple species of dust then we will still have 48 as the value of `nrcells`. The number of values to be read, if you have 2 dust species, is then simply 2*`nrcells` = 2*48 = 96.

16.4 INPUT/OUTPUT: dust_temperature.dat

The dust temperature file is an intermediate result of RADMC-3D and follows from the thermal Monte Carlo simulation. The name of this file is `dust_temperature.dat` (see Chapter *Binary I/O files* for the binary version of this file, which is more compact). It can be used by the user for other purposes (e.g. determination of chemical reaction rates), but also by RADMC-3D itself when making ray-traced images and/or spectra. The user can also produce his/her own `dust_temperature.dat` file (without invoking the Monte Carlo computation) if she/he has her/his own way of computing the dust temperature.

The structure of this file is identical to that of `dust_density.inp` (Section *INPUT (required for dust transfer): dust_density.inp*), but with density replaced by temperature. We refer to section *INPUT (required for dust transfer): dust_density.inp* for the details.

16.5 INPUT (mostly required): stars.inp

This is the file that specifies the number of stars, their positions, radii, and spectra. Stars are sources of netto energy. For the dust continuum Monte Carlo simulation these are a source of photon packages. This file exists only in formatted (ascii) style. Its structure is:

```
iformat                          <=== Put this to 2 !
nstars        nlam
rstar[1]      mstar[1]      xstar[1]      ystar[1]      zstar[1]
  .             .             .             .             .
  .             .             .             .             .
rstar[nstars  mstar[nstars] xstar[nstars] ystar[nstars] zstar[nstars]
lambda[1]
  .
  .
```

(continues on next page)

(continued from previous page)

```
lambda[nlam]
flux[1,star=1]
  .
  .
flux[nlam,star=1]
flux[1,star=2]
  .
  .
flux[nlam,star=2]
  .
  .
  .
  .
flux[nlam,star=nstar]
```

which is valid only if `iformat==2`. The meaning of the variables:

- `iformat`: The format number, at present better keep it at 2. If you put it to 1, the list of wavelengths (see below) will instead be a list of frequencies in Herz.
- `nstars`: The number of stars you wish to specify.
- `nlam`: The number of frequency points for the stellar spectra. At present this must be identical to the number of walvelength points in the file `wavelength_micron.inp` (see Section *INPUT (required): wavelength_micron.inp*).
- `rstar[i]`: The radius of star i in centimeters.
- `mstar[i]`: The mass of star i in grams. This is not important for the current version of RADMC-3D, but may be in the future.
- `xstar[i]`: The `x`-coordinate of star i in centimeters.
- `ystar[i]`: The `y`-coordinate of star i in centimeters.
- `zstar[i]`: The `z`-coordinate of star i in centimeters.
- `lambda[i]`: Wavelength point i (where $i \in [1, \mathrm{nlam}]$) in microns. This must be identical (!) to the equivalent point in the file `wavelength_micron.inp` (see Section *INPUT (required): wavelength_micron.inp*). If not, an error occurs.
- `flux[i,star=n]`: The flux F_ν at wavelength point i for star n in units of $\mathrm{erg\,cm^{-2}, s^{-1}, Hz^{-1}}$ as seen from a distance of 1 parsec = 3.08572×10^{18} cm (for normalization).

Sometimes it may be sufficient to assume simple blackbody spectra for these stars. If for any of the stars the first (!) flux number (`flux[1,star=n]`) is negative, then the absolute value of this number is taken to be the blackbody temperature of the star, and no further values for this star are read. Example:

```
2
1            100
6.96e10      1.99e33        0.      0.    0.
0.1
  .
  .
1000.
-5780.
```

will make one star, at the center of the coordinate system, with one solar radius, one solar mass, on a wavelength grid ranging from 0.1 micron to 1000 micron (100 wavelength points) and with a blackbody spectrum with a temperature equal to the effective temperature of the sun.

Note: The position of a star can be both inside and outside of the computational domain.

16.6 INPUT (optional): stellarsrc_templates.inp

This is the file that specifies the template spectra for the smooth stellar source distributions. See Section *Distributions of zillions of stars*. The file exists only in formatted (ascii) style. Its structure is:

```
iformat                          <=== Put this to 2 !
ntempl
nlam
lambda[1]
  .
  .
lambda[nlam]
flux[1,templ=1]
  .
  .
flux[nlam,templ=1]
flux[1,templ=2]
  .
  .
flux[nlam,templ=2]
  .
  .
  .
  .
flux[nlam,templ=ntempl]
```

which is valid only if `iformat==2`. The meaning of the variables:

- `iformat`: The format number, at present better keep it at 2. If you put it to 1, the list of wavelengths (see below) will instead be a list of frequencies in Herz.
- `ntempl`: The number of stellar templates you wish to specify.
- `nlam`: The number of frequency points for the stellar template spectra. At present this must be identical to the number of walvelength points in the file `wavelength_micron.inp` (see Section *INPUT (required): wavelength_micron.inp*).
- `lambda[i]`: Wavelength point i (where $i \in [1, \mathrm{nlam}]$) in microns. This must be identical (!) to the equivalent point in the file `wavelength_micron.inp` (see Section *INPUT (required): wavelength_micron.inp*). If not, an error occurs.
- `flux[i,templ=n]`: The 'flux' at wavelength i for stellar template n. The units are somewhat tricky. It is given in units of erg / sec / Hz / gram-of-star. So multiply this by the density of stars in units of gram-of-star / cm^3, and divide by 4*pi to get the stellar source function in units of erg / src / Hz / cm^3 / steradian.

Sometimes it may be sufficient to assume simple blackbody spectra for these stellar sources. If for any of the stellar sources the first (!) flux number (`flux[1,templ=n]`) is negative, then the absolute value of this number is taken to be the blackbody temperature of the stellar source, and the following two numbers are interpreted as the stellar radius and stellar mass respectively. From that, RADMC-3D will then internally compute the stellar template. Example:

```
2
1
100
0.1
  .
```

(continues on next page)

(continued from previous page)

```
  .
1000.
-5780.
6.9600000e+10
1.9889200e+33
```

will tell RADMC-3D that there is just one stellar template, assumed to have a blackbody spectrum with solar effective temperature. Each star of this template has one solar radius, one solar mass.

16.7 INPUT (optional): stellarsrc_density.inp

This is the file that contains the smooth stellar source densities. If you have the file `stellarsrc_templates.inp` specified (see Section *INPUT (optional): stellarsrc_templates.inp*) then you *must* also specify `stellarsrc_density.inp` (or its binary form, see Chapter *Binary I/O files*). The format of this file is very similar to `dust_density.inp` (Section *INPUT (required for dust transfer): dust_density.inp*), but instead different dust species, we have different templates. For the rest we refer to Section *INPUT (required for dust transfer): dust_density.inp* for the format. Just replace `ispec` (the dust species) with `itempl` (the template).

16.8 INPUT (optional): external_source.inp

This is the file that specifies the spectrum and intensity of the external radiation field, i.e. the 'interstellar radiation field' (see Section *The interstellar radiation field: external source of energy*). Its structure is:

```
iformat                          <=== Put this to 2 !
nlam
lambda[1]
  .
  .
lambda[nlam]
Intensity[1]
  .
  .
Intensity[nlam]
```

which is valid only if `iformat==2`. The meaning of the variables:

- `iformat`: The format number, at present better keep it at 2. If you put it to 1, the list of wavelengths (see below) will instead be a list of frequencies in Herz.
- `nlam`: The number of frequency points for the stellar template spectra. At present this must be identical to the number of walvelength points in the file `wavelength_micron.inp` (see Section *INPUT (required): wavelength_micron.inp*).
- `lambda[i]`: Wavelength point i (where $i \in [1, \mathrm{nlam}]$) in microns. This must be identical (!) to the equivalent point in the file `wavelength_micron.inp` (see Section *INPUT (required): wavelength_micron.inp*). If not, an error occurs.
- `Intensity[i]`: The intensity of the radiation field at wavelength i in units of erg / cm^2 / sec / Hz / steradian.

16.9 INPUT (optional): heatsource.inp

This file, if present (it is an optional file!), gives the internal heat source of the gas-dust mixture in every cell. For formatted style (`heatsource.inp`) the structure of this file is as follows.:

```
iformat                                  <=== Typically 1 at present
nrcells
heatsource[1]
..
heatsource[nrcells]
```

As with most input/output files of RADMC-3D, you can also specify the input data in binary form (`heatsource.binp`), see Chapter *Binary I/O files*.

The physical unit of `heatsource` is $\mathrm{erg\,cm^{-3}\,s^{-1}}$. The total luminosity of the heat source would then be the sum over all cells of `heatsource` times the cell volume.

16.10 INPUT (required): wavelength_micron.inp

This is the file that sets the discrete wavelength points for the continuum radiative transfer calculations. Note that this is not the same as the wavelength grid used for e.g. line radiative transfer. See Section *INPUT (optional): camera_wavelength_micron.inp* and/or Chapter *Line radiative transfer* for that. This file is only in formatted (ascii) style. It's structure is:

```
nlam
lambda[1]
  .
  .
lambda[nlam]
```

where

- `nlam`: The number of frequency points for the stellar spectra.
- `lambda[i]`: Wavelength point i (where $i \in [1, \mathrm{nlam}]$) in microns.

The list of wavelengths can be in increasing order or decreasing order, but must be monotonically increasing/decreasing.

IMPORTANT: It is important to keep in mind that the wavelength coverage must include the wavelengths at which the stellar spectra have most of their energy, and at which the dust cools predominantly. This in practice means that this should go all the way from 0.1 μm to 1000 μm, typically logarithmically spaced (i.e. equally spaced in $\log(\lambda)$). A smaller coverage will cause serious problems in the Monte Carlo run and dust temperatures may then be severely miscalculated. Note that the 0.1 μm is OK for stellar temperatures below 10000 K. For higher temperatures a shorter wavelength lower limit must be used.

16.11 INPUT (optional): camera_wavelength_micron.inp

The wavelength points in the `wavelength_micron.inp` file are the global continuum wavelength points. On this grid the continuum transfer is done. However, there may be various reasons why the user may want to generate spectra on a different (usually more finely spaced) wavelength grid, or make an image at a wavelength that is not available in the global continuum wavelength grid. Rather than redoing the entire model with a different `wavelength_micron.inp`, which may involve a lot of reorganization and recomputation, the user can specify a file called `camera_wavelength_micron.inp`. If this file exists, it will be read into RADMC-3D, and the user can now ask RADMC-3D to make images in those wavelength or make a spectrum in those wavelengths.

If the user wants to make images or spectra of a model that involves gas lines (such as atomic lines or molecular rotational and/or ro-vibrational lines), the use of a `camera_wavelength_micron.inp` file allows the user to do the line+dust transfer (gas lines plus the continuum) on this specific wavelength grid. For line transfer there are also other ways by which the user can specify the wavelength grid (see Chapter *Line radiative transfer*), and it is left to the user to choose which method to use.

The structure of the `camera_wavelength_micron.inp` file is identical to that of `wavelength_micron.inp` (see Section *INPUT (required): wavelength_micron.inp*).

Note that there are also various other ways by which the user can let RADMC-3D choose wavelength points, many of which may be even simpler and more preferable than the method described here. See Section *Specifying custom-made sets of wavelength points for the camera*.

16.12 INPUT (required for dust transfer): dustopac.inp and dustkappa_*.inp or dustkapscatmat_*.inp or dust_optnk_*.inp

These files specify the dust opacities to be used. More than one can be specified, meaning that there will be more than one co-existing dust species. Each of these species will have its own dust density specified (see Section *INPUT (required for dust transfer): dust_density.inp*). The opacity of each species is specified in a separate file for each species. The `dustopac.inp` file tells which file to read for each of these species.

16.12.1 The dustopac.inp file

The file `dustopac.inp` has the following structure, where an example of 2 separate dust species is used:

```
iformat                          <=== Put this to 2
nspec
-----------------------------
inputstyle[1]
iquantum[1]                      <=== Put to 0 in this example
<name of dust species 1>
-----------------------------
inputstyle[2]
iquantum[2]                      <=== Put to 0 in this example
<name of dust species 2>
```

where:

- `iformat`: Currently the format number is 2, and in this manual we always assume it is 2.
- `nspec`: The number of dust species that will be loaded.
- `inputstyle[i]`: This number tells in which form the dust opacity of dust species i is to be read:
  - *1* Use the `dustkappa_*.inp` input file style (see Section *The dustkappa_*.inp files*).

  - *10* Use the `dustkapscatmat_*.inp` input file style (see Section *The dustkapscatmat_*.inp files*).

- `iquantum[i]`: For normal thermal grains this is 0. If, however, this grain species is supposed to be treated as a quantum-heated grain, then non-zero values are to be specified. *NOTE: At the moment the quantum heating is not yet implemented. Will be done in the future, if users request it. Until then, please set this to 0!*

- `<name of dust species i>`: This is the name of the dust species (without blank spaces). This name is then glued to the base name of the opacity file (see above). For instance, if the name is `enstatite`, and `inputstyle==1`, then the file to be read is `dustkappa_enstatite.inp`.

16.12.2 The dustkappa_*.inp files

If you wish to use dust opacities that include the mass-weighted absorption opacity $\kappa_{\rm abs}$, the (optionally) mass-weighted scattering opacity $\kappa_{\rm scat}$, and (optionally) the anisotropy factor g for scattering, you can do this with a file `dustkappa_*.inp` (set input style to 1 in `dustopac.inp`, see Section *The dustopac.inp file*). With this kind of opacity input file, scattering is included either isotropically or using the Henyey-Greenstein function. Using an opacity file of this kind does *not* allow for full realistic scattering phase functions nor for polarization. For that, you need `dustkapscatmat_*.inp` files (see Section *The dustkapscatmat_*.inp files*). Please refer to Section *More about scattering of photons off dust grains* for more information about how RADMC-3D treats scattering.

If for dust species `<name>` the `inputstyle` in the `dustopac.inp` file is set to 1, then the file `dustkappa_<name>.inp` is sought and read. The structure of this file is:

```
# Any amount of arbitrary
# comment lines that tell which opacity this is.
# Each comment line must start with an # or ; or ! character
iformat                     <== This example is for iformat==3
nlam
lambda[1]         kappa_abs[1]        kappa_scat[1]       g[1]
   .                   .                   .                .
   .                   .                   .                .
lambda[nlam]     kappa_abs[nlam]    kappa_scat[nlam]     g[nlam]
```

The meaning of these entries is:

- `iformat`: If `iformat==1`, then only the lambda and kappa_abs colums are present. In that case the scattering opacity is assumed to be 0, i.e. a zero albedo is assumed. If `iformat==2` also kappa_scat is read (third column). If `iformat==3` (which is what is used in the above example) then *also* the anisotropy factor g is included.

- `nlam`: The number of wavelength points in this file. This can be any number, and does not have to be the same as those of the `wavelength_micron.inp`. It is typically advisable to have a rather large number of wavelength points.

- `lambda[i]`: The wavelength point i in micron. This does not have to be (and indeed typically is not) the same as the values in the `wavelength_micron.inp` file. Also for each opacity this list of wavelengths can be different (and can be a different quantity of points).

- `kappa_abs[i]`: The absorption opacity $\kappa_{\rm abs}$ in units of $\rm cm^2$ per gram of dust.

- `kappa_scat[i]`: The scattering opacity $\kappa_{\rm abs}$ in units of $\rm cm^2$ per gram of dust. Note that this column should only be included if `iformat==2` or higher.

- `g[ilam]`: The mean scattering angle $\langle\cos(\theta)\rangle$, often called g. This will be used by RADMC-3D in the Henyey-Greenstein scattering phase function. Note that this column should only be included if `iformat==3` or higher.

Once this file is read, the opacities will be mapped onto the global wavelength grid of the `wavelength_micron.inp` file. Since this mapping always involve uncertainties and errors, a file `dustkappa_*.inp_used` is created which lists the opacity how it is remapped onto the global wavelength grid. This is only for you as the user, so that you

can verify what RADMC-3D has internally done. Note that if the upper or lower edges of the wavelength domain of the `dustkappa_*.inp` file is within the domain of the `wavelength_micron.inp` grid, some extrapolation will have to be done. At short wavelength this will simply be constant extrapolation while at long wavelength a powerlaw extrapolation is done. Have a look at the `dustkappa_*.inp_used` file to see how RADMC-3D has done this in your particular case.

16.12.3 The dustkapscatmat_*.inp files

If you wish to treat scattering in a more realistic way than just the Henyey-Greenstein non-polarized way, then you must provide RADMC-3D with more information than is present in the `dustkappa_xxx.inp` files: RADMC-3D will need the full scattering Müller matrix for all angles of scattering (see e.g. the books by Mishchenko, or by Bohren & Huffman or by van de Hulst). For *randomly oriented particles* only 6 of these matrix elements can be non-zero: Z_{11}, $Z_{12} = Z_{21}$, Z_{22}, Z_{33}, $Z_{34} = -Z_{43}$, Z_{44}, where 1,2,3,4 represent the I,Q,U,V Stokes parameters. Moreover, for randomly oriented particles there is only 1 scattering angle involved: the angle between the incoming and outgoing radiation of the scattering event. This means that we must give RADMC-3D, (for every wavelength and for a discrete set of scattering angles) a list of values of these 6 matrix elements. These can be provided in a file `dustkapscatmat_xxx.inp` (set input style to 10 in `dustopac.inp`, see Section *The dustopac.inp file*) which comes * instead of* the `dustkappa_xxx.inp` file. Please refer to Section *More about scattering of photons off dust grains* for more information about how RADMC-3D treats scattering.

If for dust species `<name>` the `inputstyle` in the `dustopac.inp` file is set to 10, then the file `dustkapscatmat_<name>.inp` is sought and read. The structure of this file is:

```
# Any amount of arbitrary
# comment lines that tell which opacity this is.
# Each comment line must start with an # or ; or ! character
iformat             <== Format number must be 1
nlam
nang                <== A reasonable value is 181 (e.g. angle = 0.0,1.0,...,180.0)

lambda[1]         kappa_abs[1]       kappa_scat[1]     g[1]
   .                   .                  .              .
   .                   .                  .              .
lambda[nlam]    kappa_abs[nlam]   kappa_scat[nlam]   g[nlam]

angle_in_degrees[1]
   .
   .
angle_in_degrees[nang]

Z_11  Z_12  Z_22  Z_33  Z_34  Z_44   [all for ilam=1 and iang=1]
Z_11  Z_12  Z_22  Z_33  Z_34  Z_44   [all for ilam=1 and iang=2]
Z_11  Z_12  Z_22  Z_33  Z_34  Z_44   [all for ilam=1 and iang=3]
 .     .     .     .     .     .
 .     .     .     .     .     .
Z_11  Z_12  Z_22  Z_33  Z_34  Z_44   [all for ilam=1 and iang=nang]

Z_11  Z_12  Z_22  Z_33  Z_34  Z_44   [all for ilam=2 and iang=1]
 .     .     .     .     .     .
 .     .     .     .     .     .
Z_11  Z_12  Z_22  Z_33  Z_34  Z_44   [all for ilam=2 and iang=nang]

....
....
....
```

(continues on next page)

(continued from previous page)

```
Z_11  Z_12  Z_22  Z_33  Z_34  Z_44   [all for ilam=nlam and iang=1]
 .     .     .     .     .     .
 .     .     .     .     .     .
Z_11  Z_12  Z_22  Z_33  Z_34  Z_44   [all for ilam=nlam and iang=nang]
```

The meaning of these entries is:

- `iformat`: For now this value should remain 1.
- `nlam`: The number of wavelength points in this file. This can be any number, and does not have to be the same as those of the `wavelength_micron.inp`. It is typically advisable to have a rather large number of wavelength points.
- `nang`: The number of scattering angle sampling points. This should be large enough that a proper integration over scattering angle can be carried out reliably. A reasonable value is 181, so that (for a regular grid in scattering angle θ) you have as scattering angles $\theta = 0, 1, 2, \cdots, 180$ (in degrees). But if you have extremely forward- or backward peaked scattering, then maybe even 181 is not enough.
- `lambda[ilam]`: The wavelength point `ilam` in micron. This does not have to be (and indeed typically is not) the same as the values in the `wavelength_micron.inp` file. Also for each opacity this list of wavelengths can be different (and can be a different quantity of points).
- `angle_in_degrees[iang]`: The scattering angle sampling point `iang` in degrees (0 degrees is perfect forward scattering, 180 degrees is perfect backscattering). There should be `nang` such points, where `angle_in_degrees[1]` must be 0 and `angle_in_degrees[nang]` must be 180. In between the angle grid can be anything, as long as it is monotonic.
- `kappa_abs[ilam]`: The absorption opacity $\kappa_{\rm abs}$ in units of $\rm cm^2$ per gram of dust.
- `kappa_scat[ilam]`: The scattering opacity $\kappa_{\rm scat}$ in units of $\rm cm^2$ per gram of dust. RADMC-3D can (and will) in fact calculate $\kappa_{\rm scat}$ from the scattering matrix elements. It will then check (for every wavelength) if that is the same as the value listed here. If the difference is small, it will simply adjust the `kappa_scat[ilam]` value internally to get a perfect match. If it is larger than 1E-4 then it will, in addition to adjusting, make a warning. if it is larger than 1E-1, it will abort. Note that the fewer angles are used, the worse the match will be because the integration over angle will be worse.
- `g[ilam]`: The mean scattering angle $\langle\cos(\theta)\rangle$, often called g. RADMC-3D can (and will) in fact calculate g from the scattering matrix elements. Like with `kappa_scat[ilam]` it will adjust if the difference is not too large and it will complain or abort if the difference is larger than some limit.
- `Z_{xx}` These are the scattering matrix elements in units of $\rm cm^2\,g^{-1}\,ster^{-1}$ (i.e. they are angular differential cross sections). See Section *More about scattering of photons off dust grains* for more details.

NOTE: This only allows the treatment of *randomly oriented particles*. RADMC-3D does not, for now, have the capability of treating scattering off fixed-oriented particles. In fact, for oriented particles it would be impractical to use dust opacity files of this kind, since we would then have at least *three* scattering angles, which would require huge table. In that case it would be presumably necessary to compute the matrix elements on-the-fly.

Note that the scattering-angle grid of the `dustkapscatmat_xxx.inp` files can be chosen non-regular, e.g. to put a more finely spaced grid close to $\theta = 0$ (forward scattering) and $\theta = \pi$ (backscattering). This can be useful for large grains and/or short wavelengths, where forward scattering can be extremely strongly peaked. Since multiple dust species can each have a different scattering θ-grid, it requires you to give an additional file to `RADMC-3D` that represents the scattering θ-grid for all grains. This file is called `scattering_angular_grid.inp`. The format is as follows:

```
1            <=== Format number, must be 1
181          <=== Nr of theta grid points
0.0          <=== First angle (in degrees). Must be 0
```

(continues on next page)

(continued from previous page)

```
1.0
2.0
...
...
...
179.0
180.0        <=== Last angle (in degrees). Must be 180
```

NOTE: This file is not compulsory. If it is not given, then `RADMC-3D` will make its own internal scattering angle grid.

16.13 OUTPUT: spectrum.out

Any spectrum that is made with RADMC-3D will be either called `spectrum.out` or `spectrum_<somename>.out` and will have the following structure:

```
iformat                        <=== For now this is 1
nlam

lambda[1]        flux[1]
   .               .
   .               .
lambda[nlam]    flux[nlam]
```

where:

- `iformat`: This format number is currently set to 1.
- `nlam`: The number of wavelength points in this spectrum. This does not necessarily have to be the same as those in the `wavelength_micron.inp` file. It can be any number.
- `lambda[i]`: Wavelength in micron. This does not necessarily have to be the same as those in the `wavelength_micron.inp` file. The wavelength grid of a spectrum file can be completely independent of all other wavelength grids. For standard SED computations for the continuum typically these will be indeed the same as those in the `wavelength_micron.inp` file. But for line transfer or for spectra based on the `camera_wavelength_micron.inp` they are not.
- `flux[i]`: Flux in units of $\mathrm{erg\,s^{-1}\,cm^{-2}\,Hz^{-1}}$ at this wavelength as measured at a standard distance of 1 parsec (just as a way of normalization).

NOTE: Maybe in the future a new iformat version will be possible where more telescope information is given in the spectrum file.

16.14 OUTPUT: image.out or image_****.out

Any images that are produced by RADMC-3D will be written in a file called `image.out`. The file has the following structure (for the case without Stokes parameters):

```
iformat                        <=== For now this is 1 (or 2 for local observer mode)
im_nx        im_ny
nlam
pixsize_x    pixsize_y
lambda[1]  ......... lambda[nlam+1]

```

(continues on next page)

(continued from previous page)

```
image[ix=1,iy=1,img=1]
image[ix=2,iy=1,img=1]
  .
  .
image[ix=im_nx,iy=1,img=1]
image[ix=1,iy=2,img=1]
  .
  .
image[ix=im_nx,iy=2,img=1]
image[ix=1,iy=im_ny,img=1]
  .
  .
  .
image[ix=im_nx,iy=im_ny,img=nlam]

image[ix=1,iy=1,img=1]
  .
  .
  .
  .
image[ix=im_nx,iy=im_ny,img=nlam]
```

In most cases the nr of images (nr of wavelengths) is just 1, meaning only one image is written (i.e. the img=2, ... img=nlam are not there, only the img=1). The meaning of the various entries is:

- `iformat`: This format number is currently set to 1

for images from an observer at infinity (default) and 2 for a local observer. Note: For full-Stokes images it is 3, but then also the data changes a bit, see below.

- `im_nx,im_ny`: The number of pixels in x and in y direction of the image.
- `nlam`: The number of images at different wavelengths that

are in this file. You can make a series of images at different wavelengths in one go, and write them in this file. The wavelength belonging to each of these images is listed below. The `nlam` can be any number from 1 to however large you want. Mostly one typically just makes an images at one wavelength, meaning `nlam`=1.

- `pixsize_x,pixsize_y`: The size of the pixels in cm (for an observer at infinity) or radian (for local observer mode). This means that for the observer-at-infinity mode (default) the size is given in model units (distance within the 3-D model) and the user can, for any distance, convert this into arcseconds: pixel size in arcsec = (pixel size in cm / 1.496E13) / (distance in parsec). The pixel size is the full size from the left of the pixel to the right of the pixel (or from bottom to top).
- `lambda[i]`: Wavelengths in micron belonging to the various images in this file. In case `nlam`=1 there will be here just a single number. Note that this set of wavelengths can be completely independent of all other wavelength grids.
- `image[ix,iy,img]`: Intensity in the image at pixel `ix`, `iy` at wavelength `img` (of the above listed wavelength points) in units of $\mathrm{erg\,s^{-1}\,cm^{-2}\,Hz^{-1}\,ster^{-1}}$. *Important:* The pixels are ordered from left to right (i.e. increasing x) in the inner loop, and from bottom to top (i.e. increasing y) in the outer loop.

You can also make images with full Stokes parameters. For this you must have dust opacities that include the full scattering matrix, *and* you must add the keyword `stokes` to the `radmc3dimage` command on the command-line. In that case the `image.out` file has the following form:

```
iformat                     <=== For Stokes this is 3
im_nx        im_ny
nlam
```

(continues on next page)

(continued from previous page)

```
pixsize_x    pixsize_y
lambda[1]  ......... lambda[nlam+1]

image_I[ix=1,iy=1,img=1] image_Q[ix=1,iy=1,img=1] image_U[ix=1,iy=1,img=1] image_
↪V[ix=1,iy=1,img=1]
  .
  .
image_I[ix=im_nx,iy=1,img=1] (and so forth for Q U and V)
image_I[ix=1,iy=2,img=1] (and so forth for Q U and V)
  .
  .
image_I[ix=im_nx,iy=2,img=1] (and so forth for Q U and V)
image_I[ix=1,iy=im_ny,img=1] (and so forth for Q U and V)
  .
  .
  .
image_I[ix=im_nx,iy=im_ny,img=nlam] (and so forth for Q U and V)

image_I[ix=1,iy=1,img=1] (and so forth for Q U and V)
  .
  .
  .
  .
image_I[ix=im_nx,iy=im_ny,img=nlam] (and so forth for Q U and V)
```

That is: instead of 1 number per line we now have 4 numbers per line, which are the four Stokes parameters. Note that `iformat`=3 to indicate that we have now all four Stokes parameters in the image.

16.15 INPUT: (minor input files)

There is a number of lesser important input files, or input files that are only read under certain circumstances (for instance when certain command line options are given). Here they are described.

16.15.1 The `color_inus.inp` file (required with comm-line option 'loadcolor')

The file `color_inus.inp` will only be read by RADMC-3D if on the command line the option `loadcolor` or `color` is specified, and if the main action is `image`.

```
iformat                      <=== For now this is 1
nlam
ilam[1]
  .
  .
ilam[nlam]
```

- `iformat`: This format number is currently set to 1.
- `nlam`: Number of wavelength indices specified here.
- `ilam[i]`: The wavelength index for image i (the wavelength index refers to the list of wavelengths in the `wavelength_micron.inp` file.

16.15.2 INPUT: `aperture_info.inp`

If you wish to make spectra with wavelength-dependent collecting area, i.e. aperture (see Section *Can one specify more realistic 'beams'?*), then you must prepare the file `aperture_info.inp`. Here is its structure:

```
iformat                        <=== For now this is 1
nlam
lambda[1]       rcol_as[1]
  .               .
  .               .
lambda[nlam]    rcol_as[nlam]
```

with

- `iformat`: This format number is currently set to 1.
- `nlam`: Number of wavelength indices specified here. This does *not* have to be the same as the number of wavelength of a spectrum or the number of wavelengths specified in the file `wavelength_micron.inp`. It can be any number.
- `lambda[i]`: Wavelength sampling point, in microns. You can use a course grid, as long as the range of wavelengths is large enough to encompass all wavelengths you may wish to include in spectra.
- `rcol_as[i]`: The radius of the circular image mask used for the aperture model, in units of arcsec.

16.16 For developers: some details on the internal workings

There are several input files that can be quite large. Reading these files into RADMC-3D memory can take time, so it is important not to read files that are not required for the execution of the particular command at hand. For instance, if a model exists in which both dust and molecular lines are included, but RADMC-3D is called to merely make a continuum SED (which in RADMC-3D never includes the lines), then it would be a waste of time to let RADMC-3D read all the gas velocity and temperature data and level population data into memory if they are not used.

To avoid unnecessary reading of large files the reading of these files is usually organized in a 'read when required' way. Any subroutine in the code that relies on e.g. line data to be present in memory can simply call the routine `read_lines_all(action)` with argument `action` being 1, i.e.:

```
call read_lines_all(1)
```

This routine will check if the data are present: if no, it will read them, if yes, it will return without further action. This means that you can call `read_lines_all(1)` as often as you want: the line data will be read once, and only once. If you look through the code you will therefore find that many `read_***` routines are called abundantly, whenever the program wants to make sure that certain data is present. The advantage is then that the programmer does not have to have a grand strategy for when which data must be read in memory: he/she simply inserts a call to the read routines for all the data she/he needs at that particular point in the program, (always with action=1), and it will organize itself. If certain data is nowhere needed, they will not be read.

All these `read_***` routines with argument `action` can also be called with `action=2`. This will force the routine to (re-)read these data. But this is rarely needed.

CHAPTER

SEVENTEEN

BINARY I/O FILES

17.1 Overview

By default all input and output files of RADMC-3D are in ASCII (i.e.text) form. This makes it easier to verify if the files are ok. Also, it is easier to produce files with the right format and read the output of RADMC-3D. The disadvantage is that ASCII files are substantially larger than strictly required to store their information content. For large models, i.e.models with many grid points, this may lead to unpractically large files.

RADMC-3D supports a more compact data format: binary data. In this form, a double precision variable occupies just 8 bytes, while a single precision variable occupies just 4 bytes.

Unfortunately, Fortran-90 and Fortran-95 did, for a long time, not support true binary files. Instead they offered 'f77-unformatted' files, which uses 'records', and is harder to read than true binary files. Recently, however, many Fortran-90 and Fortran-95 compilers have introduced a true binary format, which is called 'streaming access'. It is, actually, a Fortran-2003 feature, but has been retroactively implemented into Fortran-90 and Fortran-95. The gfortran and g95 compilers have it. Also the ifort compiler has it. Presumably others as well.

RADMC-3D offers a binary I/O capability. A file containing three double precision variables will have a length of exactly 24 bytes. Files with this format will have extensions such as `.binp`, `.bdat` or `.bout`.

Here is a (presumably incomplete) list of files that have binary versions:

Name	ascii	binary
`dust_density`	`.inp`	`.binp`
`dust_temperature`	`.inp`	`.binp`
`dust_temperature`	`.dat`	`.bdat`
`gas_density`	`.inp`	`.binp`
`gas_temperature`	`.inp`	`.binp`
`electron_numdens`	`.inp`	`.binp`
`ion_numdens`	`.inp`	`.binp`
`levelpop_***`	`.dat`	`.bdat`
`numberdens_***`	`.inp`	`.binp`
`gas_velocity`	`.inp`	`.binp`
`microturbulence`	`.inp`	`.binp`
`stellarsrc_density`	`.inp`	`.binp`
`mean_intensity`	`.out`	`.bout`
`heatsource`	`.inp`	`.binp`

17.2 How to switch to binary (or back to ascii)

Specifying whether RADMC-3D should use ASCII or binary *input* is easy: It will simply look which extension each input file has, and read it accordingly. If you present RADMC-3D file input files with extension `.binp`, it will read these files as binaries.

More tricky is how to tell RADMC-3D to use binary files on *output*. By default, RADMC-3D will always write ASCII style (`.out` and `.dat`). However, if you add the following line to the `radmc3d.inp` file:

```
rto_style = 3
```

it will instead use binary output (`.bout` and `.bdat`). And, for completeness (though it is the default anyway), if you set `rto_style=1` RADMC-3D will write output in ASCII form. Note that `rto_style = 2` is the old Fortran unformatted data format, which is deprecated.

For the binary form of output you can also tell RADMC-3D to use single-precision for the main data, to produce smaller output files. This is done by adding the following line to the `radmc3d.inp` file:

```
rto_single = 1
```

By default RADMC-3D will always output double precision in the binary format.

Note: Images are still outputted in ascii even if you have `rto_style=3`. This is because images are rarely files of huge size, and ascii files are easier to analyze and check. However, sometimes images can be still quite big (e.g.if you make multi-frequency images). Then it might still be useful to output binary. If you want to also have the images in binary format, you must set

```
writeimage_unformatted = 1
```

in the `radmc3d.inp` file, *or* you add a keyword *imageunform*.

17.3 Binary I/O file format of RADMC-3D

The general format of the files listed in Section *Overview* is similar to the ASCII versions, just binary this time. There is *one* additional number in the binary version: Right after the format number comes an integer that gives the precision of the main data. This number is either 4, meaning that the main data consists of 4-byte floating point numbers (i.e.single precision), or 8, meaning that the main data consists of 8-byte floating point numbers (i.e.double precision). Other than that additional number, the order of the data is the same.

The following rules apply:

- With the exception of the `amr_grid.binp` file (see below), all integers are 8-byte integers.
- Floating point numbers for the main data (i.e.the data that represents the space-dependent variables) are either 4-byte (single) or 8-byte (double) precision numbers. Which of the two is specified in the second integer of the file (the integer right after the format number, see above).
- All other floating point numbers are double precision (i.e.8-byte floats).
- For AMR-grids the `amr_grid.binp` file contains a huge list of 0 or 1 numbers (see Section *Oct-tree-style AMR grid*). Since it is silly to use 8-byte integers for numbers that are either 0 or 1, the numbers in this list are 1-byte integers (bytes).

Example: According to Section *INPUT (required for dust transfer): dust_density.inp* the ASCII file `dust_density.inp` file has the following format:

```
iformat                                 <=== Typically 1 at present
nrcells
nrspec
density[1,ispec=1]
..
density[nrcells,ispec=1]
density[1,ispec=2]
..
..
..
density[nrcells,ispec=nrspec]
```

According to the above listed rules the binary file `dust_density.binp` file then has the following format:

```
<int8:iformat=1>
<int8:precis=8>
<int8:nrcells>
<int8:nrspec>
<dbl8:density[1,ispec=1]>
..
<dbl8:density[nrcells,ispec=1]>
<dbl8:density[1,ispec=2]>
..
..
..
<dbl8:density[nrcells,ispec=nrspec]>
```

where the `<int8:precis=8>` means that this is an 8-byte integer that we call 'precis' (the name is irrelevant here), and it has value 8, and `<dbl8:density[1,ispec=1]>` means that this is a double-precision number (8-byte float). In other words: the first 8 bytes of the file contain the format number (which is 1 at present). The second 8 bytes contain the number 8, telling that the main data (i.e.the `density` data) are double precision variables. The third set of 8 bytes gives the number of cells, while the fourth set gives the number of dust species. The data of `density` starts as of the 33rd byte of the file. If you want to compress the file even further, and you are satisfied with single-precision data, then the file would look like:

```
<int8:iformat=1>
<int8:precis=4>
<int8:nrcells>
<int8:nrspec>
<flt4:density[1,ispec=1]>
..
<flt4:density[nrcells,ispec=1]>
<flt4:density[1,ispec=2]>
..
..
..
<flt4:density[nrcells,ispec=nrspec]>
```

Another example: According to Section *Special-purpose feature: Computing the local radiation field* RADMC-3D can compute the mean intensity of radiation at each grid point at a set of pre-defined frequencies, and write this out to an ASCII file called `mean_intensity.out`. The contents of this file are:

```
iformat                                 <=== Typically 2 at present
nrcells
nfreq                                   <=== Nr of frequencies
freq_1 freq_2 ... freq_nfreq            <=== List of frequencies in Hz
meanint[1,icell=1]
```

(continues on next page)

(continued from previous page)

```
meanint[1,icell=2]
...
meanint[1,icell=nrcells]
meanint[2,icell=1]
meanint[2,icell=2]
...
meanint[2,icell=nrcells]
...
...
...
meanint[nfreq,icell=1]
meanint[nfreq,icell=2]
...
meanint[nfreq,icell=nrcells]
```

By setting `rto_style=3` in the `radmc3d.inp` file, however, RADMC-3D will instead produce a binary file called `mean_intensity.bout`, which has the contents:

```
<int8:iformat=2>
<int8:precis=8>
<int8:nrcells>
<int8:nfreq>
<dbl8:freq_1>
<dbl8:freq_2>
...
<dbl8:freq_nfreq>
<dbl8:meanint[1,icell=1]>
<dbl8:meanint[1,icell=2]>
...
<dbl8:meanint[1,icell=nrcells]>
<dbl8:meanint[2,icell=1]>
<dbl8:meanint[2,icell=2]>
...
<dbl8:meanint[2,icell=nrcells]>
...
...
...
<dbl8:meanint[nfreq,icell=1]>
<dbl8:meanint[nfreq,icell=2]>
...
<dbl8:meanint[nfreq,icell=nrcells]>
```

If you also set `rto_single=1` in the `radmc3d.inp` file, then you will get:

```
<int8:iformat=2>
<int8:precis=4>
<int8:nrcells>
<int8:nfreq>
<dbl8:freq_1>
<dbl8:freq_2>
...
<dbl8:freq_nfreq>
<flt4:meanint[1,icell=1]>
<flt4:meanint[1,icell=2]>
...
<flt4:meanint[1,icell=nrcells]>
```

(continues on next page)

(continued from previous page)

```
<flt4:meanint[2,icell=1]>
<flt4:meanint[2,icell=2]>
...
<flt4:meanint[2,icell=nrcells]>
...
...
...
<flt4:meanint[nfreq,icell=1]>
<flt4:meanint[nfreq,icell=2]>
...
<flt4:meanint[nfreq,icell=nrcells]>
```

Note that only the mean intensity data (the main data) are single precision floats.

CHAPTER

EIGHTEEN

COMMAND-LINE OPTIONS

This chapter deals with all the possible command-line options one can give when calling the `radmc3d` code.

18.1 Main commands

In addition to the radmc3d.inp file, which contains many 'steering' parameters, one can (and even must) give RADMC-3D also command-line options. The most important (and compulsory) options are the 'command' what RADMC-3D should do. At the moment you can choose from:

- `mctherm`: Runs RADMC-3D for computing the dust temperatures using the Monte Carlo method.
- `spectrum`: Runs RADMC-3D for making a spectrum based on certain settings. This option requires further command-line specifications. See chapter *Making images and spectra*.
- `sed`: Runs RADMC-3D for making a SED based on certain settings. This option requires further command-line specifications. Note that a SED is like a spectrum, but for continuum processes only (no lines). See chapter *Making images and spectra* for more details.
- `image`: Runs RADMC-3D for making an image. This option requires further command-line specifications. See chapter *Making images and spectra*.
- `movie`: Like `image`, but now for a series of different vantage points. Useful for making movies in one go, without having to call RADMC-3D time and again. *NOTE: This command is still under development*. See chapter *Making images and spectra*.
- `mcmono`: (Only expect use). Runs RADMC-3D for computing the local radiation field at each location in the model. This is only useful for when you wish to couple RADMC-3D to models of chemistry or so, which need the local radiation field. See Section *Special-purpose feature: Computing the local radiation field*.

Example:

```
radmc3d mctherm
```

runs the RADMC-3D code for computing the dust temperatures everywhere using the Monte Carlo method.

There are also some additional commands that may be useful for diagnostics:

- `subbox_****`: where **** is one of the following: `dust_density`, `dust_temperature`. But other quantities will follow in later versions. See Section *Making a regularly-spaced datacube ('subbox') of AMR-based models*.
- `linelist`: Write a list of all the lines included in this model.

18.2 Additional arguments: general

Here is a list of command line options, on top of the above listed main commands (Note: We'll try to be complete, but as the code develops we may forget to list new options here):

- `setthreads` [for MC] The next number sets the number of OpenMP parallel threads to be used.
- `npix`: [for images] The next number specifies the number of pixels in both x and y direction, assuming a square image.
- `npixx`: [for images] The next number specifies the number of pixels in x direction only.
- `npixy`: [for images] The next number specifies the number of pixels in y direction only.
- `nrrefine`: [for images and spectra] Specifies a maximum depth of refinement of the pixels (see Section *The issue of flux conservation: recursive sub-pixeling*).
- `fluxcons`: [for images and spectra] Puts nrrefine (see above) to a large value to assue flux conservation (see Section *The issue of flux conservation: recursive sub-pixeling*).
- `norefine`: [for images and spectra] Puts nrrefine (see above) to 0 so that each pixel of the image corresponds only to 1 ray. This is fast but not reliable and therefore not recommended (see Section *The issue of flux conservation: recursive sub-pixeling*).
- `nofluxcons`: [for images and spectra] As `norefine` above.
- `noscat`: This option makes RADMC-3D ignore the dust scattering process (though not the scattering extinction!) in the images, spectra and Monte Carlo simulations. For images and spectra this means that no scattering Monte Carlo run has to be performed before each image ray tracing (see Section *Scattered light in images and spectra: The 'Scattering Monte Carlo' computation*). This can speed up the making of images or spectra enormously. This is even more so if you make images/spectra of gas lines with LTE, LVG or ESCP methods, because if no scattering Monte Carlo needs to be made, ray-tracing can be done multi-frequency for each ray, and the populations can be calculated once in each cell, and used for all frequencies. That can speed up the line rendering enormously – of course at the cost of not including dust scattering. For lines in the infrared and submillimeter, if no large grains are present, this is usually OK, because small grains (smaller than about 1 micron) have very low scattering albedos in the infrared and submillimeter.
- `ilambda` or `inu`: [for images] Specify the index of the wavelength from the `wavelength_micron.inp` file for which a ray-trace image should be made.
- `color`: [for images] Allows you to make multiple images (each at a different wavelength) in one go. This will make RADMC-3D read the file `color_inus.inp` (see Section *INPUT: (minor input files)*) which is a list of indices `i` referring to the `wavelength_micron.inp` file for which the images should be made. See Section *Specifying custom-made sets of wavelength points for the camera* for details.
- `loadcolor`: [for images] Same as `color`.
- `loadlambda`: [for images] Allows you to make multiple images (each at a different wavelength) in one go. This will make RADMC-3D read the file `camera_wavelength_micron.inp` to read the precise wavelength points at which you wish to make the images. In contrast to `loadcolor`, which only allows you to pick from the global set of wavelength used by the Monte Carlo simulation (in the file `wavelength_micron.inp`), with the `camera_wavelength_micron.inp` files you can specify any wavelength you want, and any number of them. See Section *Specifying custom-made sets of wavelength points for the camera* for details.
- `sizeau`: [for images and spectra] The next number specifies the image size in model space in units of AU (=1.496E13 cm). This image size is measured from the image left to right and top to bottom. This gives always square images. This image size in au is observer distance independent. The corresponding image size in arcsec is: image size in arcsec = image size in AU / (distance in parsec).
- `sizepc`: [for images and spectra] Same as `sizeau`, but now in parsec units.

- zoomau: [for images and spectra] The next four numbers set the image window precisely by specifying the xleft, xright, ybottom, ytop of the image in units of AU. The zero point of the image (the direction of the 2-D image point located at (0.0,0.0) in image coordinates) stays the same (i.e. it aims toward the 3-D point in model space given by pointau or pointpc). In this way you can move the image window left or with or up or down without having to change the pointau or pointpc 3-D locations. Also for local perspective images it is different if you move the image window in the image plane, or if you actually change the direction in which you are looking (for images from infinity this is the same). *Note:* If you use this option without the truepix option RADMC-3D will always make square pixels by adapting npixx or npixy such that together with the zoomau image size you get approximately square pixels. Furthermore, if truezoom is not set, RADMC-3D will alleviate the remaining tiny deviation from square pixel shape by slightly (!) adapting the zoomau window to obtain exactly square pixels.
- zoompc: [for images and spectra] Same as zoomau, but now the four numbers are given in units of parsec.
- truepix: [for images and spectra] If with zoomau or zoompc the image window is not square then when specifying npix one gets non-square pixels. Without the truepix option RADMC-3D will adapt the npixx or npixy number, and subsequently modify the zoom window a bit such that the pixels are square. With the truepix option RADMC-3D will not change npixx nor npixy and will allow non-square pixels to form.
- truezoom: [for images and spectra] If set, RADMC-3D will always assure that the exact zoom window (specified with zoomau or zoompc) will be used, i.e. if truepix is *not* set but truezoom is set, RADMC-3D will only (!) adapt npixx or npixy to get *approximately* square pixels.
- pointau: [for images and spectra] The subsequent three numbers specify a 3-D location in model space toward which the camera is pointing for images and spectra. The (0,0) coordinate in the image plane corresponds by definition to a ray going right through this 3-D point.
- pointpc: [for images and spectra] Same as pointau but now in units of parsec.
- incl: [for images and spectra] For the case when the camera is at infinity (i.e. at a large distance so that no local perspective has to be taken into account) this inclination specifies the direction toward which the camera for images and spectra is positioned. Incl = 0 means toward the positive z-axis (in cartesian space), incl=90 means toward a position in the x-y-plane and incl=180 means toward the negative z-axis. The angle is given in degrees.
- phi: [for images and spectra] Like incl, but now the remaining angle, also given in degrees. Examples: incl=90 and phi=0 means that the observer is located at infinity toward the negative y axis; incl=90 and phi=90 means that the observer is located at infinity toward the negative x axis; incl=90 and phi=180 means that the observer is located at infinity toward the positive y axis (looking back in negative y direction). Rotation of the observer around the object around the z-axis goes in clockwise direction. The starting point of this rotation is such that for incl=0 and phi=0 the (x, y) in the image plane correspond to the (x, y) in the 3-D space, with x pointing toward the right and y pointing upward. Examples: if we fix the position of the observer at for instance incl=0 (i.e. we look at the object from the top from the positive z-axis at infinity downward), then increasing phi means rotating the object counter-clockwise in the image plane.
- posang: [for images] This rotates the camera itself around the $(0, 0)$ point in the image plane.
- imageunform: Write out images in binary format
- imageformatted: Write out images in text form (default)
- tracetau: [for images] If this option is set, then instead of ray-tracing a true image, the camera will compute the optical depth at the wavelength given by e.g. inu and puts this into an image output as if it were a true image. Can be useful for analysis of models.
- tracecolumn: [for images] Like tracetau but instead of the optical depth the simple column depth is computed in $\mathrm{g/cm^2}$. *NOTE: for now only the column depth of the dust.*
- tracenormal: [for images: Default] Only if you specified tracetau or tracecolumn before, and you are in child mode, you may sometimes want to reset to normal imaging mode.

- `apert` or `useapert`: [for images/spectra] Use the image-plane aperture information from the file `aperture_info.inp`.
- `noapert`: [for images/spectra] Do *not* use an image-plane aperture.
- `nphot_therm`: [for MC] The nr of photons for the thermal Monte Carlo simulation. But it is better to use the `radmc3d.inp` for this (see Section *INPUT: radmc3d.inp*), because then you can see afterward with which photon statistics the run was done.
- `nphot_scat`: [for MC] The nr of photons for the scattering Monte Carlo simulation done before each image (and thus also in the spectrum). But it is better to use the `radmc3d.inp` for this (see Section *INPUT: radmc3d.inp*), because then you can see afterward with which photon statistics the run was done.
- `nphot_mcmono`: [for MC] The nr of photons for the monochromatic Monte Carlo simulation. But it is better to use the `radmc3d.inp` for this (see Section *INPUT: radmc3d.inp*), because then you can see afterward with which photon statistics the run was done.
- `countwrite`: [for MC] The nr of photons between 'sign of life' outputs in a Monte Carlo run. Default is 1000. That means that if you have `nrphot=10000000` you will see ten-thousand times something like `Photonnr: 19000` on your screen. Can be annoying. By adding `countwrite 100000` to the command line, you will only see a message every 100000 photon packages.

18.3 Switching on/off of radiation processes

You can switch certain radiative processes on or off with the following command-line options (though often the `radmc3d.inp` file also allows this):

- `inclstar`: [for images and spectra] Include stars in spectrum or images.
- `nostar`: [for images and spectra] Do *not* include stars in spectrum or images. Only the circumstellar / interstellar material is imaged as if a perfect coronograph is used.
- `inclline`: Include line emission and extinction in the ray tracing (for images and spectra).
- `noline`: Do not include line emission and extinction in the ray tracing (for images and spectra).
- `incldust`: Include dust emission, extinction and (unless it is switched off) dust scattering in ray tracing (for images and spectra).
- `nodust`: Do not include dust emission, extinction and scattering in ray tracing (for images and spectra).
- `maxnrscat 0`: (if dust is included) Do not include scattering in the images/spectra created by the camera. With `maxnrscat 1` you limit the scattering in the images/spectra to single-scattering. With `maxnrscat 2` to double scattering, etc. Can be useful to figure out the relative importance of single vs multiple scattering.

CHAPTER

NINETEEN

WHICH OPTIONS ARE MUTUALLY INCOMPATIBLE?

For algorithmic reasons not all options / coordinate systems and all grids are compatible with each other. Here is an overview of which options/methods work when. Note that only options/methods for which this is a possible issue are listed.

19.1 Coordinate systems

Some coordinate systems exclude certain possibilities. Here is a list.

Option/Method:	Cart 3D	Sph 3D	Sph 2D (axisymm)	Sph 1D
Second order ray-tracing	yes	yes	yes	yes
Isotropic scattering	yes	yes	yes	yes
An-isotropic scattering for thermal Monte Carlo	yes	yes	yes	yes
An-isotropic scattering for monochromatic Monte Carlo	yes	yes	yes	yes
An-isotropic scattering for images and spectra	yes	yes	yes	no
Full Stokes scattering for thermal Monte Carlo	yes	yes	yes	yes
Full Stokes scattering for monochromatic Monte Carlo	yes	yes	yes	yes
Full Stokes scattering for images and spectra	yes	yes	yes	no
Gas lines	yes	yes	yes	yes
Gas lines and Doppler-shift line catching	yes	yes	no	no

19.2 Scattering off dust grains

The inclusion of the effect of scattering off dust grains in images and spectra typically requires a separate Monte Carlo computation for each image. This is done automatically by RADMC-3D. But it means that there are some technical limitations.

Option/Method:	No scattering	Isotropic approximation	Full anisotropic/Stokes scattering
Fast multi-frequency ray tracing for spectra (auto)	yes	no	no
Multiple images at different vantage point at once	yes	yes	yes
Local observer	yes	yes	no

19.3 Local observer mode

The local observer mode (Sect. *For public outreach work: local observers inside the model*) is a special mode for putting the observer in the near-field of the object, or even right in the middle of the object. It is not meant to be really for science use (though it can be used for it, to a certain extent), but instead for public outreach stuff. However, it is kept relatively basic, because to make this mode compatible with all the functions of RADMC-3D would require much more development and that is not worth it at the moment. So here are the restrictions:

Option/Method:	Local observer mode
Dust isotropic scattering	yes
Dust an-isotropic scattering	no
Multiple images at different vantage point at once	yes
Second-order ray-tracing	yes
Doppler-catching of lines	no

CHAPTER

TWENTY

ACQUIRING OPACITIES FROM THE WWW

Opacities are the basic ingredients necessary for any model with RADMC-3D. The example models in this package contain example opacities, but for professional usage of RADMC-3D it may be necessary to get specific opacity data from the web. These opacity data are usually in a wide variety of formats. To enable RADMC-3D to read them usually requires a conversion into RADMC-3D-readable form (see Section *INPUT (required for dust transfer): dustopac.inp and dustkappa_*.inp or dustkapscatmat_*.inp or dust_optnk_*.inp* for dust opacities and Section *INPUT: Molecular/atomic data: The molecule_XXX.inp file(s)* for gas line opacities).

To make it easier for the user to create RADMC-3D-readable input files from opacity data downloaded from the web, we now feature a new directory `opac/` in the RADMC-3D distribution in which, for several of the most common WWW databases, we provide Python routines for the conversion. Please read the `README_*` files in this directory and its subdirectories for details.

Note also that Carsten Dominik made a very nice and easy-to-use tool to generate dust opacities on the Linux/Mac command line. It is called `optool`, and can be found on github at https://github.com/cdominik/optool . It can produce RADMC-3D-ready dust opacity files.

CHAPTER

TWENTYONE

VERSION TRACKER: DEVELOPMENT HISTORY

This version overview is very rough, and has only been started as of version 0.25.

- *Version 0.25*
  - Second order integration, based on a vertex-based grid (as opposed to the usual cell-based grid), implemented. This gives much smoother images, and you don't see the blocky cell structure anymore in the images. It requires extra memory, though. See Section *Second order ray-tracing (Important information!)*.
  - The number of photons for scattering Monte Carlo (i.e. the small MC run done before each image, if dust scattering is active) can now be chosen to be smaller for when you make a spectrum instead of an image. Reason: Since you anyway integrate over the images for a spectrum, you do not need the image to 'look nice', i.e. you can afford more photon noise. You can set this in `radmc3d.inp` by setting `nphot_spec=10000`, for instance. See Section *Scattered light in images and spectra: The 'Scattering Monte Carlo' computation*.
- *Version 0.26*
  - For line transfer: Added the 'doppler catching' method to the code. This prevents bad numerical artifacts in images/spectra of regions with large velocity gradients, where the doppler-shift between two neighboring cells exceeds the intrinsic line width of the material in the cell. See Section *Preventing doppler jumps: The 'doppler catching method'*.
  - NOTE: Up to, and including, version `0.26_23.02.11` this method (and for that matter any second order integration of line transfer) was not stable when strong shocks or contact discontinuities were encountered. This was because interpolation of the source function $S_\nu \equiv j_\nu/\alpha_\nu$ was done. Experimentation showed that interpolation of the emissivity j_ν is much more stable. As of version `0.26_27.02.11` this is fixed.
- *Version 0.27*
  - For line transfer: Implemented the possibility to use a Voigt line profile instead of just a Gaussian. This was implemented by Thomas Peters, and slightly modified by CPD. It uses the Voigt approximation by Humlicek JQSRT 27, 437 (1982) as programmed by Schreier, JQSRT 48, 743 (1992). It requires a user-defined subroutine `userdef_compute_lorentz_delta()` that sets the value of the Lorentz profile delta. This implementation is not yet documented, and may still be subject to modification.
  - Implemented the 'Large Velocity Gradient' (LVG) method (also called the Sobolev method) of approximate non-LTE line transfer.
  - Implemented the optically thin populations method.
  - Implemented the possibility of reading linelist molecular data instead of full molecular data. *Still needs testing.*
  - Finally implemented the positive `lines_mode` modes, i.e. in which the level populations are computed and stored globally before the ray-tracing. This has been latently in the code somewhat, but unfinished. Now it is implemented. The advantage is: it may be under some conditions much faster than the on-the-fly

computation of the populations during the ray-tracing (the negative `lines_mode` modes). Also it allows you to write the populations to file, so that you can examine them. Disadvantage: It is memory hungry.

- The level subset capacilities are now limited to only the storage of the levels in the global arrays (for positive `lines_mode` modes), and to the lines that will appear in images/spectra. For the rest, the full set of levels are always used from now on.
- Added a directory '`opac/`' which contains programs for generating your own dust opacities using optical constants from the web, and for generating your own molecular/atomic input data files using data from several web pages. The data from the web are not included, but there are README files that point you to the web sites.
- Tested the 'fisheye' fulldome (OMNIMAX) projection. It seems to work! Thanks to Mario Flock.
- Several small (and bigger) bugfixes
  - Fixed bug that showed up when no dust is included.
  - Fixed bug that caused RADMC-3D to crash when using no stars.
  - Fixed bug that caused RADMC-3D to crash when making images at very short wavelengths with nearly zero thermal emission.
  - Fixed bug in the AMR module when using second order integration or the doppler catching method with certain kinds of AMR-arrangements of cells.
  - Fixed many bugs when using a 'piece of a cake' model, i.e. using spherical coordinates in 3-D, but having the ϕ-grid going not over the full $0-2\pi$ range but e.g. just from 0 to $\pi/4$. It is rather rare that one really wants to use such grids (certainly not for real physical models, I presume), but for visualization of data it might be useful: for instance for visualizing a 3-D disk MHD model, which is cut open so you can see also to the midplane. Now it works. Thanks to Mario Flock.
  - Fixed bug with the aperture mode for spectra. Thanks to Daniel Harsono.
  - Fixed many bugs in linelist mode; now it works. Thanks to Attila Juhasz.
  - Fixed a bug in LVG mode that caused it to fail when AMR was used. Thanks to Anika Schmiedeke.
  - Fixed a tiny bug in `idl/radmc3dfits.pro`: filename was unused. Thanks to Stella Offner.
  - Retroactive bugfix from version 0.28 (see below): LVG and AMR mode.

  For details and for smaller bugfixes, read the `src/Radmc_3D_LOG.txt` document.

- *Version 0.28*
  - A number of people complained that even without AMR the code requires a huge amount of memory. That is because even if no AMR is used, the cells are connected via the AMR tree. Since the AMR cells contain information about which are the neighboring cells, and each cell has 6 neighbors, and slots for 8 child-cells (which are unused in case of a regular grid) this wastes a lot of memory space. The first big improvement in version 0.28 is that, from now on, the AMR tree is only set up and used if the grid indeed has refinement. If RADMC-3D notices that the grid is regular, it will not allocate space for the AMR tree, and everywhere in the code where the cell-management is done the code will switch to regular grid mode. There is now a flag `amr_tree_present` that says whether the AMR tree is present or not. Throughout the code there are now if-statements to switch between using or not-using the AMR tree. This may make the code a tiny bit slower, but this is only a minor reduction of speed. But as a result it should now be much easier to load huge regular grid models into memory.
  - A small (but potentially nasty) bug was found and fixed for the case when you use LVG mode on a grid with AMR-refinements. For the regular grid case (even in version 0.27, when it still used the AMR tree) this bug should not have caused problems, but perhaps you might want to check nevertheless. Note: This bug is now also retroactively fixed in version 0.27. See, as always, `src/Radmc_3D_LOG.txt` for details.

  - Added the possibility to visualize the location (along the line of sight) of the $\tau = 1$ surface (or any $\tau = \tau_s$ surface for that matter). See new Section *Visualizing the \tau=1 surface*. This can be very useful for getting a 3-D feeling for *where* certain emission comes from.

- *Version 0.29*
  - The big change in this version is that the whole stuff with the global storage of level populations has been improved. In earlier versions of RADMC-3D, either the populations of *all* levels of a molecule were stored globally (potentially requiring huge amounts of memory), or you would have to select a 'subset' of levels to store globally. This subset selection had to be done by the user ('manually', so to speak). You would have had to think a-priori which lines you wish to model, and which levels they connect, and then, in the `lines.inp` file you would have to select these levels by hand. That was cumbersome and prone to error. To avoid having to do this you could use 'on-the-fly' calculation of populations (by making the `lines_mode` negative), but that sometimes caused the code to become terribly slow. *Now this is dramatically improved:* From now on you can forget about the 'on-the-fly' calculation of populations. Just use the 'normal' way by which RADMC-3D first calculates the populations and then starts the ray-tracing. The subset-selection is now done automatically by RADMC-3D, based on which wavelengths you want to make the image(s) or spectra for (see Section *Background information: Calculation and storage of level populations*). Now the on-the-fly methods are no longer default and should not be used, unless absolutely necessary. Also the 'manual' subset selection is no longer necessary (though still possible if absolutely desired).
  - Added the subbox and sample capabilities to the level populations. See Sections *Making a regularly-spaced datacube ('subbox') of AMR-based models* and *Alternative to subbox: arbitrary sampling of AMR-based models*. Note that, in order to make it easier to identify which levels were written to file, the file formats of `***_subbox.out` and `***_sample.out` have been slightly modified: A list of identification numbers is added before the main data. For the dust temperature and dust density this list is simply 1 2 3 4 ... (dust species 1, dust species 2, dust species 3 ...), which is trivial. For the level populations (e.g. the file `levelpop_co_subbox.out` and `levelpop_co_sample.out` for the CO molecule) this list is, however, essential when not all levels were computed (see Section *Background information: Calculation and storage of level populations*). So if only level 4 and level 8 are stored, then the identification list is 4 8.
  - Fixed a bug which caused the code to crash when you put a star substantially far outside of the domain and try to make an image or spectrum. Thanks, Erika Hamden, for the bug report.
  - Fixed a bug that prevented the `lines_mode=50` mode from working. Now it works, and we can ask RADMC-3D to read the level populations from file (rather than calculating them internally). Also a new section was added to this manual describing this option (Section *Non-LTE Transfer: Reading the level populations from file*).
  - Added VTK output options (see chapter *Visualization with VTK tools (e.g. Paraview or VisIt)*) for allowing 3-D visualization of your model setups using e.g. Paraview, a freely available visualization tool.
  - Fixed a bug that occurred sometimes if a spectrum was made at inclination 90 and phi 90. Thanks Stella Offner for reporting this bug.

- *Version 0.30*
  - Fixed bugs in the Henyey-Greenstein scattering mode.
  - Introduced the new binary I/O feature: No more hassle with f77-unformatted records! The new binary mode is much simpler and more straightforward. This will help reducing the file sizes for large models. See Chapter *Binary I/O files*.

- *Version 0.31*
  - Added the possibility, in cartesian coordinates, to 'close the box', in the sense of making the domain boundaries thermal walls. Each of the 6 boundaries can be set separately, so you can also have just one

thermall wall. Also the temperatures can be set separately for each of the 6 boundaries. See Section *Thermal boundaries in Cartesian coordinates*.

- Added two new coordinate systems:
  - Cartesian 1-D plane-parallel (the only remaining active coordinate is z). The x and y dimensions are infinitely extended and have translational symmetry. The photons can, however, travel in full 3-D as always. See Section *1-D Plane-parallel models*.
  - Cartesian 2-D pencil-parallel (the two remaining active coordinate are y and z). The x dimension is infinitely extended and has translational symmetry. The photons can, however, travel in full 3-D as always.
  - For the 1-D plane-parallel mode it is possible to include parallel beams of radiative flux impinging on the 1-D atmosphere.
  - Attila Juhasz has improved the VTK output: Now it also supports 3-D spherical coordinates. Thanks, Attila!

- *Version 0.32*

  This is an intermediate version in which some stuff for the near-future modus of polarization is implemented.

- *Version 0.33*
  - Some minor technical changes to the doppler-catching integration of lines (storing the upper and lower level population instead of the jnubase and anubase variables).
  - Added the classical escape probability to the LVG mode (see Section *Non-LTE Transfer: The Large Velocity Gradient (LVG) + Escape Probability (EscProb) method* for details).
  - Sped up the filling of the matrix of the statistical equilibrium equation.
  - Vastly improved the LVG (and esc prob) method: Instead of the simple 'lambda iteration style' iteration as it was before, the A_{ik} is now multiplied with β_{ik} (the escape probability of the line i->k) and the J_{ik} is replaced by $J_{ik}^{\rm background}$. This means that the solution is almost instant, requiring only 2 or 3 iterations.

- *Version 0.34*

  Implemented the Modified Random Walk method, based on Min, Dullemond, Dominik, de Koter & Hovenier (2009) A&A 497, 155, and simplified by Robitaille (2010) A&A 520, 70. But beware: Still in the testing phase! By default it is switched off.

- *Version 0.35*
  - Implemented polarized scattering off randomly oriented particles. But beware: Still in the testing phase!
  - Fixed a bug in the modified random walk method (thanks to Daniel Harsono for spotting the problem and thanks to Attila Juhasz for finding the fix!)
  - Fixed two bugs that made it impossible to use second order integration with axially symmetric spherical coordinates and/or a finite-size star (thanks to Rolf Kuiper for reporting the bug).
  - Added the `sloppy` command line option to spectrum and image making in spherical coordinates. This was necessary because RADMC-3D is always trying to make 100% sure that all cells are picked up by the subpixels. In spherical coordinates these cells can be extremely non-cubic (they can be extremely flat or needle-like), which means that under some projections RADMC-3D feels obliged to do extreme subpixeling, which can make image- and spectrum-making extremely slow. By adding the `sloppy` keyword on the command line, RADMC-3D will limit it's pubpixeling which could speed up the calculation very much (but of course at your own risk!).

- *Version 0.38*
  - Implemented OpenMP parallellization of the thermal Monte Carlo (by Adriana Pohl). Still beta-version.

  - Bugfix in the mean intensity computation (mcmono) mode (thanks to Gwendoline Stephan).
  - Bugfix in the mean intensity computation (mcmono) mode (thanks to Seokho Lee).
  - Major bugfix in aperture mode (thanks to So ren Frimann).
  - Unformatted image format is from now on C-style binary instead of F77-style unformatted.
  - The viewimage tool is now ported to Qt by Farzin Sereshti, meaning that you can now use viewimage without having an IDL license. Viewimage is a very powerful tool to interactively make and view images of your model at different wavelengths and viewing angles. It can be found in the directory `viewimage_QT_GUI/`.
  - A Python package for RADMC-3D was developed by Attila Juhasz. It is included as of RADMC-3D version 0.38 in the directory `python/`.

- *Version 0.39*
  - Polarization mode is incompatible with mirror mode (in spherical coordinates). An error message is now included to catch this.
  - Minor bugfix in `pick_randomfreq_db()` (thanks to Seokho Lee).
  - Optimization of the OpenMP parallellization and extension of the OpenMP parallellization to the Scattering Monte Carlo computation (both by Farzin Sereshti).
  - Bugfix in `amrray_module.f90`: Sometimes one got 'Photon outside of cell' error due to a numerical precision round-off error. This bug is now (mostly?) fixed.
  - Bugfix in `sources_module.f90`: When using second order integration (or doppler catching) for line transfer in spherical coordinates, the line doppler shift was not transformed to spherical coordinates. This is now fixed.
  - Several bugfixes in the modified random walk method by John Ramsey. The method crashed for extreme optical depth problems due to out-of-cell events. Still not 100% perfect, but better.
  - John Ramsey also proposed two small fixes to the Planck function routines so that the events of overflow are caught. Note: This might change the results (in a tiny way: at the machine precision level) to the extent that a model run by an old version might not yield the same values to machine precision, but the differences should not matter in any meaningful way.

- *Version 0.40*
  - The RADMC-3D package is now 'officially' converting from IDL to Python wrappers. The Python modules were already there since a long time (thanks to Attila Juhasz!). But as of version 0.40 we will no longer update/maintain the IDL scripts (though they remain there and should remain working), and instead use python as the main setup and analysis tools for RADMC-3D. The full conversion will still take some time, but should be finished by the end of version 0.40.
  - Under some circumstances the simple 2x2 pixel plus sub-pixeling method for making spectra (default method) can be dangerous. For some grid geometries this can lead to under-resolving of the images that are integrated to obtain the flux, leading to a too low flux. So as of now 15.09.2016 the spectra and SEDs are always by default made with 100x100 images (and sub-pixeling of course). One can set the number of pixels with npix. So if you do `radmc3dsednostarnpix2` you get the original behavior again.
  - Bugfix in `montecarlo_module.f90`: The internal heat source method (which is still being tested) had a bug. The bug manifested itself for optically thin cells with non-negligible internal heat production. The energy was not immediately added to the cell. It only got added upon re-absorption of that photon package. Now this is fixed.
  - I now added some documentation for the heat source method, which is useful for e.g. disk viscous accretion heating.

  - Bugfix in `montecarlo_module.f90`: When using mirror symmetry in spherical coordinates in the θ-coordinate (i.e. modeling only the upper part of the disk and letting RADMC-3D assume that the lower part is identical), the distributed source luminosity was computed only for the top quadrant, and wasn't multiplied by 2. For most applications this does not cause problems, but for the heat source (see above), for continuous stellar sources and for the thermal origin of the isotropic scattering luminosity (for non-isotropic scattering, mirror symmetry was not allowed anyway), this could lead to a factor of 2 underestimation (only if mirror symmetry was used, i.e. if the θ coordinate was going only up to $\pi/2$). This is now fixed. To test if the fix works one can simply make the same model again, but now without using mirror symmetry (and thus using twice as many cells in θ, to cover both the upper and lower half of the object). This should yield (apart from some Monte Carlo noise) the same results.
  - Improved the stability of the Modified Random Walk (MRW) method a bit further.
  - Bug fix: scattering mode 3 (tabulated phase function, but not full polarization) had a bug which caused images of scattered light to be multiplied by some arbitrary number. Reason: as a phase function it returned Z_{11} instead of $4\pi Z_{11}/\kappa_{\rm scat}$. Most people use either isotropic scattering (scattering mode 1), or Henyey-Greenstein (scattering mode 2) or full polarization (scattering mode 5), all of which are ok. At any rate: the problem is now fixed, so scattering mode 3 should now also work.

- *Version 0.41*
  - Implemented a first testing version of the aligned grains: only polarized thermal emission so far. Still very much a testing version.
  - Implemented a method to also allow full Stokes vector polarized scattering in the 2-D axisymmetric mode in spherical coordinates. Until now the full scattering mode (scattering mode 5) was only possible in full 3-D. Note however that anisotropic scattering in 2-D axisymmetric models requires scattering mode 5, which is the full scattering mode. It is still not possible to use intermediate scattering modes (like henyey-greenstein or any scattering mode between 2 and 4) in 2-D axisymmetry. But those intermediate modes are anyway more for testing than for real models, so that should be ok.
  - Bugfixes to the OpenMP stuff. In particular the OpenMP parallellization of the scattering MC crashed. This is now fixed. In general the OpenMP stuff was a bit cleaned up.
  - Bugfix in thermal Monte Carlo with full polarization mode: needed to reset the photon package after each thermal absorption/re-emission event. Usually the effect is subtle, but had to be fixed.
  - Bugfix in reading the `scattering_angular_grid.inp`: the `theta` angles should be converted into radian. But this file was not officially offered before anyway.
  - Attila Juhasz has made a large improvement of his python package for RADMC-3D. See the `python/` directory. This is version 0.29 of his package. This package now also supports reading and writing AMR grids.
  - Bugfix in VTK for 3-D spherical coordinates (thanks Attila Juhasz!). Now it should work.

- *Version 2.0*

  Version 2.0 is the version after 0.41. We skip version 1.0, because version 1.0 could be mistaken for the first version of the code. Version 2.0 is mostly the same as 0.41, but with a few differences.

  - IDL support is removed permanently. From now on, the front-end functionality is only in Python. We assume Python 3.
  - Version 0.30.2 of the `radmc3dPy` Python package (written by Attila Juhasz) has been implemented. It is also being improved, mainly to make its use easier (i.e. with more automatic default behavior).
  - A very simple `simpleread.py` reading library is provided as a 'light version' of radmc3dPy. It contains only some basic reading functions, and only for ascii output (no binary files).

- Some of the standard-output is shortened. You can also call a Monte Carlo run with `radmc3d` with the command line options `countwrite 100000` to make RADMC-3D write a message only every 10^5 photon packages instead of every thousand.
- We removed the fortran-unformatted data format from the manual, and will remove it from the code in later versions. Use either text (ascii) format or binary format.
- The manual is now converted to Sphinx, from which the LaTeX version and the HTML version can be automatically created.
- [as of 11.11.2021] BUGFIX: For OpenMP parallel thermal Monte Carlo computation of the dust temperatures for multiple grain species or sizes, when `iranfreqmode=1` (as opposed to the default value of `iranfreqmode=0`), the dust temperatures could acquire errors because the `pick_randomfreq_db()` subroutine uses the array `db_cumul(:)` as thread private, but without having it declared as such. This led to interference between threads. This is now fixed.
- [as of 30.01.2022] Added unstructured grid support.

CHAPTER

TWENTYTWO

INDICES AND TABLES

- genindex
- modindex
- search