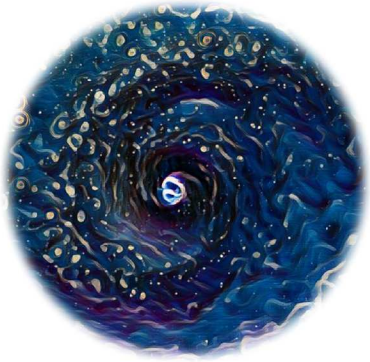
Chapter 4



The Synthesizer

Synthetic observations from 3D numerical simulations

4.1 Introduction

4.1.1 What is a synthetic observation?

A synthetic observation is a theoretical construct created by combining observational setups and models. In other words, it is a simulation of what scientists would see if they were able to observe a physical system under particular conditions. It allows astronomers to investigate the impact of different observational setups before proposing and performing the real observations. Since astronomers have full control over the simulated setup, they can use it to find the most convenient observing setup when writing proposals or to directly compare models with already observed astronomical sources.

In order to properly compare simulations and observations, your observed models need to look as realistic as possible. This means, one needs to post-process the simulation output or analytical models by adding a series of instrument-related effects, such as resolution and sensitivity. All these values vary from telescope to telescope and must be as tailored as needed for every par-

ticular observation setup. According to Haworth et al. (2018), the most typical post-processing steps are:

1. **Spectral transmission:** Real telescopes do not have uniform sensitivity to all frequencies within a given bandwidth. As a result, it is necessary to use a filter that takes into account the spectral sensitivity of the detector. A box filter can provide a reasonable first-order approximation of such a filter.

2. **Beam convolution:** In single-dish telescopes, the response to the observed brightness distribution is determined by the antenna pattern or Point Spread Function (PSF), which can be reasonably approximated as a two-dimensional Gaussian beam. For interferometric arrays, the PSF is obtained through the Fourier transform of the sampled *uv*-plane. In order to properly compare simulations to observations, the physical resolution of the simulation needs to be downsampled to the telescope angular resolution and antenna efficiency.

3. **Finite pixel resolution:** The angular pixel size of a radiative transfer model needs to be at least same as small as the physical resolution of the simulation output, in order to catch all physical structures. However, the angular pixel size of an astronomical image is calculated such that it does not under- or over-sample the telescope's beam (Nyquist sampling) and is probably much bigger than the model's pixel size. Downsampling and flux rescaling of the radiative transfer output is therefore another necessary step.

4. **Addition of Noise:** The addition of noise is maybe the most important step during the image post-processing and should be based on the technical specifications of the simulated instrument. Various types of noise, including non-thermal noise such as mutual interference of wavefronts in speckle patterns seen in coronographic imaging, as well as noise added by side-lobes, and thermal noise, should be taken into consideration. Thermal noise is particularly relevant to consider as it plays a significant role in determining whether structures can be observed or not. If the level of noise exceeds the maximum signal, it can prevent the detection of structures that might be otherwise assumed to be detectable.

Post-processing simulation outputs with radiative transfer modelling and addition of instrument-related effects can soon become a tedious and repetitive task, specially when you need to do parameter space exploration. Since the order of each of these steps is usually the same, the whole post-processing pipeline can be easily automated, leaving the parameter controls all in one space. This is basically the motivation I had to create the Synthesizer, a tool that automates this pipeline without the need to learn and create input files for all the available different codes formats.

4.2 The code

The Synthesizer is a program used to calculate synthetic images from a numerical model directly from the command-line. It pipelines all the steps involved in a synthetic observation (see Fig. 4.1), from the generation of a physical model, either from an analytical prescription or from a 3D simulation output, to the generation of a realistic telescope observation. The program can be used

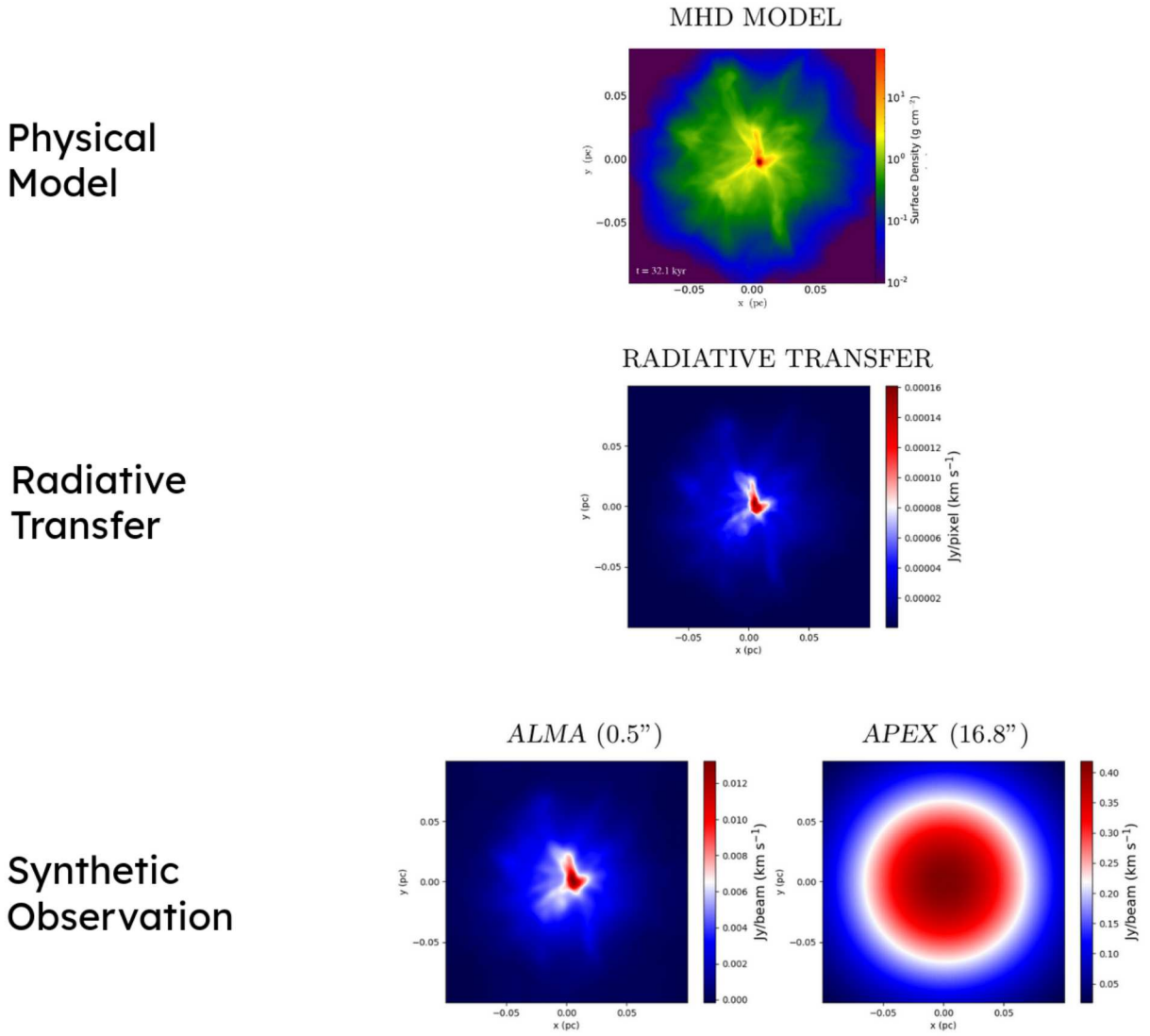


Figure 4.1: All the steps involved in the process of generating a synthetic observation, starting from the model, generating the flux via radiative transfer and finally convolving the image to a desired primary or synthesized beam, plus the addition of thermal receiver noise.

either in one or all of the modules available (see section 4.3). This means, you can use it to only calculate your model, and refine it, without the need to run all the extra steps (radiative transfer, opacity calculation, etc.), you can use it to post-process an independently prepared model or to investigate the effect of changing the physical properties of your model. The synthesizer comes also handy for users that might not be interested in modelling or generating synthetic observations, but in need of an easy to use data visualization tool. For instance, it allows for 2D and 3D visualization of your analytical models and simulation outputs, without the need to program your own plotting routines.

4.2.1 Requirements

The Synthesizer consists of four main modules (see section 4.3), out of which two are self-contained features fully written in Python. This means, that to use the `--grid` and `--opacity` options, no further installation is needed beyond the main package installation. For the other two `--raytrace` and `--synobs` options, two different softwares need to be separately installed. These are the commonly used parallelized radiative transfer code RADMC3D (Dullemond et al. 2012)[1] and the Common Astronomy Software Applications (CASA; McMullin et al. 2007) [2], needed for the raytrace and synobs modules, respectively. When attempting to use either of the two modules without the already-installed corresponding software, a warning and help message will be prompted by the Synthesizer. Installation of them both is quite easy and quick, but it needs to be done manually by the user.

4.2.2 Installation

The Synthesizer is fully written in Python 3 and is structured in the form of a Python package, publicly available through the Python Package Index[3] and its public repository[4]. Installation of the latest stable version can be easily done via the python-pip command:

```
$ pip install astro-synthesizer
```

In order to catch up on more recent updates or to modify the code to your own needs, installation of a development version can also be done from source with the following commands:

```
$ git clone https://github.com/jzamponi/synthesizer
$ cd synthesizer
$ pip install -e .
```

The `-e` flag added to the last install command means the code will be installed in *editable* mode. This is optional and allows the user to have modifications to the source code taking immediate effect without the need to reinstall. This is particularly useful for users interested in modifying the predefined physical models. Once installed, an executable python file called `synthesizer` will be created in one of the folders added to your `PATH`. This is most likely `~/.local/bin`

[1] `https://github.com/dullemond/radmc3d-2.0`
[2] `casa.nrao.edu`
[3] `https://pypi.org/project/astro-synthesizer`
[4] `https://github.com/jzamponi/synthesizer`

by default but it can be customized providing the `--prefix` flag (see `pip install --help` for further details). To make sure the program is now properly installed and callable, simply type `$ synthesizer` in your terminal and you should get the following output:

```
$ synthesizer
  [synthesizer] Nothing to do. Main options: --grid, --opacity,
                --monte-carlo, --raytrace or --synobs
  [synthesizer] Run synthesizer --help for details.
```

4.2.3 Running the code

Using the code is quite easy, that is its philosophy. All options and parameters can be provided directly from the command line without the need to create input files or learning new file formats. A list of all possible options and commands can be obtained by typing `$ synthesizer --help`. A copy of this output is provided in Table 4.1.

As a quick initial example, the following command will create a protoplanetary disk model (within a cartesian rectangular box) both in density (default) and temperature (optional):

```
$ synthesizer --grid --model ppdisk --temperature --opacity
  --show-grid-2d --show-grid-3d --raytrace --synobs --show-rt --show-synobs
```

The code will first create the model grid. Once this is done it will show 2D midplane slices of the density and temperature in two different projections. You can close this window pressing `q`. It will then, open a Mayavi scene where the 3D density model is rendered and will dump a help message to the terminal. This figure is completely interactive and customized only to a minimum level. Once examined and closed, it will open a new scene rendering this time the dust temperature. After the figure is closed, the code will dump all the model information into text files ready to be fed to RADMC3D. These files are *amr_grid.inp*, *dust_density.inp* and (optional) *dust_temperature.dat*. The `--opacity` option tells the code to calculate a dust opacity table for bare silicate grains (by default) with a minimum and maximum grain size of 0.1μm and 10μm, respectively (by default). This opacity is also written in a RADMC3D-compliant syntax to the file *dustkappa_s-10um.inp*. Given the `--raytrace` option, the Synthesizer will now call RADMC3D to run a monochromatic (1.3 mm by default) continuum radiative transfer calculation. Provided the `--show-rt` option, the resulting image will be opened and displayed once the calculation is done. The `--synobs` option represents the last step of the synthetic observation, as illustrated in Fig. 4.1, and consists on formatting the output from RADMC3D into a CASA-compliant format, i.e., to a header-customized FITS File containing all the relevant information for CASA. The Synthesizer then creates a template CASA script called *casa_script.py* with two main function calls, these are the two CASA tasks `simobserve` and `tclean`, plus some extra post-processing. Simobserve reads in a flux model, namely, the output from the radiative transfer calculation (called *radmc3d_I.fits*), and generates a measurement set (.ms) within a new folder called *synobs_data*. A measurement set is the CASA format for interferometric visibilities (by default). The visibilities are then provided to tclean and deconvolved to generate a final image. This image is saved in the new *synobs_I.fits* file. Provided the `--show-synobs` option, the final image will be opened and displayed. Once the full pipeline is finished, we encourage the

user to take a closer inspection to the standard output in order to get an idea of the parameters involved in the calculation.

4.3 Modules

4.3.1 Gridding (`--grid`)

The core functionality of the Synthesizer is to generate flux distributions from a physical model via radiative transfer calculations using the RADMC3D code. By design, RADMC3D does the raytracing and Monte-Carlo photon propagation on a spatial grid, as most common radiative transfer codes do. Because of this, all physical models, regardless of their numerical origin (analytical prescriptions or particle or grid-based simulations), must be constructed on a grid. In the following sections we describe how does the synthesizer generates such gridded models, even in the case of originally non grid-based simulations.

Using analytical models

One of the most common approaches to radiative transfer modelling is via analytical descriptions of a physical object. One of its main advantages is that it is quick to create them at a reasonable resolution. This facilitates the parameter space exploration and allows the further tailoring of the models to your own scientific case. The Synthesizer comes bundled with a few pre-defined models that can serve as a starting point for more sophisticated descriptions. Some of them are: a constant density box, a sphere with a power-law radial density profile, a filament, a prestellar core (see Fig. 4.2), and a protoplanetary disk (see Fig. 4.3). More models are to come in a future version of the program, such as bespoke profiles for most DSHARP disks (Andrews et al. 2018) and the textbook prestellar core L1544 (Chacón-Tanarro et al. 2019; Caselli et al. 2019).

The Synthesizer does also allow to calculate synthetic polarized fluxes produced by the alignment of elongated dust grains. To do this, the physical model needs to include information about the vector field along which grains will align. Vector fields can be directly converted from the realistic velocity or magnetic field of a 3D simulation by passing the `--alignment` option, as long as this information is available in the snapshots. If no vector field is available, they can be easily created by adding the `--vector-field` option, followed by a keyword indicating the name of the field. Available keywords are x, y, z (for uniform fields), toroidal, hourglass, helicoidal (superposition of toroidal and hourglass), dipole and quadrupole. When a vector field is added to the model, the field streamlines are overplotted on top of the density colormaps, provided the `--show-grid-2d` option. An example of such images for some of the available vector fields is shown in Fig. 4.4.

Using grid-based hydro codes

The generation of models from grid-based hydrodynamical simulation outputs is in principle quite straightforward. In this gridding mode, the Synthesizer acts merely as a converter interface. For the radiative transfer to work, numerical limitations referring to the grid resolution, geometry

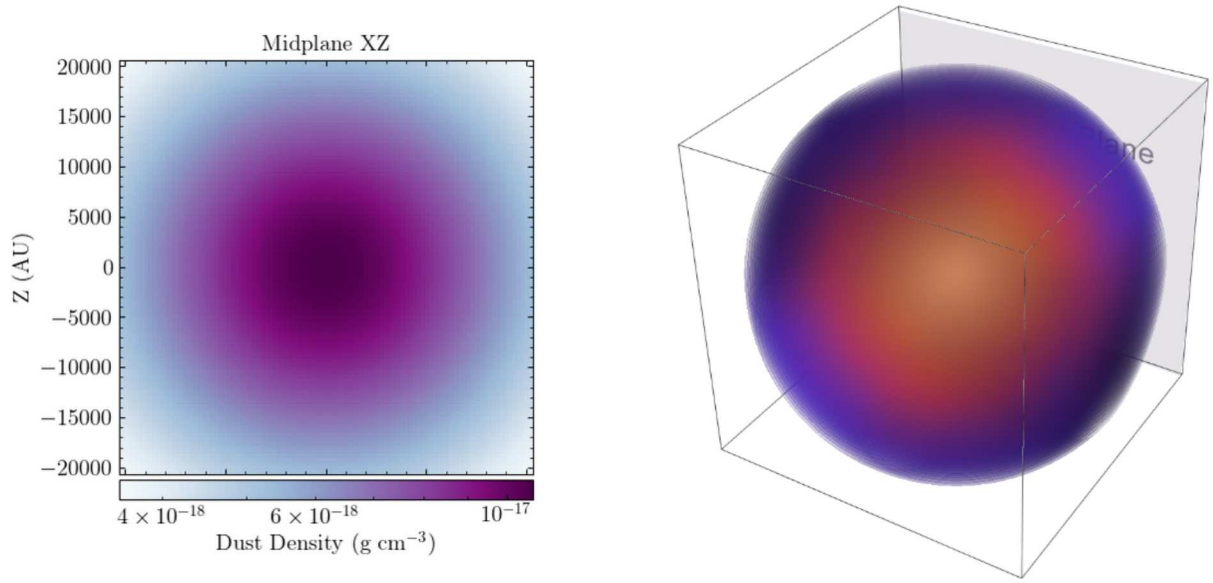


Figure 4.2: Analytical model a prestellar core, described as a Bonnort-Ebert sphere. Panels show a 2D midplane slice of the dust density and a 3D rendering of the same structure, from left to right.

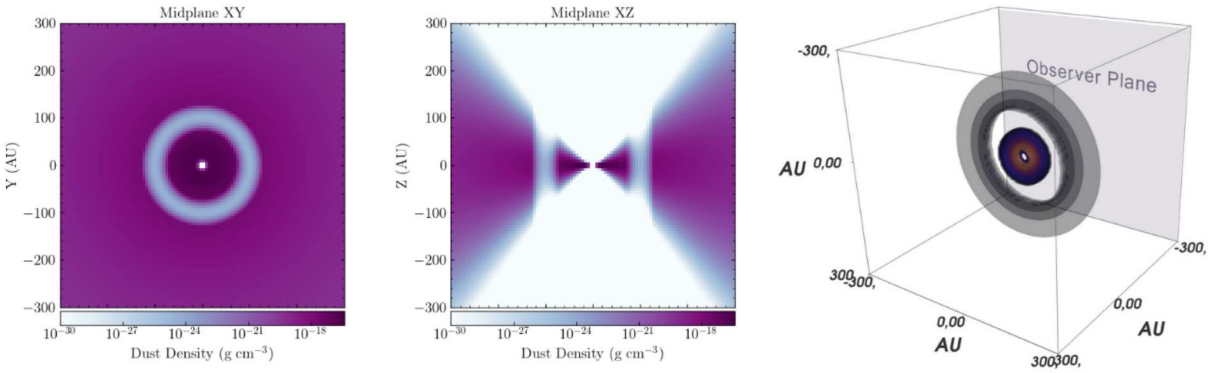


Figure 4.3: Analytical model of a protoplanetary disk with a gap. Left and middle panels show the 2D midplane slices of the model in face-on and edge-on projections. The right panel shows a 3D rendering of the disk density, indicating the plane of observation in gray.

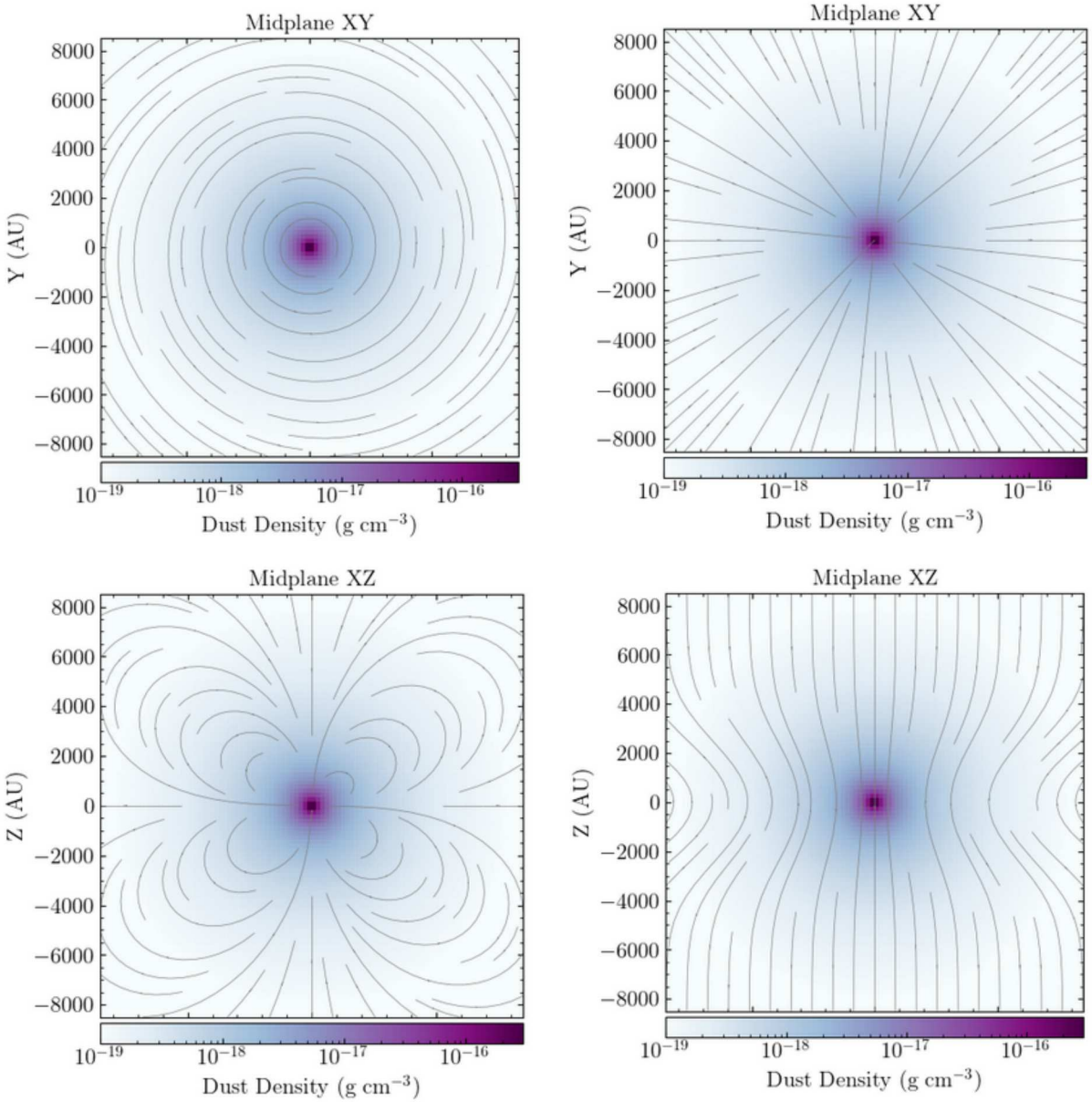


Figure 4.4: Some of the analytical vector fields available within the Synthesizer. The top row shows a toroidal and a radial vector field, from left to right, respectively. Bottom row shows a quadrupole and hourglass field morphology.

and memory size depend on what types of grids are supported by RADMC3D. In this regard, all the Synthesizer does is to read in all the grid and density information from the simulation output and translate it into the RADMC3D grid file format. Grid types currently supported by RADMC3D are regular cartesian and spherical grids, statically refined spherical grids and Oct-tree and layered cartesian grids (commonly generated by hydro codes with Adaptive Mesh Refinement; AMR). Beta versions of RADMC3D do offer support for unstructured grids such as Delaunay triangulations and Voronoi tessellations, however this not yet fully tested. The Synthesizer does not support conversion of unstructured meshes until it is fully supported by RADMC3D. The Synthesizer is very young and at the moment only snapshots from the grid-based MHD code ZeusTW (Krasnopolsky et al. 2010) are supported. Creating interfaces for new codes is easy to implement but it takes time. For AMR codes, initially only grid and density (and optionally temperature) information is needed to generate the input for RADMC3D. Future versions of the code will support conversion for some of the most commonly used hydro codes, such as Athena++ (Stone et al. 2020), FLASH (Fryxell et al. 2000), ENZO (Bryan et al. 2014) and RAMSES (Teyssier 2002) (see section 4.5). In order to use simulation outputs from grid-based codes, the `--source` and `--amrfile` options must be given. An example command would look like

```
$ synthesizer --grid --source athena --amrfile cbd.out1.00001.athdf ...
```

Using particle-based hydro codes

The core functionality of this module lies in its interpolation implementation for particle-based codes, namely Smoothed Particle Hydrodynamics (SPH) codes. The Synthesizer reads in 3D positions (in cartesian coordinates) and density values for an ensemble of particles. It then uses the *interpolate* module from the Python SciPy (Virtanen et al. 2020) library to generate a regularly-spaced 3D density cube by linearly (default) interpolating all the particle density values. The same is repeated for temperature values and vector field components when requested. In order to avoid interpolating low-density regions of low scientifical interest, the options `--bbox` or `--rout` can be given, followed by a physical scale in au. These options reject particles outside of a given range. Both options act as a zoom-in implementation, where particles outside a given bounding-box (`rout`) or outter radius (`rout`) are excluded from the interpolation. This command can also be combined with the `--ncells` option to increase the resolution of a given region. This is particularly useful when using grids with a large spatial and physical dynamic range. An example of this implementation is shown in Fig. 4.5, for the case of an SPH protostellar disk formed out of the collapse of an initially spherical prestellar core. Because the interpolation of SPH particles needs only particle positions and physical quantities, there is not actual "conversion" involved in the generation of the grid. All the code needs is a way to read in one-dimensional data arrays containing the particle coordinates and values. This means that creating interfaces to more SPH codes is quite easy to implement. The most common approach is through HDF5 (Hierarchichal Data Format 5) file readers, which is the de facto standard for large simulation outputs storage. The Synthesizer does currently offers compatibility with output files from SPHng (Bate et al. 1995), GIZMO (Hopkins 2015), GADGET-2 (Springel 2005) and AREPO (Springel 2010). In

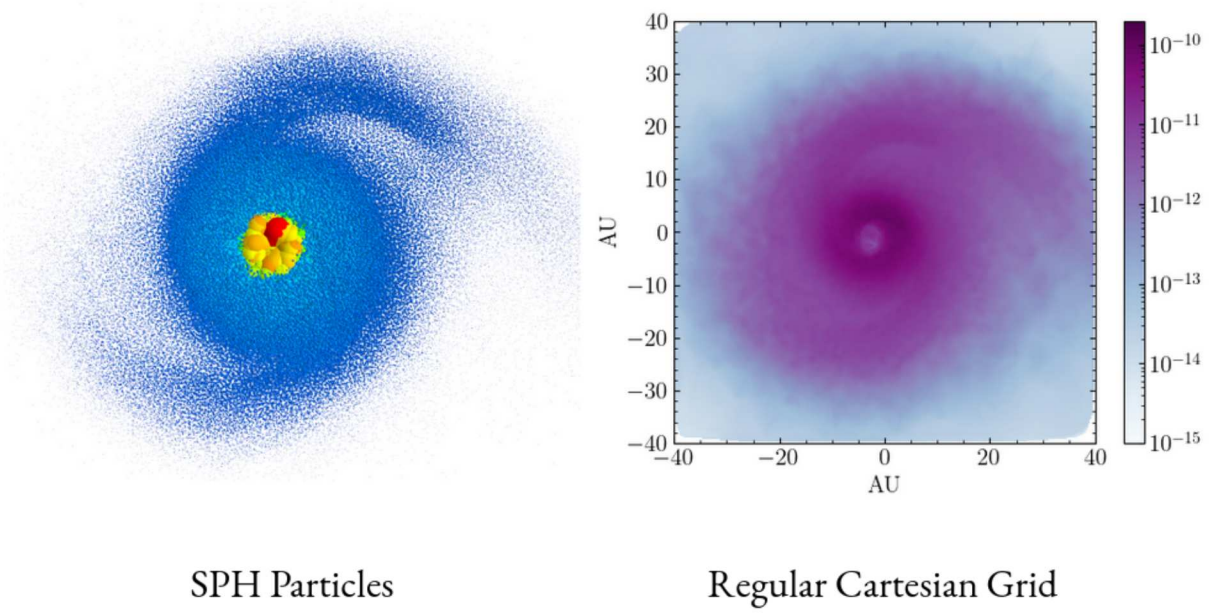


Figure 4.5: Interpolation of SPH particles onto a regular cartesian grid as done by the gridding module. This example shows the gridding of an SPH protostellar disk formed 18 kyr after the collapse of a Bonnor-Ebert sphere.

order to use simulation outputs from particle-based codes, the `--source` and `--sphfile` options must be given. An example command would look like this

```
$ synthesizer --grid --source arepo --sphfile snap_001.hdf5 ...
```

4.3.2 Dust Opacity (`--opacity`)

The second module available within the Synthesizer is in charge of creating dust opacity tables. This module is accessible via the `synthesizer --opacity` command.

How to calculate dust opacities

The simulated fluxes produced from a density model are the result of the radiative transfer equation, solved along a certain line-of-sight, plus the contribution of the emission scattered within the grid. The core quantity involved in the radiative transfer equation is the opacity of the radiating material. In the context of this thesis, dust continuum observations, the main contributor to the opacity is the dust opacity, which in turn depends on factors like the composition and the size of the grains. The Synthesizer includes a full module to calculate dust opacity tables for different mixtures and it is robust enough to receive its own name. This module is called DustMixer, and it is been shortly introduced in Chapter 3, section B.1. The DustMixer is a full command-line driven tool as well, allowing to easily experiment with various dust models and the effects of polarization by scattering or grain alignment. In this section, we describe the process of calculating

dust opacity tables starting from laboratory measurements of specific material. These properties are represented as a complex number m, composed by two real and imaginary parts of the form

$$m(\lambda) = n(\lambda) - ik(\lambda). \tag{4.1}$$

They both receive the name of refractives indices, or also known as optical constants, and can be obtained from the literature (e.g., Draine & Lee 1984, Ossenkopf & Henning 1994, Weingartner & Draine 2001, Jager et al. 1998, Draine 2003a, Woitke et al. 2016 and Birnstiel et al. 2018) or from the nicely compiled repository of optical constants from the Optool[5] program. Optool (Dominik et al. 2021) is another publicly available command-line driven tool for calculating dust opacities with different mixtures. Optool is in fact the main motivation for the DustMixer to be created, as a tool easy to embed within the Synthesizer, and offers larger flexibility and customization for dust ensembles than the DustMixer does in its current version.

The DustMixer starts by reading in data tables of n and k values (see Fig. 4.6), for a specific material (e.g., silicates, graphites, molecular ices, etc), tabulated over a range of wavelengths. It then (see Fig. 4.6) uses the Bohren & Huffman (1983) algorithm BHMIE to convert optical constants into dust efficiencies Q, for a specific grain size, under the assumption that grains are spherical. Dust efficiencies represent the electromagnetic response of a medium (a spherical dust grain in this case) to an incident plane electromagnetic wave. The incident radiation causes the grain to radiate electromagnetic waves of itself. This is in principle an Electrodynamics problem, but the nature of this interaction can be modelled in simplified ways for cases when the grain size is either much larger or much smaller than the incident wavelength. The main quantity involved in this scattering event is the scattering efficiency $Q_{\rm sca}$. For grain sizes much smaller than the incident wavelength ($a << \lambda$), this event is called *Rayleigh scattering*, and the scattering and absorption efficiencies can be obtained by

$$Q_{\rm sca} = \frac{8}{3}\left(\frac{2\pi a}{\lambda}\right)^4 \left|\frac{m^2-1}{m^2+2}\right|^2 \qquad\qquad Q_{\rm abs} = 4\left(\frac{2\pi a}{\lambda}\right) {\rm Im}\left|\frac{m^2-1}{m^2+2}\right| \tag{4.2}$$

where a is the radius of the dust grain. On the other hand, if the particle is much larger than the incident wavelength ($a >> \lambda$), the scattering can be treated as a geometric reflection and light can be propagated simply using raytracing. The middle case when the grain size is comparable to the incident wavelength ($a \sim \lambda$) is the most difficult to model and it was firstly introduced by Gustav Mie (Mie 1908; thereafter named Mie scattering), who described the sphere's response to light as a solution to Maxwell's equations. In the Mie regime, because the size of the grain is comparable to the wavelength, the incident electromagnetic wave arrives at different parts of the sphere's surface with different phases, and the net interaction must be modelled as an infinite sum of independently interacting dipoles. This way, the outgoing wave from the dielectric sphere is represented as the expansion of a scalar field using spherical harmonics but for a vector field. Similar to a scalar field, this involves numerically solving Legendre polynomials and Bessel functions, with expansion coefficients a_n and b_n, with n the expansion index going from $n = 1$ to $n \to \infty$, also known as scattering coefficients. This calculation requires to similarly decompose

[5] `https://github.com/cdominik/optool`

the incident wave into infinite spherical harmonics, with scattering coefficients c_n and d_n. By coupling the expressions for the incoming and outgoing radiation fields (see Bohren & Huffman 1983) and applying boundary conditions for a dielectric sphere, we can obtain the expressions for the outgoing scattering coefficients for a given n

$$a_n = \frac{m\psi_n(mx)\psi_n'(x) - \psi_n(x)\psi_n'(mx)}{m\psi_n(mx)\xi_n'(x) - \xi_n(x)\psi'(mx)} \tag{4.3}$$

$$b_n = \frac{\psi_n(mx)\psi_n'(x) - m\psi_n(x)\psi_n'(mx)}{\psi_n(mx)\xi_n'(x) - m\xi_n(x)\psi'(mx)} \tag{4.4}$$

where ψ and ξ are the Riccati-Bessel functions. With these two expressions we can write the scattering dust efficiencies from Mie theory as

$$Q_{\rm sca} = \frac{2}{k^2a^2}\sum_{n=1}^{\infty}(2n+1)\left(|a_n|^2 + |b_n|^2\right) \tag{4.5}$$

with $k = 2\pi/\lambda$ the wave number of the incident wave. A second product of Mie theory and the BHMIE algorithm is the dust extinction efficiency

$$Q_{\rm ext} = \frac{2}{k^2a^2}\sum_{n=1}^{\infty}(2n+1){\rm Re}(|a_n|^2 + |b_n|^2) \tag{4.6}$$

The final relevant quantity involved in the radiative transfer equation is the absorption dust efficiency. This can be obtained from the scattering and extinction efficiency as

$$Q_{\rm abs} = Q_{\rm ext} - Q_{\rm sca}. \tag{4.7}$$

The BHMIE algorithm explicitly uses the theory of Mie scattering to obtain the scattering efficiencies regardless of the size of the grains, which in the context of small grains deliver the same results as the analytic expression for Rayleigh scattering (equation 4.2). It does also use Mie theory for grain sizes larger than the incident wavelength, without a geometric approximation. This makes the calculation of dust efficiencies highly computationally expensive for large grains (e.g., millimetric scattering on millimetric and centimetric dust grains as observed by ALMA and JVLA). The calculation of such sums (4.5 and 4.6) is computationally expensive, specially for larger grains. The bigger the grain size the more terms need to be included in the sum for it to converge. The computational expense is determined by the so called size parameter $x = 2\pi a/\lambda$. For $x \simeq 1$, $n \simeq x$ terms are needed. For larger size parameters $x >> 1$, this might take much longer. In the Synthesizer, on sequential mode (i.e., single-thread calculation), calculating dust efficiencies for a $10\,\mu$m sized grain takes no longer than 1 minute, and only a few seconds for even smaller grains. For 1 mm sized grains this can scale up to 20 or 40 minutes depending on the resolution of the wavelength grid (100 or 200 bins, respectively). In order to speed this up, the Synthesizer employs a multiprocess parallelization scheme to simultaneously calculate efficiencies for an ensemble of grain sizes. This however, still slows down on the bins corresponding to the largest grain sizes, and currently only works when no scattering matrices are needed, but

it does result in a faster calculation. To avoid re-calculating full opacity tables on every single synthetic observation, precomputed high-resolution dust opacity tables have been made available in a public repository[6]. This does not need to be manually downloaded by the user. If no explicit call to the DustMixer is made (via the `--opacity` option), the Synthesizer will first check whether an opacity table is already existent in the local directory, and if not, it will attempt to download the corresponding table from the public respository, as long as it exists.

The final step in the calculation of dust opacities is the conversion of dust efficiencies into mass-weighted dust opacities κ (see Fig. 4.6). This is maybe the quickest step to perform numerically, as it only involves the conversion of the adimensional dust efficiencies Q and cross sections $\sigma = 2\pi a^2$ (cm^2) into opacities κ (cm^{2}g^{-1}). For a single grain size, this is done as

$$\kappa_{\rm abs} = 2\pi a^2 Q_{\rm abs} \tag{4.8}$$

$$\kappa_{\rm sca} = 2\pi a^2 Q_{\rm sca} \tag{4.9}$$

$$\kappa_{\rm ext} = 2\pi a^2 Q_{\rm ext}. \tag{4.10}$$

If the DustMixer is called without customized by parameters, it will default to a power-law grain size distribution for 100 grains sizes between 0.1 μm and 10 μm. The results of this calculation are shown in Fig. 4.6. This default setup can be easily reproduced with the following command

```
$ synthesizer --opacity --show-nk --show-dust-eff --show-opac
```

The `--show-*` flags are completely optional and only serve as illustration. This command represents the default setup, which can be also explicitly expanded as

```
$ synthesizer --opacity --amin 0.1 --amax 10 --na 100 --q -3.5 --lmin 0.1 --
   lmax 1e5 --nlam 200 --material s
```

To explicitly calculate dust opacities for a single grain size, either the minimum and maximum grain size must be set equal or the number of grain size bins (`--na`) must be set to 1, as

```
$ synthesizer --opacity --amin 100 --amax 100 --na 1
```

Dust opacities for a range of grain sizes, as the default case, are obtained by calculating the dust efficiencies Q for each grain size and then integrating them over a size distribution. Currently, only power-law size distribution of the form $N(a) \propto a^q$ are implemented, with $q = -3.5$ the MRN (Mathis et al. 1977) interstellar size distribution slope as default. In such case, the size integrated dust opacity κ_λ is obtained as

$$\kappa_\lambda = \frac{1}{C} \int_{a_{\rm min}}^{a_{\rm max}} \pi a^2 Q_\lambda(a) \varphi(a) da \tag{4.11}$$

with $\varphi(a) = a^q$ and C a mass normalization constant of the form

$$C = \frac{4}{3}\pi\rho \int_{a_{\rm min}}^{a_{\rm max}} \varphi(a) a^3 da \tag{4.12}$$

where ρ is the density of the considered material (silicate by default).

[6] `https://github.com/jzamponi/utils/tree/main/opacity_tables`

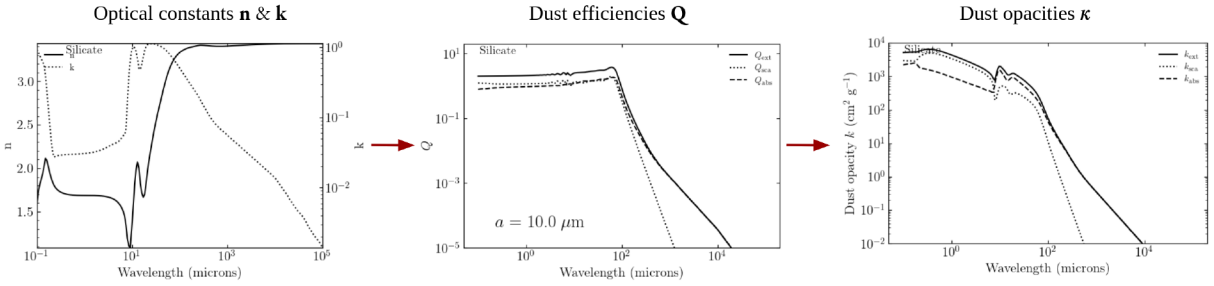


Figure 4.6: The process of calculating opacities. The first step (left) is to obtain laboratory optical constants of a given material. These are then converted into efficiencies (middle) for a given grain size and finally converted into mass-normalized dust opacities (cm^2 g^{-1}; right). The optical constants here shown represent silicate dust (Draine 2003b). The dust efficiencies are calculated for a single grain size of 10 μm and the dust opacities are obtained integrating the efficiencies for grains of 0.1 μm up to 10 μm, using a power-law size distribution ($N(a) \propto a^{-3.5}$), for 100 size bins. This calculation takes 30 seconds to run.

The resultant thermal emission from a radiative transfer model is obtained by solving the radiative transfer equation along a given line of sight. The propagation of this flux is determined by the extinction opacity $\kappa_{\rm ext}$. However, the total flux of a model should in principle consider the contributions from both the thermal and the scattered flux. Because following the direction of scattered rays in all directions (raytracing) would largely increase the computational time, one of most commonly employed numerical approaches is the use of Monte-Carlo calculations (Steinacker et al. 2013). In this way, scattering events are modelled by launching a finite amount of photons from a radiating source (either the protostar or the dust itself) and letting them enter the grid cells, depositing or removing a finite amount of energy, and being re-launched in a new random direction (Bjorkman & Wood 2001), until all photons have escaped the grid. Those photons reaching the observing plane are then counted towards the scattering flux. The direction of the outcoming photons after an scattering event is not isotropically distributed (isotropic scattering) in most astrophysical contexts. The most commonly used probability distribution function for the outcoming scattering angle θ is called the *Henyey-Greenstein* phase function (Henyey & Greenstein 1941), and is fully parametrized by a single parameter g. This scattering phase function is the randomly-sampled (Monte-Carlo) distribution function and receives the form

$$p_g(\mu) = \frac{1}{2}\frac{1-g^2}{(1+g^2-2g\mu)^{3/2}} \tag{4.13}$$

where $\mu = \cos\theta$ represents the scattering angle. This approach is numerically convenient when modelling the final Stokes I-only flux or when generating temperature distributions by radiative heating. In reality, every single scattering angle should be sampled from the $p(\mu)$ function so that

$$\int_{-1}^{+1} p(\mu)d\mu = 1. \tag{4.14}$$

This is particularly important when dealing with polarization of light and the full Stokes representation of a light packet, since scattering tends to change the polarization status of the incident

photon. However, when only the Stokes I light information is needed, i.e., the net flux produced from a model, modelling scattering events using the Henyey-Greenstein approach is good enough to deliver accurate scientific results. This parameter g is another byproduct of the BHMIE algorithm previously mentioned and is also dumped by the DustMixer as an extra column in the opacity table.

How to include scattering matrices for polarization

In some cases, one might be interested in studying the final polarization status of light generated by a modelled. This could be important when studying the effects of dust self-scattering (see chapter 3) or the effect of alignment of elongated grains. In such cases, full information of the light Stokes parameters is needed at the end of the simulation. In the specific scenario of modelling polarization by scattering, the Henyey-Greenstein scattering phase function is not appropriate because it does not provide information about the Q, U, and V stokes components, but only from the Stokes I intensity. In order for a radiative transfer code (as RADMC3D for the Synthesizer) to keep track of the polarization status of light during the Monte-Carlo modelling of scattering, it needs information about the *scattering matrix*. The scattering matrix Z_{ij} is the tensor that describes how will an incident photon $F = (F_\mathrm{I}, F_\mathrm{Q}, F_\mathrm{U}, F_\mathrm{V})^T$ transform into an outcoming photon $W = (W_\mathrm{I}, W_\mathrm{Q}, W_\mathrm{U}, W_\mathrm{V})^T$ after an scattering event. The outcoming wave W is a function of two scattering angles θ and ϕ and of the distance from the scattering particle, so its flux falls as $1/r^2$. This interaction can be described as

$$\begin{pmatrix} W_\mathrm{I} \\ W_\mathrm{Q} \\ W_\mathrm{U} \\ W_\mathrm{V} \end{pmatrix} = \frac{m_\mathrm{grain}}{r^2} \begin{pmatrix} Z_{11} & Z_{12} & Z_{13} & Z_{14} \\ Z_{21} & Z_{22} & Z_{23} & Z_{24} \\ Z_{31} & Z_{32} & Z_{33} & Z_{34} \\ Z_{41} & Z_{42} & Z_{43} & Z_{44} \end{pmatrix} \begin{pmatrix} F_\mathrm{I} \\ F_\mathrm{Q} \\ F_\mathrm{U} \\ F_\mathrm{V} \end{pmatrix} \tag{4.15}$$

with m_grain the mass of a spherical grain of radius a. For spherical grains or similarly for randomly oriented non-spherical grains, symmetry arguments (Bohren & Huffman 1983) reduce the amount components of the scattering matrix needed to describe the scattering event to only six, $Z_{11}, Z_{12}, Z_{22}, Z_{33}, Z_{34}$ and Z_{44}. In Mie theory, this even reduces to four relevant quantities because $Z_{22} = Z_{11}$ and $Z_{44} = Z_{33}$. These four components of the scattering matrix can be obtained by

$$Z_{11} = \frac{k^2 m_\mathrm{grain}}{2} \left(|S_2|^2 + S_1|^2 \right) \tag{4.16}$$

$$Z_{12} = \frac{k^2 m_\mathrm{grain}}{2} \left(|S_2|^2 - S_1|^2 \right) \tag{4.17}$$

$$Z_{33} = \frac{k^2 m_\mathrm{grain}}{2} \left(S_2^* S_1 + S_2 S_1^* \right) \tag{4.18}$$

$$Z_{34} = \frac{k^2 m_\mathrm{grain}}{2} \left(S_2^* S_1 - S_2 S_1^* \right) \tag{4.19}$$

where $k = 2\pi/\lambda$ and

$$S_1 = \sum_{n=1}^{\infty} \frac{2n+1}{n(n+1)}(a_n\pi_n + b_n\tau_n) \tag{4.20}$$

$$S_2 = \sum_{n=1}^{\infty} \frac{2n+1}{n(n+1)}(a_n\tau_n + b_n\pi_n) \tag{4.21}$$

$$\tag{4.22}$$

with

$$\pi_n(\theta) = \frac{P_n^1(\theta)}{\sin\theta} \tag{4.23}$$

$$\tau_n(\theta) = \frac{dP_n^1(\theta)}{d\theta} \tag{4.24}$$

$$\tag{4.25}$$

where $P_n^1(\theta)$ are the associated legendre polynomials. The S_1 and S_2 infinite sums are, as expected, another byproduct of the BHMIE algorithm along with the dust efficiencies and Henyey-Greenstein scattering parameter. They are similarly calculated for a specific grain size and then integrated over the size distribution. The phase function used in this case is

$$p(\mu) = k^2 m_{\text{grain}} \frac{Z_{11}(\cos\theta)}{\sigma_{\text{sca}}} \tag{4.26}$$

where

$$\sigma_{\text{sca}} = \int_0^{2\pi} Z_{11}(\theta,\phi)d\phi. \tag{4.27}$$

In the Synthesizer, adding the information about the scattering matrix to the dust opacity table is enabled with the `--polarization` option as

```
$ synthesizer --opacity --polarization
```

An additional parameter subject to customization is the number of angles used to sample the scattering matrix elements. This can be set with the option `--nang` and currently defaults to 181 angles. When calling RADMC3D for a polarized flux calculation, it will first make sure that the listed dust opacities κ_{sca} are consistent with the listed scattering matrix components Z_{11}. This test is passed when the difference between the angular integral of Z_{11} and the corresponding κ_{sca}, given by the condition

$$\kappa_{\text{sca}} \simeq 2\pi \int_{-1}^{+1} Z_{11}(\mu)d\mu, \tag{4.28}$$

is smaller than a predefined epsilon (1e-4).

4.3.3 Radiative Transfer (`--raytrace`)

The third module available within the Synthesizer is in charge of running the radiative transfer calculation for an already create density and dust model. This is accessed via the `--raytrace` option. The Synthesizer relies fully on RADMC3D to do the raytracing and Monte-Carlo and limits itself simply to the preparation of all the necessary input files and the post-processing of the results. Currently, only dust continuum radiative transfer is available within the Synthesizer. Future versions of the code will include support for molecular line modelling (see section 4.5).

In order to run a raytracing plus scattering monte-carlo calculation, one can simply use the default setup with the following command

```
$ synthesizer --raytrace
```

This example assumes that a *dust_temperature.dat* file has been created from the model via the `$ synthesizer ... --grid ... --temperature ...` command. If not dust temperature file is existent, RADMC3D will stop before starting the main calculations as it won't be able to create the initial radiation field. If dust temperature is not provided by the model's gas temperature, it can be easily created via Monte-Carlo radiative heating. In this mode, accessible simply by running

```
$ synthesizer --raytrace --monte-carlo
```

RADMC3D will first perform a thermal Monte-Carlo run by launching photon packets from a user-defined radiating source (central 4000 K protostar by default, customizable with the `--star` option) and heating the model until all photons are either absorbed or escape the grid. Once this temperature by radiation is created, it will proceed to do the raytracing. The Monte-Carlo heating step does not need to be repeated for every raytracing calculation. It is anyway recommended to test convergence of the Monte-Carlo heating by increasing the number of photons until no significant difference is seen in the temperature distribution. This distribution can be easily visualized on a new call with the command `$ synthesizer --temperature -show-grid-2d -show-grid-3d`.

Almost all RADMC3D parameters can be customized from the Synthesizer front-end (see Table 4.1 for details). For those parameters not explicitly defined, we provide an option called `--radmc`, meant to be followed by a sequence of command-line arguments to be passed to the standard RADMC3D call. In Fig. 4.7 is shown an example case of an SPH protostellar disk (the one shown in the gridding step in Fig. 4.5 but this time after the output of the radiative transfer step) observed at 1300 microns (default). This example show the disk's flux along with polarization vectors produced by dust alignment with the velocity field. Next to it is shown a map of optical depth also obtained using the Synthesizer. The optical depth map, obtained as $\tau = \int \rho_{\rm dust} \kappa_{\rm ext} dz$, with z the line-of-sight, used the extinction dust opacity $\kappa_{\rm ext}$ of an ensemble of grains grown up to $100\,\mu$m, which leads to a highly optically thick emission. The explicitly expanded command used to generate such images goes as follows

```
$ synthesizer --raytrace --tau --lam 1300 --incl 0 --npix 300 --sizeau 50 --
    show-rt --alignment
```

In Fig. 4.7 the colorscale represents the Stokes I flux from the disk model. Polarization vectors are generated from the information about the polarized emission, which can be obtained

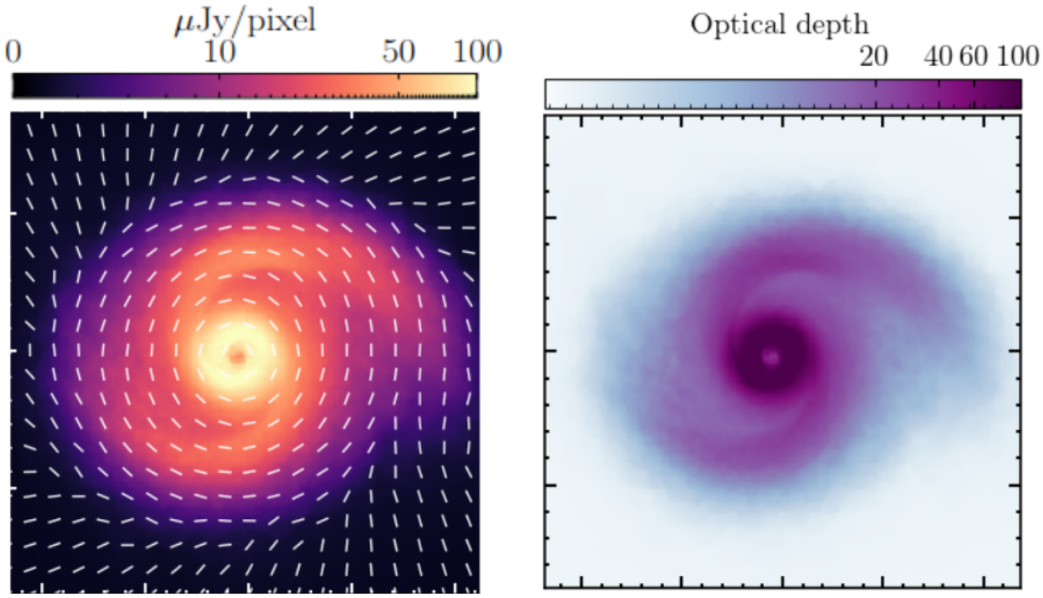


Figure 4.7: Example of results from the raytracing module within the Synthesizer. On the left is shown ideal synthetic flux map from a protostellar disk model, with polarization vectors overlaid, generated by dust alignment with the velocity field. On the right is shown a optical depth map for the same disk model, assuming grain sizes of up to 100 μm, which make the disk be extremely optically thick.

from the Q and U components as

$$P_{\rm I} = \sqrt{Q^2 + U^2}. \tag{4.29}$$

The vector angles are obtained from Q and U as well, as

$$P_{\rm angle} = \frac{1}{2} \arctan\left(\frac{U}{Q}\right). \tag{4.30}$$

The length of the vectors is commonly scaled by the polarization fraction. This is obtained as the ratio of the polarized intensity $P_{\rm I}$ over the total intensity $I \geq \sqrt{Q^2 + U^2 + V^2}$ as

$$P_{\rm frac} = \frac{P_{\rm I}}{I}. \tag{4.31}$$

In some cases, as in Fig. 4.7, when the polarization fraction spans a too large dynamic range within the image, it might be convenient to set all vectors to a constant length, in order to avoid empty and overcrowded regions. This is the default behaviour of the Synthesizer for the image displayed with the `--show-rt` option.

4.3.4 Synthetic Observation (`--synobs`)

The last module within the Synthesizer, and last step in the whole process of simulating observations (see Fig. 4.1) is the application of instrument and telescope-specific related effects.

Single-dish radio and optical/infrared observations

In the context of single-dish radio telescopes (e.g., APEX, IRAM 30-m, Green Bank Telescope, etc.) or single aperture optical/infrared telescopes (VLT, Herschel, Hubble, JWST, etc.), convolution with a beam and thermal noise addition are the two main effects needed to make a synthetic observation look as realistic as possible. The Synthesizer provides support for both of these techniques, however, only those applications for telescopes in the radio and sub-/millimeter range have been tested so far. The principle is basic and has been nicely detailed by Zamponi et al. (2022). As a first step, one can feed the Synthesizer the option `--resolution` followed by the size of a symmetric Gaussian beam in arcseconds. This will be used to convolve the map previously created with `--raytrace`. The convolution is done using the `convolve_fft` function from the Astropy Python Package (Astropy Collaboration et al. 2022). This function needs the standard deviation σ of a Gaussian profile to create a 2D Gaussian kernel, and performs the convolution in Fourier space in favour of speed for large images ($\gtrsim$50 pixels per side). This standard deviation can be obtained from the entered resolution θ as

$$\sigma = \frac{\theta}{\sqrt{8 \ln 2}}. \tag{4.32}$$

After convolution, the original flux F_{unconv} (in units of Jy/pixel, for radio observations) needs to be rescaled to a beam-dependent flux F_{conv}, multiplying by the ratio of the area of a Gaussian beam over the area of a square pixel as

$$\frac{F_{\mathrm{conv}}}{\mathrm{Jy\,pixel^{-1}}} = \frac{\pi}{4 \ln 2} \frac{\theta^2}{\mathrm{arcseconds}} \left(\frac{\mathrm{pixelsize}}{\mathrm{arcsec}} \right)^{-2} \frac{F_{\mathrm{unconv}}}{\mathrm{Jy\,pixel^{-1}}}. \tag{4.33}$$

The second step is to add noise to our ideal images based on the telescope's receiver temperature. Within the Synthesizer this is currently only implemented for radio single-dish telescopes using the radiometer equation

$$T_{\mathrm{rms}} = \frac{T_{\mathrm{sys}}}{\Delta \nu t_{\mathrm{int}}}, \tag{4.34}$$

where T_{rms} is the observation noise to be added to the new convolved image, in units of K, T_{sys} is the system temperature, $\Delta\nu$ is the spectral bandwidth to be considered (or channel resolution in case of line observations) and t_{int} is the time on source (provided to the Synthesizer via the `--obs-time` option). The system temperature T_{sys} for a single-dish measurement, is given by the following equation

$$T_{\mathrm{sys}} = T_{\mathrm{A}} + \left(T_{\mathrm{rec}} + T_{\mathrm{atm}} \left[1 - \exp\left(\frac{-\tau_0}{\sin(El)} \right) \right] \right) \exp\left(\frac{-\tau_0}{\sin(El)} \right), \tag{4.35}$$

where T_{A} is the antenna temperature, T_{rec} is the receiver temperature characterized by the Friis equation

$$T_{\mathrm{rec}} = T_1 + \frac{T_2}{G_1} + \frac{T_3}{G_2 G_3} + \cdots,$$

with $1, 2, ..., n$ identifiers of the ordered electronic components in the receiver and G the gain of each of them, T_{atm} is the integrated physical atmospheric temperature, τ_0 is the atmospheric

zenith opacity and *El* is the elevation of the source. Because of $T_{\rm sys}$ being highly telescope specific, different radio telescopes offer their own system temperature calculators. Future versions of the Synthesizer will include either either core implementation of the system temperature calculation for most common telescope receivers or the option to provide such values as command line arguments.

Sub-/millimeter inteferometric observations

Simulating the antenna response and image reconstruction for a full interferometric array is a quite challenging task, both in terms of computational time and data manipulation. Luckily, this option is already provided by CASA. In CASA, inteferometric (and also single-dish) synthetic observations are split into two main steps: first, the generation of the complex visibilities for a specific sky model, and second, the deconvolution of the dirty image generated by the convolution of the real sky intensity distribution and the finite interferometric sampling (the visibilities). The first step is calculated using the CASA task `simobserve`. This function receives parameters related to the observing setup, such as, the sky coordinates of the synthetic model, the observing frequency and bandwidth, the time per scan and the time on source, the observing mode (interferometer or single-dish), the antenna configuration and customization of the parameters related to the generation of thermal noise, as in equation 4.35. An example call to the simobserve task goes as follows

```
simobserve(
      project = 'band6',
      skymodel = 'radmc3d_I.fits',
      incenter = '223GHz',
      inwidth = '0.12GHz',
      setpointings = True,
      integration = '2s',
      totaltime = '1.25h',
      indirection = 'J2000 16h32m22.63 -24d28m31.8',
      hourangle = 'transit',
      obsmode = 'int',
      antennalist = 'alma.cycle4.7.cfg',
      thermalnoise = 'tsys-manual',
      graphics = 'both',
      overwrite = True,
      verbose = True
)
```

When calling the Synthesizer with the `--synobs` option and no external script has been specified, it generates a minimal template CASA script called *casa_script.py*. This script serves as a starting point for the user to create a more customized version. This new version can then be feed to the Synthesizer on a new call via

```
$ synthesizer --synobs --lam 3000 --script my_new_script.py --show-synobs
```

for a 3000 microns observation example. Some parameters, such as the wavelength, need to be explicitly reentered since the Synthesizer does not trace history of previous calls. Otherwise it will always default to 1300 microns and will mismatch the already existing project, stored

in *synobs_data*. The Synthesizer includes also few more templates tailored to real ALMA and JVLA observations and previously used by Zamponi et al. (2021) and Zamponi et al. (Submitted), also available within the code distribution[7]. To use these scripts instead of the default minimal one, add the option `--use-template`. Given the simobserve argument `graphics = "file"` or `graphics = "both"`, simobserve will generate and optionally display two diagnostic plots during runtime. The first one, shown in the upper row of Fig. 4.8, shows what is the needed beam coverage of the observation given the size of the model and the telescope's field-of-view. In some cases only one pointing is needed, some other time a mosaic must be done. This is automatically detected and performed by simobserve. The second plot, upper right panel of Fig. 4.8, shows information about the source elevation (given the observing site and source celestial coordinates), the array configuration as seen by the source, the *uv*-coverage and the Point-Spread-Function (PSF). Once this step is done, the Synthesizer will create a folder called *synobs_data* where all these figures and the measurement set will be stored.

The second step is performed by calling the CASA task `tclean`. This function receives parameters related to the cleaning process of a dirty image, such as, the image size in pixels, the pixel size in arcseconds, the reference frequency (to match that from simobserve), the spectral mode (mfs for continuum, cube for line emission), the type of gridder used for the Fourier Transform of the visibilities, the type of deconvolver, type of visibility weighting and the cleaning threshold. An example call to tclean is included in the previously mentioned templates and reads as follows

```
tclean(
    vis = 'synobs_data/synobs_data.alma.cycle4.7.noisy.ms',
    imagename = 'synobs_data/clean_I',
    imsize = 400,
    cell = '0.008arcsec',
    reffreq = '223GHz',
    specmode = 'mfs',
    gridder = 'standard',
    deconvolver = 'multiscale',
    scales = [1, 8, 24],
    weighting = 'briggs',
    uvrange = '120~2670klambda',
    robust = 0.0,
    niter = 10000,
    threshold = '2.5e-4Jy',
    mask = 'synobs_data/synobs_data.alma.cycle4.7.skymodel',
    interactive = False,
    verbose = True
)
```

This step does not generate diagnostic plots on the run, but it does save several byproducts of the deconvolution to file. All these files are saved within the project folder *synobs_data*. Some of them are: the final cleaned image (shown in the lower left panel of Fig. 4.8) called *clean_I.image* and also saved to FITS file within the parent directory as *synobs_I.fits*, a residual map `clean_I.residual` (see lower right panel of Fig. 4.8), a file containing the cleaning mask

[7] `https://github.com/jzamponi/synthesizer/tree/main/synthesizer/synobs/templates`

used *clean_I.mask*, a file containing the telescope's Primary Beam *clean_I.pb* and the primary beam corrected final image *clean_I.image_pbcor*. As mentioned, *synobs_I.fits* is the end result of the whole synthetic observation process illustrated in Fig. 4.1. This FITS image is the science ready model users can do for their own scientific purposes, being certain that it accounts for the proper resolution and sensitivity treatment when comparing high-resolution numerical simulations to real observations.

4.4 Science cases

This chapter would not be complete without a proper demonstration that the Synthesizer can deliver publication-ready results. The code started as a collection of previous slightly versatile scripts, highly specific for certain codes and setups. Earlier versions of such scripts have been used in previous works, such as Zamponi et al. (2021) and Zamponi et al. (2022), before coming into shape as a standalone Python package. The Synthesizer came to birth as the need to easily automate various tedious and isolated manual tasks and therefore its philosophy is to make the process of modelling observations more manageable for future users, letting them focus their time on the science and not on the file and code formatting. Below I list a couple of projects in which the Synthesizer has been used, regardless of their status as published, submitted, in preparation or future work. All the following figures and results are included here with full consent of the corresponding owners.

Polarization by self-scattering

The first and biggest scientific case for which the code has been used is that described in full in chapter 3 and Zamponi et al. (Submitted). Because of having a full chapter dedicated to this only, I refrain myself from elaborating on the details of the project and refer the reader to sections 3.3.2, 3.3.5 and 3.4 for further details. In summary, I have used the Synthesizer to model the polarized emission produced by dust self-scattering in the Class 0 protostellar disk IRAS 16293-2422 B, using a radiation-hydrodynamics simulation of protostellar disk formation as a physical model. The disk model is hot, with central (<10 au) temperatures above 400 K, and massive (0.3 $M_\odot$). Previous ALMA and JVLA observations (Sadavoy et al. 2018 and Liu et al. 2018a, respectively) have suggested that the observed azimuthal polarization pattern could be produced by the dust scattering of the dust thermal emission itself. If this is the case, being the polarization fraction (2-4%) similar between both 1.3 and 7 mm observed wavelengths, then dust grains must have grown to at least 200 or even up to 2000 microns in order to produce polarization by self-scattering detectable at both millimeter wavelengths. I employed the Synthesizer to automate the modelling of several parameters involved in the self-scattering polarization and to study their parameter space, aiming to assess the scenario of dust being grown to millimeter sizes in a Class 0 disk. The results of the modelling indicate that regardless of the dust composition and size, the disk model used (under the assumption that it properly represents the disk in IRAS 16293 B) is too massive and therefore too optically thick (τ>100) to produce polarization fractions by self-scattering of around 2-4% at both 1.3 and 7 mm. The maximum polarization fraction ob-

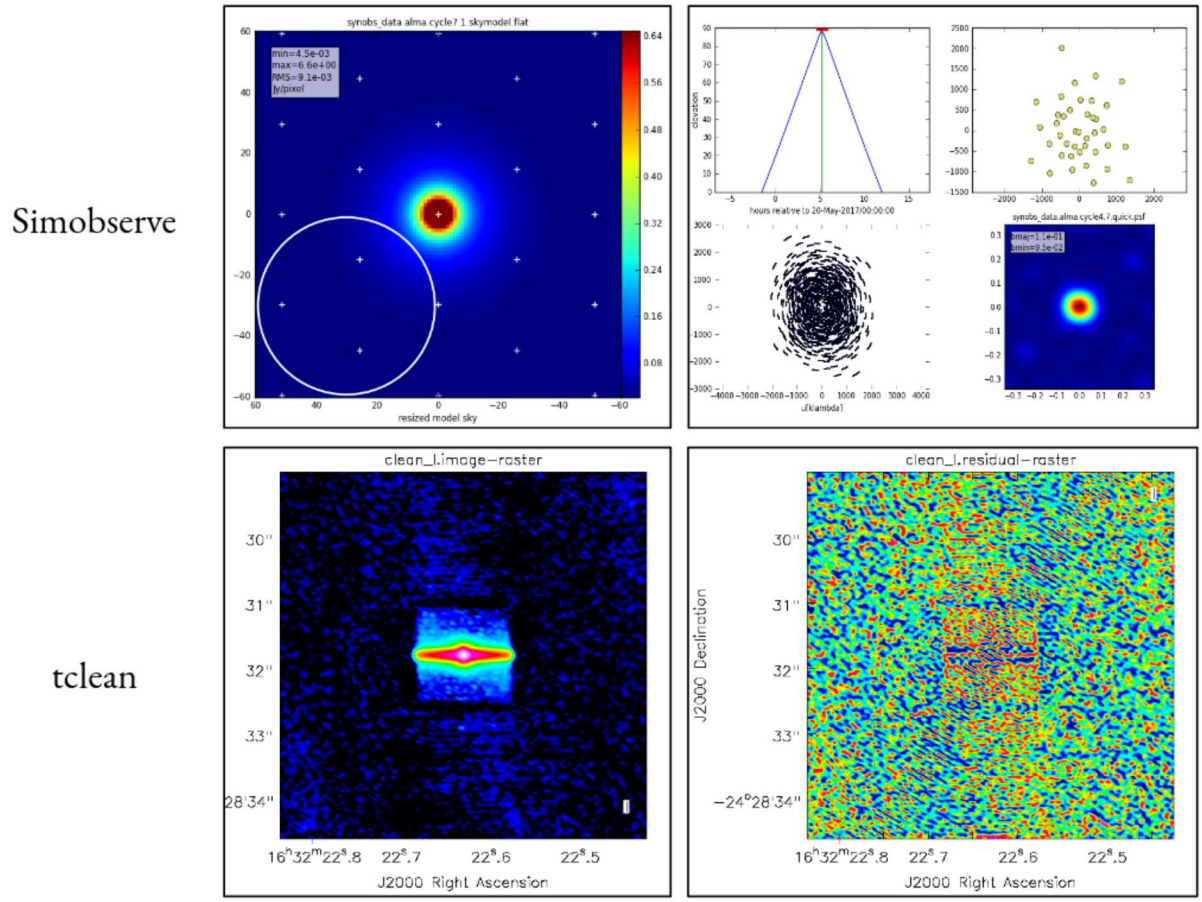


Figure 4.8: Diagnostic plots of the two main steps involved in a interferometric synthetic observation: simobserve (top row) and tclean (bottom clean). The first (upper left) panel is displayed at the beginning of execution, to show what's the sky coverage needed given the model size and the field of view. Second panel (top right) show information about the source elevation, antenna array, *uv*-coverage and PSF. Third (bottom left) panel shows the end result of tclean, i.e., the cleaned map, using a default template cleaning mask. The fourth (bottom left) panel show the residuals of the cleaning process.

tained from the models comes from a dust population of 100 μm sized silicate-graphite grains. However, this maximum fraction reaches values no larger than 0.2% from the output of the ideal intensity models (from the radiative transfer code). After feeding this model to the CASA synthetic observation module, mimicking the corresponding ALMA and JVLA observing setups, no polarized intensity is detected above the corresponding sensitivity levels. This means, that based on our results, self-scattering is unlikely to be the responsible mechanism for the observed polarization towards IRAS 16293 B. Future projects, see below, should focus on studying the polarization produced by dust alignment with either the underlying velocity, radiation or magnetic field in order to further characterize the polarization observed.

AREPO Protostellar Core

The Synthesizer has also been used to post-process magneto-hydrodynamic simulations of protostellar core formation, using the code AREPO. These simulations are performed by a PhD student at MPA, Alex Mayer, on the aim to test the newly-created interface between the Synthesizer and AREPO snapshots. The example shown serves more as a testing box, or a proof-of-concept than a science case. No further science related to the observability of modelled phenomena has been done with this simulation, however, this certainly opens a new window for AREPO users to easily couple their simulations with observations at no extra cost. All of these Synthesizer commands can be added at the end of the simulation post-processing scripts. This comes particularly handy when generating predictions for observational proposals. All intermediate results from the gridding of the particle/cell-center locations, the 2D plotting and 3D rendering, the raytracing and the final ALMA observations are shown in Fig. 4.9, and can be reproduced with the following command

```
$ synthesizer --grid --source arepo --sphfile snap_070.hdf5 --temperature --
    show-grid-2d --show-grid-3d
```

Magnetic field morphologies in the high-mass star-forming region G31.41

The Synthesizer is also currently being used to characterize the morphology of the magnetic field in the high-mass star forming clump G31.41+031 (see Fig. 4.10). This source is one of the first high-mass star forming regions suggested to be threaded by an hourglass magnetic field, shown by early SMA 0.89 mm observations (Girart et al. 2009). However, this observed hourglass field seem to be completely misaligned from the velocity axis, by approximately 90° (Cesaroni et al. 2011 and Moscadelli et al. 2013). More recent ALMA 1.3 mm observations (Beltran et al. 2019) support the same scenario but their vector angles mismatch those observed at 7 mm JVLA observations (Sia et al., priv. comm.). The magnetic field derived from the 7 mm polarization vectors is indeed parallel to the outflow orientation and has a much larger spatial coverage than the ALMA observation, where the vector angles remain almost uniform even up to 0.4 pc. The ALMA observations only cover the central 0.1 pc, where the vector flip occurs. The aim of the project is to understand why do polarization vectors flip within the central 0.1 pc when observed at 1.3 versus 7 mm. One possible scenario (Siu et al. in prep.) is that the central 0.1 pc, when targeted by ALMA, have a higher optical depth than the larger 0.4 pc scales observed at 7 mm,

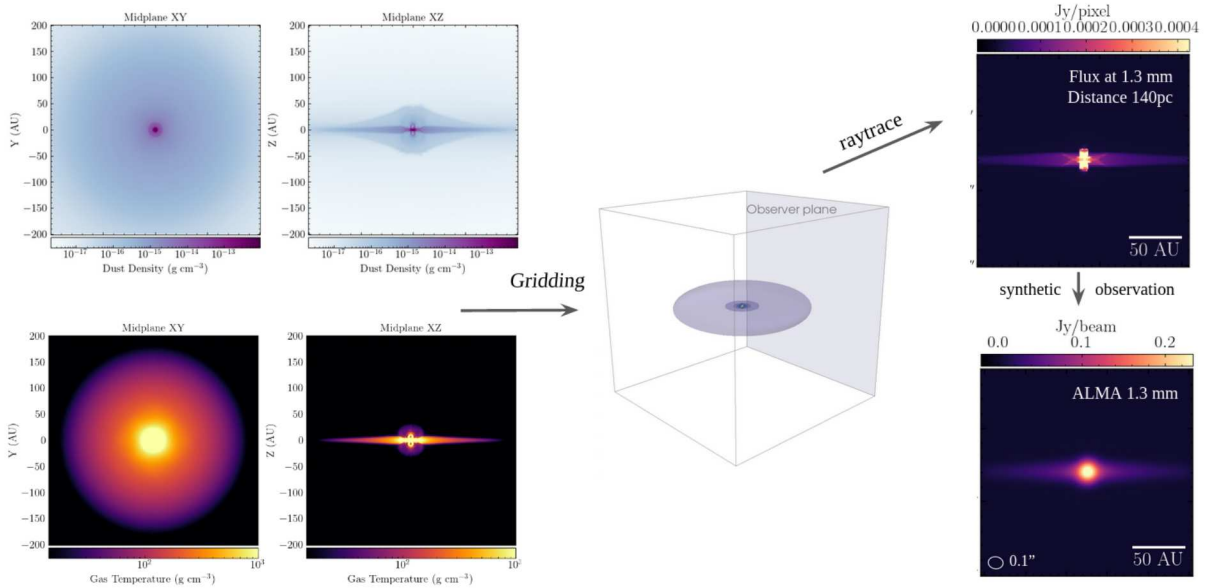


Figure 4.9: AREPO simulation of protostellar core formation post-processed using the Synthesizer. All the intermediate results included in the synthetic observations are shown here for illustration. The magneto-hydrodynamic simulations are performed by Alex Mayer (PhD student at MPA) using the code AREPO and reproduced here with permission from the author.

changing from optically thin to optically thick emission. The polarized emission coming from an optically thick medium can actually be produced by a different phenomenon than the commonly assumed case of emission from aligned grain. The latter assumes that elongated grains align their major axis perpendicular to the magnetic field and produce E-vectors perpendicular to the field lines. A 90° rotation is then assumed to derive the B-field vectors, but the picture changes if the polarization is produced in the form of absorption instead of emission. This can happen towards optically thick regions that reduce the net polarization of light and emit unpolarized light. This light is then absorbed by foreground envelope grains preferentially along the grain's long axis, letting polarized light pass through parallel to the field lines, and potentially explaining the central outflow-magnetic field misalignment. In order to test this scenario, Siu et al. (in prep.) is using the Synthesizer with a power-law density model and an hourglass field (both included in the code), to simulate the 1.3 and 7 mm observations (see Fig. 4.10).

4.4.1 Data visualization

For users not interested in generating synthetic observations, the Synthesizer can also be used as a standalone visualization tool. It relies on the Python libraries Matplotlib and Mayavi for the 2D and 3D plotting, respectively. It has already been mentioned how to tell the code to display intermediate plots between all the modules. Here, we describe how can the code be used without the need to run any of the of the main modules --grid, --raytrace, --opacity or --synobs again. If a model has been created, but no display option has been given, plotting options alone can

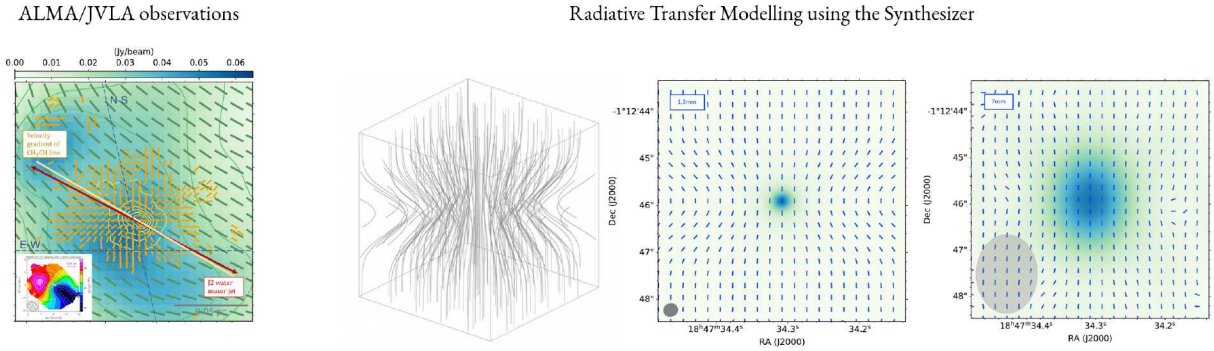


Figure 4.10: *(left)* ALMA/JVLA polarization observations of G31.41+031 at 1.3 and 7 mm. *(right)* Polarized radiative transfer modelling from analytical models using the Synthesizer. Reproduced with permission from Siu et al. (in prep.)

be provided in a new call and the Synthesizer will read from the lastly-created grid, FITS or datatable files. This applies for all `--show-*` options.

Midplane slices and 3D rendering of already existing models

All model grids, regardless of them being analytical prescriptions or full hydro simulations, can be visualized using the Synthesizer without the need to run all modules again. The visualization options allow to also open previously created grid files, such as, *dust_density.inp* and *dust_temperature.inp* without having to recalculate them, as

```
$ synthesizer --show-grid-2d --show-grid-3d
```

Optically thick $\tau = 1$ surface

The Synthesizer can also be used to visualize the optically thick $\tau = 1$ surface of your model, if it contains one. An example of this feature is shown in Fig. 4.11 for a massive SPH protostellar disk, as the one shown in Figs. 4.5 and 4.7. This is particularly useful when studying optically thick physical models that don't seem to show the embedded features we expect to see. When dealing with non-homogeneous optically thick models, it is important to understand the depth of the $\tau = 1$ surface along the line-of-sight because structures beyond this surface are in principle present but are not visible. Visualization of this surface can be done for any already created grid model, without the need to run any of the main modules, as

```
$ synthesizer --show-grid-3d --tau
```

4.5 Current and future code features

Below I summarize all the features currently implemented and tested in the code. I also provide a list of features currently implemented but in their beta status, i.e., lacking robust testing and

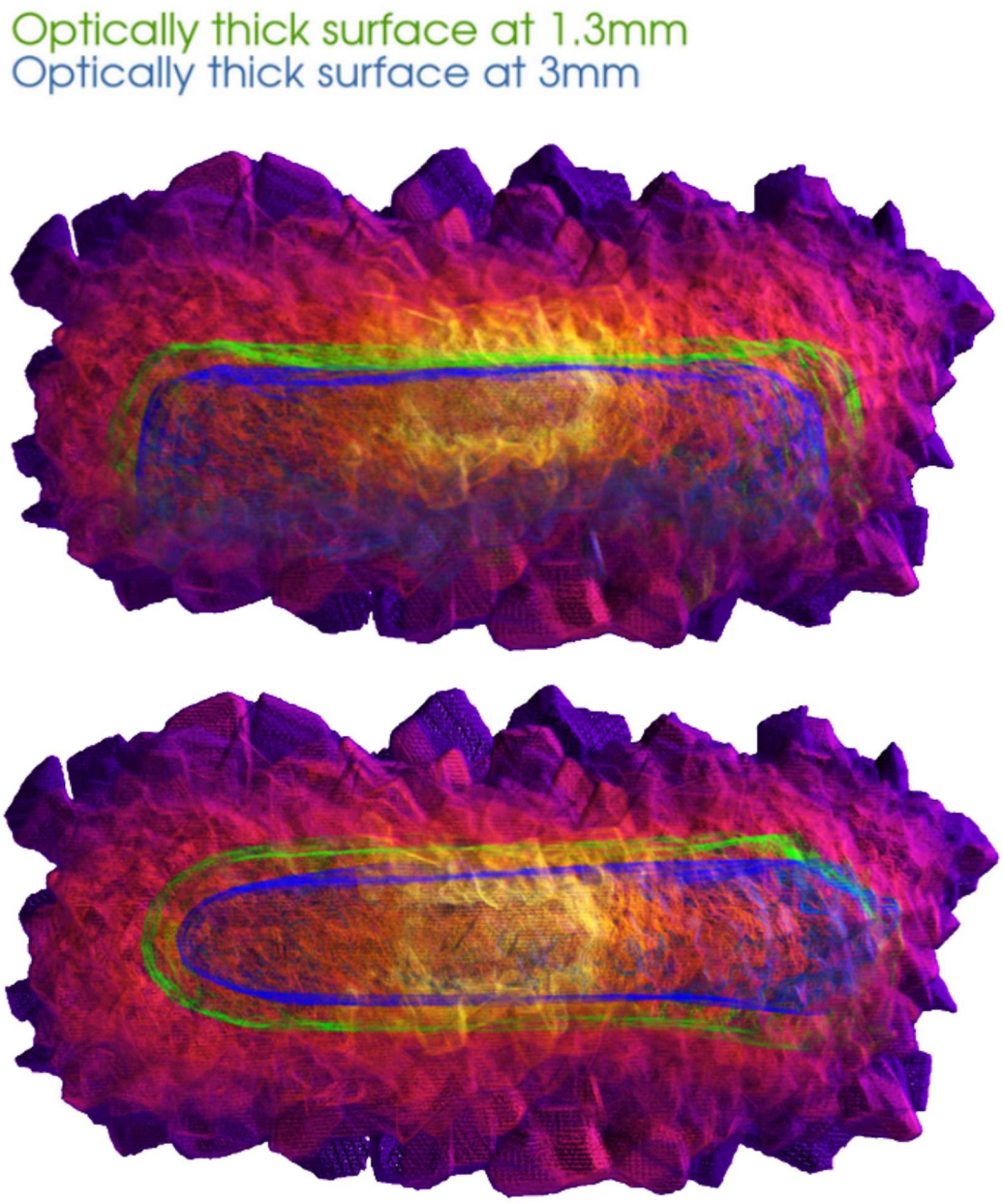


Figure 4.11: 3D rendering of the dust temperature and $\tau = 1$ surfaces along a face-on (top) and edge-on (bottom) projections for an SPH protostellar disk. This is one of the visualization modes obtained with the options `--tau` and `--show-grid-3d`.

problem catching. After that, I provide a list of features planned to be implemented in the future, that will make the code suitable for a wider range of science cases. At the end of this section, Table 4.1 lists all the possible command line arguments along with a short description, similar to what can be obtained with the command `$ synthesizer --help`.

Currently implemented features:

- Generation of monochromatic synthetic continuum images
- Generation of synthetic images with polarization by scattering
- Generation of synthetic images with polarization by dust alignment
- Gridding of particle locations and physical quantities onto a Cartesian regular gridding
- 2D plotting of the vector field stream lines from both analytical prescriptions and entered simulation vector fields
- 3D visualization of simulation outputs and analytical models
- Visualization of the $\tau = 1$ surface within the model, for optically thick models.
- Calculation of dust opacity tables for different grain sizes and materials, including full scattering matrices for polarization and alignment efficiencies.
- Support for realistic observations mimicking currently existing single-dish radio telescopes
- Support for realistic radio interferometric observations, with the option to vary antenna array configurations to achieve different angular resolutions

Current features in beta:

- Visualization of the vector field streamlines within the 3D rendered scene
- Creation of user-defined analytical models and vector fields
- Automatic detection of an appropriate antenna array configuration that would deliver a user-desired angular resolution
- Interfaces for simulation outputs from grid-based codes.
- Option to use spherical grids instead of the default regular cartesian grids.
- Mixing of optical constants (Bruggeman or Maxwell-Garnett Mixing).

Future implementations:

- Addition of new interfaces for different commonly-used hydro grid-based codes, such as, RAMSES, ENZO, ATHENA++, FLASH.

- Addition of new interfaces for different commonly-used hydro particle-based codes, such as, PHANTOM and GRADSPH.
- Addition of support for molecular line observations
- Addition of more tailored analytical models for most disks from the DSHARP survey along with PDS-70.
- Addition of toy analytical models used to test polarization, as those from Kataoka et al. (2015)
- Addition of ready-made common dust mixtures, such as DSHARP (Birnstiel et al. 2018) and DIANA (Woitke et al. 2016).
- Addition of support for telescopes outside of the sub-/millimeter range, such as, Herschel, Hubble, JWST and VLT instruments.

Option	Description
-h, --help	Show this help message and exit
-g, --grid	Create an input grid for the radiative transfer
--model MODEL	Keyword for a predefined density model.
--sphfile SPHFILE	Name of the input SPH file (snapshot from a particle-based code).
--amrfile AMRFILE	Name of the input AMR grid file (snapshot from a grid-based code).
--source CODENAME	Name of the code used to generate the input file.
--ncells NCELLS	Number of cells in every direction.
--bbox BBOX	Size of the half-length of a bounding box in au.
--rout ROUT	Size of the outer radial boundary in au (i.e., zoom in).
--g2d G2D	Set the gas-to-dust mass ratio.
--vector-field FIELD	Create a vector field for alignment of elongated grains.
--temperature	Write the dust temperature from the model.
--show-particles	Render the SPH particle positions weighted by density.
--show-grid-2d	Plot the midplane of the newly created grid.
--show-grid-3d	Render the new Cartesian grid in 3D.
--vtk	Call RADCM3D to create a VTK file of the newly created grid.
--render	Visualize the VTK file using ParaView.
-op, --opacity	Call dustmixer to generate a dust opacity table.
--material MATERIAL	Dust optical constants. Can be a predefined key, a path or a URL.
--amin AMIN	Minimum value for the grain size distribution.
--amax AMAX	Maximum value for the grain size distribution.
--na NA	Number of size bins for the logarithmic grain size distribution.
--q Q	Slope of the grain size distribution in logspace.
--nang NANG	Number of scattering angles used to sample the dust efficiencies.
--show-opacity	Plot the resulting dust opacities.
--show-nk	Plot the input dust optical constants.
--nopb	Disable printing of an opacity progress bar. Useful when logging to file.
-mc, --monte-carlo	Call RADMC3D to raytrace the new grid and plot an image.
--nphot NPHOT	Set the number of photons for scattering and thermal Monte Carlo.
--nthreads NTHREADS	Number of threads used.
-rt, --raytrace	Call RADMC3D to raytrace the new grid and plot an image
--lam LAM	Wavelength used to generate an image in units of micron
--lmin LMIN	Lower end of the wavelength grid in microns.
--lmax LMAX	Upper end of the wavelength grid in microns.
--nlam NLAM	Number of wavelengths to build a logarithmically spaced grid.
--npix NPIX	Number of pixels per side of new image
--incl INCL	Inclination angle of the grid in degrees
--phi PHI	Inclination angle over a second axis in degrees
--sizeau SIZEAU	Physical size of the image in AU

Table 4.1: List of all possible command line arguments to be passed to the synthesizer, along with a minimal description. This table is generated from the output of the command `$ synthesizer --help`.

--distance DISTANCE	Physical distance in pc, used to convert fluxes into Jy/pixel.
--star x y z Rstar Mstar Teff	6 parameters used to define a radiating star (values should be given in cgs)
--tau	Generate a 2D optical depth map.
--tau-surf tau	Generate a 3D surface at optical depth = tau.
--show-tau-surf	Render the 3D the optical depth = tau surface.
--polarization	Enable polarized RT and full scattering matrix opacity tables
--alignment	Enable polarized RT of thermal emission from aligned grains.
--noscat	Turn off the addition of scattered flux to the thermal flux
--sublimation SUBLIMATION	Percentage of refractory carbon that evaporates from the dust
--sootline SOOTLINE	Temperature at which carbon is supposed to sublimate
--dust-growth	Enable dust growth within the soot-line
--show-rt	Plot the intensity map generated by ray-tracing
--radmc3d [RADMC3D ...]	Additional commands to be passed to RADMC3D.
-so, --synobs	Call CASA to run a synthetic observation from the new image
--dont-observe	Skips CASA's Simobserve task.
--dont-clean	Skips CASA's tclean task to clean and image simobserve's output.
--dont-export	Don't export tclean's output as a fits image.
--obs-time OBS_TIME	Set the observing time in hours. Default is 1h.
--resolution RESOLUTION	Set a desired angular resolution in arcseconds.
--obsmode int,sd	Whether to observe with a radio interferometer or a single-dish.
--telescope {alma,aca,vla,sma, noema,atca,meerkat,vlba}	Radio telescope to use. Default: ALMA
--script SCRIPT	Name, path or url to a CASA script for the synthetic observations.
--use-template	Use one of the template CASA scripts contained within the synthesizer.
--show-synobs	Plot the ALMA/JVLA synthetic image generated by CASA.
-ow, --overwrite	Overwrite opacities, RADMC3D and CASA input files.
--quiet	Disable verbosity. Do not output anything.
--version	Show program's version number (v0.0.3) and exit.

Table 4.2: Continuation of Table 4.1.