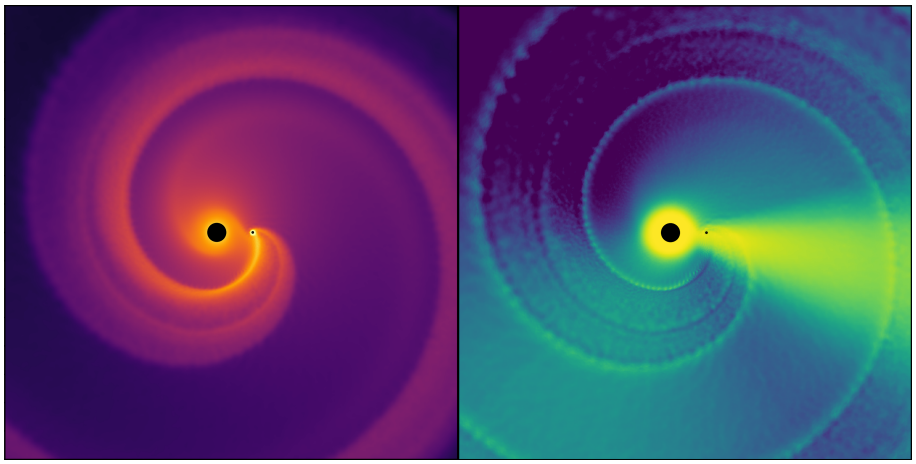
KU LEUVEN

FACULTY OF SCIENCES

SPH Approach to Modelling Dust Attenuation of the Wind Acceleration in AGB Binaries



Mats ESSELDEURS

Supervisor: Dr. W. Homan
Institut d'Astronomie et d'Astrophysique, ULB
Instituut voor Sterrenkunde, KU Leuven

Co-supervisor: Dr. F. De Ceuster
Instituut voor Sterrenkunde, KU Leuven

Co-supervisor: Dr. L. Siess
Institut d'Astronomie et d'Astrophysique, ULB

Master thesis submitted in
fulfillment of the requirements
for the degree in Master of Science
in Astronomy & Astrophysics

Academic year 2021-2022

Acknowledgements

After five years of studying physics and astronomy, there are no regrets. I have been able to experience many years, build strong friendships that I will never forget. It is hard to name everyone but I would like to tank my housemates, William, Pili and Orjan, in particular for the wonderful years and the support I have received from you this past year. In addition, Babet, I also want to thank you for all the support and the pressure relief you were able to give in the last, stressful thesis year.

The past year of this I was able to make my thesis in collaboration with great people. First and foremost I would like to thank my promoter Ward for all the great meetings, all the enthusiasm, and pleasant feedback about all my work. Especially for the manuscript, your help was extremely valuable. Besides Ward, I would also like to thank my co-promoters Frederik and Lionel, because they also contributed to these fun and productive meetings, and improved this thesis to the level I present to you here. I would also like to thank Silke and Jolien for their valuable insights into my thesis, as well as being a point of contact within team L.E.E.N.

I would also like to give a huge thank you to my parents, as well as, Karolien and Tom. Thank you for allowing me to walk this path, with all your support and trust. I could never have been able to stand here without you! Also an extra thank you to you Karolien for all the texts you have read and improved upon over the past years.

Abstract

Circumstellar envelopes of asymptotic giant branch (AGB) stars have for a long time been modelled assuming a spherical symmetry. However, recent observations of the inner winds of these stars have shown that they exhibit a variety of complex structures, such as spirals, equatorial density enhancements, disks, bipolar outflows, etc. These structures are believed to originate from the presence of a companion, either stellar and/or planetary. Due to the inherently three-dimensional (3D) nature of this phenomenon, its investigation requires advanced 3D-simulations. It has already been shown that using hydrodynamic simulations, some of these wind morphologies can be obtained. However, the computational cost of truly self-consistent calculations, including the crucial chemical and radiation processes, is currently still computationally prohibitive. Therefore, incremental modelling improvements using ever more refined approximations can significantly advance the quality and physical consistency of the simulations. One such problem revolves around the transfer of momentum from the stellar photons to the dust particles in the AGB wind. Until recently, this coupling has always either been ignored, or treated in the optically thin limit, which only requires knowledge on the local quantities. However, properly accounting for the dust opacity and its attenuation of the stellar radiation field can drastically affect the effective radiation pressure on the dust. In turn, this can significantly alter the dynamics and morphology of these stellar winds.

In the context of a large collaboration that revolves around upgrading the treatment of dusty winds within the smoothed-particle-hydrodynamics code `PHANTOM`, I have worked on levelling up the way in which dust acceleration is calculated. To calculate the attenuation of the stellar radiation field, knowledge on the distribution and optical properties of the matter in-between each particle and the star is required. To this end the ray-tracing algorithm of the radiative transfer code `MAGRITTE` was extracted, made compatible with the SPH philosophy, and coupled to `PHANTOM`. We investigated different options to speed up the ray-tracer with only a minimal loss in accuracy. We show that the best results are found when rays are traced outwards from the star in a uniform distribution, set by `HEALPix`. Because not all particles are struck by a ray, we also investigated different interpolation approaches. We find that interpolation scaling with the inverse square of the perpendicular distance to the four closest rays gives the most desirable result. Finally, we demonstrate the validity of our new approach in a simulation.

Laymen Abstract

While stars with masses similar to our sun age, they evolve through different evolutionary stages. One of these stages, just before the end of their lifetime is the asymptotic giant branch phase, a phase which most of the stars in the night sky go through. They grow to immense sizes, to scales about as large as the orbit of our own planet. Therefore the gas in the star is loosely bound, and the outer layers are blown into space (a so called stellar wind). This happens due to the force that the stellar light exerts on tiny soot- or sand-like particles that form in these fluffy outer layers. For a long time these AGB stellar winds were taught to be spherical in shape, however new recent images show more complex structures like spirals and disks. One of the explanations for these complex structures uses the presence of an other star or planet (a companion) to shape the wind in these structures. It is in this study this thesis aims to improve our understanding on the formation of these structures.

The aim of this thesis is to account for the fading of light. Fading means that when light goes through zones with a lot of matter, gradually diminishes in intensity, much like a receding flashlight in a thick mist. This is important because if light is faded, the force pushing the material in the wind is reduced, or might even fall away completely.

The calculation of the fading of light is complicated, as it requires information on all the material in-between the location where one wants to calculate this radiation force, and the bright star. We show that the most efficient way to calculate this fading is by drawing lines starting at the star, going outwards into space. And only along these lines the fading is actually calculated. For all the other matter around the star that is not does not touch a line, we find the four closest rays and use those to try to predict the fading in-between. Doing so makes the calculation fast enough to be performed in sync with fluid-motion simulations (a digital way to predict the flow of gases and fluids subject to certain conditions like boundaries and gravity), which is crucial to understand the influence of a companion in the wind structure of asymptotic giant branch stars.

List of Abbreviations

Acronyms

AGB	Asymptotic Giant Branch
AMR	Adaptive Mesh Refinement
BHL	Bondi-Hoyle-Lyttleton (BHL) accretion
CFL	Courant-Friedrich-Lews (CFL) condition
COM	Center Of Mass
CSE	CircumStellar Environment
HB	Horizontal Branch
HR	Hertzsprung-Russell (HR) diagram
IRS	Inwards Ray-tracing Scheme
ISM	Interstellar Medium
LTE	Local Thermodynamic Equilibrium
MS	Main Sequence
PN	Planetary Nebula
post-AGB	post-Asymptotic Giant Branch
RGB	Red Giant Branch
RLOF	Roche Lobe OverFlow
SPH	Smoothed Particle Hydrodynamics
WLOF	Wind Roche Lobe OverFlow

Symbols

M_*	stellar mass

L_*	stellar luminosity
$\dot{M}$	mass loss rate
R_d	dust condensation radius
e	eccentricity
ρ	density
T	temperature
a	semi-major axis
M_1	primary mass
M_2	secondary mass
h	smoothing length
W	smoothing kernel
Λ	energy cooling term
γ	polytropic index
μ	mean molecular weight
F	Additional long-range forces
α	artificial wind launching factor
Γ_d	optically thin Eddington factor for dust
κ	opacity
τ	optical depth
o	`HEALPix` order

Units

au	astronomical unit: 1 au $= 1.496 \cdot 10^{11}$ m
$\mathsf{M}_\odot$	solar mass: 1 $\mathsf{M}_\odot = 1.989 \cdot 10^{30}$ kg
$\mathsf{R}_\odot$	solar radius: 1 $\mathsf{R}_\odot = 6.957 \cdot 10^{8}$ m
$\mathsf{L}_\odot$	solar luminosity: 1 $\mathsf{L}_\odot = 3.828 \cdot 10^{26}$ Watt

List of Figures

Contents

Chapter 1

Introduction

Asymptotic giant branch (AGB) stars are a late evolutionary stage of low- and intermediate-mass stars. These stars are characterised by an enormous mass loss, shaping their environment. In this thesis, I refine the treatment of the radiation pressure driving these winds to the hydrodynamic modelling of these outflows in a binary system. Before we wish to understand how binary interaction can influence the wind morphology of this outflow, we first need to understand how a normal unperturbed system behaves. For this reason the stellar evolution of a single star is discussed first (Section 1.1), then the mechanism of the outflow. The dust driven wind is explained in more detail in Section 1.2. Next some observations of these outflows are discussed in Section 1.3 to then explain the influence of a binary companion in Section 1.4. Going further to a small introduction on how these systems can be numerically modelled in Section 1.5. This chapter is concluded with the motivation and outline of this thesis (Section 1.6).

1.1 Single star evolution

As AGB stars are an evolutionary stage of low- and intermediate-mass stars ($0.8\ M_\odot \lesssim M_* \lesssim 8\ M_\odot$), the evolution of these stars (as a single star) is discussed here. The evolutionary stages of stars in other mass ranges are similar, each with their own complexities and branches. Here only the low- and intermediate-mass stars are discussed, as only these evolve through the AGB phase, which is the topic of this thesis. The section is based on the book of Carroll & Ostlie (1996).

1.1.1 Star formation

From an initial cloud of gas, either primordial gas of the big bang or from earlier generation of stars, new stars can be born. Given sufficient mass, such a cloud can undergo contraction due to its self-gravity. Contraction is possible if self-gravity wins over repulsive forces, such as the cloud's thermal repulsion, rotation of the cloud or potential galactic magnetic fields. Initially the collapse happens isothermally in free-fall. During this first isothermal stage, fragmentation can take place, where inhomogeneities can separately collapse, resulting in multi-body systems. This free-fall continues until the release of gravitational energy is strong enough to make the process adiabatic, resulting in the heating of the fragment's core. Provided sufficient mass,

the gravitational collapse of this core can be of such magnitude that the temperatures and pressures build up, activating nuclear burning, turning the once fluffy cloud into a dense, light emitting star. When the collapsed cloud is too light, it will stabilise using electron degeneracy pressure to counter the gravitational force. This pressure is exerted by the inability of fermions to occupy the same quantum state (Pauli exclusion principle). These objects are called brown dwarfs, but fall below the low- and intermediate-mass range, and are thus not explained further.

1.1.2 Main-Sequence Star

The first element that ignites is hydrogen, which initiates the first stage of stellar evolution: the Main-Sequence (MS). These stars populate a distinct region in the Hertzsprung-Russell (HR) diagram (see Figure 1.1, see the observational HR diagram on the left), from which the MS gets its name. The most massive stars are on the upper left and the least massive on the lower right. More massive stars have more fuel to burn, but also have more gravity to counter. Therefore the energy output of massive stars is significantly larger, resulting in a lower lifetime of these stars. Lower mass stars therefore live longer than high mass stars.

To have a star in hydrostatic equilibrium, the gravitational force needs to be countered. The repulsive force is created via a pressure gradient in the star, where this pressure gradient is maintained by a temperature gradient via the energy transport of this nuclear burning. The energy created by the nuclear burning in the core of the star is transported out via multiple options. Radiation pressure arises from local radiative luminosity that is absorbed by the medium. If the energy and/or opacity is sufficiently low, then the energy is transported outwards via radiation only. However, if the energy flux/opacity is too high, then convection activates. Convection causes bubbles of hot plasma to move around, transporting the energy

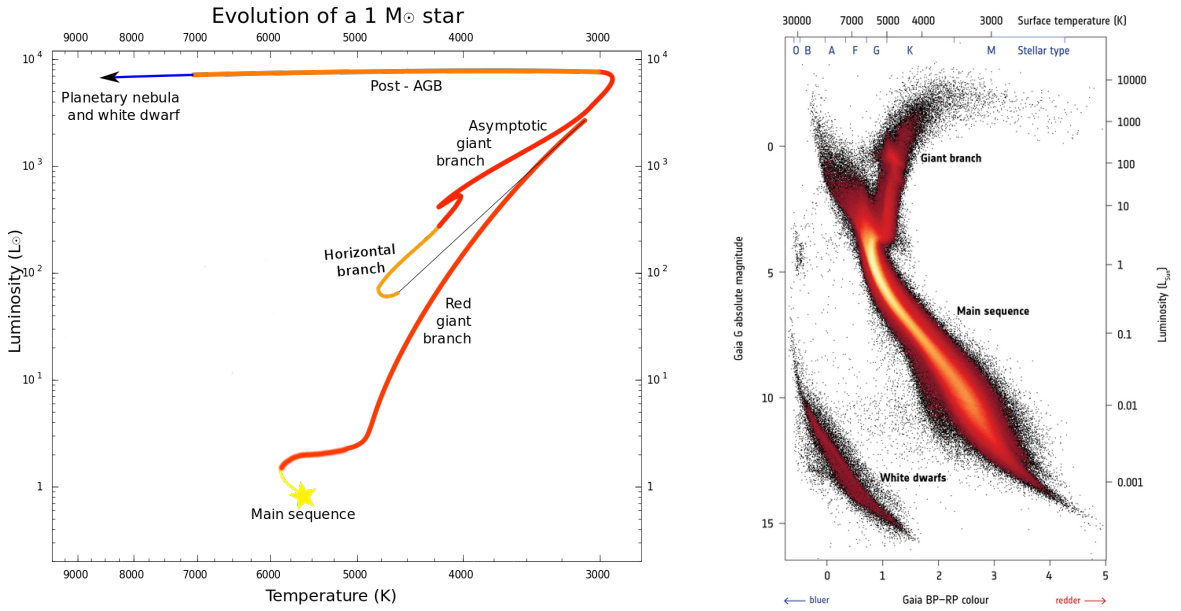


Figure 1.1: HR diagrams, both theoretical (left) and observational (right). The left represents the evolutionary track of a 1 $M_{\odot}$ star in the HR diagram, adopted from Wikimedia Commons user Lithopsian, while on the right observed stars are placed on the HR diagram, credit: ESA/Gaia.

and creating the pressure gradient in this way. Both cases counter gravity, making it possible to sustain a hydrostatic equilibrium, a star.

In this first burning stage, the nuclear burning fuses hydrogen (^{1}H, or p as this is just a proton) into helium (^{4}He, or α). Depending on the exact core temperature, different hydrogen burning processes are dominant. For stars with a core temperature below 20 MK ($M_* \lesssim 1.2$ $M_\odot$) the proton-proton (pp)-chains dominate. This process converts hydrogen into helium without external catalysts. For temperatures larger than 20 MK ($M_* \gtrsim 1.2$ $M_\odot$) the CNO-cycles dominate. Here carbon, oxygen and nitrogen act as catalysts, enhancing the burning rate. These elements can originate in earlier nuclear burning in other stars that returned their material to the interstellar medium (ISM) before the creation of the star. (Iliadis, 2015)

The hydrogen burning described above is the most efficient of all burning stages, meaning that with only a few hydrogen elements a lot of energy can be released. This results in a slow burning process. For this reason stars live most of their lives in this phase. Once hydrogen is depleted in the stellar core, hydrogen burning ceases and the stellar core collapses due to the remaining gravity. This initiates the further evolution of the star, at which point it moves away from the MS and onto the red giant branch.

1.1.3 Red Giant Branch Star

When hydrogen burning ends in a stellar core, all the hydrogen in the center of the core is fused into helium. But at this point the core is not hot nor dense enough to ignite helium yet meaning that it has no force to counter gravity. Therefore the core contracts, heating it up at the same time. Hydrogen burning goes on in a shell around the core, but the core itself is not supported and keeps on shrinking. Looking at the hydrogen burning shell, this shell is in a region hotter than where core hydrogen burning happens. This means that it happens at a higher rate, creating a larger energy output (higher luminosity) of the star. As radiation pressure increases, the outer layers of the star expand, resulting in a lower effective temperature in these outer layers. These stars appear red and are thus called red giants, and move upwards in the Red Giant Branch (RGB) (see Figure 1.1). Depending on their initial mass, RGB stars evolve along one of two distinct tracks:

- $M_* \lesssim 2$ $M_\odot$: These stars have non-convective cores, and can therefore profit from electron degeneracy pressure to counter the gravitational force in the core. This creates an isothermal core, that results in a slow heating. Eventually the core is hot enough to ignite helium burning, throughout the entire core at the same time (as it is isothermal). The star now leaves the RGB via a so called helium flash.
- $M_* \gtrsim 2$ $M_\odot$: These stars have a core too hot to transport their energy via radiation only, resulting in a convective core, which is therefore not able to reach a degenerate state. They also reach a sufficiently hot core to ignite helium, leaving the RGB phase, but since the core is not isothermal, this is not as instantaneous, thus no helium flash happens.

1.1.4 Horizontal Branch Star

When helium is ignited in a stellar core, the core expands and cools down. This means that the output energy drops, lowering the luminosity and shrinking the star. Therefore the star

moves away from the RGB to the start of the horizontal branch (HB) (see Figure 1.1). During the evolution in the HB, a star keeps shrinking (rising the effective temperature) while keeping its luminosity roughly constant. Therefore the name horizontal branch, as the star moves horizontally in the HR diagram.

Helium burning happens at temperatures higher than about 10 MK, using the triple-α process. In this process three ^{4}He nuclei fuse together and produce ^{12}C, where this can interact at the same time with an other α-particle to produce ^{16}O.

Helium burning is less efficient which means that more helium atoms need to fuse to have the same energy output. Although the hydrogen shell burning that started in the RGB phase is still ongoing, the fast burning of helium in the core results in a faster HB phase compared to the MS phase. When core helium burning stops, the star moves to the asymptotic giant branch.

1.1.5 Asymptotic Giant Branch Star

When all the helium in the center of the core is depleted, a CO electron-degenerate core is created. Similar to hydrogen shell burning, helium keeps on burning around the core, in a helium burning shell. When this happens, the star becomes an Asymptotic Giant Branch (AGB) star. At this point there are five layers in the star: a CO electron-degenerate core, a He burning shell, a quiescent He intershell, a H burning shell and finally a convective H envelope. These layers can be seen in Figure 1.2 for a 5 $M_{\odot}$ star.

This layered structure is not stable. Most of the time the helium shell is not hot enough to sustain thermonuclear burning, while the hydrogen shell keeps on burning due to its lower

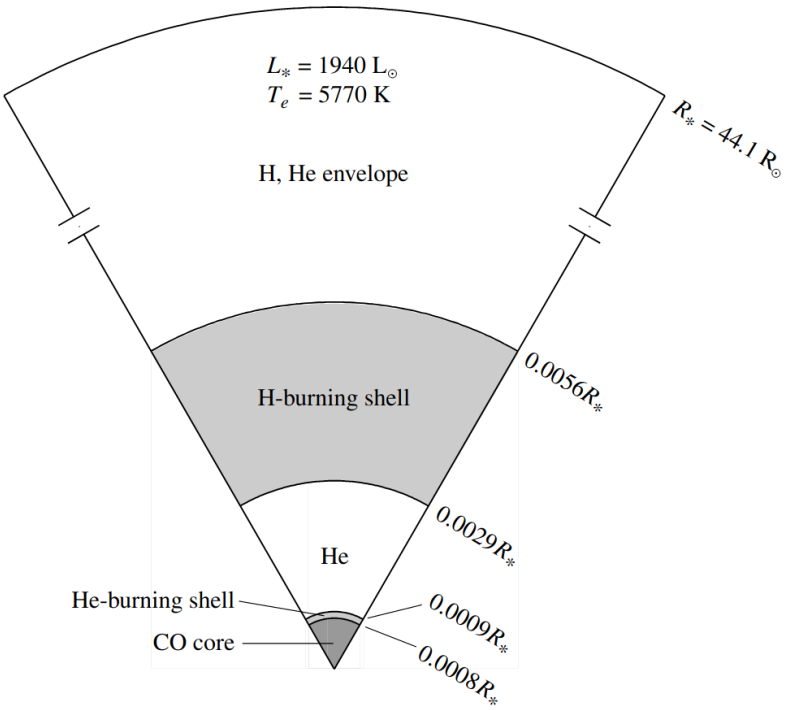


Figure 1.2: The interior structure of a 5 $M_{\odot}$ star in the AGB phase not to scale. Scales of the core and shells increased by a factor 100. (Carroll & Ostlie, 1996)

ignition temperature. This means that the helium shell grows in mass as helium is produced in the layer below, and heats up due to gravitational contraction. In a similar way as the helium flash, here the helium burning starts in a helium shell flash, more commonly referred to as a thermal pulse. All the energy produced in the helium burning shell is absorbed by the hydrogen burning shell, expanding this layer and cooling it down. The hydrogen burning temporarily stops due to the cooling. At the same time the energy of the helium shell flash expands the radius of the core, pushing the helium shell and the intershell regions into the convective zone. This mixes the layers and is called the third dredge-up[1], and brings material such as carbon and oxygen to the upper layers of the star. The helium burning extinguishes, contracting the hydrogen layer, re-igniting the hydrogen burning shell where the process starts over. Stars in this process are called thermally pulsing AGB stars.

Because the byproduct of He burning is predominantly carbon, the mixing of material due to a dredge-up influences the C/O abundance ratio at the surface of the star. At the start of the AGB star, the C/O abundance is still the abundance the star started with, and since the cosmic abundance of oxygen is greater than the carbon abundance, most of the AGB stars on the beginning of the track are oxygen-rich. The AGB star is called a carbon-rich star if C/O is above unity, and called an oxygen-rich star if C/O is below unity. Due to the thermal pulses, relatively more carbon is brought up to the envelope increasing C/O. All stars that stay on the AGB long enough thus eventually become carbon-rich stars.

An other phenomenon that characterises AGB stars is their significant mass loss. Due to their extremely convective envelope which only harbours a few enormous convective cells, the surface of the star becomes highly dynamic, resulting in pulsations. These pulsations create shocks pushing the outer envelope layers to cool regions where dust can form. The opacity of these dust particles is sufficiently high so they efficiently interact with the stellar radiation and are blown away from the star. The outward motion of the dust particles through the gas initiates a drift force on the remaining gas particles, dragging them along, away from the star. In this way, a dust driven wind arises (for a more detailed explanation, see Section 1.2), blowing away a significant amount of material with mass loss rates of about 10^{-8} to 10^{-4} $M_{\odot}yr^{-1}$ and wind velocities of 5 to 25 km s^{-1} Höfner & Olofsson (2018). This creates a large, nebulous cloud of matter surrounding the star, called the circumstellar environment (CSE).

1.1.6 Post-AGB Evolution

The AGB phase stops when its stellar wind has stripped the star from its envelope. Now the thermal pulses and mass loss is halted, and the star enters the post-AGB phase. Hotter regions of the star now become visible, increasing the effective temperature of the star throughout this phase, while keeping a constant luminosity (see Figure 1.1). The object is thus a central remnant object, surrounded by a large and dusty CSE. This dust is so significant that for some post-AGB stars the light coming from the central object is absorbed or scattered by this dust before it reaches us as observers, and only the scattered light from the dust is observed (e.g. the Red Rectangular Nebula in Figure 1.3).

This evolutionary phase is followed by the planetary nebula (PN) phase, that is characterised by its ionised CSE. This ionisation comes from the remnant core with effective temperature

[1]There are two earlier dredge up phases similar to this one, one in the RGB phase, and one at the start of the AGB phase.

Figure 1.3: Images of a post-AGB, the Red Rectangular Nebula (left) and a planetary nebula, the Cat's Eye Nebula (right). Image credits: NASA

higher than about 30 kK, emitting most of its radiation in the UV. The CSE now had more time to expand, making these structures significantly larger than the post-AGB CSE, where most PNe have sizes of the order of light years. Both evolutionary phases show a large-scale morphology that is predominantly non-spherical. Often the nebulae have bi- or multi-polar shapes, whose origins remain unknown to date. Examples for a post-AGB star and a PN can be seen in Figure 1.3.

Eventually the nuclear burning stops in the bare core, cooling it down again. As the effective temperature decreases, so does the UV flux, rendering the nebulae invisible, and only the remnant object (the degenerate CO core) - called a white dwarf - remains observable.

1.2 Dust Driven Wind

AGB stars are characterised by their significant dust driven wind. A schematic overview of the processes happening in a dust driven wind can be seen in Figure 1.5. Here each part of the process is explained in more detail, based on Chapter 7 of Lamers & Cassinelli (1999).

Extended atmosphere

Due to the unstable convective envelope of the AGB star, strong pulsation modes arise throughout the surface of the star. This creates complex dynamics in the atmosphere of the star where these pulsations can induce strong shock waves pushing material from the atmosphere away from the surface (see Figure 1.4). The layer to which material is pushed, is called the extended atmosphere as schematically represented in Figure 1.5. The speed at which the material is pushed, remains below the escape velocity, thus the material falls back onto the star. This mechanism alone is not enough to drive a wind. Therefore an additional mechanism is needed, namely dust formation.

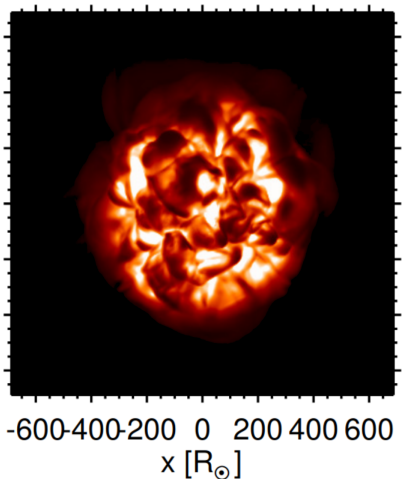


Figure 1.4: Intensity map of the atmosphere of an AGB star, modelled by Freytag et al. (2017).

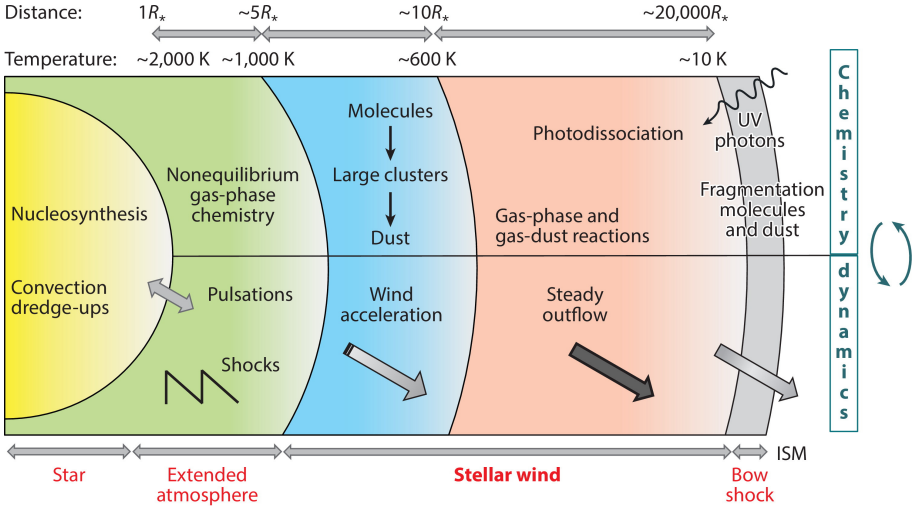


Figure 1.5: Schematic representation of all the processes important in launching an AGB dust driven wind. (Decin, 2021)

Dust formation

When sufficiently dense material is located in a sufficiently cool environment, it can condensate into dust grains. This means that atoms and molecules stick together, creating a macroscopic sized object. In AGB stars this is possible in high density regions with a low temperature, conditions thought to occur in the post-shock regions (Freytag et al., 2017), making it possible to form dust close to the star. As dust nucleation depends on the reactions that are able to occur, it heavily depends on compositions in the AGB atmosphere, and in particular the C/O abundance ratio which, as discussed in Section 1.1.5, depends on the initial mass and age of the AGB star. Held together by a triple bond, the CO molecule is extremely strong and has the ability to lock in all the available C and O molecules. If C/O is above unity, only C atoms are left, thus molecules/dust can only form with the leftover carbon. If C/O is below unity then all the carbon is expected to be locked in the CO molecules, so the remaining oxygen can form molecules/dust. As explained in Section 1.1.5, this ratio can increase during an AGB lifetime due to the dredge-up. As an example for dust formation of a carbon-rich star, carbon can be the atom that sticks together in the reaction

$$\mathrm{C_N} + \mathrm{C}_i \rightarrow \mathrm{C_{N+}}_i \tag{1.1}$$

such that the carbon grain can grow. The carbon dust and formation reactions are quite well understood, but for oxygen this remains unknown to date.

Around the star there is an approximate radius where dust forms. This radius is referred to as the dust condensation radius R_d. These dust grains have a significantly higher opacity than the surrounding gas, due to either the direct absorption of photons (and their momentum), or via the scattering of light by the reflective grains. The transfer of momentum from the stellar light to the grains causes the dust to be blown away by radiation.

Wind Acceleration

Radiation pressure is dependent on the opacity, as well as the luminosity of the star (more information on the radiation pressure can be found in Section 3.1.1). Due to the high luminosity of an AGB star in the infrared, and the coincident high opacity of the dust in the infrared,

the radiation pressure can be significant. The transfer of momentum from the stellar light to the dust particles results in an outward acceleration of the grains which can overcome gravity, resulting in an outflow of dust.

While dust particles move outwards, the gas, which does not possess this broad opacity in the infrared, remains gravitationally bound to the star. This creates a difference in velocity resulting in a lot of collisions between both types of particles, via a drag force. This drag force transfers a portion of the linear momentum from the dust to the gas. In this way both the gas and dust are moving outwards, in total a steady outflow of material.

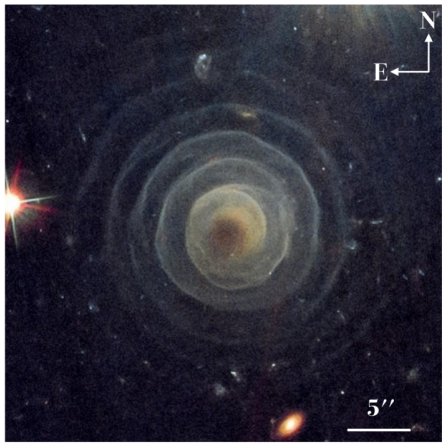
Chemistry

Now that the wind is launched, and the gas and dust are moving outwards, the question remains which chemical elements are moved outwards, feeding the ISM. Chemistry is not only affected by the temperature of the gas emitted by the AGB star itself, it can also be altered by a possible stellar companion, as well as UV photons coming from outside the system (Van de Sande et al., 2021; Van de Sande & Millar, 2022). Over 105 molecules and 16 types of dust are already observed (Decin, 2021). This complex chemistry results in 85% of the interstellar gas and 35% of the interstellar dust (Tielens, 2005). It is important to understand this chemistry since on the one hand it contributes to the enrichment of the ISM, but also because it is via the low-energy transitions in these molecules that most observations of these stellar winds are performed.

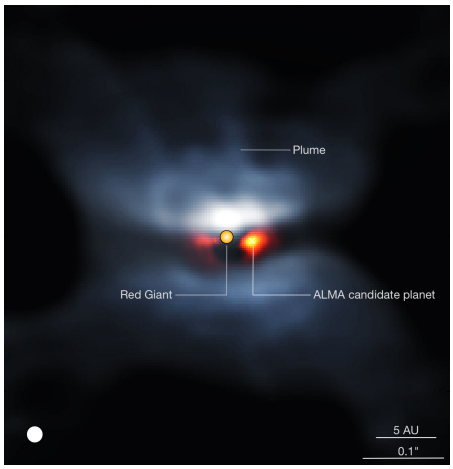
1.3 AGB observations

The picture painted in Section 1.2 is purely radial, so one could expect AGB outflows to be spherically symmetric. About 80% of AGB stellar winds show an overall spherical symmetry on the largest scales (Neri et al., 1998), however observations of the inner winds (on scales of a $\sim$10 to $\sim$100 au) of AGB stars, complex structures become visible, similar to the ones observed in PNe (e.g. Decin et al., 2020). Looking at the AGB progenitors, less than about 20% remain spherically symmetric (Parker et al., 2006; Sahai et al., 2011), probably due to these intermediate scale structures.

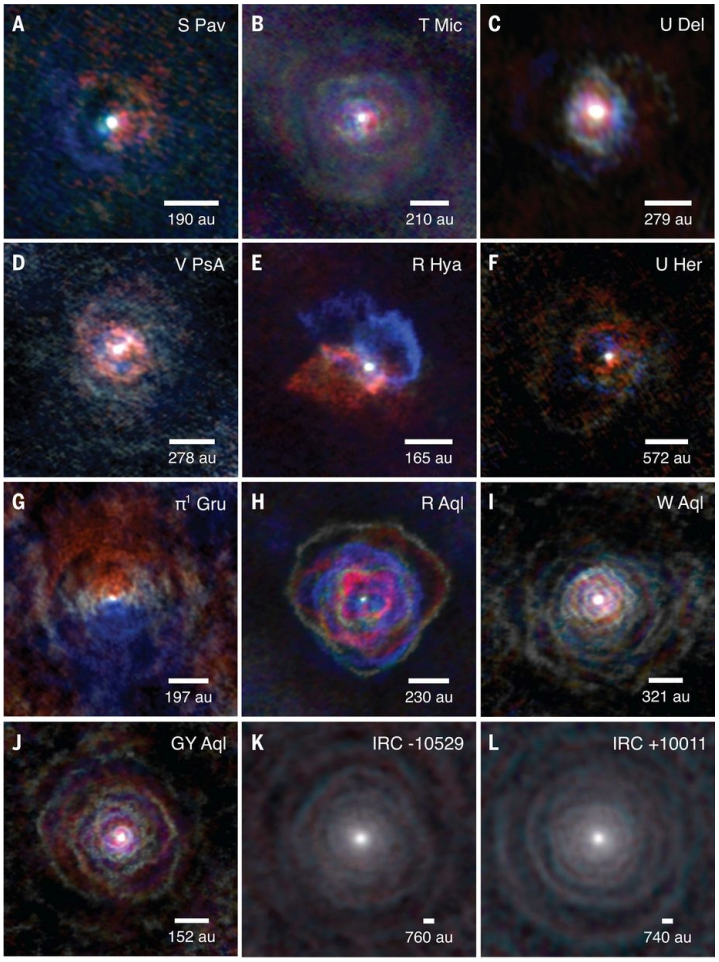
Looking in more detail to AGB wind observations, complex structures like spirals can be seen in for instance LL Pegasi (Guerrero et al., 2020) in Figure 1.6a, R Scl (Maercker et al., 2016), EP Aqr (Homan et al., 2018), also signals of a rotating disk in for instance L2 Puppis (Kervella et al., 2016) in Figure 1.6b, or R Hya (Homan et al., 2021) but also hourclass structures in Pi1 Gru and R Hya (Homan et al., 2020) and so on. Additional recent observations of other structures observed in AGB stars can be seen in Figure 1.6c, from the ALMA ATOMIUM Large Program investigating these morphologies. These structures can not be explained by single star evolution, only asymmetries can arise from non-radial pulsations. looking at the shapes of post-AGB evolutionary systems, they are explained due to a binary companion (e.g. Van Winckel, 2003; Duchêne & Kraus, 2013). For some observations of the complex structures the presence of a binary companion is observed as well (e.g. Homan et al., 2020). For this reason it is interesting to investigate the hypothesis of a companion as a shaping mechanism of these structures. Previous observational investigations of a companion as the shaping mechanism already give promising results (e.g. Ramstedt et al., 2014; Decin et al., 2020), and therefore an investigation into binary systems is necessary.



(a) LL Pegasi



(b) L2 Puppis



(c) ALMA ATOMIUM Large Program

Figure 1.6: High resolution observations of AGB stars. In (a) LL Pegasi (Guerrero et al., 2020), in (b) L2 Puppis Kervella et al. (2016), and (c) observations of the ALMA ATOMIUM Large Program (Decin et al., 2020)

1.4 Binary Systems

For massive stars ($M_* \gtrsim 8$ $M_\odot$) the vast majority has at least one stellar companion (Sana et al., 2012), while for lower-mass stars up to the AGB phase the answer is less clear. However recent statistics predict that stars with masses above about 1.5 $M_\odot$ have on average more than one companion with mass above about 5 Jupiter masses (Decin et al., 2020). Looking at the progenitors of AGB stars, more than 50% is accompanied by a stellar companion (Van Winckel, 2003; Duchêne & Kraus, 2013). Therefore the effect of a companion on the evolution of AGB systems must be quantified. This section focuses on the basics of binary interaction, based on the book of Hilditch (2001).

A binary system consists of two stars orbiting around each other. They are bound by mutual gravitational interaction, and their orbit is described by Kepler's laws

$$\begin{cases} r = \dfrac{a(1-e^2)}{1+e\cos(\theta)} = a(1-e\cos(E)) \\ \tan^2\dfrac{\theta}{2} = \dfrac{1+e}{1-e}\tan^2\dfrac{E}{2} \end{cases} , \quad E - e\sin(E) = \frac{2\pi}{P}(t-T) \tag{1.2}$$

where r is the distance between the two stars, a is the semi-major axis, e the eccentricity ($e < 1$ for a bound system), θ the true anomaly, E the eccentric anomaly, P the period of the system and T the time of periastron passage. This motion is the relative motion, placing the origin in one star, and following the other. We can describe the motion around the common center of mass (COM) as well, which is typically the reference frame of an observer, where this (barycentric) motion is related to the relative motion by

$$r_1 = \frac{M_1}{M_1+M_2}r \ , \quad r_2 = \frac{M_2}{M_1+M_2}r \ , \quad \theta_1 = \theta_2 = \theta \ . \tag{1.3}$$

1.4.1 The Roche Lobe Model

Approximating the two stars as point sources, the construction of a gravitational equipotential surface reveals insights on the behaviour of matter around a binary system. This is known as the Roche model, and can be obtained from the sum of the gravity forces and the Coriolis force. By assuming the system to be circular ($e = 0$), the equipotential can be expressed in the co-rotating frame as

$$\Phi(\vec{r}) = -G\frac{M_1}{|\vec{r}-\vec{r}_1|} - G\frac{M_2}{|\vec{r}-\vec{r}_2|} - \frac{1}{2}|\vec{\omega}\times\vec{r}|^2 \tag{1.4}$$

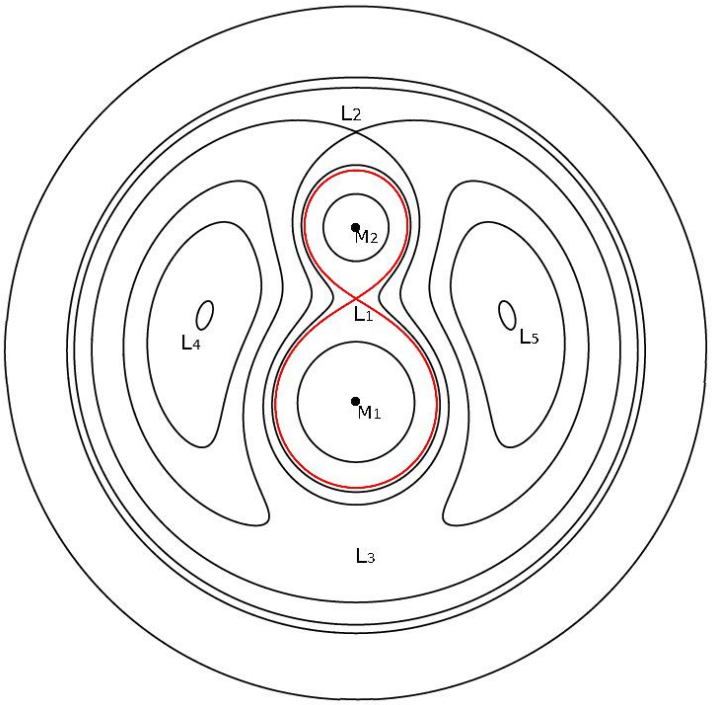


Figure 1.7: Equipotential lines of the Roche potential in the orbital plane. L_i are the five Lagrangian points, and the red line is the Roche lobe. (Hilditch, 2001)

where $\vec{r}$ is the position vector from the COM and ω the angular velocity of the system. The typical arrangement of contour lines of this potential in the orbital plane can be seen in Figure 1.7.

There are five stationary points in the Roche potential, due to the exact balance of the gravitational force and the centripetal force. They are called the Lagrangian points and are shown by L_1 to L_5 in Figure 1.7.

Going through L_1 there is one special equipotential surface called the Roche lobe shown in red in Figure 1.7. This surface represents the maximal size of the star before the gravitational attraction of the companion star becomes strong enough to absorb mass of the first star. The extent to which the stellar, or circumstellar matter fills the Roche lobe leads to three primary classifications of binary systems. These are a detached system where both stars are inside their Roche lobe, a semi-detached system where one of the stars fills its Roche lobe, or a contact (or over-contact) system where both stars overshoot their Roche lobe. The filling of the Roche lobe by either one or two stars implies very tight interaction between the two stars making up the binary. In an over-contact system it is clear that the binaries directly interact with each other. For a semi-detached system, the Roche lobe filling star loses mass directly from the gravitational pull of the companion. This means that the companion accretes and gains mass in this process. This can happen via stable or unstable mass transfer, but in both cases, a significant amount of mass is stripped from the star by the companion. It should be noted that a detached system does not mean that there is no interaction between both stars. For instance, the presence of a stellar wind such as in the case of an AGB star can influence a star's surroundings significantly, thus influencing the companion.

1.4.2 Binary interaction through stellar winds

Binary interaction happening due to stellar winds can be described by either of two scenarios. For fast (free, above the escape velocity) winds, the interaction of the Roche potential can be neglected and only the portion of the radial wind flowing closest to the companion can be gravitationally pulled towards the companion, giving rise to Bondi-Hoyle-Lyttleton (BHL) accretion (Hoyle & Lyttleton, 1939; Bondi & Hoyle, 1944).

To derive the BHL accretion, it is assumed that the companion is moving at supersonic speeds with respect to a medium. In this way a critical impact parameter (called the Hoyle-Littleton radius) can be defined, given as (Edgar, 2004)

$$R_{HL} = \frac{2GM}{v_\infty^2} \tag{1.5}$$

where all the particles passing the companion closer than this radius are accreted. This results in an accretion rate of

$$\dot{M}_{HL} = \pi R_{HL}^2 v_\infty \rho_\infty = \frac{4\pi G^2 M^2 \rho_\infty}{v_\infty^3} \ . \tag{1.6}$$

A more rigorous derivation leads to the Bondi-Hoyle accretion rate (Edgar, 2004)

$$\dot{M}_{HL} = \pi R_{HL}^2 v_\infty \rho_\infty = \frac{4\pi G^2 M^2 \rho_\infty}{(c_\infty^2 + v_\infty^2)^{3/2}} \ . \tag{1.7}$$

For this formalism to hold, the wind velocity should be much larger than the movement of the companion ($v_{wind} \gg v_{orb}$). If this is not the case, further analysis should be done.

For slow winds, the Roche potential can not be neglected. In this case the accretion is enhanced due to mass transfer of the wind through L1. This phenomenon is called wind Roche lobe overflow (WRLOF) (Mohamed & Podsiadlowski, 2007, 2012). It can be interpreted as a semi-detached binary system, where it is not the star that fills its lobe, but the AGB stellar wind. In this case more material is attracted to the companion, passing trough L_1. This results in an enhancement of the accretion rate, where numerical simulations have shown an increase with a factor of about 100.

1.5 Modelling AGB outflows

In many astrophysical contexts, a large range of complex feedback loops exist between micro- and macro-physical processes, making it hard to simplify the systems to analytic models. The way to combine these different kinds of physics is through numerical modelling. In the context of understanding AGB stellar winds, there are three primary numerical fields that dominate theoretical research: hydrodynamics, radiation transfer, and chemistry. Ideally these three numerical pillars would be computationally interlinked, and simultaneously solved. However, each of these components is extremely computationally demanding, making self-consistent modelling of AGB winds currently impossible. We briefly discuss the state-of-the-art modelling of these three numerical problems.

Hydrodynamic modelling is the art of solving the fluid equations, and gives insight on the flow of gas in astrophysical systems. Binary interactions have provided insight into the physics that govern the wind-companion interactions, and the ensuing consequences on e.g. the wind morphology and the evolution of the system. Previous studies already produce some of the observed complex structures (e.g. Theuns & Jorissen, 1993; Mastrodemos & Morris, 1999; Chen et al., 2017, 2020; El Mellah et al., 2020; Maes et al., 2021; Malfait et al., 2021) as can be seen in Figure 1.8 where a spiral structure is obtained. Being of high relevance to this thesis (see Section 1.6), the details of hydrodynamic modelling is explained in Chapter 2.

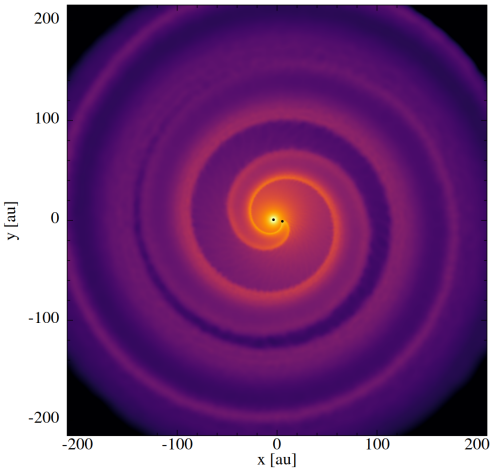


Figure 1.8: Hydrodynamic AGB outflow model forming a spiral. (Maes et al., 2021)

Besides hydrodynamics, there is also the study of the chemistry, which focuses on solving a large set of chemical- and photo-reactions under various physical conditions in order to predict the abundances of a multitude of molecular, atomic, and ionised species. Due to the large span of densities and relatively cool temperatures, the chemistry of AGB outflows is rich in molecules. Understanding this chemistry is important for interpreting the observations, since most observations are performed through the low-energy transitions of the prevailing molecules in the cool wind. So it is crucial to know the locations of different types of molecules, and how these might be used to trace certain features in the wind. In addition, the chemistry in the system has a significant influence on the wind outflow

because it is the details of the chemistry that determine the efficiency with which dust is formed, as well as its chemical composition. This, in turn, affects how the dust interacts with the radiation, and consequently the dynamics of the wind. To properly model and understand these effects, a large chemical network must be constructed with a lot of chemical reactions. Simulations of the chemistry of AGB outflows are being done e.g. Van de Sande et al. (2021); Van de Sande & Millar (2022), but due to their computational cost, only 1D simulations of chemistry are currently computationally feasible.

The field of radiation transfer focuses on understanding the interaction between light and matter. Because the excitation properties of the gas affect the way in which it radiates, and vice versa, this problem is numerically speaking the most difficult to solve. Full 3D radiation transfer solvers (e.g. `MAGRITTE`; De Ceuster et al., 2020a,b) exist and can be used in post-processing to investigate how simulations would manifest as observables, as well as to obtain radiative cooling rates, rates of radiative acceleration, or to obtain dust temperature profiles. Real-time coupling between full radiation transfer and the two other numerical pillars described above are, crudely put, computationally impossible. In AGB outflows, the rates of radiative acceleration are of extreme importance to model the dynamics of the outflow, and will be the topic of this thesis. A more in-depth explanation of radiation transfer can be found in Chapter 3.

1.6 Motivation, Contribution Statement and Outline of this Thesis

High resolution observations show that complex structures exist in the inner portions of the outflows of AGB stars. Given the overall axial symmetry of the observed structures, it is hypothesised that the origin of these structures could be due to the presence of a close stellar or planetary companion, interacting with the wind to shape the complex structures (Decin et al., 2020). Hydrodynamic studies have been undertaken to model this interaction (e.g. Theuns & Jorissen, 1993; Mastrodemos & Morris, 1999; Chen et al., 2017, 2020; El Mellah et al., 2020; Maes et al., 2021; Malfait et al., 2021) but due to the high computational cost, a lot of the important physics governing these dust driven winds have been either approximated, or even completely neglected. However, recently efforts are being made to improve the modelling of these winds by steadily adding or upgrading the lacking physics into hydrodynamic models, paving the way towards self-consistent AGB wind models. It is within this context that a collaboration has been established around the smoothed particle hydrodynamics (SPH) code `PHANTOM` (Price et al., 2018) (see Section 2). It is possible to implement the lacking physics following one of two different philosophies. Either implementing all the physics at once and immediately calculating models as realistically as possible, or implementing physics step by step, carefully investigating each step in order to in the end have a better understanding of all the components and their influence on the system. Following the latter approach, in Leuven codes are being developed to accelerate the chemistry and radiation transfer calculations so these can be coupled to the SPH solver. At the ULB colleagues are working on implementing dust nucleation, cooling, dust-gas mixtures, and stellar surface pulsations into `PHANTOM`, as well as improving the numerics of the implementation (for a more extensive overview, see Maes et al., 2022).

It is within this context that this thesis project has been worked out, focusing on adding a more

refined treatment of the dust attenuation of the stellar radiation flux to the SPH models of binary AGB outflows, aiming to better describe the wind acceleration mechanism. In the past hydro simulations have always assumed the optically thin limit (meaning $\tau = 0$, see Section 3.1) for the entire simulation. More refined simulations show that lifting this approximation can give rise to phenomena like circumbinary structures or disks (Chen et al., 2017, 2020). These structures arise when the radiation pressure becomes negligible if the optical depth takes over, resulting in the possibility of accumulating stationary circumbinary matter. These large scale structures are common for post-AGB stars (e.g. Van Winckel, 2003) but are recently also observed around AGB stars (e.g. Kervella et al., 2016; Homan et al., 2021).

For this purpose I have extracted the ray-tracer implemented in `MAGRITTE` (De Ceuster et al., 2020a,b), and added it to `PHANTOM`. First I translated the algorithm from C++ to Fortran95, than made some adaptations to the ray-tracer to make the interaction with `PHANTOM` more consistent with the SPH approach. Next I implemented different ray-tracing schemes to speed up the calculation, to then create a ray-interpolation scheme. Next up with the help of my collaborators at the ULB, I plugged the full implementation inside `PHANTOM`, to then simulate a system as a proof of concept.

This thesis is structured in the following chapters. In Chapter 2 the general methodology of hydrodynamic modelling an AGB outflow in `PHANTOM` is explained, going deeper into hydrodynamic modelling itself, the specifics of `PHANTOM` itself, as well as how the physical setup is created in `PHANTOM`. In Chapter 3 radiation transfer is explained in the context of AGB outflows, to further explain the `MARGRITTE` ray-tracer, as well as how rays can be traced to get the most accurate solution in the least computationally demanding way. In Chapter 4 the impact of the attenuation of radiation pressure on the morphologies of AGB outflows is explained, to conclude in Chapter 5.

Chapter 2

SPH Modelling of AGB outflows using PHANTOM

In modelling AGB outflows, we need to investigate how the fluid equations react to a binary system. This is exactly what hydrodynamic modelling can do. Therefore an introduction to hydrodynamic modelling is given in Section 2.1, both analytical and numerical. Further in Section 2.2 more information is given on smoothed particle hydrodynamics (SPH), a way of solving hydrodynamics. Next up in Section 2.3 the SPH code used in this thesis PHANTOM is explained. Finally the implementation of a binary AGB outflow is explained in Section 2.4.

2.1 Hydrodynamic Modelling

2.1.1 Analytical fluid hydrodynamics

Under the assumption of a continuum of a Newtonian (non-viscous) fluid, the conservation laws of mass, linear momentum and internal energy can be constructed. Two distinct fluid descriptions can be used to represent these equations, the Eulerian and the Lagrangian descriptions. In both descriptions, the conservation laws can be written as (Eulerian on the left and Lagrangian on the right)

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0 \\ \rho \left(\dfrac{\partial}{\partial t} + \vec{v} \cdot \nabla \right) \vec{v} = -\nabla p + \vec{\mathbf{F}} \\ \left(\dfrac{\partial}{\partial t} + \vec{v} \cdot \nabla \right) u = (\gamma - 1) \dfrac{u}{\rho} \left(\dfrac{\partial}{\partial t} + \vec{v} \cdot \nabla \right) \rho + \mathbf{\Lambda} \end{cases} \iff \begin{cases} \dfrac{\mathrm{d}\rho}{\mathrm{d}t} + \rho \nabla \cdot \vec{v} = 0 \\ \rho \dfrac{\mathrm{d}\vec{v}}{\mathrm{d}t} = -\nabla p + \vec{\mathbf{F}} \\ \dfrac{\mathrm{d}u}{\mathrm{d}t} = (\gamma - 1) \dfrac{u}{\rho} \dfrac{\mathrm{d}\rho}{\mathrm{d}t} + \mathbf{\Lambda} \end{cases} \tag{2.1}$$

where ρ is the density, $\vec{v}$ the velocity, u the internal energy and p the pressure of the fluid. The Eulerian description describes the motion of a fluid as seen fixed in space (partial derivatives), and the Lagrangian description follows a co-moving frame (total derivatives). They can be transformed into each other using

$$\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla) \vec{v} = \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} \, . \tag{2.2}$$

In addition to the local fluid properties, the long range forces are represented by $\vec{F}$, and are for instance gravity or radiation pressure. The polytropic index γ and the mean molecular weight μ describes a relation between pressure, temperature and energy. This is determined by chemistry, radiation transfer etc. The cooling term Λ represents additional processes that can alter the internal energy. This can for instance be heating due to friction between gas and dust, heating and cooling due to chemical processes etc. Here $\vec{F}$, γ, μ and Λ are extremely important terms, as they provide the coupling of hydrodynamics with other processes like chemistry and radiation transfer (see Chapter 3).

As a final equation, the equation of state is constructed, giving the relation between pressure, density and internal energy. For an ideal gas, this is given by

$$p = (\gamma - 1)\rho u \ . \tag{2.3}$$

Using all these properties, some other useful relations can be obtained, such as

$$c_s = \sqrt{\gamma \frac{P}{\rho}} \ , \quad p = \frac{\rho k_B T}{\mu m_H} \tag{2.4}$$

where c_s is the local sound speed, k_B is the Boltzmann's constant, μ is the mean molecular weight and m_H is the mass of a hydrogen atom. Here the final equation comes from the ideal gas law, the equation of state (2.3).

2.1.2 Fluid Numerics

When describing a fluid numerically, the two approaches explained in Section 2.1.1 give rise to fundamentally different ways of solving the fluid equations. The Eulerian formalism starts from a fixed geometrical grid, where each grid point keeps track of the local fluid properties like density, pressure, temperature etc. To go from one timestep to the next, it updates all the properties based on in- and outflow of matter and energy from and into the neighbouring grid points. On the other hand the Lagrangian formalism describes the fluid as individual particles, which are traced in their co-moving frame. All the particles possess bulk fluid properties such as mass, pressure, temperature. Position and velocity, together with the fluid properties are updated in each timestep, where the mass remains constant. Because these particles essentially represent parcels of fluid, and not individual gas particles, their properties are smoothed out over a volume with a typical length-scale. Therefore this description is called Smoothed Particle Hydrodynamics (SPH).

Both descriptions have advantages and disadvantages. Grid-based codes can be made to refine their grid using any flow parameter (called Adaptive Mesh Refinement (AMR)), making it possible to tailor the simulation on for instance the temperature, whereas SPH resolution automatically follows density (mass). AMR is better in describing steep gradients and shocks, whereas SPH smooths them out. On the other side, SPH can be implemented to inherently follow the conservation of mass, momentum and energy (see Section 2.2), where these need to be forced in AMR. Specifically for modelling wind-companion perturbations in AGB outflows, both formalisms have already been used, for example in El Mellah et al. (2020) and Chen et al. (2020) for AMR and in (Maes et al., 2021) and Malfait et al. (2021) for SPH.

In this study, SPH is used. This is done since we want to study the interaction of the wind on binary separation scale, while keeping track of the remaining outflow of material. Therefore, we require a resolution scaling with density. Furthermore it is of extreme importance that the conservation laws are satisfied, to make sure there is no long term drift of energies or momenta in the simulations. We acknowledge that shocks are harder to model, but this is of less importance than the above statements (as long as non-equilibrium chemistry is not accounted for, at least). Therefore the next chapter goes deeper into SPH.

2.2 Smoothed Particle Hydrodynamics

We investigate the mathematical representation of systems using discrete particles, where smoothing the properties needs to be introduced. The theory of SPH can be built up and it can be shown that the analytical conservation laws are inherently fulfilled in SPH.

2.2.1 Smoothing kernel

Starting from the trivial equation

$$f(\vec{r}) = \int_V f(\vec{r}')\delta(\vec{r} - \vec{r}')\mathrm{d}\vec{r}' \tag{2.5}$$

where f can represent any scalar function defined in the volume V, $\vec{r}$ is a vector inside this volume, and δ represents the Dirac delta distribution. A normalised smoothing kernel W with a smoothing length h can be defined as

$$\lim_{h\to 0} W(\vec{r}, h) = \delta(\vec{r}) \tag{2.6}$$

where h is the smoothing length, a measure for the finite length of influence. If h is large, the property is smoothed out a lot, while if h is small the property can be described more sharply. It can be shown that for a symmetric kernel $W(\vec{r}) = W(-\vec{r})$, which we will keep on assuming, equation (2.5) expands in

$$f(\vec{r}) = \int_V f(\vec{r}')W(\vec{r} - \vec{r}', h) + \mathcal{O}(h^2)\mathrm{d}\vec{r}' \ . \tag{2.7}$$

Discretising the integral into a sum over particles with mass $m = \rho(\vec{r}')\mathrm{d}\vec{r}'$, this further reduces to

$$f(\vec{r}) \approx \sum_i \frac{m_i}{\rho_i} f(\vec{r}_i)W(\vec{r} - \vec{r}_i', h) \ , \tag{2.8}$$

where the index i refers to the i^{th} particle. This equation is what lies at the heart of SPH. As example, the density can now be calculated as

$$\rho(\vec{r}) \approx \sum_i m_i W(\vec{r} - \vec{r}_i', h) \ . \tag{2.9}$$

A nice example of a kernel is a Gaussian kernel, where h represents the variance.

$$W(r, h) = \frac{1}{\pi^{3/2}h^3} \exp\left\{\left(-\frac{r^2}{h^2}\right)\right\} \tag{2.10}$$

Upon taking $h \to 0$ the Gaussian goes to the delta function. The problem is that a Gaussian has an infinite range, thus for all the particles, knowledge of the properties of all the other particles is necessary. If the kernel is constructed with a finite length of influence (for instance $W = 0$ for $|\vec{r} - \vec{r}_i'| > 2h$, see Section 2.3), it is sufficient to only sum over all the particles within this sphere of influence. This set of particles is called the SPH neighbours.

2.2.2 Gradients and Divergence

Starting from equation (2.8), we can now calculate derivatives of the property described by f. The gradient of a scalar field can be written as

$$\nabla f(\vec{r}) \approx \sum_i \frac{m_i}{\rho_i} f(\vec{r}_i) \nabla W(\vec{r} - \vec{r}_i', h) \ , \tag{2.11}$$

as m_i, ρ_i and $f(\vec{r}_i)$ are particle properties and do not depend on the spacial coordinates anymore. Similarly the divergence and curl of a vector field is given by

$$\nabla \cdot \vec{F}(\vec{r}) \approx \sum_i \frac{m_i}{\rho_i} \vec{F}_i(\vec{r}_i) \cdot \nabla W(\vec{r} - \vec{r}_i', h) \tag{2.12}$$

$$\nabla \times \vec{F}(\vec{r}) \approx \sum_i \frac{m_i}{\rho_i} \vec{F}_i(\vec{r}_i) \times \nabla W(\vec{r} - \vec{r}_i', h) \ . \tag{2.13}$$

Some calculations become significantly easier using this formulation, for instance the conservation laws can be rewritten, such that they always hold for symmetrical kernels.

2.2.3 Conservation laws

In the SPH formalism, the conservation laws in equation (2.1) can be written out explicitly.

Conservation of Mass

Starting with the conservation of mass

$$\frac{\mathrm{d}\rho}{\mathrm{d}t} + \rho \nabla \cdot \vec{v} = 0 \ , \tag{2.14}$$

and the density using the SPH kernel

$$\rho_j = \sum_i m_i W(\vec{r}_j - \vec{r}_i, h) = \sum_i m_i W_{ij} \ , \tag{2.15}$$

the full time derivative of ρ can be written out as

$$\frac{\mathrm{d}\rho_j}{\mathrm{d}t} = \sum_i m_i \left[\frac{\partial W_{ij}}{\partial \vec{r}_j} \cdot \frac{\mathrm{d}\vec{r}_j}{\mathrm{d}t} + \frac{\partial W_{ij}}{\partial \vec{r}_i} \cdot \frac{\mathrm{d}\vec{r}_i}{\mathrm{d}t} + \frac{\partial W_{ij}}{\partial h} \cdot \frac{\mathrm{d}h}{\mathrm{d}t} \right] \ , \tag{2.16}$$

where the derivative of the kernel has been written out using the chain rule. Further the derivative of h with respect to time is neglected, as this term has influence only in second

order of h (Nelson & Papaloizou, 1994) where the same is true for the gradient of h. This can further be rewritten as

$$\frac{\mathrm{d}\rho_j}{\mathrm{d}t} = \sum_i m_i \left[\nabla_i W_{ij} \cdot \vec{v}_i + \nabla_j W_{ij} \cdot \vec{v}_j\right] = -\sum_i m_i \left[\nabla_j W_{ij} \cdot \vec{v}_{ij}\right] \tag{2.17}$$

by using the antisymmetry in the gradient of the kernel (as the kernel itself is symmetric) and introducing $\vec{v}_{ij} = \vec{v}_i - \vec{v}_j$. Going further

$$\frac{\mathrm{d}\rho_j}{\mathrm{d}t} = -\rho_j \nabla_j \cdot \vec{v}_j = 0 \tag{2.18}$$

by using equation (2.12) to write everything only in function of the particle j. This proves that for each particle in an SPH simulation, conservation of mass is automatically fulfilled.

Conservation of Linear Momentum

For the conservation of momentum we start from the Lagrange functional $\mathcal{L}$, constructed as the kinetic energy minus the potential energy, to later on use the Euler-Lagrange equations. $\mathcal{L}$ is given by

$$\mathcal{L} = \int_V \frac{1}{2}\rho \left[\vec{v} \cdot \vec{v} - u\right] \, \mathrm{d}\vec{r} = \sum_i m_i \left[\frac{1}{2}\vec{v}_i \cdot \vec{v}_i - u\right] \tag{2.19}$$

over all the particles i. The equations of motion for a particle j can now be found using the Euler-Lagrange equations

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \vec{v}_j}\right) = \frac{\partial \mathcal{L}}{\partial \vec{r}_j} \ . \tag{2.20}$$

Since the internal energy u is only dependent on P and ρ (see the equation of state (2.3)) which are both only dependent on the position $\vec{r}$, making u only dependent on $\vec{r}$. This can be used to obtain the derivative of $\mathcal{L}$ with respect to velocity

$$\frac{\partial \mathcal{L}}{\partial \vec{v}_j} = \frac{\partial}{\partial \vec{v}_j}\left(\sum_i m_i \left[\frac{1}{2}\vec{v}_i \cdot \vec{v}_i - u\right]\right) = m_j \vec{v}_j \ , \tag{2.21}$$

and position

$$\frac{\partial \mathcal{L}}{\partial \vec{r}_j} = -\sum_i \left[\frac{\partial u_i}{\partial P_i}\frac{\partial P_i}{\partial \vec{r}_j} + \frac{\partial u_i}{\partial \rho_i}\frac{\partial \rho_i}{\partial \vec{r}_j}\right] \tag{2.22}$$

$$= -\sum_i \left[\frac{\partial u_i}{\partial P_i}\frac{\mathrm{d}P_i}{\mathrm{d}\rho_i} + \frac{\partial u_i}{\partial \rho_i}\right]\frac{\partial \rho_i}{\partial \vec{r}_j} \tag{2.23}$$

$$= -\sum_i \frac{P_i}{\rho_i^2}\frac{\partial \rho_i}{\partial \vec{r}_j} \ . \tag{2.24}$$

where in the last equation we used $\frac{\partial u_i}{\partial P_i}\frac{\mathrm{d}P_i}{\mathrm{d}\rho_i} + \frac{\partial u_i}{\partial \rho_i} = \frac{P_i}{\rho_i^2}$, which follows from the equation of state (2.3). Now by writing out ρ_i explicitly using the kernel, the equation splits up again, and becomes

$$\frac{\partial \mathcal{L}}{\partial \vec{r}_j} = -m_j \sum_i m_i \left(\frac{P_j}{\rho_j^2} + \frac{P_i}{\rho_i^2}\right) \nabla_j W_{ij} \tag{2.25}$$

where this equation is pairwise symmetric in i,j due to the antisymmetry of the gradient of the kernel. This means that

$$\frac{\mathrm{d}\vec{v}_j}{\mathrm{d}t} = -\sum_i m_i \left(\frac{P_j}{\rho_j^2} + \frac{P_i}{\rho_i^2}\right) \nabla_j W_{ij} \tag{2.26}$$

is pairwise symmetric as well. For this reason the loss of linear momentum is always gained by a different particle, the total linear momentum is thus conserved.

Conservation of Angular Momentum

The angular momentum is given by $\vec{L} = \vec{r} \times m\vec{v}$. Calculating the time derivative of $\vec{L}$ of each particle j

$$\frac{\mathrm{d}\vec{L}_j}{\mathrm{d}t} = m_j \vec{v}_j \times \vec{v}_j + m_j \vec{r}_j \times \frac{\mathrm{d}\vec{v}_j}{\mathrm{d}t} \tag{2.27}$$

$$= -m_j \vec{r}_j \times \sum_i m_i \left(\frac{P_j}{\rho_j^2} + \frac{P_i}{\rho_i^2}\right) \nabla_j W_{ij} \tag{2.28}$$

$$= -m_j \sum_i m_i \left(\frac{P_j}{\rho_j^2} + \frac{P_i}{\rho_i^2}\right) \vec{r}_j \times \nabla_j W_{ij} \tag{2.29}$$

where this equation is again pairwise symmetric in i,j due to the antisymmetry of the gradient of the kernel. Thus again all angular momentum that is lost to a particle, is gained by an other particle. This conserves the total angular momentum.

conservation of energy

The total energy of a system is given by $E = \rho(u + v^2/2)$, the sum of the internal energy and the kinetic energy. In SPH this becomes

$$\frac{\mathrm{d}E}{\mathrm{d}t} = \sum_i m_i \left(\vec{v}_i \cdot \frac{\mathrm{d}\vec{v}_j}{\mathrm{d}t} + \frac{P_i}{\rho_i^2} \frac{\mathrm{d}\rho_i}{\mathrm{d}t}\right) \tag{2.30}$$

$$\begin{aligned} = -\sum_i m_i \vec{v}_i \cdot \sum_j m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2}\right) \nabla_j W_{ji} \\ + \sum_i m_i \frac{P_i}{\rho_i^2} \sum_j m_j \left(\vec{v}_i - \vec{v}_j\right) \cdot \nabla_j W_{ji} \end{aligned} \tag{2.31}$$

$$= \sum_i \sum_j m_i m_j \left(\frac{P_j}{\rho_j^2} \vec{v}_i + \frac{P_i}{\rho_i^2} \vec{v}_j\right) \cdot \nabla_j W_{ji} = 0 \ . \tag{2.32}$$

In the final step the antisymmetry of the gradient of the kernel is used again. This means that the total energy in the system is conserved.

This proves that all conservation laws are inherently solved in the SPH formalism.

2.3 PHANTOM

PHANTOM[1] (Price et al., 2018) is one of many SPH codes in the field. Other codes are for example GADGET (Springel et al., 2001) with a focus on cosmology, GASOLINE (Wadsley et al., 2017) with a focus on galaxy formation, among others. PHANTOM has been designed with the intent of stellar, planetary and accretion discs as well as Galactic astrophysics. This wider focus is ideal for our study. Furthermore PHANTOM is designed for low-memory, high-efficiency. It is parallelised, using OpenMP parallel programming, making it possible to divide computations to multiple threads, reducing the computation time significantly. Lastly it is an open source code, making it easy to implement extra physics to it, as well as the constant development from other fields of astrophysics. Due to the large user base of the code, which spans research field such as turbulence, star formation, cluster formation, studies of galaxies, interaction with other developers facilitates the development and testing of new code features, an attribute that should not be underestimated. This means that PHANTOM is a state-of-the-art code, ideal for the simulations of the outflow of AGB binaries.

2.3.1 Smoothing length and SPH kernel

In PHANTOM a variable smoothing length is implemented, related to the local particle number density as

$$h_i = h_{fact} n_i^{1/3} = h_{fact} \left(\frac{m_i}{\rho_i} \right)^{1/3} \tag{2.33}$$

where h_{fact} is a proportionality factor, that can be tweaked. This creates two equations that relate h and ρ (see also equation (2.9)). Using both equations the density and smoothing length can be calculated numerically, using Newton-Raphson in PHANTOM. This means that the resolution follows the density of the system. High density regions are thus probed with higher resolution. This can be related to the mean number of neighbours according to

$$\bar{N}_{neigh} = \frac{4}{3}\pi (R_{kern} h_{fact})^3 \tag{2.34}$$

where R_{kern} is the maximal interaction radius of the chosen kernel, resulting in a sphere of influence of $\frac{4}{3}\pi(R_{kern}h)^3$. For the kernel, a new formalism is used to account for the variable smoothing length. The kernel is written as

$$W_{ij}(\mathbf{r}_{ij}, h) = \frac{C_{norm}}{h^3} f(q) \tag{2.35}$$

where C_{norm} is a normalisation constant, q is a dimensionless quantity given as $|\mathbf{r}_{ij}|/h$ and f a function representing the variability of the kernel as a function of q. There are multiple kernels implemented in PHANTOM, where the default one is the M_4 cubic spline given by

$$f(q) = \begin{cases} 1 - \frac{3}{2}q^2 + \frac{3}{4}q^3, & 0 \leq q < 1; \\ \frac{1}{4}(2-q)^3, & 1 \leq q < 2; \\ 0, & 2 \leq q \end{cases} \tag{2.36}$$

[1] https://github.com/danieljprice/phantom

where $C_{norm} = 1/\pi$, and $R_{kern} = 2$. These kernels have a finite sphere of influence, what the Gaussian kernel lacks as explained in Section 2.2. This is important as for an infinite sphere of influence, all the particles feel the contribution of all the particles, making the computation extremely demanding. By default for this kernel $h_{fact} = 1.2$, resulting in $\bar{N}_{neigh} = 57.9$. Other kernels can be found in figure 1 and table 1 of Price et al. (2018).

2.3.2 Time integration

In hydrodynamic simulations, integration with respect to time is necessary to evolve the solutions. In PHANTOM this time integration is done using a Leapfrog integrator, which starts from velocity Verlet scheme (Verlet, 1967) originally created for molecular dynamics. This scheme has the benefit of being reversible and symplectic, meaning phase space volume is conserved, important for instance in long-term orbital mechanics. In PHANTOM this is implemented as

$$\begin{cases} \mathbf{v}^{n+1/2} = \mathbf{v}^n + \dfrac{\Delta t}{2}\mathbf{a}^n \\ \mathbf{r}^{n+1} = \mathbf{r}^n + \Delta t \mathbf{v}^{n+1/2} \\ \mathbf{v}^* = \mathbf{v}^{n+1/2} + \dfrac{\Delta t}{2}\mathbf{a}^n \\ \mathbf{a}^{n+1} = \mathbf{a}(\mathbf{r}^{n+1}, \mathbf{v}^*) \\ \mathbf{v}^{n+1} = \mathbf{v}^* + \dfrac{\Delta t}{2}(\mathbf{a}^{n+1} - \mathbf{a}^n) \end{cases} \tag{2.37}$$

where $\mathbf{a}$ is the acceleration. The timestep over which is integrated is dependent on how fast signals travel in the current simulation setup. The timestep is chosen using the usual Courant–Friedrichs–Lewy (CFL) condition for Eulerian codes (Courant et al., 1928)

$$\Delta t = \mathsf{min}(\Delta t_{C,i})_i \ , \quad \Delta t_{C,i} = C_{cour}\frac{h_i}{v_{sig,i}} \ , \tag{2.38}$$

where $C_{cour} = 0.3$ by default and $v_{sig,i}$ is the signal speed of the particle i given by

$$v_{sig,i} = \alpha_i^{AV} c_{s,i} + \beta^{AV} \mathsf{max}(|\mathbf{v}_{ij} \cdot \mathbf{r}_{ij}|/|\mathbf{r}_{ij}|)_j \tag{2.39}$$

where $c_{s,i}$ is the local speed of sound, $\alpha_i^{AV} c_{s,i} \in [0, 1]$ dependent on nearby shocks, and $\beta^{AV} = 2$ (by default) used to prevent particle penetration. This timestep is for the close range forces, and can sometimes be larger than the timestep for some of the long range forces (for instance gravity). Since long range forces are much faster to compute a sub-stepping scheme is activated, where long range and short range forces (or external and point-mass forces as

they are called in PHANTOM) are evolved separately. The algorithm becomes

$$\begin{cases} \boldsymbol{v} = \boldsymbol{v} + \dfrac{\Delta t_{\rm sph}}{2}\boldsymbol{a}^n_{\rm sph} \\ \text{over substeps} \begin{cases} \boldsymbol{v} = \boldsymbol{v} + \dfrac{\Delta t_{\rm ext}}{2}\boldsymbol{a}^m_{\rm ext} \\ \boldsymbol{r} = \boldsymbol{r} + \Delta t_{\rm ext}\boldsymbol{v} \\ \quad \text{get } \boldsymbol{a}_{\rm ext}(\boldsymbol{r}) \\ \boldsymbol{v} = \boldsymbol{v} + \dfrac{\Delta t_{\rm ext}}{2}\boldsymbol{a}^{m+1}_{\rm ext}, \end{cases} \\ \quad \text{get } \boldsymbol{a}_{\rm sph}(\boldsymbol{r}), \\ \boldsymbol{v} = \boldsymbol{v} + \dfrac{\Delta t_{\rm sph}}{2}\boldsymbol{a}^n_{\rm sph} \end{cases} \tag{2.40}$$

so that more timesteps can be done for the external forces than the point-mass forces.

2.3.3 Neighbour finding

The most time-consuming part of SPH codes, is the neighbour finding needed to evaluate and evolve the quantities that depend on the local medium, such as pressure. In PHANTOM this is tackled using a kd-tree[2], divided into leaf nodes. This means that all the particles are structured and separated in a tree-structured way into cells, until the particles are in a cell that contains less than N_{min} particles, by default $N_{min} = 10$ in PHANTOM. A visual representation is shown in Figure 2.1. Once the kd-tree is built, neighbour search can be done. The searching algorithm is built to find particles inside the sphere of influence, so for a given smoothing length it first finds all the cells potentially having particles inside this sphere, then goes through all the particles in these cells to find the actual particles in the sphere. This neighbour search is of the order of $\mathcal{O}(Nlog(N))$ in computation time, for large N, significantly faster than the binary neighbour search $\mathcal{O}(N^2)$.

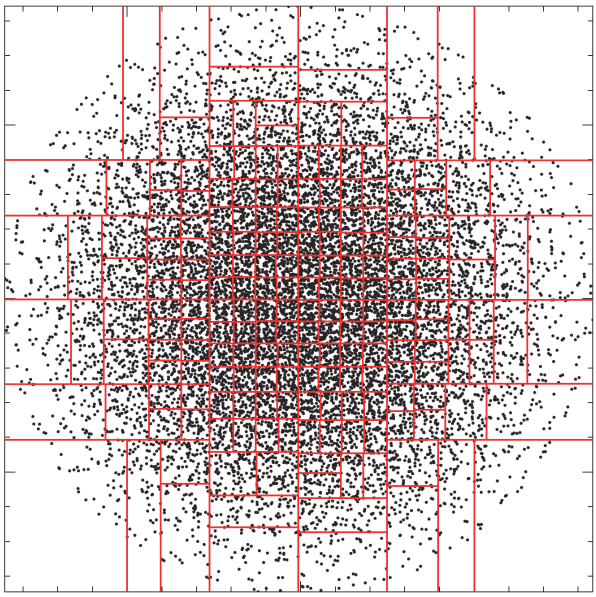


Figure 2.1: A 2D representation of a kd-tree structure, divided until $N_{min} = 100$. (Price et al., 2018)

2.3.4 Shock capturing

A well-known weakness of SPH is the handling of shocks. IN PHANTOM this is improved upon by the implementation of an artificial viscosity. This means that an additional term is added to the equation of motion as well as the conservation of energy. Both terms are implemented in such a way that they keep the conservation of momentum and energy of the entire system constant. More information can be found in section 2.2.7-2.2.9 of Price et al. (2018).

[2]a quick introduction to kd-trees can be found in https://en.wikipedia.org/wiki/K-d_tree

2.4 Hydrodynamics of binary AGB outflows

To implement our AGB outflow in a binary system, we need to mold the existing PHANTOM utilities to represent our physical system. First we need to correctly represent the AGB wind launching. Secondly the boundary conditions as well as the correct dynamical representation of a companion is needed. This setup is adopted from Siess et al. (2022), used in previous studies such as (Maes, 2020; Malfait, 2020; Maes et al., 2021; Malfait et al., 2021; Donné, 2021).

2.4.1 Technical setup of an AGB outflow

Sink particles

Stars in the simulation are represented by so-called sink particles which are treated separately from the other SPH particles. Sink particles only interact via gravity with other sink particles allowing them to evolve on shorter timesteps compared to the SPH particles. Sink particles are also allowed to have different masses, accounting correctly for the gravitational interaction of the star.

Wind injection shells

To correctly model a stellar wind, SPH particles are launched at an input velocity close to a numerical representation of the stellar surface. This is done using wind injection shells, that create an artificial pressure and density gradient to reach the given velocity. These are shells with n particles (to be chosen, see Resolution), located on a wind launching radius R_{inj}. These shells are fixed in space, and only exist to provide a negative pressure gradient for other particles. To reduce the chance of introducing numerical artefacts, the shells are rotated with respect to each other, to not induce a preferred direction. The particles on the last of these fixed shells are launched with a timing that matches the input mass-loss rate to simulate new material injected via the wind. The input velocity fixes the time interval in-between shell launches, and the input mass loss rate fixes the mass of these particles.

Resolution

The tangential resolution of the simulation can be chosen depending on the amount of particles on a shell n. This is determined by $n = 10(2q-1)^2+2$ with q an integer to create a uniformly distributed shell. This means that each particle has a distance of $d_\perp = \frac{2}{2q-1}(\phi\sqrt{5})^{-1/2}$ to its neighbouring particles on a shell of unit radius where $\phi = (1+\sqrt{5})/2$ is the golden ratio. The radial resolution, which determines the distance between consecutive shells, can also be set, and is scaled to $d_\perp$. This is implemented as $d_r = w_{ss} d_\perp$ where w_{ss} is a value larger than zero. This sets the smoothing length for these particles to $h = h_{fact} d_r^{1/3} d_\perp^{2/3}$, constraining the resolution close to the star. The rotation angles to reduce the numerical artefacts differs for the chosen resolution, where the optimal rotation angles can be found in table 2 of Siess et al. (2022).

Vacuum outer boundary layer

SPH inherently does not require an outer boundary, so essentially the outflowing particles can remain on their trajectories indefinitely. However, to not make the model too large, an outer boundary can be activated. This boundary is given as a sphere with radius R_{out}, after which

all particles are removed from the system. This creates a vacuum at this radius, which can influence the neighbourhood of the sphere. As long as the wind is supersonic, the information cannot travel downstream, so the vacuum is no problem for supersonic winds.

Output

PHANTOM creates output files (known as dumps) of the simulations. At every chosen time interval, a dump file is constructed containing the 3D position, mass as well as smoothing length of each particle (this also gives the density using equation (2.33)). Every 5 dump files, PHANTOM creates a full dump containing more information on the system. For each particle, it states all the variables PHANTOM keeps track of, like the 3D velocities, internal energy, temperature, opacity, optical depth and so on. Separately from the dump files, sink files are generated as well, giving information on the sink particles in the simulation, such as their location throughout the simulation. For the interpretation of these dump files, the visualisation tool SPLASH (Price, 2007) is used, as well as a Python plotting tool developed by Maes (2020); Malfait (2020); Donné (2021), further refined and accelerated for this thesis[3].

2.4.2 Adding a companion

To add a companion to the simulation, an additional sink particle is introduced.

Reference frame

The simulation setup works in a 3D Cartesian coordinate system. Here the origin is taken as the COM of the binary system. Around the origin both stars evolve in a Keplerian orbit. At the start of the simulation the stars are placed along the x-axis (y=z=0), and the orbital plane is located in the x-y-plane (z=0).

Accretion

Both stars are able to accrete mass (sink particles are able to change their mass during the simulation), from inside an accretion radius R_{acc}. Not all particles are accreted as some particles are fast enough to escape the star's gravity of the sink. Accretion only happens when four criteria are fulfilled:

- The particle is automatically accreted if it is in the range $r \leq f_{acc} R_{acc}$ where $f_{acc} \in [0, 1]$ (default 0.8 in PHANTOM). This region is considered close enough to the star so it is not able to escape anymore. If the particle is in the range $f_{acc} R_{acc} < r \leq R_{acc}$, the following three criteria should hold as well:
- The specific angular momentum of the particle should be less than the angular momentum of the sink particles Keplerian orbit;
- The particle should be gravitationally bound to the sink particle ($e_{i,j} = \frac{v_i^2}{2} - \frac{GM_{sink,j}}{r} < 0$);
- The particle should be most bound to the sink particle considered here. All other $e_{i,j}$ for other particle-sink pairs should be larger than the considered particle-sink pair.

Apart from the boundary layers in the system, and the hydrodynamics happening in the simulation, the cooling term and the long range forces still need to be specified.

[3] https://github.com/lsiess/SPH_plotting_pipeline

2.5 Additional physics in binary AGB outflows (Λ, γ, μ and $\vec{F}$)

In the conservation of energy and the equation of motion (2.1) the parameters Λ, γ μ and $\vec{F}$ still need to be set for our simulations. It is via these quantities that the additional physics of radiative transfer and chemistry influence the hydrodynamics (see Section 1.5). As discussed, a truly self-consistent approach is computationally unfeasible, thus important approximations enter the simulations though the evaluation of these quantities. We give a brief overview of the current way in which phantom handles these important physics, and what steps are being undertaken to improve the current implementation.

2.5.1 Cooling Λ

The cooling term Λ represents the cooling and heating of the medium, that is not due to compression (this is included by the other terms in the conservation of energy). Concretely, it describes the changes to the internal energy of the gas via mechanisms that either decrease the energy (via e.g. the emission of photons), or increase it (via e.g. friction with dust). In PHANTOM a set of rudimentary cooling prescriptions can be activated, which include a basic HI-cooling (Spitzer & Jura, 1978), as well as a cooling formalism based on different cooling mechanisms following tables of Omukai et al. (2010), implemented by Donné (2021). Currently many more cooling and heating terms are being implemented by colleagues at the ULB, which include molecular-, atomic-, ionised-, and fundamental-particle-based cooling and heating. In addition, the hydrodynamical formalism is being upgraded to the so-called one-fluid approximation of dust-gas mixtures to track the dust drift, which is thought to be an important heating source. But further discussing this is beyond the scope of this thesis.

2.5.2 Polytropic index γ and mean molecular weight μ

The polytropic index captures the relation between pressure, density and temperature, and the mean molecular weight captures the composition of the gas. The polytropic index γ is still an input variable, where $\gamma = 1$ represents an isothermal flow, and $\gamma = 7/5$ an adiabatic diatomic gas. In principle γ is dependent on the chemistry and excitation state of the gas, but as chemistry is not taken into account here, we adopt a constant value. γ can be evaluated either by setting it to the expected adiabatic value (i.e. 7/5, diatomic gas), but observations of AGB outflows have shown that the temperature of the wind broadly decays with a power of $\xi \approx 0.6 - 0.7$ (Millar, 2004). Given that the gas temperature depends on γ as (Maes, 2020)

$$\begin{aligned} T(r) &\propto r^{-\xi} \\ T(r) &\propto r^{-3(\gamma-1)} \end{aligned} \tag{2.41}$$

It can be shown that a value of $\gamma = 1.2$ captures the expected temperature trend. Such a low value for γ, which lies between the isothermal and adiabatic case, is to be expected, since these stellar winds are known to be subject to highly efficient cooling mechanisms. The default value for μ in PHANTOM is set to 2.38, which represents a standard mixture of H_2, helium, and other trace elements. A more refined treatment of the chemistry and the excitation of the gas' level populations will allow for a better prediction of the precise values of γ and μ throughout the wind.

2.5.3 Additional forces $\vec{F}$

Additional long range forces are represented by $\vec{F}$. Currently, the external force $\vec{F}$ for a binary system with a mass-losing AGB star is implemented in PHANTOM as follows

$$F_r = -\frac{GM_{AGB}}{r_1^2}(1-\alpha) - \frac{GM_{comp}}{r_2^2} \tag{2.42}$$

where r_1 and r_2 represent the distance to the AGB star and the companion, respectively, M_{AGB} and M_{comp} the mass of the AGB star and the companion, respectively, and α is an ad-hoc scalar that represents the efficiency of the wind launching mechanism. In a first approximation α is typically set to 1, representing a radiation-pressure-free wind where the gravity of the AGB star is completely balanced out. This approximation is often used in the context of hydro modelling of wind-companion interactions since it grossly encapsulates the entire dust-formation and radiative acceleration calculation, which are otherwise extremely expensive to calculate (e.g. Mastrodemos & Morris, 1998; Kim & Taam, 2012; Liu et al., 2017). With PHANTOM, nice results have been obtained using $\alpha = 1$, for instance (Maes et al., 2021), (Malfait et al., 2021). A more refined approach provides a better description for the wind acceleration mechanism, by substituting α by the optically thin Eddington factor Γ_d

$$\alpha \rightarrow \Gamma_d = \frac{(\kappa_{dust} + \kappa_{gas})L_{AGB}}{4\pi c G M_{AGB}} \tag{2.43}$$

where κ is the local opacity (dust and gas), and L_{AGB} and M_{AGB} are the luminosity and mass of the AGB star. Typically, however, α remains in the expression for $\vec{F}$ as an additional force since material still needs to reach the dust condensation radius before the dust driving can take place. The stellar luminosity dilutes as $1/4\pi r^2$, which is purely geometric dilution, and does not account for the removal of photons via e.g. dust. Hence, this expression assumes that the photons do not interact with the medium they traverse before depositing their momentum, so it is commonly referred to as the optically thin limit. The removal of photons via absorption or scattering can be accounted for by introducing a correction factor $e^{-\tau}$ to the optically thin Eddington factor, whose origin and meaning will be addressed in the next section. The additional long range force term becomes

$$F_r = -\frac{GM_{AGB}}{r_1^2}(1-\alpha-\frac{(\kappa_{dust} + \kappa_{gas})L_{AGB}}{4\pi c G M_{AGB}}e^{-\tau}) - \frac{GM_{comp}}{r_1^2} \tag{2.44}$$

To evaluate this force, knowledge of the opacities κ_{gas} and κ_{dust} as well as the optical depth τ have to be known for every point in the simulation.

Dust Opacity

The local removal of photons by a traversed medium can be described by the opacity κ. This opacity is highly dependent on the chemical composition, as well as the temperature of the medium and can represent absorption or scattering. For dust-gas mixtures, such as the outflows of AGB stars, it is useful to split up the opacity of the dust κ_{dust} and the gas κ_{gas}. To properly obtain the opacity of the gas, a full analysis of the chemical and excitation state of the gas has to be performed. This is extremely difficult and computationally expensive, so it is generally bypassed by assuming a constant value for κ_{gas}. It can be shown that a typical value

is $\kappa_{gas} = 2 \cdot 10^{-4}$ cm^2/g (Bowen, 1988). The opacity of the dust is much larger, and is the one that launches the wind. In order to properly calculate κ_{dust}, full chemistry and dust nucleation must be solved. This is relatively possible for C-rich environments, it is still impossible in the much more complex O-rich environments. Carbon-rich dust nucleation is implemented in PHANTOM, and emerges from solving a small carbon-rich chemical network, and growing the dust according to the formalism constructed by Gail & Sedlmayr (1988). However, for a typical wind setup the dust opacity rises as soon as dust is formed, and then stays roughly constant. This typically occurs around a certain dust-condensation temperature which strongly depends on the chemical compositing of the grains, but typically ranges between 1000 K and 1800 K. Since we're not interested in the details of the microphysics for this thesis, these calculations, that come with an added computational cost, can be circumvented by approximating the behaviour of κ using the analytic Bowen-type prescription of the dust opacity, given by a logistic function (Bowen, 1988)

$$\kappa = \kappa_{dust} + \kappa_{gas} = \frac{\kappa_{dust,max}}{1 + exp[(T - T_{cond})/\delta]} + \kappa_{gas} \tag{2.45}$$

where T_{cond} is the condensation temperature, δ a parameter influencing the width of the transition as well as $\kappa_{dust,max}$ the maximal dust opacity. These parameters depend on the composition of the dust, for instance C-rich or O-rich (see Section 1.1.5).

Optical depth

PHANTOM is coupled with the dust radiation transfer code MCFost (Pinte et al., 2009). This is a Monte Carlo based code, where dust radiation transport is solved using probabilistic methods. Though this code can provide us with the optical depths we require, it is slow because it solves the full radiation transport problem. And since we are only interested in the radial optical depth, we have chosen to leverage the expertise in the institutes in Leuven and Brussels by extracting the ray-tracer from MAGRITTE (De Ceuster et al., 2020a,b). In the following section more details on the implementation are given.

Chapter 3

Radiative Dust Acceleration in SPH

Radiative acceleration is crucial in launching a dust driven wind. This acceleration is given by the theory of radiation transfer. Therefore an introduction to radiation transfer is given in Section 3.1. This acceleration can be estimated using ray-tracing, explained in Section 3.2. Further an investigation is done on how to speed up the calculation. The neighbour-finding algorithm, ray-sampling, adaptive ray-tracing and ray-interpolation are investigated (Section 3.4.1, 3.4.2, 3.4.3 and 3.4.4, respectively). Finally the ray-tracer is coupled to `PHANTOM` (Section 3.5) and a scaling analysis as well as a benchmark is presented in Section 3.5.1 and 3.5.2. The final code entering `PHANTOM` is can be seen in Appendix A, and the code for the other algorithms can be seen in Appendix B.

3.1 Radiation Transfer

The study of photons and their interaction with the medium they pass through, is the field of study called radiation transfer. This section is based on Gray (2005); De Ceuster (2022). Radiation transfer can be described using the radiation transport equation

$$\frac{\mathrm{d}I_\nu}{\mathrm{d}s} = \eta_\nu - \alpha_\nu I_\nu \tag{3.1}$$

where I_ν is the local intensity, η_ν and α_ν the local emissivity and absorption of the medium at a certain frequency ν and $\mathrm{d}s$ an infinitesimal section of the path along which the ray propagates. η_ν quantifies the amount of photons emitted in the local segment, and α_ν quantifies the amount of photons that are being absorbed. α_ν can be calculated using the local opacity and density $\alpha_\nu \equiv \kappa_\nu \rho$. For radiation, it is not the distance that is important, but how much light is attenuated over that distance. For this reason it is useful to express the problem as a function of a new unitless distance coordinate, namely the optical depth

$$\tau_\nu \equiv \int \alpha_\nu \mathrm{d}s = \int \kappa_\nu \rho \mathrm{d}s \tag{3.2}$$

which describes how much light is attenuated, independent on the emission in the system. The radiation transport equation then becomes

$$\frac{\mathrm{d}I_\nu}{\mathrm{d}\tau} = S_\nu - I_\nu \tag{3.3}$$

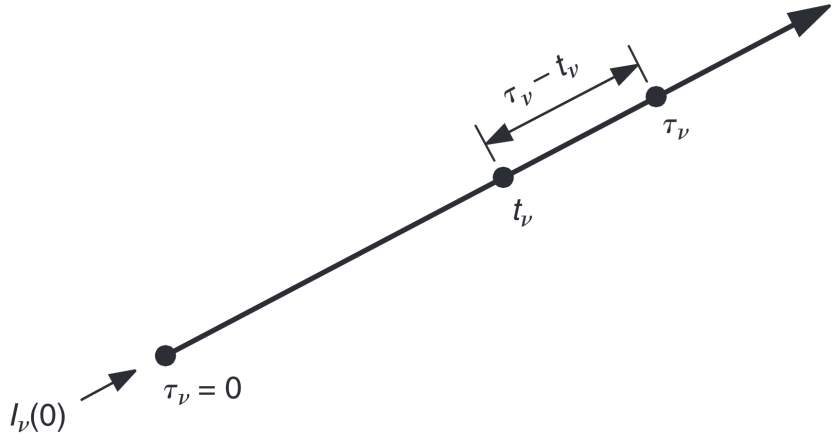


Figure 3.1: Visual representation of the formal solution of the radiation transport equation. (Gray, 2005)

where $S_\nu = \eta_\nu/\alpha_\nu$ is the source function. The differential equation (3.3) can be rewritten along a specific ray parameterised by the path length s, by integrating

$$I_\nu(s) = I_\nu(0)e^{-\tau_\nu(s)} + \int_0^{\tau_\nu(s)} S_\nu(s')e^{-[\tau_\nu(s)-t_\nu]}\,\mathrm{d}t_{nu} \tag{3.4}$$

where $I_\nu(0)$ is the intensity at the start of the ray. This equation is known as the formal solution and can physically be interpreted using Figure 3.1. The original intensity is attenuated, and e-folds with respect to the optical depth. At the same time new photons are sent out along the ray, given by the source function. But these new photons need to travel as well to the location of s, and thus are e-folded as well with respect to the correct optical depth.

If local thermodynamic equilibrium (LTE) is assumed, meaning that at any given location in the system, the local neighbourhood is in thermodynamic equilibrium, but not necessarily the entire system (global thermodynamic equilibrium), this source function acts as a black body at the local temperature, given by Planck's law

$$S_\nu = \frac{2h\nu^3}{c^2}\frac{1}{e^{h\nu/kT}-1} = B_\nu(T)\ . \tag{3.5}$$

When the source function is assumed to be constant throughout the medium, it can be taken outside of the integral and it can be solved exactly. The formal solution is then given as

$$I_\nu(s) = I_\nu(0)e^{-\tau_\nu(s)} + S_\nu\left(1 - e^{-\tau_\nu(s)}\right) \tag{3.6}$$

where this can again be interpreted as the initial intensity being attenuated and the source function being added to the local intensity.

3.1.1 Radiation pressure for a gray point source

In the case where only absorption is considered, this source function S_ν becomes zero as it is proportional to the local emission. The solution to the radiation transport equation reduces to

$$I_\nu(\vec{r}) = I_\nu(0)e^{-\tau_\nu(\vec{r})}\ . \tag{3.7}$$

If photons are absorbed, their momentum is absorbed as well. This can be interpreted as an acceleration on the medium, where the radiative force (also called radiation pressure or radiation acceleration) is given by

$$F_{\text{rad}} = \frac{1}{c} \int \mathcal{F}_\nu \kappa_\nu d\nu \tag{3.8}$$

where c is the speed of light, $\mathcal{F}_\nu$ is the local flux and κ_ν the opacity for the frequency ν.

When assuming a frequency independent (gray) case, and assuming the photons emerge from a point source, the equations become

$$\tau(\vec{r}) = \int_0^r \kappa\rho \mathrm{d}s \ , \quad \mathcal{F}(\vec{r}) = \frac{L}{4\pi r^2} e^{-\tau(\vec{r})} \ , \quad \vec{F}_{\text{rad}} = \frac{\kappa L}{4\pi c r^2} e^{-\tau(\vec{r})} \vec{\hat{r}} \ . \tag{3.9}$$

where L is the luminosity of the star, κ the opacity, c the speed of light and ρ the density. The first equation becomes frequency-independent since in a grey medium all frequencies react the same way. The second equation takes the flux of a point source ($\mathcal{F} = L/4\pi r^2$) which is attenuated by the optical depth. Finally the force can be obtained using equation 3.8.

For situations where radiation pressure is important, like for instance the AGB dust driven wind investigated here, the optical depth grows as well. If the optical depth grows, the radiation pressure e-folds, resulting in less radiation pressure further out in the medium. It is thus important to account for this attenuation of light.

3.1.2 Ray-tracing

Numerically, equation (3.9) needs to be solved using all the information in-between the point source and the position where the acceleration must be calculated. To do this a ray-tracer can be used. This means that a line is traced through the numerical spacial domain, and the properties along this so-called ray are incremented to obtain the information at the given location. For the optical depth, the local opacity and density must be summed along the line, until the location is reached. In our case, if we assume all the light to come from the AGB star, and approximate it as a point source, we can use this approach to calculate the optical depth to find the properties of equation (3.9).

3.2 MAGRITTE ray-tracer

SPH particles are not located on a grid, hence we require a ray-racer that does not require an underlying grid. One such ray-tracing algorithm is implemented in MAGRITTE (De Ceuster et al., 2020a,b). MAGRITTE is a 3D radiation transfer code, flexibly developed to handle both grid and gridless data, thus can work with SPH data. Ideally, MAGRITTE would be fully coupled to PHANTOM, but due to the immense computational cost of performing the full gas and dust radiative transfer at every hydrodynamical time step, the combination is not yet feasible. We have thus opted to extract only the ray-tracer to calculate the dust attenuation. The algorithm is visually explained in Figure 3.2. Starting from a position O, a ray is traced in the R-direction. The algorithm searches for the nearest neighbours, in this case the six points whose cell borders to the cell of O. From these neighbours it selects the point with the closest

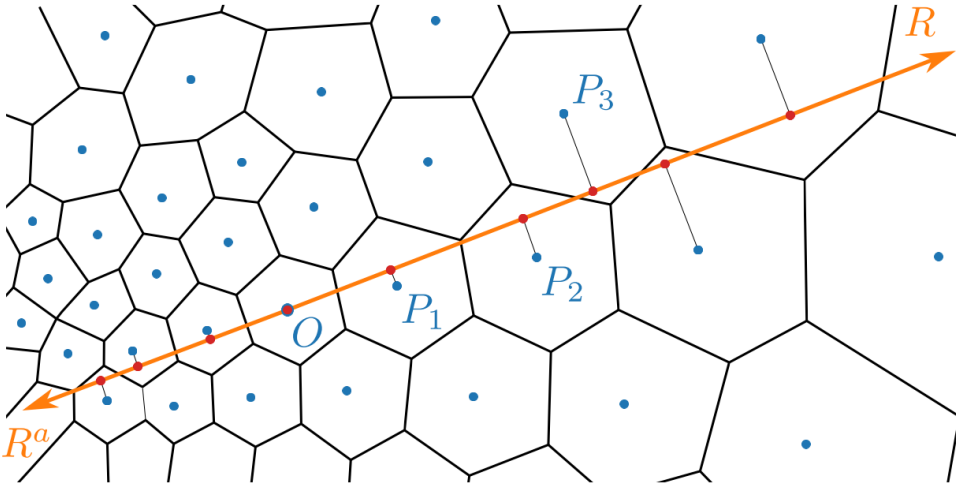


Figure 3.2: Visual representation of the ray-tracing algorithm of `MAGRITTE`. Starting in the point O, and tracing the ray in the R-direction, reaching P_1, P_2, P_3 and so on. Figure adapted from De Ceuster et al. (2020b)

perpendicular distance to the ray, provided that the point lies in the direction of the ray. This search is repeated until a given location, or the edge of the numerical domain is reached.

To calculate the optical depth, we need to calculate the integral

$$\tau = \int_{R_*}^{r} \kappa\rho \, \mathrm{d}s \ . \tag{3.10}$$

To calculate this integral, a segment will be defined. A segment is taken to be the space in-between two subsequent points on the ray, for instance in-between O and P_1 in Figure 3.5. This integral can be approximated by a sum over all the segments created using the ray-tracer. The optical depth can then be discretised as

$$\tau = \sum_{R_*}^{r} \kappa_i \rho_i \ \Delta s_i \tag{3.11}$$

where Δs_i is the length of the segment, κ_i the opacity in the segment and ρ_i the density in the segment. Here Δs can easily be calculated from geometrical considerations, κ_i and ρ_i require a little more attention.

The opacity of a segment is calculated as the mean of the opacity at the starting point, and the ending point of the segment. As the implementation is done in an SPH environment, the opacity is calculated using the SPH smoothing kernel (equation 2.8)

$$\kappa(\vec{r})\rho(\vec{r}) = \sum_{j \in \text{ SPH neigh}} m_j \kappa_j W(\vec{r} - \vec{r}_j, h_j) \ . \tag{3.12}$$

3.3 Ray-Tracer: best physical representation

In this section we describe how to best represent the physical problem via the discretised calculation of τ using the above mentioned algorithm. This way is very computationally

intensive, but it serves as a reference solution to quantify the accuracy of later necessary simplifications to speed up the calculation.

The ray-tracer as explained above does not make a specific assumption on how many nearest neighbour to use in calculating the segments along the ray. If the full set of SPH neighbours are considered, the segments making up the ray will on average be larger than when a more compact set of neighbours is used. By taking a smaller set of neighbours, local conditions are better traced as smaller segments are taken. Ideally we would thus use the minimal set of neighbours. These can be found using the Delaunay triangulation (Delaunay, 1934), which is a way to connect particles via triangles such that all the angles are as small as possible. All particles that are directly connected to the original particle, are considered to be the nearest neighbours. Delaunay triangulation is computationally extremely expensive, but gives the smallest possible set of neighbours. If we use this representation, we minimise the segments sizes, and hence best trace the physical conditions along the ray, which means that the ray-tracer gives the most accurate result.

To calculate τ for each point in the numerical domain, which in the context of SPH would be each individual SPH particle, the most physically correct approach is to trace a ray for each particle to the AGB star. A visual representation of what this looks like for 4 particles can be seen in Figure 3.3. In this way, τ is immediately found for all the particles. This implementation is similar to the algorithm of Kessel-Deynet & Burkert (2000). However, This algorithm is very computationally intensive, and can not be done in real-time for our simulations. Because it represent the most physical description of the discretised problem, we shall henceforth use it as reference to which the speedup adaptations presented in the following sections will be compared. We shall refer to this setup as the inwards ray-tracing scheme (IRS).

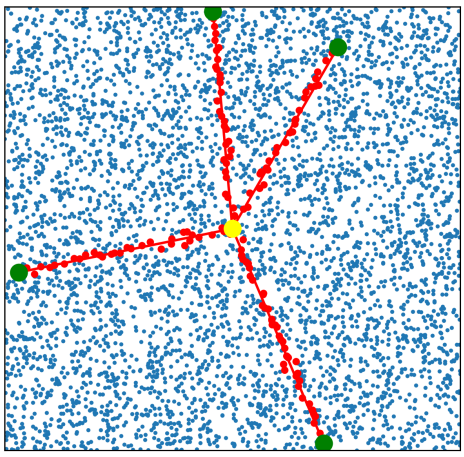


Figure 3.3: Visual representation of the rays traced for the reference solution.

3.4 Ray-tracer: optimising computation time

To speed up the calculation, multiple aspects of the algorithm have been optimised. To quantify the performance of the optimised module, an error analysis is done using an already existing SPH AGB binary simulation, namely dump number 600 of model v05e50 ($v_{ini} = 5$ km/s, $e = 0.5$) of Malfait et al. (2021). This is a far evolved snapshot of the most complex morphology found in this study containing 1 182 044 SPH particles, so that as many structures as possible are sampled in the error analysis. For the entire chapter, when using the relative error, this is the relative error on τ given as

$$\delta_{rel} = \frac{1}{N_{part}} \sum_{i}^{N_{part}} abs\left(\frac{\tau_{IRS,i} - \tau_i}{(\tau_{IRS,i} + \tau_i)/2}\right) \quad . \tag{3.13}$$

3.4.1 Neighbour-finding algorithm

The Delaunay triangulation of the cloud of SPH particles is not calculated by the SPH solver

in PHANTOM, and is therefore computationally expensive. However, since PHANTOM already constructs a kd-tree at the beginning of each timestep, it seems reasonable to use this pre-existing data structure to obtain the SPH neighbours. The SPH neighbour set contains more particles than the Delaunay neighbour set, so the sampling of the local conditions along the ray is done more crudely. Hence the SPH neighbours results in a less accurate solution, where this relative error can be quantified as 1% with respect to the ray-tracer that uses the Delaunay triangulation. The larger sampling, results in less loops to find the next point on a ray, but every loop is more computationally expensive as more neighbours are looped over. Both algorithms are thus approximately equally fast if the calculation time of the neighbour set is not taken into account. But since the Delaunay triangulation must still be performed, a speedup by a factor 2 is reached when the pre-existing kd-tree is used. This makes this approach preferable over the original IRS.

3.4.2 Inward versus outward tracing

Tracing rays from the particles to the AGB star is also inefficient because the optical depths obtained for the particles closer to the AGB star are re-calculated when considering particles further out. A way to get rid of this inefficient calculation is by tracing a pre-determined number of rays starting from the star outwards, and interpolating τ on these rays to the particles that were not hit by a ray. A visual representation can be seen in Figure 3.4 using only three rays. This algorithm is based on the internal structure of most ray-tracing 3D radiation transfer algorithms (e.g. Altay & Theuns, 2013; De Ceuster et al., 2020b), where for each particle rays are traced outwards, to find the influence of its surroundings. But since we're only interested in the radially outward propagation of photons from the AGB star, we only trace the rays associated with this geometry, inverting the problem. Starting from the surface of the star (at radius R_*), rays are traced uniformly outwards, and the optical depth is calculated, accumulated and saved for each segment, for each ray. Then, for each particle in the numerical domain the closest ray is found and its value for τ is obtained by radially projecting the point onto the ray, and the value of τ at this radial projection is given to that particle. Most of these projections will not directly coincide with the saved optical depths along the ray, so τ is calculated as the linear interpolation using the radial distance from the star to the two closest segments. The accuracy of this approach highly depends on the number of rays, as well as their spatial distribution. However, the ability to choose these sampling properties can be used to finetune the accuracy vs. the computational cost of the simulation.

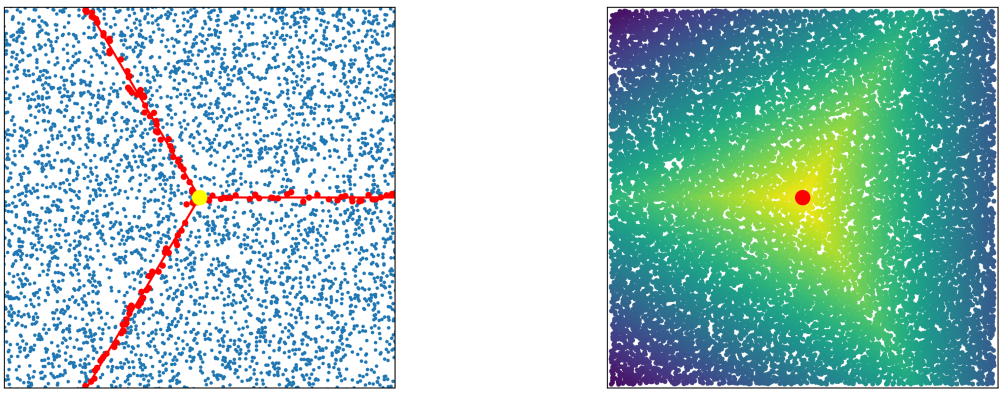


Figure 3.4: Visual representation outwards ray-tracing scheme, for $n_{rays} = 3$. The rays are shown on the left, and the optical depth on the right (darker is more optical depth).

3D uniform ray distribution

To obtain a 3D uniform distribution of rays, we use the `HEALPix` (Górski et al., 2005) package, designed specifically for this problem. Tracing rays uniformly in 2D is rather trivial: the full 2π is divided by n_{rays}, so that there is a ray each $2\pi/n_{rays}$ (for 3 rays, see Figure 3.4). In 3D this is less trivial. `HEALPix` divides the full 2-sphere into uniform pixels of equal area, with unit vectors pointing to the center of each pixel (see Figure 3.5). These unit vectors can be used as directions for the rays. In the nested scheme, `HEALPix` can refine the spatial division by increasing the number of pixels such that each pixel area is split into four. Each time this happens, the order of the scheme increases by one. So, at order zero, the 2-sphere is divided into 12 pixels, and each increase in order augments the number of pixels to $n_{rays} = 12 \cdot 4^o$ with o this `HEALPix` order. `HEALPix` also provides a routine to get the pixel associated with every point on the 2-sphere. This means that when the pixel unit vectors are used as directions along which to trace our rays, every point in the simulation can automatically (computation time of order $\mathcal{O}(1)$) be associated with a ray.

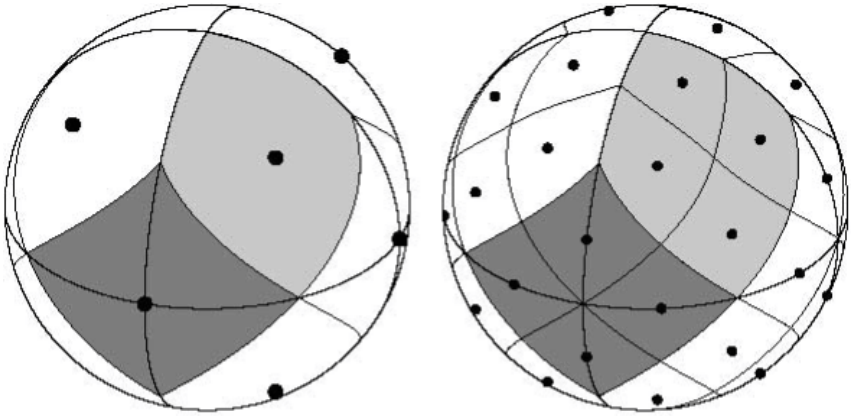


Figure 3.5: Visual representation of the `HEALPix` pixels for $o = 0, 1$. (Górski et al., 2005)

Interpolation accuracy

When tracing rays outwards, as opposed to inwards (Section 3.4.1), there is not always a ray passing through every point in the simulation. Interpolating every point onto the chosen rays is inevitably associated with a loss in accuracy. The more rays that are traced, the closer a ray is situated to every particle, the more accurate the interpolation. Therefore we expect a decrease in relative error with an increasing the number of rays. On the other hand, tracing the rays takes up most of the computation time, therefore the computation time increases significantly when tracing more rays. This effect

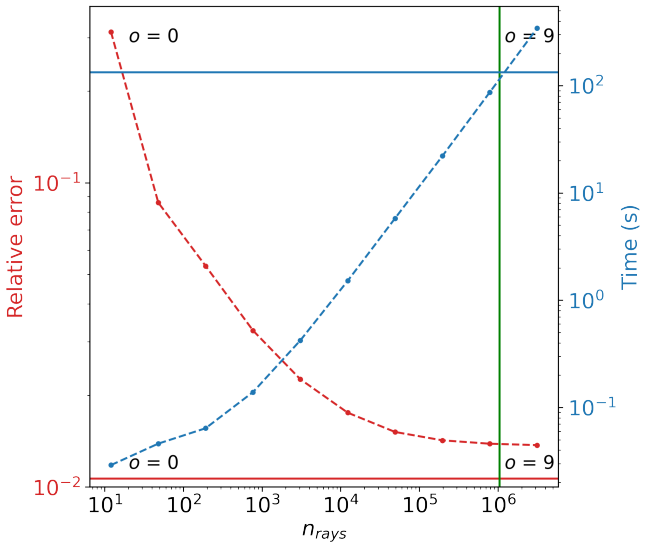


Figure 3.6: The relative error and the computation time as a function of n_{rays}. The solid lines represent the values the SPH neighbours, the dots represent the outwards algorithm for increasing o.

can be seen in Figure 3.6, where the relative error between the HEALpix scheme and the IRS, as well as the computation time are shown as a function of the number of rays. Every point on the curve represents subsequent `HEALPix` order, starting from $o = 0$ to $o = 9$. As expected, the computation time increases with increasing the number of rays, whereas the relative error converges to the 1% error made upon taking the SPH neighbours.

In Figure 3.6, for $o \leq 8$ ($n_{rays} = 786\,432$), the computation time of the algorithm is less than the computation time for the IRS. This plot can be used to optimise the balance between speedup and loss of accuracy. For instance, taking `HEALPix` order 5 results in a speedup by a factor 250 with respect to the IRS, but only amounts to a loss of accuracy of 2%, which is only an additional percent to the expected error associated with the expansion to the SPH neighbours described above. For our calculations, we consider these conditions the computational sweet spot.

As a next step, two extra improvements for this algorithm are presented as well. First adaptive ray-tracing is investigated which makes it possible to sample more rays in crucial regions. Secondly an improved ray-interpolation is investigated, which looks at different interpolation schemes to interpolate particles on rays.

3.4.3 Adaptive ray-tracing

Uniform ray-tracing might not be ideal, as it is possible that there exist regions with a lot of particles and structure, where it is important to sample rays more densely, as well as the reverse. To handle this type of variability, adaptive ray-tracing can be implemented, much like the adaptive mesh refinement that is done in AMR codes. There are multiple ways to approach adaptive ray-tracing, where here we base ourselves on the work done by De Ceuster et al. (2020a) that determines the distribution of rays before tracing them outwards. Alternative adaptive ray-tracing algorithms exist (e.g. Abel & Wandelt, 2002) (used for instance in Altay & Theuns, 2013) that refine rays during the ray-tracing. The algorithm of De Ceuster et al. (2020a) starts from a chosen minimal order o_{min}, and refines pixels that satisfy a certain criterion until a maximal order o_{max} is reached. This maximal order is given by the refinement level (o_{ref}), where $o_{max} = o_{min} + o_{ref}$. An example can be seen in Figure 3.7 that represents the 2-sphere projected onto a 2D plane.

`HEALPix` does not keep track of what order all the rays are at, so this adaptive approach loses

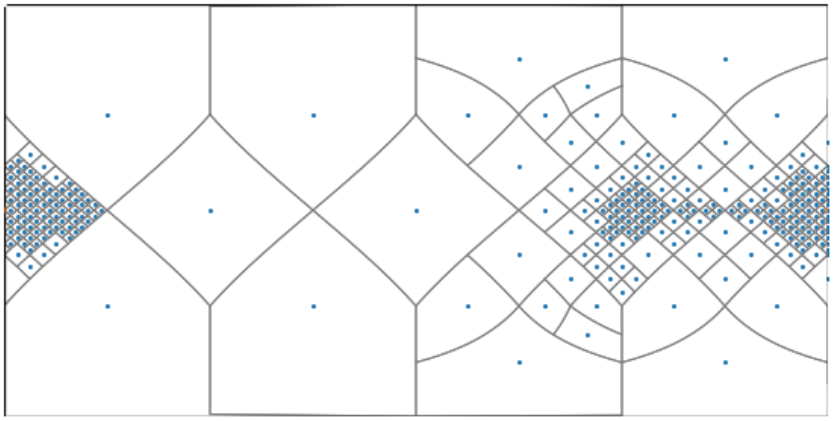


Figure 3.7: Example of the rays that is traced in the adaptive ray-tracing algorithm. It represents the full 2-sphere in the Cartesian projection onto the 2D plane.

the advantage to immediately find the ray closest to any point. This can be solved by keeping track of what rays are being sampled: We construct a list containing the unit vectors of each ray is be traced. At the same time we construct an array where each element represents a pixel at the level o_{max}. In each element of this array there is an link to the ray representing this pixel. After the ray-tracing is done and the closest ray for a particle is searched, the closest pixel in o_{max} is found first, so that the correct ray can be found using this newly constructed array.

We use the cumulative particle column density along a ray as refinement criterion, since it is reasonable to expect that more particles (= more mass) along the column will result in larger changes in τ. We investigate two refinement approaches. The first follows the algorithm implemented by De Ceuster et al. (2020a), where the 2-sphere is initially homogeneously discretised at an initial refinement level o_{min}. The pixels are ordered according to their column density, and the densest half of them are refined. This is repeated until a desired maximal refinement is reached. This criterion we call the halving criterion. The second criterion uses a more physical reasoning: in each pixel the particle column density is calculated, and all pixels with column density larger than the spatially average column density ($n_{cell} > N_{part}\frac{4\pi}{n_{rays}}$) are refined. This criterion we call the overdense criterion.

The performance of both refinement criteria are shown in Figures 3.8 and 3.9. On the left figure the connected dots represent the same refinement level (the same o_{ref}), where each subsequent dot represents the next o_{min}. On the right figure, connected dots have the same base distribution (the same o_{min}) where each subsequent dot represents the next refinement level. The first (leftmost) dot of each color represents no refinement ($o_{ref} = 0$, a uniform distribution). We wish to have an algorithm that performs better than the uniform ray-sampling, meaning that it is preferred to increment o_{ref} rather than o_{min}, and this is what we will call an improvement.

Starting with the halving criterion (see Figure 3.8), first looking at the figure on the right. The blue line here represents the uniform sampling (as it has $o_{ref} = 0$). It can be seen that for

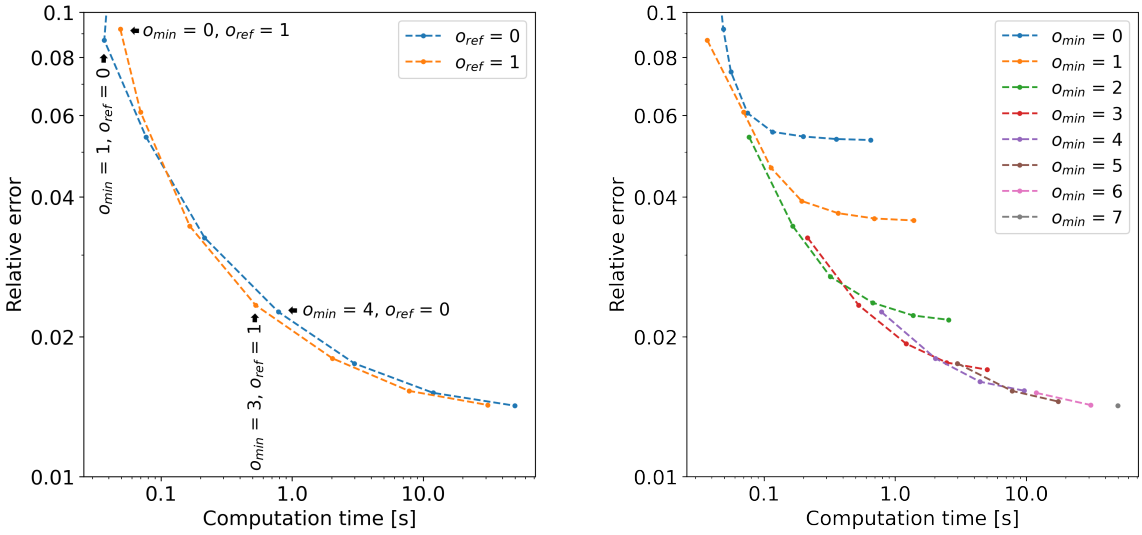


Figure 3.8: Performance of the adaptive ray-tracing scheme, using the halving criterion on the left, and the overdense criterion on the right. Connected dots represent the same o_{min} where each subsequent dot represents an extra o_{ref}, starting at 0.

$o_{min} < 2$ (the first two dots, starting leftmost), the $o_{ref} = 0$ curve performs more desirably compared to the $o_{ref} = 1$ curve ($o_{min} = 1$, $o_{ref} = 0$ is computationally more interesting as $o_{min} = 0$, $o_{ref} = 1$). This happens since the overhead computation time to find the important rays, takes longer compared to the computation gain from tracing less rays. For higher o_{min}, the adaptive scheme performs slightly better than the uniform scheme. The behaviour at higher o_{ref} can be understood using the figure on the right. Following connected dots (following a constant o_{min} for increasing o_{ref}) starting with the upper-leftmost dot, the same improvement for low o_{ref} can be seen for for instance $o_{min} = 3$, where the first refinement level is an improvement as explained before, but if o_{ref} keeps on increasing the improvement in accuracy stagnates. This is because the increase in accuracy due to the refinement becomes negligible compared to the left-over error in the previous orders. This indicates that the refinement criterion is not well suited for high refinement.

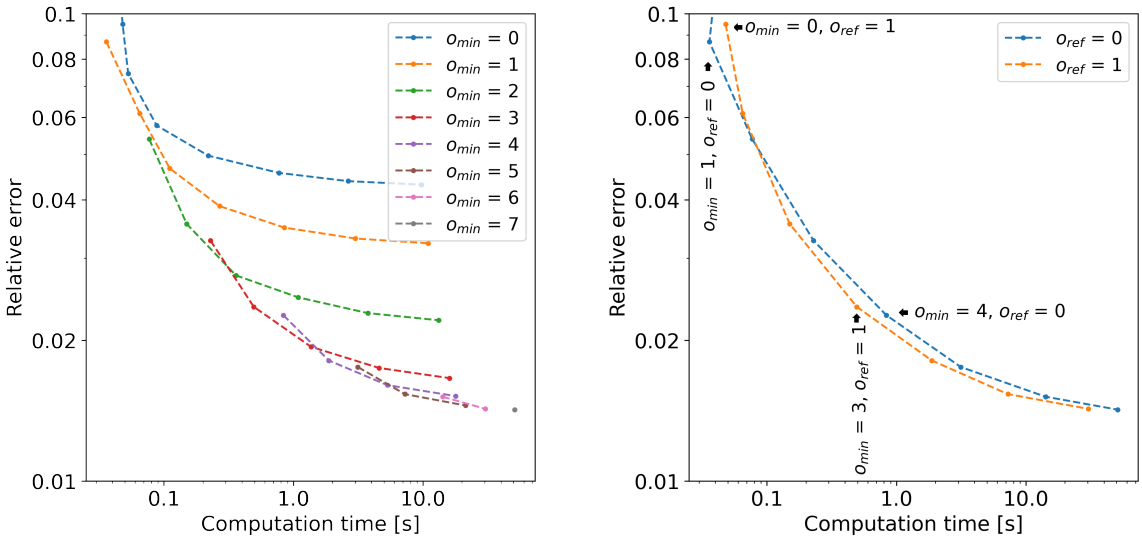


Figure 3.9: Performance of the adaptive ray-tracing scheme, using the halving criterion on the left, and the overdense criterion on the right. Connected dots represent the same o_{min} where each subsequent dot represents an extra o_{ref}, starting at 0.

For the overdense criterion, also no improvement (Figure 3.9 on the right) is found for $o_{min} < 2$ for the same reason as in the halving criterion, although for $o_{min} = 1$ there is a measurable decrease in computation time. For higher o_{min} again the curve $o_{ref} = 1$ shows a more desirable performance. This improvement is more significant as for the halving criterion, as the two curves shown lay further apart. Now investigating even higher values of o_{ref} in the figure on the left, the same phenomenon occurs where the relative error stagnates and though the relative error stagnates at a lower value, the computation time is significantly higher as well. This is because by refining overdense regions, a significant amount of subpixels will be overdense as well. This results in most of the rays being in the order o_{min} or o_{max} meaning that computation time and relative error start to follow the behaviour of o_{max}, while still carrying the relative error of the o_{min} region. This means that the refinement criterion performs worse for high refinement, but better for low refinement compared to the halving criterion.

Overall the improvement associated with adaptive refinement is not significant. However, it is possible that a better criterion can be found that makes the adaptive refinement approach worthwhile. This is something for future research. Because we have not been able to find such

a criterion, we have decided to not make use of the adaptive approach, but rather investigate the effect of ray-interpolation.

3.4.4 Ray-interpolation

Multiple rays can be used for the interpolation, in stead of just one. Doing so should better capture way variation of τ in the medium in-between rays. We investigate the effect of using either the 4 closest rays, or the 9 closest rays. Example rays can be seen in Figure 3.10. `HEALPix` provides a routine to calculate the nearest rays for each ray, giving us the nine closest rays out of which the four closest can be filtered out. Interpolation in-between the rays is done following

$$\tau_i = \frac{1}{\sum_j r_j^{-k}} \sum_j \frac{1}{r_j^k} \tau_j \tag{3.14}$$

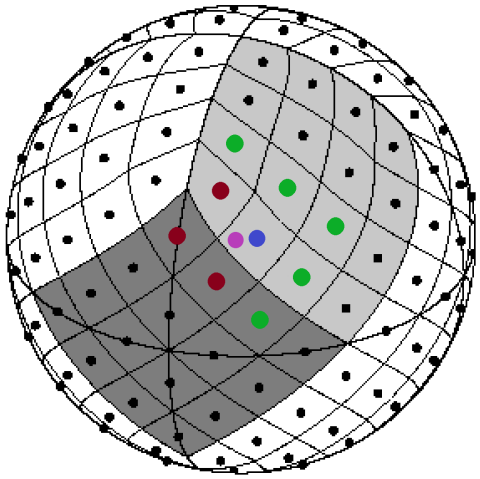


Figure 3.10: Representation of the closest rays to a particle, where the pink dot represents the particle, the blue dot the closest ray, the red together with the blue the 4 closest rays and red blue and green the 9 closest rays.

where r is the perpendicular distance from the particle to the ray and k represents the weighting of interpolation. For lower values of k, the effect of rays further away from the particle have a stronger impact on the final τ. This means that for smooth features the ray-interpolation works better, resulting in a lower relative error. We investigate weighting according to $k \in [1, 2, 3, 4]$.

In Figure 3.11 the relative error is plotted as a function of the number of rays used in the interpolation, for $o = 5$ (see 3.4.2). The relative error for the interpolation for 9 rays, compared to 4 rays, does not improve. in fact, for $k = 1$ it even becomes slightly worse. However, the improvement of the interpolation using 4 rays is significant compared to only 1 ray. We thus conclude that using a four-point interpolation is preferable over either a 9-point interpolation, or a closest-ray interpolation. Looking at the different values of k, the improvement in relative error is largest for high values of k, we therefore conclude that low values of k are to be preferred over high values of k.

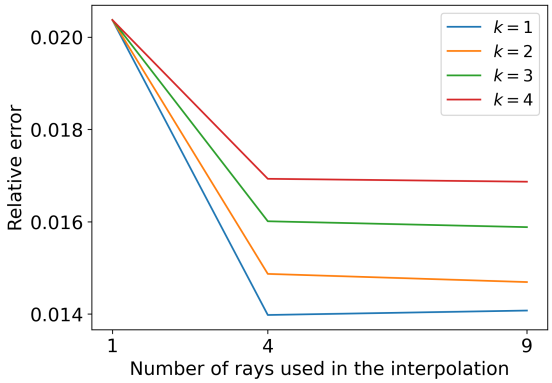


Figure 3.11: Relative error as a function of the number of rays used in the interpolation for $o = 5$ and different values of k.

While aiming for a lower relative error with less computation time, a careful assessment of the smoothing of sharp edges should be taken into account, as we expect these to occur in our simulations, for instance close to the companion. To investigate how well sharp edges are preserved, an artificial local sphere of extreme high opacity is placed in the system. This creates a sharp-edged shadow behind the sphere (similar to the artificial shadow explained in

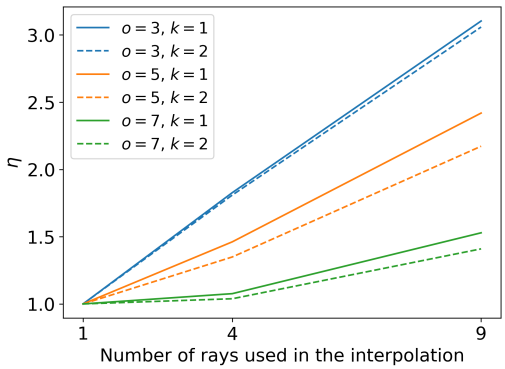


Figure 3.12: η (see text) in function of the number of rays used in the interpolation.

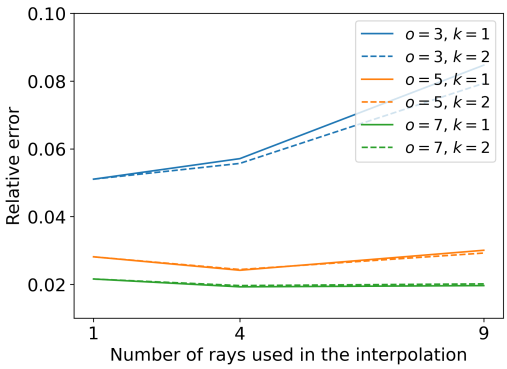


Figure 3.13: Relative error of the model with an artificial high opacity sphere.

Section 3.5.3, see Figure 3.19). To evaluate the ability to conserve the sharp edge of this shadow, we count how many particles have an optical depth that differs sufficiently from the reference model to have a relative error above 1. This number is scaled to the number obtained for interpolating with 1 ray, and we call this fraction η. η will describe the preservation of this sharp edge, as if it is smoothed out, as if it is smoothed the lower part of the shock is significantly increased and the higher part of the shock is significantly reduced. The result can be seen in Figure 3.12, where η increases when accounting for more rays in the interpolation. This is expected since taking more rays into account smooths the sharp edge more. In this figure order 3, 5 and 7 are shown, where the increase of η is reduced for higher order. This is also expected, as the spacial size of the interpolation becomes smaller for higher orders thus better sampling the sharp features in the model. η is also dependent on the chosen k, as for lower k the relative influence of rays at a further position is felt less, resulting in less smoothing. In Figure 3.13 the relative error is shown for the model with an artificial sphere of high opacity. The relative error increases when taking 9 rays, compared to 4 due to the extra smoothing of the shocks. This is why the relative error increases compared to the reference model for $k = 1$ as well, as $k = 1$ smooths the shocks the most. For this reason we conclude that the interpolation performs best using 4 rays. Since we expect to have some sharp edges in the simulation (like for instance close to the companion), we conclude that $k = 2$ provides the optimal balance between accuracy and edge resolution.

Using the four-point interpolation with a weighting of $k = 2$, the relative error as a function of the non-adaptive `HEALPix` order is shown in Figure 3.14. For low order, the improvement in accuracy due to the extra interpolation is not worth the increase in computation time. This extra computation time comes from taking multiple interpolations for each particle. For $o > 2$, the increase in computation time is small compared to the significant improvement in accuracy.

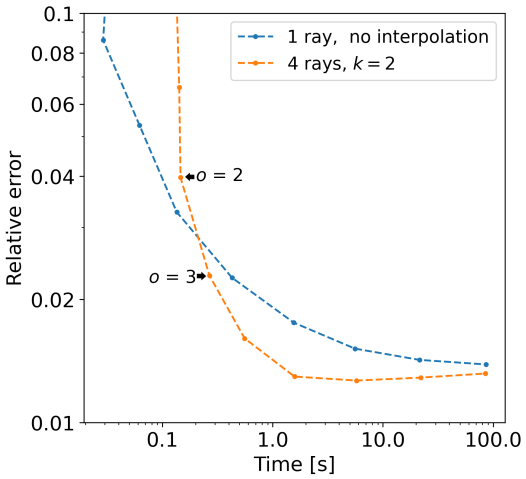


Figure 3.14: The relative error as a function of the computation time. Each dot represents a subsequent o, for both no interpolation, as 4 rays $k = 4$.

3.5 Final Ray-tracing algorithm and implementation in PHANTOM

SPH nearest neighbours from the pre-calculated kd-tree improve the computation time by a factor 2. Tracing rays outwards using the HEALPix discretisation of the 2-sphere can result in a significant speedup, while keeping the relative error with respect to the IRS small. We found that adaptive ray-tracing can result in a marginal improvement in computing time, but only for low refinement ($o_{ref} = 1$). Finally, increasing the rays that contribute to the interpolation to 4, and weighting the contribution of each ray by the inverse square of the distance, was found to give the best compromise between increasing the accuracy of the interpolated value of τ while safeguarding sharp edges that might exist in the numerical set-up. Given that adaptive ray-tracing does not provide a significant improvement to the computation time, and ray-interpolation does, we have chosen to adopt ray-interpolation in the total algorithm.

Having obtained out optimal configuration of the ray-tracer, the coupling with PHANTOM was done. The algorithm as implemented in PHANTOM can be found in Appendix A. The calculation of τ is done when the kd-tree is already built. This means that the calculation can only be done in the second part of the PHANTOM timestep (see Section 2.3.2), and stored for each particle. As radiation pressure is a long range force, it is called upon in the substeps. During a substep τ is assumed to be constant as we expect no significant changes during a full timestep.

3.5.1 Scaling

In scientific computations parallelization is of extreme importance. This means that computations are split over multiple threads of execution or computational threads, often just referred to as threads, to divide the calculation and reduce computation time. As PHANTOM is OpenMP parallelized, the ray-tracer was also OpenMP parallelize. The loop over all the rays, as well as over all the particles is parallelized. To test how well this parallelization performs, a scaling test is done which can be seen in Figure 3.15 for both PHANTOM and the ray-tracer.

Our ray-tracing algorithm clearly scales much better than PHANTOM. We investigate the relative fraction of computation time of this algorithm for HEALPix order 5 compared to the computa-

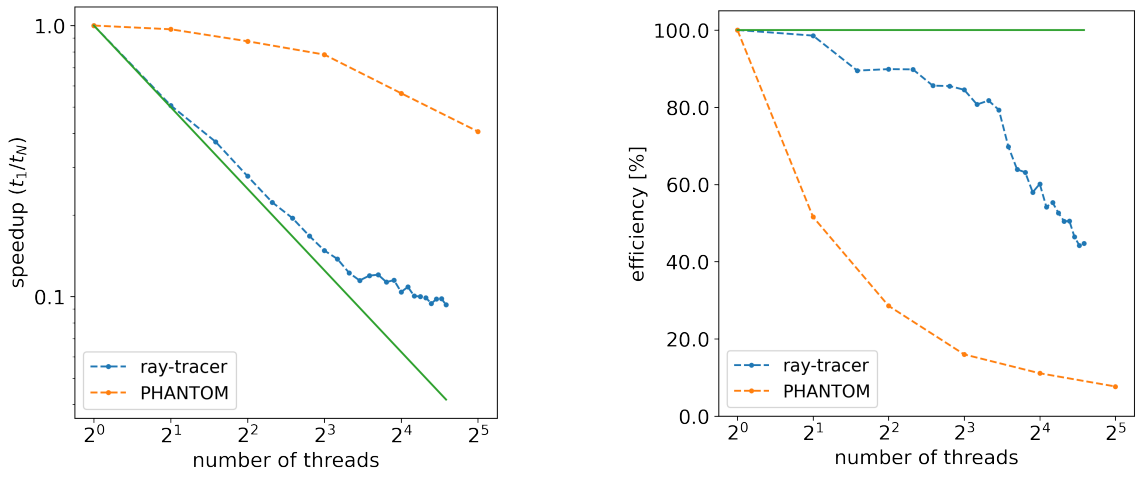


Figure 3.15: Scaling test of the ray-tracer in blue, PHANTOM in orange, and the ideal scaling in green.

tion time of a typical PHANTOM timestep, as a function of the number of threads. This can be seen in Figure 3.16. When only 1 core is used, the calculation of τ takes about 40% times the computation time of a typical PHANTOM timestep that does not call upon the ray-tracer. Increasing the number of threads this decreases to just under 10%. This implies that under fully parallelised conditions, i.e. beyond 8 threads, the total expected increase in computation time of the entire simulation will also be of the order of about 10%. Given the significant improvement of the physics that such a ray-racer can provide, the additional cost of 10% computing time is considered worthwhile.

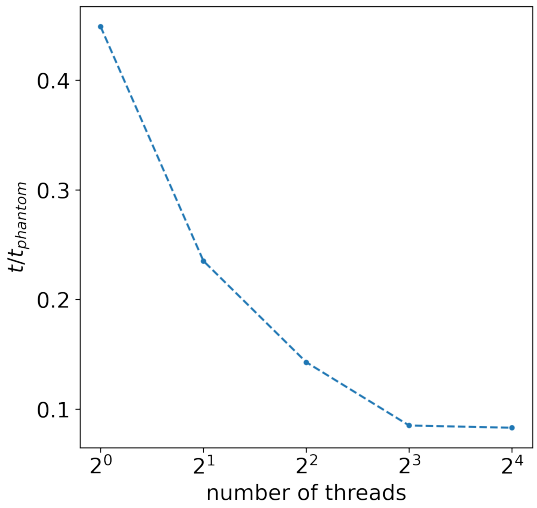


Figure 3.16: Time fraction of the computation time of τ and a usual PHANTOM timestep

3.5.2 Benchmarking

To test the implementation in PHANTOM the ray-tracer was benchmarked to the analytic solution of a stationary wind (without companion) with an added correction factor of $e^{-\tau}$ attenuating the Eddington factor. The result can be found in Figure 3.17, where the dots represent the solution of the SPH solver, and the lines represent analytic (1D) solutions to the stationary wind equations. Here the blue dots show the SPH gas velocity without dust attenuation of the acceleration, and the red dots with the ray-tracer active. The dots follow the curves of the analytic solutions well showing that the implementation of the ray-tracer into PHANTOM acts as predicted by the stationary wind solution. The attained terminal velocity for the solutions with and without dust attenuation are significant, seen as an increase of more than 20%. This illustrates the importance of properly accounting for this dust attenuation when researching the dynamics of dust-driven stellar winds.

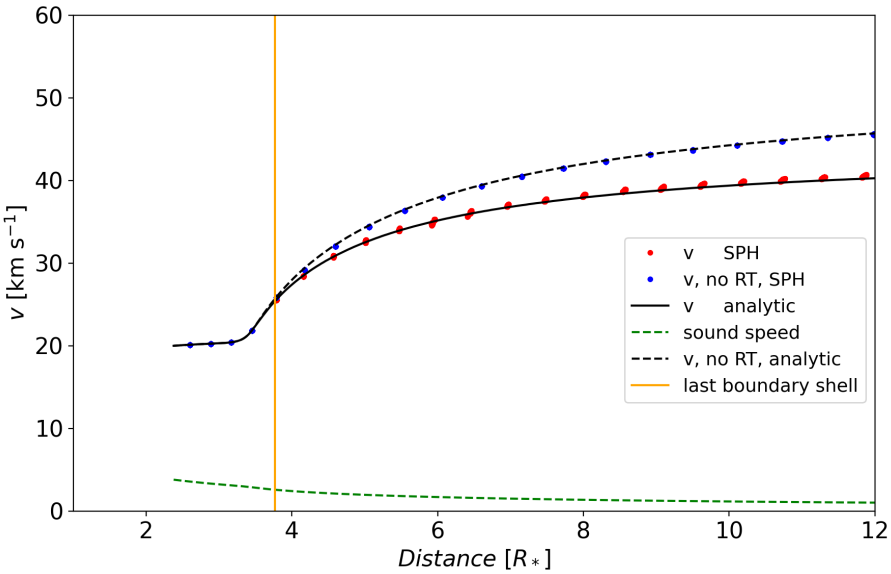


Figure 3.17: Bench marking test describing a spherical outflow.

3.5.3 Geometrical a shadow behind the companion

In a binary simulation, the physical body of the companion will cast a shadow radially outwards. As the companion is just a sink particle in the simulation, this shadow is not yet formed. To ensure the shadow is created, the shadow is artificially enforced from geometrical considerations.

At the beginning of the calculation, the opening angle of the companion θ_0 (see Figure 3.18) is calculated based on the input physical size of the companion R_comp. For each particle the angle θ between the center of the companion and the particle, as seen from the AGB sink vantage point is calculated, and checked if it is smaller than θ_0.

$$\theta_0 = \sin^{-1}\left(\frac{R_{comp}}{r_{comp}}\right) \ , \quad \theta_i = \cos^{-1}\left(\vec{r}_{comp} \cdot \vec{r}_i\right) \ , \tag{3.15}$$

where R_{comp} is the radius of the companion, $\vec{r}_{comp}$ is the distance vector from the AGB star, to the companion. If this is true, the ray is inside the cone, but can still be in front of the companion. Therefore the radial distance is is checked compared to r_{sphere}, which is given by

$$r_{sphere,i} = r_{comp}\cos(\theta_i) - \sqrt{R_{comp}^2 - r_{comp}^2\sin(\theta_i)^2} \ . \tag{3.16}$$

If a ray passes through the companion, the ray is halted at $r = r_{sphere}$. This creates a geometric shadow behind the companion, as can be seen in Figure 3.19.

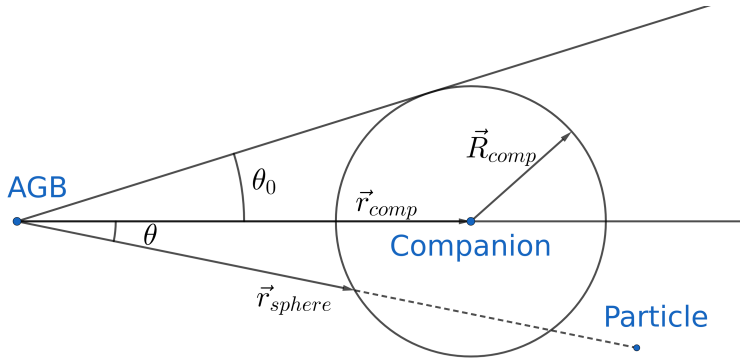


Figure 3.18: Geometrical representation of the variables used to construct a shadow.

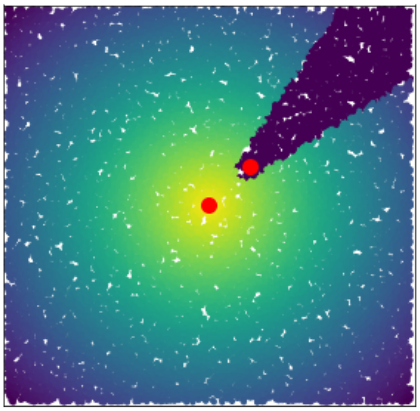


Figure 3.19: A shadow constructed behind a companion

Chapter 4

Impact on the morphology

Now that the optical depth can be calculated, we can run actual binary simulations with PHANTOM. As a proof of concept, the parameter setup following one of the models in Chen et al. (2020) is used and can be found in Table 4.1. First a reference simulation is ran where Bowen dust opacity profile is used, which can be seen in Section 4.1. Next the ray-tracing is activated, to attenuate the wind acceleration represented by the Eddington factor in Section 4.2. Finally in Section 4.3 the same simulation is re-iterated using cooling.

Table 4.1: Parameters used in the simulation.

M_1 [$M_\odot$]	M_2 [$M_\odot$]	R_{eff} [au]	v_{ini} [km/s]	$\dot{M}$ [$M_\odot$/yr]	T [K]	L_1 [$L_\odot$]
1.5	1	1.27	7	$5 \cdot 10^{-7}$	3000	4300

4.1 Reference model

As a first reference simulation, the ray-tracer is not yet activated. Here the Bowen dust opacity is used, representing a C-rich outflow with $T_{cond} = 1500$ K, $\delta = 60$ K and $\kappa_{dust,max} = 6$ g/cm^2. Together with the optically thin Γ_d, α is still in the simulation and set to 1. This is done to ensure an outflow of material since we have no pulsations (see Section 2.4). The results of the simulation can be seen in Figure 4.1 where the density is plotted in a slice through the orbital plane and perpendicular to orbital plane. A spiral structure is obtained in the orbital plane, and a shell structure in the edge-on plane. This spiral arises due to the gravity of the companion, where there are two edges, one leading and one tailing the spiral arm. Both edges have a different outwards moving velocity, and eventually merge. The same can be seen in the edge-on plane, where the each shell is the presence of the high density spiral arm.

Radial plots of the density, radial velocity, as well as the temperature along both the x-axis (through both stars), are shown in Figure 4.2. Along this axis a lot of structure can be seen in the density as well as the temperature, where the spiral arms are located. The temperature (especially close to the companion) here is high, as there is no cooling activated. The radial velocity is relatively stable throughout the spiral arms, and is about constant on larger scales than the semi-major axis.

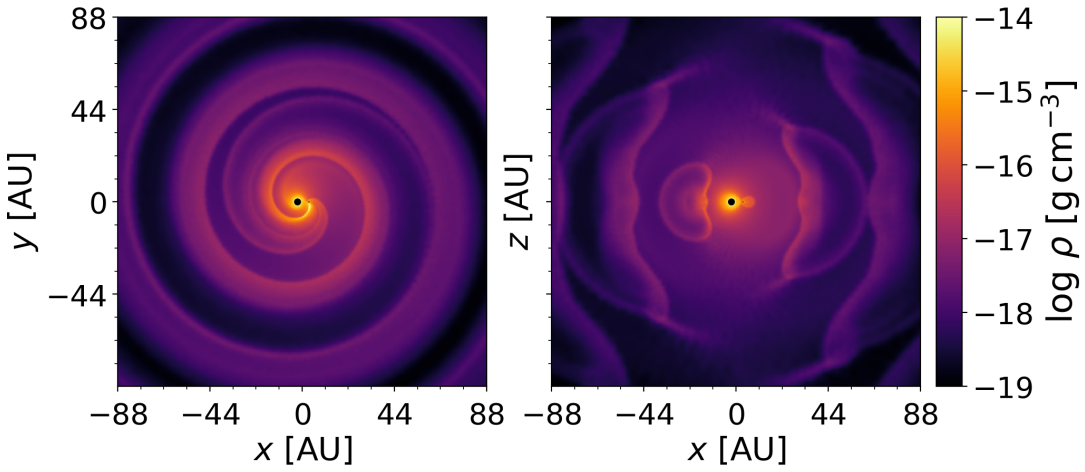


Figure 4.1: Density of the simulation for a Bowen dust profile in a slice through the orbital plane on the left, and perpendicular to orbital plane. Both slices go through both stars, along the x-axis.

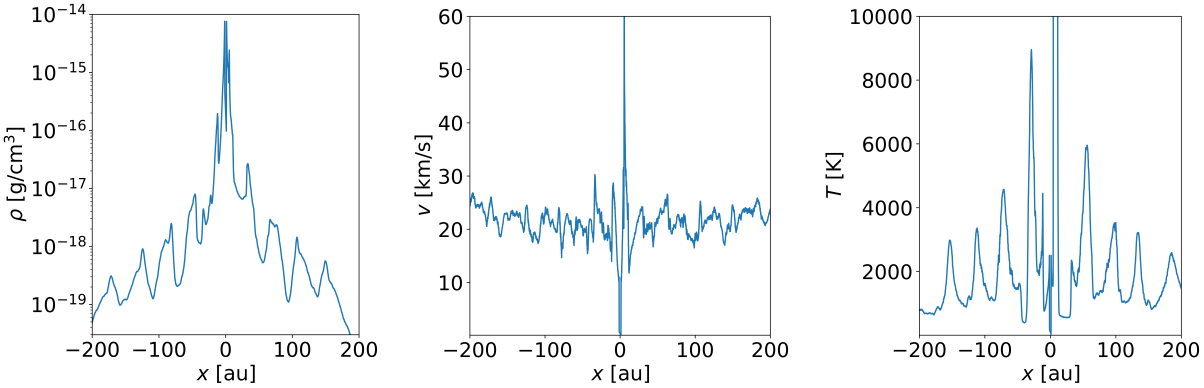


Figure 4.2: Radial structure plots showing the density, velocity and the temperature along the x-axis of the reference simulation.

4.2 Reference model + ray-tracer

In this simulation, the ray-tracer as explained in Section 3.5 is activated using order 5 (see Section 3.5.1). The result of the simulation can be found in Figure 4.3 where the density, opacity and optical depth is plotted, again for the orbital plane and the edge-on plane. When looking at the optical depth, this should be monotonously increasing as a function of radius. This is not the case as the same structures as in the density are visible. This is due to a plotting artefact as the resolution in SPH scales with density, and not the actual optical depth changing in these regions. Radial structure plots can be seen in Figure 4.4. The difference between the previous simulation and this simulation is small. This is because α is still put as 1 at the moment, so the morphology is still dominated by this parameter. By pushing α to 0, changes in the morphology might become visible, but unfortunately time constrains have prevented us from exploring how low we can push the value of alpha to enhance the effect of the ray-tracer.

The optical depth τ can be seen in Figure 4.3 on the bottom row. Most of the attenuation of

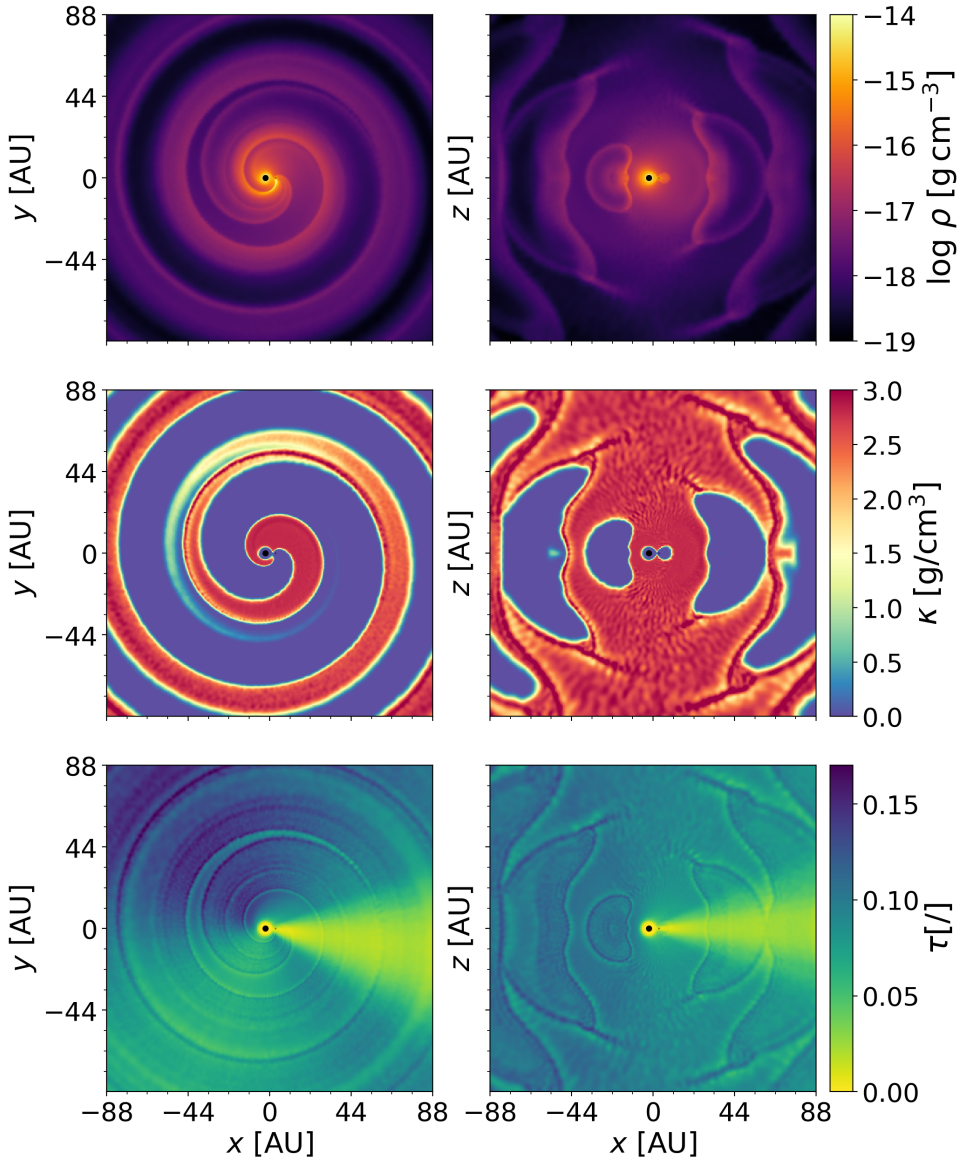


Figure 4.3: Density, opacity and optical depth of the simulation including ray-tracing in a slice through the orbital plane on the left, and perpendicular to orbital plane. Both slices go through both stars, along the x-axis.

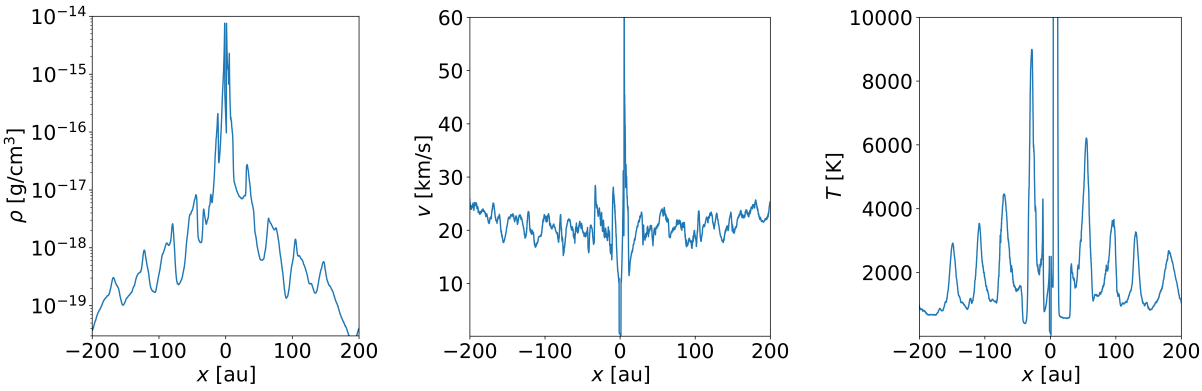


Figure 4.4: Radial structure plots showing the density, velocity and the temperature along the x-axis of the simulation including the ray-tracing.

the light happens close to the companion, in the region just before the spiral structure. The reason for this is that this is the region with the highest density, where dust is able to form and where τ becomes large. The thickness of the region between the dust condensation radius and the first arm of the spiral will dominate the value of τ.

In addition, a cone of low τ visible behind the companion. To explain this the low-tau beam, a zoom in of Figure 4.3 is shown in Figure 4.5. The opacity κ seen the middle row, in the orbital plane and the edge on plane. Close to the AGB star, it is hot, and dust is not yet able to form. Close to the companion the temperature is also too high due to the temperature increase due to the pressure build up close to the companion. Both regions are large enough so that there is a small region between the AGB star and the companion, where it is too hot to form dust. Therefore the opacity in this region remains small, and light can pass through nearly unattenuated. Radially further out the density has decreased sufficiently to make the contributions to τ behind the companion negligible.

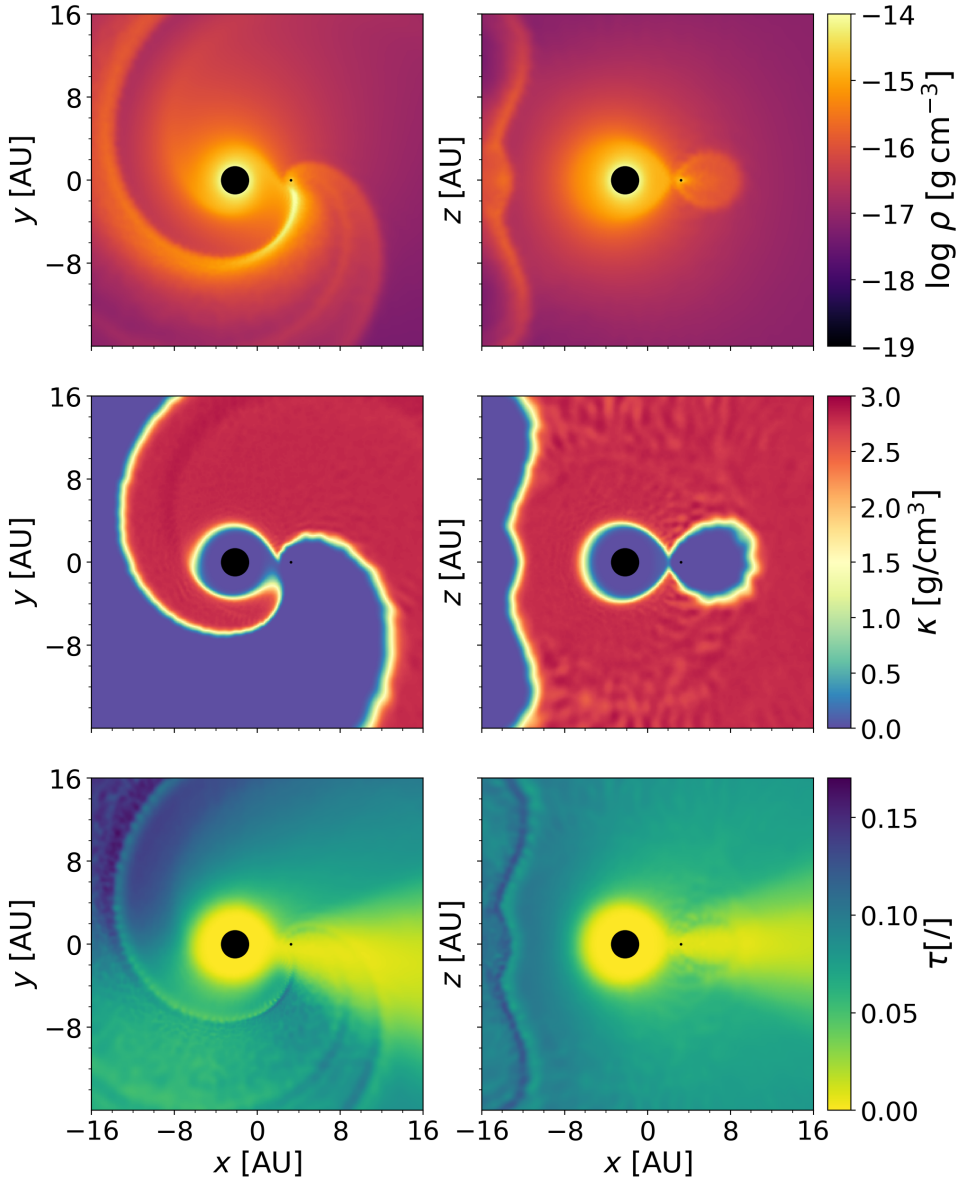


Figure 4.5: Zoom in of Figure 4.3.

This is contrary to simulations done by (Chen et al., 2017, 2020), the main difference lies in the treatment of the cooling of the medium around the companion. In particular, the use of HI electronic cooling, with additional Bremsstralung and H_2O molecular cooling depress the temperature sufficiently to create a dusty accretion disk. This disk now creates a shadow behind the companion, instead of a beam. Adding more cooling to our simulations might thus bring our simulations closer together.

4.3 Reference model + ray-tracer + cooling

Since a basic HI-cooling description is implemented in PHANTOM (see Section 2.5), this can be activated. The results are shown in Figures 4.7 and 4.8 and a zoom in of the density in Figure 4.6. This cooling only activates for temperatures higher than 3000K, and thus will only affect the regions close to the companion. For the global morphology of the simulation, as well as the radial structure there are no significant changes. What is different in these simulations is the accretion disk forming as can be seen in Figure 4.6. The accretion disk forms as the cooling makes it possible for the material to reach this densities without building up too much pressure. Nevertheless, activation of only HI cooling seemed to not have been sufficient to cool the medium around the companion below the dust condensation temperature. Hence, even with HI cooling, the beam of low tau persists. Further away from the companion no real changes are apparent, where the ray-tracer acts the same in this simulation.

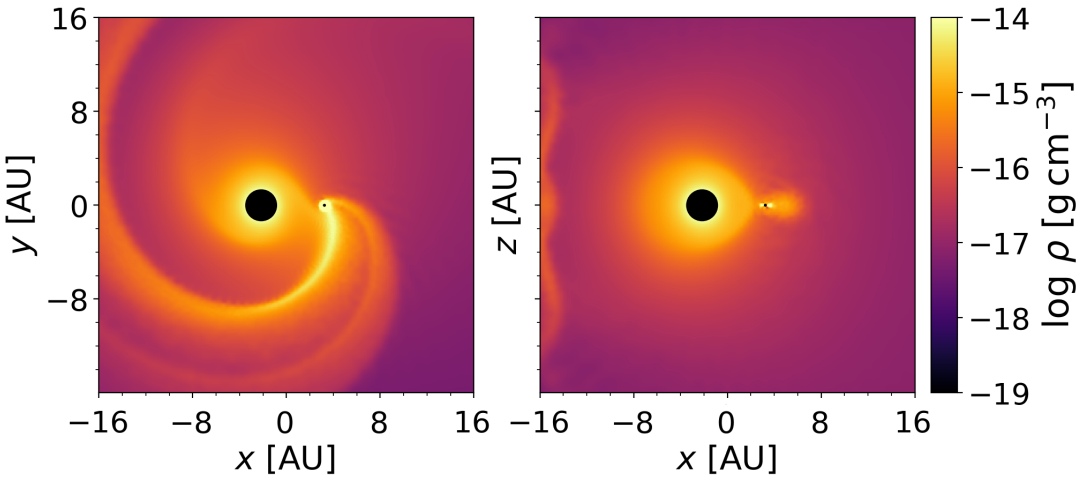


Figure 4.6: Density of the simulation including ray-tracing as well as HI-cooling in a slice through the orbital plane on the left, and perpendicular to orbital plane. Both slices go through both stars, along the x-axis.

In these few exploratory simulations simulations the activation of the ray-tracer does not influence the morphology of the wind significantly, but this is probably a consequence of the choice of wind/binary parameters, and the limited physics that are currently in PHANTOM.

An impotent caveat in these simulations is α that is always set to unity, and thus will dominate the morphology. So even if the radiation is attenuated, the outwards acceleration is still dominate by this artificial potential. To reduce α, pulsations and cooling have to be improved in PHANTOM. Only then will the effects of including dust attenuation on the dynamics become visible and will we be able to better appreciate the effect of including this ray-tracer.

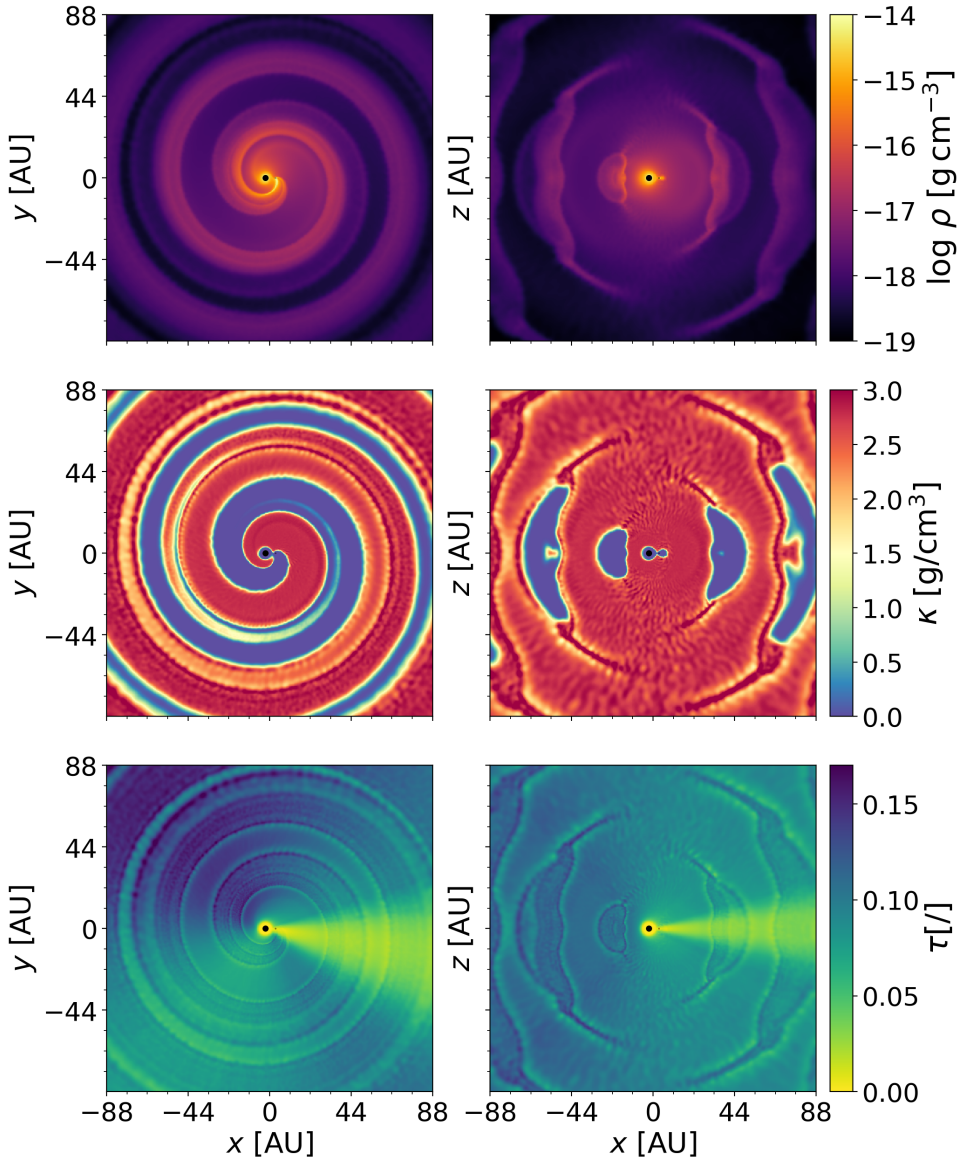


Figure 4.7: Density, opacity and optical depth of the simulation including ray-tracing as well as HI-cooling in a slice through the orbital plane on the left, and perpendicular to orbital plane. Both slices go through both stars, along the x-axis.

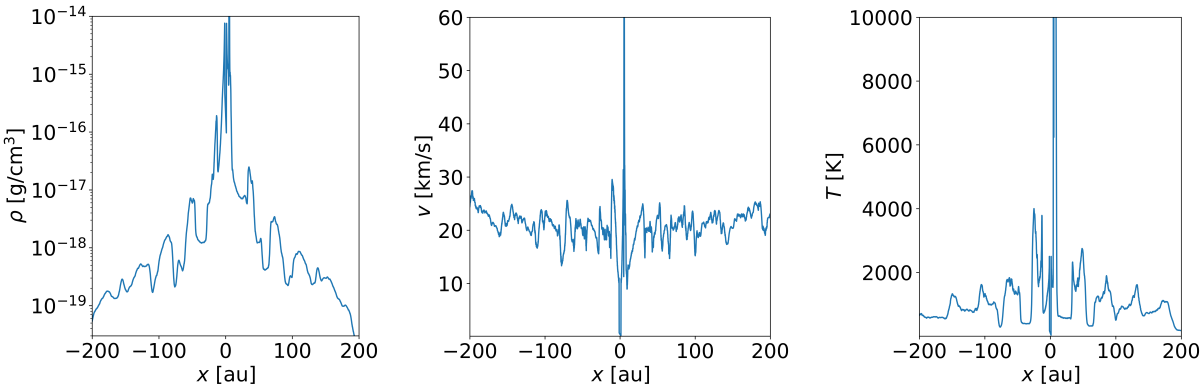


Figure 4.8: Radial structure plots showing the density, velocity and the temperature along the x-axis of the simulation including the ray-tracing as well as HI-cooling.

Chapter 5

Conclusion

Asymptotic giant branch (AGB) stars are a late evolutionary stage of stars with an initial mass between 0.8 and 8 solar masses (low- and intermediate-mass stars). During this phase, these stars can lose up to 80% of their own mass via a cool, molecular and dusty wind (Höfner & Olofsson, 2018). These dust-driven winds contribute to 85% of the interstellar gas and 35% of the interstellar dust abundances of the interstellar medium (ISM) (Tielens, 2005). When the stars have been fully stripped by the stellar wind, the AGB phase terminates and the stars move onto the post-AGB track on the Hertzsprung-Russell diagram, before turning into a planetary nebula (PN). On large scales, it has been shown that about 80% of the AGB stars show an overall spherical symmetry (Neri et al., 1998), while significantly less of their progenitors posses this symmetry (less than about 20% Parker et al. (2006); Sahai et al. (2011)). It is currently thought that this discrepancy possibly arises from low-mass binary interaction (Nordhaus & Blackman, 2006). Indeed, high spatial resolution observations reveal that the inner wind of these stars possess a high degree of morphological complexities, such as spirals, disks, bipolar outflows, hourglass-shaped shock fronts, etc. (e.g. Ramstedt et al., 2014; Decin et al., 2020).

To investigate the influence of the interaction of a low-mass binary, hydrodynamic simulations are done using the SPH code `PHANTOM` (Price et al., 2018). In previous studies, the acceleration of the wind has generally been calculated in the optically thin limit ($\tau = 0$), where in this thesis we upgrade the acceleration treatment to account for the attenuation of the stellar light by the dusty medium it traverses. We find an algorithm to calculate the optical depth τ constrained by being least computationally demanding as possible. This is not trivial since τ depends on the physical conditions between the location where the radiative force is calculated, and the light emitting AGB star, thus ray-tracing is necessary. We adopt the ray-tracing scheme of the 3D radiative transfer code `MAGRITTE` (De Ceuster et al., 2020a,b) and alter it to be more consistent with the SPH approach. We take advantage of the pre-calculated `PHANTOM` kd-tree to find the SPH nearest neighbours. Using the `HEALPix` discretisation of the 2-sphere the algorithm is made to trace rays outwards from the AGB star. We find that uniformly tracing a little more than 12 000 rays gives the fastest and most accurate set-up (see Figure 3.4). We also investigated the possibility of reducing the number of traced rays by making use of adaptive ray-tracing, but this was found to only results in a marginal improvement in computing time for low refinement ($o_{ref} = 1$). We attribute the poor performance of the adaptive scheme to the refinement criteria that we explored, and suspect that the formulation of better criteria can make the adaptive scheme worthwhile. Finally we also explored different

interpolation approaches, and found that using 4 rays, and weighting the interpolation with the inverse square of the distance, gives the best compromise between increasing the accuracy of the interpolated value of τ while safeguarding sharp edges that might exist in the numerical set-up.

This algorithm is implemented and coupled to the SPH code `PHANTOM` and the effect of the ray-tracer on wind-companion interaction is simulated. Here no major differences to the morphology of the outflow are visible, probably this is due to two major limitations currently in `PHANTOM`. The first is the use of an artificial repulsive potential represented by α to carry the SPH particles from the surface of the AGB star to the dust condensation zone without falling back. This is necessary because the models do not yet account for stellar surface pulsations. Reducing this artificial force significant differences in the wind dynamics and morphology are expected to become visible. The second limitation is the limited cooling physics presently available in `PHANTOM`, which have proven not to be sufficient to cool the hot medium surrounding the companion such that dust can be formed. We expect that accounting for more cooling and heating physics can have a drastic impact on e.g. the formation of dust in an accretion disk around the companion, such as presented by (Chen et al., 2017, 2020).

Bibliography

Abel, T. & Wandelt, B. D. (2002). Adaptive ray tracing for radiative transfer around point sources. *MNRAS*, 330(3):L53–L56.

Altay, G. & Theuns, T. (2013). URCHIN: a reverse ray tracer for astrophysical applications. *MNRAS*, 434(1):748–764.

Bondi, H. & Hoyle, F. (1944). On the mechanism of accretion by stars. *MNRAS*, 104:273.

Bowen, G. H. (1988). Dynamical Modeling of Long-Period Variable Star Atmospheres. *ApJ*, 329:299.

Carroll, B. W. & Ostlie, D. A. (1996). *An introduction to modern astrophysics*. Addison-Wesley, Reading.

Chen, Z., Frank, A., Blackman, E. G., Nordhaus, J., & Carroll-Nellenback, J. (2017). Mass transfer and disc formation in AGB binary systems. *MNRAS*, 468(4):4465–4477.

Chen, Z., Ivanova, N., & Carroll-Nellenback, J. (2020). A 3D Radiation Hydrodynamic AGB Binary Model. *ApJ*, 892(2):110.

Courant, R., Friedrichs, K., & Lewy, H. (1928). Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen*, 100:32–74.

De Ceuster, F. (2022). *Simulating 3D Radiation Transport, a modern approach to discretisation and an exploration of probabilistic methods*. PhD thesis, Doctoral thesis (Ph.D), UCL (University College London).

De Ceuster, F., Bolte, J., Homan, W., Maes, S., Malfait, J., Decin, L., Yates, J., Boyle, P., & Hetherington, J. (2020a). MAGRITTE, a modern software library for 3D radiative transfer - II. Adaptive ray-tracing, mesh construction, and reduction. *MNRAS*, 499(4):5194–5204.

De Ceuster, F., Homan, W., Yates, J., Decin, L., Boyle, P., & Hetherington, J. (2020b). MAGRITTE, a modern software library for 3D radiative transfer: I. Non-LTE atomic and molecular line modelling. *MNRAS*, 492(2):1812–1826.

Decin, L. (2021). Evolution and Mass Loss of Cool Ageing Stars: a Daedalean Story. *ARA&A*, 59:337–389.

Decin, L., Montargès, M., Richards, A. M. S., Gottlieb, C. A., Homan, W., McDonald, I., El Mellah, I., Danilovich, T., Wallström, S. H. J., Zijlstra, A., Baudry, A., Bolte, J., Cannon, E., De Beck, E., De Ceuster, F., de Koter, A., De Ridder, J., Etoka, S., Gobrecht, D.,

Gray, M., Herpin, F., Jeste, M., Lagadec, E., Kervella, P., Khouri, T., Menten, K., Millar, T. J., Müller, H. S. P., Plane, J. M. C., Sahai, R., Sana, H., Van de Sande, M., Waters, L. B. F. M., Wong, K. T., & Yates, J. (2020). (Sub)stellar companions shape the winds of evolved stars. *Science*, 369(6510):1497–1500.

Delaunay, B. (1934). Sur la sphère vide. *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk 7*, pages 793–800.

Donné, D. (2021). Sph modelling of companion perturbations on cooled agb outflow: Effects of radiative cooling on wind morphology.

Duchêne, G. & Kraus, A. (2013). Stellar Multiplicity. *ARA&A*, 51(1):269–310.

Edgar, R. (2004). A review of Bondi-Hoyle-Lyttleton accretion. *New A Rev.*, 48(10):843–859.

El Mellah, I., Bolte, J., Decin, L., Homan, W., & Keppens, R. (2020). Wind morphology around cool evolved stars in binaries. The case of slowly accelerating oxygen-rich outflows. *A&A*, 637:A91.

Freytag, B., Liljegren, S., & Höfner, S. (2017). Global 3D radiation-hydrodynamics models of AGB stars. Effects of convection and radial pulsations on atmospheric structures. *A&A*, 600:A137.

Gail, H. P. & Sedlmayr, E. (1988). Dust formation in stellar winds. IV. Heteromolecular carbon grain formation and growth. *A&A*, 206:153–168.

Górski, K. M., Hivon, E., Banday, A. J., Wandelt, B. D., Hansen, F. K., Reinecke, M., & Bartelmann, M. (2005). HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere. *ApJ*, 622(2):759–771.

Gray, D. F. (2005). *The Observation and Analysis of Stellar Photospheres*. Cambridge University Press, 3 edition.

Guerrero, M. A., Ramos-Larios, G., Toalá, J. A., Balick, B., & Sabin, L. (2020). Rings and arcs around evolved stars - II. The Carbon Star AFGL 3068 and the Planetary Nebulae NGC 6543, NGC 7009, and NGC 7027. *MNRAS*, 495(2):2234–2246.

Hilditch, R. W. (2001). *An introduction to close binary stars*. Cambridge astrophysics series. Cambridge University Press, Cambridge New York.

Höfner, S. & Olofsson, H. (2018). Mass loss of stars on the asymptotic giant branch. Mechanisms, models and measurements. *A&A Rev.*, 26(1):1.

Homan, W., Montargès, M., Pimpanuwat, B., Richards, A. M. S., Wallström, S. H. J., Kervella, P., Decin, L., Zijlstra, A., Danilovich, T., de Koter, A., Menten, K., Sahai, R., Plane, J., Lee, K., Waters, R., Baudry, A., Tat Wong, K., Millar, T. J., Van de Sande, M., Lagadec, E., Gobrecht, D., Yates, J., Price, D., Cannon, E., Bolte, J., De Ceuster, F., Herpin, F., Nuth, J., Philip Sindel, J., Kee, D., Grey, M. D., Etoka, S., Jeste, M., Gottlieb, C. A., Gottlieb, E., McDonald, I., El Mellah, I., & Müller, H. S. P. (2020). ATOMIUM: A high-resolution view on the highly asymmetric wind of the AGB star π^1Gruis. I. First detection of a new companion and its effect on the inner wind. *A&A*, 644:A61.

Homan, W., Pimpanuwat, B., Herpin, F., Danilovich, T., McDonald, I., Wallström, S. H. J., Richards, A. M. S., Baudry, A., Sahai, R., Millar, T. J., de Koter, A., Gottlieb, C. A., Kervella, P., Montargès, M., Van de Sande, M., Decin, L., Zijlstra, A., Etoka, S., Jeste, M., Müller, H. S. P., Maes, S., Malfait, J., Menten, K., Plane, J., Lee, K., Waters, R., Wong, K. T., Lagadec, E., Gobrecht, D., Yates, J., Price, D., Cannon, E., Bolte, J., De Ceuster, F., Nuth, J., Philip Sindel, J., Kee, D., Gray, M. D., & El Mellah, I. (2021). ATOMIUM: The astounding complexity of the near circumstellar environment of the M-type AGB star R Hydrae. I. Morpho-kinematical interpretation of CO and SiO emission. *A&A*, 651:A82.

Homan, W., Richards, A., Decin, L., de Koter, A., & Kervella, P. (2018). An unusual face-on spiral in the wind of the M-type AGB star EP Aquarii. *A&A*, 616:A34.

Hoyle, F. & Lyttleton, R. A. (1939). The effect of interstellar matter on climatic variation. *Proceedings of the Cambridge Philosophical Society*, 35(3):405.

Iliadis, C. (2015). *Nuclear Physics of Stars*. John Wiley & Sons, Ltd.

Kervella, P., Homan, W., Richards, A. M. S., Decin, L., McDonald, I., Montargès, M., & Ohnaka, K. (2016). ALMA observations of the nearby AGB star L_2 Puppis. I. Mass of the central star and detection of a candidate planet. *A&A*, 596:A92.

Kessel-Deynet, O. & Burkert, A. (2000). Ionizing radiation in smoothed particle hydrodynamics. *MNRAS*, 315(4):713–721.

Kim, H. & Taam, R. E. (2012). A NEW METHOD OF DETERMINING THE CHARACTERISTICS OF EVOLVED BINARY SYSTEMS REVEALED IN THE OBSERVED CIRCUMSTELLAR PATTERNS: APPLICATION TO AFGL 3068. *The Astrophysical Journal*, 759(1):L22.

Lamers, H. J. & Cassinelli, J. P. (1999). *Introduction to stellar winds*. Cambridge University Press, Cambridge, digital repr. edition.

Liu, Z.-W., Stancliffe, R. J., Abate, C., & Matrozis, E. (2017). Three-dimensional hydrodynamical simulations of mass transfer in binary systems by a free wind. *The Astrophysical Journal*, 846(2):117.

Maercker, M., Vlemmings, W. H. T., Brunner, M., De Beck, E., Humphreys, E. M., Kerschbaum, F., Lindqvist, M., Olofsson, H., & Ramstedt, S. (2016). A detailed view of the gas shell around R Sculptoris with ALMA. *A&A*, 586:A5.

Maes, S. (2020). Sph modelling of agb wind-companion interactions: (sub-)stellar companions in carbon- and oxygen-type winds.

Maes, S., Homan, W., Malfait, J., Siess, L., Bolte, J., De Ceuster, F., & Decin, L. (2021). SPH modelling of companion-perturbed AGB outflows including a new morphology classification scheme. *A&A*, 653:A25.

Maes, S., Siess, L., Homan, W., Malfait, J., De Ceuster, F., Ceulemans, T., Donné, D., Esseldeurs, M., & Decin, L. (2022). Route towards complete 3D hydro-chemical simulations of companion-perturbed AGB outflows. *Proceedings of the International Astronomical Union*.

Malfait, J. (2020). Sph modelling of agb wind-companion interactions: Extreme wind speeds and eccentric orbits.

Malfait, J., Homan, W., Maes, S., Bolte, J., Siess, L., De Ceuster, F., & Decin, L. (2021). SPH modelling of wind-companion interactions in eccentric AGB binary systems. *A&A*, 652:A51.

Mastrodemos, N. & Morris, M. (1998). Bipolar preplanetary nebulae: Hydrodynamics of dusty winds in binary systems. i. formation of accretion disks. *The Astrophysical Journal*, 497(1):303–329.

Mastrodemos, N. & Morris, M. (1999). Bipolar Pre-Planetary Nebulae: Hydrodynamics of Dusty Winds in Binary Systems. II. Morphology of the Circumstellar Envelopes. *ApJ*, 523(1):357–380.

Millar, T. J. (2004). Molecule and Dust Grain Formation. In *Asymptotic Giant Branch Stars*, pages 247–289. Springer.

Mohamed, S. & Podsiadlowski, P. (2007). Wind Roche-Lobe Overflow: a New Mass-Transfer Mode for Wide Binaries. In Napiwotzki, R. & Burleigh, M. R., editors, *15th European Workshop on White Dwarfs*, volume 372 of *Astronomical Society of the Pacific Conference Series*, page 397.

Mohamed, S. & Podsiadlowski, P. (2012). Mass Transfer in Mira-type Binaries. *Baltic Astronomy*, 21:88–96.

Nelson, R. P. & Papaloizou, J. C. B. (1994). Variable Smoothing Lengths and Energy Conservation in Smoothed Particle Hydrodynamics. *MNRAS*, 270:1.

Neri, R., Kahane, C., Lucas, R., Bujarrabal, V., & Loup, C. (1998). A (12) CO (J=1 $\rightarrow$ 0) and (J=2 $\rightarrow$ 1) atlas of circumstellar envelopes of AGB and post-AGB stars. *A&AS*, 130:1–64.

Nordhaus, J. & Blackman, E. G. (2006). Low-mass binary-induced outflows from asymptotic giant branch stars. *MNRAS*, 370(4):2004–2012.

Omukai, K., Hosokawa, T., & Yoshida, N. (2010). Low-metallicity Star Formation: Prestellar Collapse and Protostellar Accretion in the Spherical Symmetry. *ApJ*, 722(2):1793–1815.

Parker, Q. A., Acker, A., Frew, D. J., Hartley, M., Peyaud, A. E. J., Ochsenbein, F., Phillipps, S., Russeil, D., Beaulieu, S. F., Cohen, M., Köppen, J., Miszalski, B., Morgan, D. H., Morris, R. A. H., Pierce, M. J., & Vaughan, A. E. (2006). The Macquarie/AAO/Strasbourg Hα Planetary Nebula Catalogue: MASH. *MNRAS*, 373(1):79–94.

Pinte, C., Harries, T. J., Min, M., Watson, A. M., Dullemond, C. P., Woitke, P., Ménard, F., & Durán-Rojas, M. C. (2009). Benchmark problems for continuum radiative transfer. High optical depths, anisotropic scattering, and polarisation. *A&A*, 498(3):967–980.

Price, D. J. (2007). splash: An Interactive Visualisation Tool for Smoothed Particle Hydrodynamics Simulations. *PASA*, 24(3):159–173.

Price, D. J., Wurster, J., Tricco, T. S., Nixon, C., Toupin, S., Pettitt, A., Chan, C., Mentiplay, D., Laibe, G., Glover, S., Dobbs, C., Nealon, R., Liptai, D., Worpel, H., Bonnerot, C., Dipierro, G., Ballabio, G., Ragusa, E., Federrath, C., Iaconi, R., Reichardt, T., Forgan, D., Hutchison, M., Constantino, T., Ayliffe, B., Hirsh, K., & Lodato, G. (2018). Phantom: A Smoothed Particle Hydrodynamics and Magnetohydrodynamics Code for Astrophysics. *PASA*, 35:e031.

Ramstedt, S., Mohamed, S., Vlemmings, W. H. T., Maercker, M., Montez, R., Baudry, A., De Beck, E., Lindqvist, M., Olofsson, H., Humphreys, E. M. L., Jorissen, A., Kerschbaum, F., Mayer, A., Wittkowski, M., Cox, N. L. J., Lagadec, E., Leal-Ferreira, M. L., Paladini, C., Pérez-Sánchez, A., & Sacuto, S. (2014). The wonderful complexity of the Mira AB system. *A&A*, 570:L14.

Sahai, R., Morris, M. R., & Villar, G. G. (2011). Young Planetary Nebulae: Hubble Space Telescope Imaging and a New Morphological Classification System. *AJ*, 141(4):134.

Sana, H., de Mink, S. E., de Koter, A., Langer, N., Evans, C. J., Gieles, M., Gosset, E., Izzard, R. G., Le Bouquin, J. B., & Schneider, F. R. N. (2012). Binary Interaction Dominates the Evolution of Massive Stars. *Science*, 337(6093):444.

Siess, L., Homan, W., Toupin, S., & Price, D. J. (2022). 3D simulations of AGB stellar winds - I. Steady winds and dust formation. *A&A*.

Spitzer, L. & Jura, M. (1978). Physical Processes in the Interstellar Medium. *Physics Today*, 31(7):48.

Springel, V., Yoshida, N., & White, S. D. M. (2001). GADGET: a code for collisionless and gasdynamical cosmological simulations. *New A*, 6(2):79–117.

Theuns, T. & Jorissen, A. (1993). Wind accretion in binary stars - I. Intricacies of the flow structure. *MNRAS*, 265:946–967.

Tielens, A. G. G. M. (2005). *The Physics and Chemistry of the Interstellar Medium*. Cambridge University Press.

Van de Sande, M. & Millar, T. J. (2022). The impact of stellar companion UV photons on the chemistry of the circumstellar environments of AGB stars. *MNRAS*, 510(1):1204–1222.

Van de Sande, M., Walsh, C., & Millar, T. J. (2021). Chemical modelling of dust-gas chemistry within AGB outflows - III. Photoprocessing of the ice and return to the ISM. *MNRAS*, 501(1):491–506.

Van Winckel, H. (2003). Post-AGB Stars. *ARA&A*, 41:391–427.

Verlet, L. (1967). Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. *Physical Review*, 159(1):98–103.

Wadsley, J. W., Keller, B. W., & Quinn, T. R. (2017). Gasoline2: a modern smoothed particle hydrodynamics code. *MNRAS*, 471(2):2357–2369.

Appendices

Appendix A

Final implementation

The final implementation plugged into the PHANTOM, programmed in FORTRAN95.

```
!--------------------------------------------------------------------------!
! The Phantom Smoothed Particle Hydrodynamics code, by Daniel Price et al. !
! Copyright (c) 2007-2022 The Authors (see AUTHORS)                        !
! See LICENCE file for usage and distribution conditions                   !
! http://phantomsph.bitbucket.io/                                          !
!--------------------------------------------------------------------------!
module raytracer
!
! This module contains all routines required to:
!   - perform radial ray tracing starting from the primary star
!   - calculate optical depts along the rays given the opacity distribution
!   - interpolate optical depths to all SPH particles
! Applicable both for single star as well as binary models
!
! WARNING: This module has only been tested on phantom wind setups
!
! :References: None
!
! :Owner: Esseldeurs Mats
!
! :Dependencies: linklist, kernel, part, healpix
!
  use healpix

  implicit none
  public :: get_all_tau

  private

 contains

  !-----------------------------------------------------------------------
  !+
  !  MAIN ROUTINE
  !  Returns the optical depth to each SPH particle, using the uniform outwards
  !  ray-tracing scheme.
  !+
  !  IN: npart:           The number of SPH particles
  !  IN: nptmass:         The number of sink particles
  !  IN: xyzm_ptmass:     The array containing the properties of the sink particle
  !  IN: xyzh:            The array containing the particles position+smooting lenght
  !  IN: xyzh:            The array containing the particles position+smooting lenght
  !  IN: kappa_cgs:       The array containing the opacities of all SPH particles
  !  IN: order:           The healpix order which is used for the uniform ray sampling
  !+
  !  OUT: tau:            The array of optical depths for each SPH particle
  !+
  !-----------------------------------------------------------------------
```

```
49      subroutine get_all_tau(npart, nptmass, xyzmh_ptmass, xyzh, kappa_cgs, order, tau)
50       use part, only: iReff
51       integer, intent(in) :: npart, order, nptmass
52       real, intent(in)    :: kappa_cgs(:), xyzh(:,:), xyzmh_ptmass(:,:)
53       real, intent(out)   :: tau(:)
54
55       if (nptmass == 2 ) then
56          call get_all_tau_companion(npart, xyzmh_ptmass(1:3,1), xyzh, kappa_cgs, &
57               xyzmh_ptmass(iReff,1), xyzmh_ptmass(1:3,2), xyzmh_ptmass(iReff,2), order,
     tau)
58       else
59          call get_all_tau_single(npart, xyzmh_ptmass(1:3,1), xyzh, kappa_cgs,
     xyzmh_ptmass(iReff,1), order, tau)
60       endif
61    end subroutine get_all_tau
62
63    !-----------------------------------------------------------------------
64    !+
65    !  Calculates the optical depth to each SPH particle, using the uniform outwards
66    !  ray-tracing scheme for models containing a single star
67    !
68    !  Relies on healpix, for more information: https://healpix.sourceforge.io/
69    !+
70    !  IN: npart:           The number of SPH particles
71    !  IN: primary:         The xyz coordinates of the primary star
72    !  IN: xyzh:            The array containing the particles position+smooting lenght
73    !  IN: kappa:           The array containing the kappa of all SPH particles
74    !  IN: Rstar:           The radius of the primary star
75    !  IN: order:           The healpix order which is used for the uniform ray sampling
76    !+
77    !  OUT: taus:           The array of optical depths to each SPH particle
78    !+
79    !-----------------------------------------------------------------------
80    subroutine get_all_tau_single(npart, primary, xyzh, kappa, Rstar, order, tau)
81       use part, only : isdead_or_accreted
82       integer, intent(in) :: npart,order
83       real, intent(in)    :: primary(3), kappa(:), Rstar, xyzh(:,:)
84       real, intent(out)   :: tau(:)
85
86       integer  :: i, nrays, nsides
87       real     :: ray_dir(3),part_dir(3)
88       real, dimension(:,:), allocatable  :: rays_dist, rays_tau
89       integer, dimension(:), allocatable :: rays_dim
90       integer, parameter :: ndim = 200
91
92       nrays = 12*4**order ! The number of rays traced given the healpix order
93       nsides = 2**order   ! The healpix nsides given the healpix order
94
95       allocate(rays_dist(ndim, nrays))
96       allocate(rays_tau(ndim, nrays))
97       allocate(rays_dim(nrays))
98
99       !-----------------------------------------------
100      ! CONSTRUCT the RAYS given the ORDER
101      ! and determine the optical depth along them
102      !-----------------------------------------------
103
104 !$omp parallel default(none) &
105 !$omp private(ray_dir) &
106 !$omp shared(nrays,nsides,primary,kappa,xyzh,Rstar,rays_dist,rays_tau,rays_dim)
107 !$omp do
108      do i = 1, nrays
109         !returns ray_dir, the unit vector identifying ray
110         !        (index i-1 becase healpix starts counting at index 0)
111         call pix2vec_nest(nsides, i-1, ray_dir)
112         !calculate the properties along the ray
113         call ray_tracer(primary,ray_dir,xyzh,kappa,Rstar,rays_tau(:,i),rays_dist(:,i),
     rays_dim(i))
114      enddo
115 !$omp enddo
116 !$omp end parallel
```

```
117
118
119         !-------------------------------------------------
120         ! DETERMINE the optical depth for each particle
121         ! using the values available on the rays
122         !-------------------------------------------------
123
124 !$omp parallel default(none) &
125 !$omp private(part_dir) &
126 !$omp shared(npart,primary,nsides,xyzh,ray_dir,rays_dist,rays_tau,rays_dim,tau)
127 !$omp do
128         do i = 1,npart
129            if (.not.isdead_or_accreted(xyzh(4,i))) then
130               part_dir = xyzh(1:3,i)-primary
131               call interpolate_tau(nsides, part_dir, rays_tau, rays_dist, rays_dim, tau(i))
132            else
133               tau(i) = -99.
134            endif
135         enddo
136 !$omp enddo
137 !$omp end parallel
138
139    end subroutine get_all_tau_single
140
141    !-----------------------------------------------------------------------
142    !+
143    !  Calculates the optical depth to each SPH particle, using the uniform outwards
144    !  ray-tracing scheme for models containing primary star and a companion
145    !
146    !  Relies on healpix, for more information: https://healpix.sourceforge.io/
147    !+
148    !  IN: npart:           The number of SPH particles
149    !  IN: primary:         The xyz coordinates of the primary star
150    !  IN: xyzh:            The array containing the particles position+smooting lenght
151    !  IN: kappa:           The array containing the opacity of all the SPH particles
152    !  IN: Rstar:           The radius of the primary star
153    !  IN: companion:       The xyz coordinates of the companion
154    !  IN: Rcomp:           The radius of the companion
155    !  IN: order:           The healpix order which is used for the uniform ray sampling
156    !+
157    !  OUT: tau:            The array of optical depths for each SPH particle
158    !+
159    !-----------------------------------------------------------------------
160    subroutine get_all_tau_companion(npart, primary, xyzh, kappa, Rstar, companion, Rcomp,
         order, tau)
161       use part, only : isdead_or_accreted
162       integer, intent(in) :: npart, order
163       real, intent(in)    :: primary(3), companion(3), kappa(:), Rstar, xyzh(:,:), Rcomp
164       real, intent(out)   :: tau(:)
165
166       integer  :: i, nrays, nsides
167       real     :: normCompanion,theta0,phi,cosphi,sinphi,theta,sep,root
168       real     :: ray_dir(3),part_dir(3),uvecCompanion(3)
169       real, dimension(:,:), allocatable  :: dirs
170       real, dimension(:,:), allocatable  :: rays_dist, rays_tau
171       integer, dimension(:), allocatable :: rays_dim
172       integer, parameter :: ndim = 200
173
174       nrays = 12*4**order ! The number of rays traced given the healpix order
175       nsides = 2**order   ! The healpix nsides given the healpix order
176
177       allocate(dirs(3, nrays))
178       allocate(rays_dist(ndim, nrays))
179       allocate(rays_tau(ndim, nrays))
180       allocate(rays_dim(nrays))
181
182       uvecCompanion = companion-primary
183       normCompanion = norm2(uvecCompanion)
184       uvecCompanion = uvecCompanion/normCompanion
185       theta0        = asin(Rcomp/normCompanion)
186       phi           = atan2(uvecCompanion(2),uvecCompanion(1))
```

```
187       cosphi        = cos(phi)
188       sinphi        = sin(phi)
189
190       !-----------------------------------------------
191       ! CONSTRUCT the RAYS given the ORDER
192       ! and determine the optical depth along them
193       !-----------------------------------------------
194
195 !$omp parallel default(none) &
196 !$omp private(ray_dir,theta,root,sep) &
197 !$omp shared(nrays,nsides,primary,kappa,xyzh,Rstar,Rcomp,rays_dist,rays_tau,rays_dim) &
198 !$omp shared(uvecCompanion,normCompanion,cosphi,sinphi,theta0)
199 !$omp do
200       do i = 1, nrays
201          !returns ray_dir, the unit vector identifying ray
202          !        (index i-1 becase healpix starts counting at index 0)
203          call pix2vec_nest(nsides, i-1, ray_dir)
204          !rotate ray vectors by an angle = phi so the main axis points to the companion
205          !This is because along the main axis (1,0,0) rays are distributed more
     uniformally
206          ray_dir = (/cosphi*ray_dir(1) - sinphi*ray_dir(2),sinphi*ray_dir(1) + cosphi*
     ray_dir(2), ray_dir(3)/)
207          theta   = acos(dot_product(uvecCompanion, ray_dir))
208          !the ray intersects the companion: only calculate tau up to the companion
209          if (theta < theta0) then
210             root  = sqrt(Rcomp**2-normCompanion**2*sin(theta)**2)
211             sep   = normCompanion*cos(theta)-root
212             call ray_tracer(primary,ray_dir,xyzh,kappa,Rstar,rays_tau(:,i),rays_dist(:,i)
     ,rays_dim(i), sep)
213          else
214             call ray_tracer(primary,ray_dir,xyzh,kappa,Rstar,rays_tau(:,i),rays_dist(:,i)
     ,rays_dim(i))
215          endif
216       enddo
217 !$omp enddo
218 !$omp end parallel
219
220       !-----------------------------------------------
221       ! DETERMINE the optical depth for each particle
222       ! using the values available on the rays
223       !-----------------------------------------------
224
225 !$omp parallel default(none) &
226 !$omp private(part_dir) &
227 !$omp shared(npart,primary,cosphi,sinphi,nsides,xyzh,ray_dir,rays_dist,rays_tau,rays_dim,
     tau)
228 !$omp do
229       do i = 1, npart
230          if (.not.isdead_or_accreted(xyzh(4,i))) then
231             !vector joining the source to the particle
232             part_dir = xyzh(1:3,i)-primary
233             part_dir = (/cosphi*part_dir(1) + sinphi*part_dir(2),-sinphi*part_dir(1) +
     cosphi*part_dir(2), part_dir(3)/)
234             call interpolate_tau(nsides, part_dir, rays_tau, rays_dist, rays_dim, tau(i))
235          else
236             tau(i) = -99.
237          endif
238       enddo
239 !$omp enddo
240 !$omp end parallel
241    end subroutine get_all_tau_companion
242
243    !-----------------------------------------------------------------------
244    !+
245    !  Calculate the optical depth of a particle.
246    !  Search for the four closest rays to a particle, perform four-point
247    !  interpolation of the optical depts from these rays. Weighted by the
248    !  inverse square of the perpendicular distance to the rays.
249    !
250    !  Relies on healpix, for more information: https://healpix.sourceforge.io/
251    !+
```

```
252    !  IN: nsides:          The healpix nsides of the simulation
253    !  IN: vec:             The vector from the primary to a point
254    !  IN: rays_tau:        2-dimensional array containing the cumulative optical
255    !                       depts along each ray
256    !  IN: rays_dist:       2-dimensional array containing the distances from the
257    !                       primary along each ray
258    !  IN: rays_dim:        The vector containing the number of points defined along
259    !                       each ray
260    !+
261    !  OUT: tau:            The interpolated optical depth at the particle's location
262    !+
263    !-----------------------------------------------------------------------
264    subroutine interpolate_tau(nsides, vec, rays_tau, rays_dist, rays_dim, tau)
265       integer, intent(in) :: nsides, rays_dim(:)
266       real, intent(in)    :: vec(:), rays_dist(:,:), rays_tau(:,:)
267       real, intent(out)   :: tau
268
269       integer :: rayIndex, neighbours(8), nneigh, i, k
270       real    :: tautemp, ray(3), vectemp(3), weight, tempdist(8), distRay_sq, vec_norm2
271       logical :: mask(8)
272
273       vec_norm2 = norm2(vec)
274       !returns rayIndex, the index of the ray vector that points to the particle
275       call vec2pix_nest(nsides, vec, rayIndex)
276       !returns ray(3), the unit vector identifying the ray with index number rayIndex
277       call pix2vec_nest(nsides, rayIndex, ray)
278       vectemp       = vec - vec_norm2*ray
279       distRay_sq    = dot_product(vectemp,vectemp)
280       call get_tau_on_ray(vec_norm2, rays_tau(:,rayIndex+1), rays_dist(:,rayIndex+1),
     rays_dim(rayIndex+1), tautemp)
281       if (distRay_sq > 0.) then
282          tau    = tautemp/distRay_sq
283          weight = 1./distRay_sq
284       else
285          ! the particle sits exactly on the ray, no need to get the neighbours
286          tau    = tautemp
287          return
288       endif
289
290       !returns the number nneigh and list of vectors neighbouring the ray number index
291       call neighbours_nest(nsides, rayIndex, neighbours, nneigh)
292       !for each neighbouring ray calculate its distance to the particle
293       do i=1,nneigh
294          call pix2vec_nest(nsides, neighbours(i), ray)
295          vectemp     = vec - vec_norm2*ray
296          tempdist(i) = dot_product(vectemp,vectemp)
297       enddo
298       neighbours       = neighbours+1
299       mask             = .true.
300       if (nneigh <8) mask(nneigh+1:8) = .false.
301       !take tau contribution from the 3 closest rays
302       do i=1,3
303          k       = minloc(tempdist,1,mask)
304          mask(k) = .false.
305          call get_tau_on_ray(vec_norm2, rays_tau(:,neighbours(k)), &
306                 rays_dist(:,neighbours(k)), rays_dim(neighbours(k)), tautemp)
307          tau    = tau + tautemp/tempdist(k)
308          weight = weight + 1./tempdist(k)
309       enddo
310       tau = tau / weight
311    end subroutine interpolate_tau
312
313
314    !-----------------------------------------------------------------------
315    !+
316    !  Interpolation of the optical depth for an arbitrary point on the ray,
317    !  with a given distance to the starting point of the ray.
318    !+
319    !  IN: distance:        The distance from the staring point of the ray to a
320    !                       point on the ray
321    !  IN: tau_along_ray:   The vector of cumulative optical depths along the ray
```

```
322    !  IN: dist_along_ray:  The vector of distances from the primary along the ray
323    !  IN: len:             The length of listOfTau and listOfDist
324    !+
325    !  OUT: tau:            The optical depth to the given distance along the ray
326    !+
327    !-----------------------------------------------------------------------
328    subroutine get_tau_on_ray(distance, tau_along_ray, dist_along_ray, len, tau)
329       real, intent(in)    :: distance, tau_along_ray(:), dist_along_ray(:)
330       integer, intent(in) :: len
331       real, intent(out)   :: tau
332
333       integer :: L, R, m ! left, right and middle index for binary search
334
335       if (distance .lt. dist_along_ray(1)) then
336          tau = 0.
337       else if (distance .gt. dist_along_ray(len)) then
338          tau = 99.
339       else
340          L = 2
341          R = len-1
342          !bysection search for the index of the closest ray location to the particle
343          do while (L < R)
344             m = (L + R)/2
345             if (dist_along_ray(m) > distance) then
346                   R = m
347             else
348                   L = m + 1
349             end if
350          enddo
351          !interpolate linearly ray properties to get the particle's optical depth
352          tau = tau_along_ray(L-1)+(tau_along_ray(L)-tau_along_ray(L-1))/ &
353                   (dist_along_ray(L)-dist_along_ray(L-1))*(distance-dist_along_ray(L-1))
354       endif
355    end subroutine get_tau_on_ray
356
357    !-----------------------------------------------------------------------
358    !+
359    !  Calculate the optical depth along a given ray
360    !+
361    !  IN: primary:         The location of the primary star
362    !  IN: ray:             The unit vector of the direction in which the
363    !                       optical depts will be calculated
364    !  IN: xyzh:            The array containing the particles position+smooting lenght
365    !  IN: kappa:           The array containing the particles opacity
366    !  IN: Rstar:           The radius of the primary star
367    !+
368    !  OUT: tau_along_ray:  The vector of cumulative optical depts along the ray
369    !  OUT: dist_along_ray: The vector of distances from the primary along the ray
370    !  OUT: len:            The length of tau_along_ray and dist_along_ray
371    !+
372    !  OPT: maxDistance:    The maximal distance the ray needs to be traced
373    !+
374    !-----------------------------------------------------------------------
375    subroutine ray_tracer(primary, ray, xyzh, kappa, Rstar, tau_along_ray, dist_along_ray,
         len, maxDistance)
376       use units, only:umass,udist
377       real, intent(in)      :: primary(3), ray(3), Rstar, xyzh(:,:), kappa(:)
378       real, optional        :: maxDistance
379       real, intent(out)     :: dist_along_ray(:), tau_along_ray(:)
380       integer, intent(out) :: len
381
382       real     :: dr, next_dr, h, dtaudr, previousdtaudr, nextdtaudr, distance
383       integer :: inext, i
384
385       h = Rstar/100.
386       inext=0
387       do while (inext==0)
388          h = h*2.
389          !find the next point along the ray : index next
390          call find_next(primary+Rstar*ray, h, ray, xyzh, kappa, previousdtaudr, dr, inext
       )
```

```
391         enddo
392
393         i = 1
394         tau_along_ray(i)  = 0.
395         distance          = Rstar
396         dist_along_ray(i) = distance
397         do while (hasNext(inext,tau_along_ray(i),distance,maxDistance))
398            distance = distance+dr
399            call find_next(primary + distance*ray, xyzh(4,inext), ray, xyzh, kappa,
     nextdtaudr, next_dr, inext)
400            i = i + 1
401            dtaudr            = (nextdtaudr+previousdtaudr)/2.
402            previousdtaudr    = nextdtaudr
403            !fix units for tau (kappa is in cgs while rho & r are in code units)
404            tau_along_ray(i)  = tau_along_ray(i-1)+dr*dtaudr*umass/(udist**2)
405            dist_along_ray(i) = distance
406            dr                = next_dr
407         enddo
408         len = i
409      end subroutine ray_tracer
410
411      logical function hasNext(inext, tau, distance, maxDistance)
412         integer, intent(in) :: inext
413         real, intent(in)    :: distance, tau
414         real, optional      :: maxDistance
415         real, parameter :: tau_max = 99.
416         if (present(maxDistance)) then
417            hasNext = inext /= 0 .and. distance < maxDistance .and. tau < tau_max
418         else
419            hasNext = inext /= 0 .and. tau < tau_max
420         endif
421      end function hasNext
422
423      !-----------------------------------------------------------------------
424      !+
425      !  First finds the local optical depth derivative at the starting point, then finds
426      !                       the next point on a ray and the distance to this point
427      !+
428      !  IN: inpoint:         The coordinate of the initial point projected on the
429      !                       ray for which the opacity and the next point will be
430      !                       calculated
431      !  IN: h:               The smoothing length at the initial point
432      !  IN: ray:             The unit vector of the direction in which the next
433      !                       point will be calculated
434      !  IN: xyzh:            The array containing the particles position+smoothing length
435      !  IN: kappa:           The array containing the particles opacity
436      !  IN: inext:           The index of the initial point
437      !                       (this point will not be considered as possible next point)
438      !+
439      !  OUT: dtaudr:         The local optical depth derivative at the given location
440      !                       (inpoint)
441      !  OUT: distance:       The distance to the next point
442      !  OUT: inext:          The index of the next point on the ray
443      !+
444      !-----------------------------------------------------------------------
445      subroutine find_next(inpoint, h, ray, xyzh, kappa, dtaudr, distance, inext)
446         use linklist, only:getneigh_pos,ifirstincell,listneigh
447         use kernel,   only:radkern,cnormk,wkern
448         use part,     only:hfact,rhoh,massoftype,igas
449         real,    intent(in)    :: xyzh(:,:), kappa(:), inpoint(:), ray(:), h
450         integer, intent(inout) :: inext
451         real,    intent(out)   :: distance, dtaudr
452
453         integer, parameter :: nmaxcache = 0
454         real  :: xyzcache(0,nmaxcache)
455
456         integer  :: nneigh, i, prev
457         real     :: dmin, vec(3), dr, raydistance, q, norm_sq
458
459         prev     = inext
460         inext    = 0
```

```
461       distance = 0.
462
463       !for a given point (inpoint), returns the list of neighbouring particles (listneigh
      )
464       !            within a radius h*radkern
465       call getneigh_pos(inpoint,0.,h*radkern,3,listneigh,nneigh,xyzh,xyzcache,nmaxcache,
      ifirstincell)
466
467       dtaudr = 0
468       dmin = huge(0.)
469       !loop over all neighbours
470       do i=1,nneigh
471          vec      = xyzh(1:3,listneigh(i)) - inpoint
472          norm_sq = dot_product(vec,vec)
473          q        = sqrt(norm_sq)/xyzh(4,listneigh(i))
474          !add optical depth contribution from each particle
475          dtaudr = dtaudr+wkern(q*q,q)*kappa(listneigh(i))*rhoh(xyzh(4,listneigh(i)),
      massoftype(igas))
476
477          !find the next particle: among the neighbours find the particle located the
      closest to the ray
478          if (listneigh(i) .ne. prev) then
479             dr = dot_product(vec,ray) !projected distance along the ray
480             if (dr>0.) then
481                !distance perpendicular to the ray direction
482                raydistance = norm_sq - dr**2
483                if (raydistance < dmin) then
484                   dmin     = raydistance
485                   inext    = listneigh(i)
486                   distance = dr
487                end if
488             end if
489          endif
490       enddo
491       dtaudr = dtaudr*cnormk/hfact**3
492    end subroutine find_next
493 end module raytracer
```

Appendix B

Utils

All the algorithms explained in Chapter 3, programmed in FORTRAN95.

```
module raytracer_all
   use healpix
   implicit none
   public :: get_all_tau_outwards, get_all_tau_inwards, get_all_tau_adaptive
   private
   contains

   !*************************************************************************!
   !************************    ADAPTIVE    ***********************************!
   !*************************************************************************!

   !-------------------------------------------------------------------------
   !+
   !  Calculate the optical depth of each particle, using the adaptive ray-
   !  tracing scheme
   !+
   !  IN: npart:            The number of SPH particles
   !  IN: primary:          The xyz coordinates of the primary star
   !  IN: xyzh:             The array containing the particles position+smooting lenght
   !  IN: kappa:            The array containing the kappa of all SPH particles
   !  IN: Rstar:            The radius of the star
   !  IN: minOrder:         The minimal order in which the rays are sampled
   !  IN: refineLevel:      The amount of orders in which the rays can be
   !                        sampled deeper
   !  IN: refineScheme:     The refinement scheme used for adaptive ray selection
   !+
   !  OUT: taus:            The list of optical depths for each particle
   !+
   !  OPT: companion:       The xyz coordinates of the companion
   !  OPT: Rcomp:           The radius of the companion
   !+
   !-------------------------------------------------------------------------
   subroutine get_all_tau_adaptive(npart, primary, xyzh, kappa, Rstar, minOrder,&
                                   refineLevel, refineScheme, taus, companion, Rcomp)
      integer, intent(in) :: npart, minOrder, refineLevel, refineScheme
      real, intent(in)    :: primary(3), kappa(:), xyzh(:,:), Rstar
      real, optional      :: Rcomp, companion(3)
      real, intent(out)   :: taus(:)

      integer      :: i, nrays, nsides, index
      real         :: normCompanion, theta0, unitCompanion(3), theta, root, dist, vec(3),
    dir(3)
      real, dimension(:,:), allocatable :: dirs
      real, dimension(:,:), allocatable :: listsOfDists, listsOfTaus
      integer, dimension(:), allocatable :: indices, rays_dim
      real, dimension(:), allocatable    :: tau, dists

      if (present(companion) .and. present(Rcomp)) then
```

```
48          unitCompanion = companion-primary
49          normCompanion = norm2(unitCompanion)
50          theta0 = asin(Rcomp/normCompanion)
51          unitCompanion = unitCompanion/normCompanion
52
53          call get_rays(npart, primary, companion, Rcomp, xyzh, minOrder, refineLevel,
     refineScheme, dirs, indices, nrays)
54          allocate(listsOfDists(200, nrays))
55          allocate(listsOfTaus(size(listsOfDists(:,1)), nrays))
56          allocate(tau(size(listsOfDists(:,1))))
57          allocate(dists(size(listsOfDists(:,1))))
58          allocate(rays_dim(nrays))
59
60          !$omp parallel do private(tau,dist,dir,dists,root,theta)
61          do i = 1, nrays
62             tau=0.
63             dists=0.
64             dir = dirs(:,i)
65             theta = acos(dot_product(unitCompanion, dir))
66             if (theta < theta0) then
67                root = sqrt(normCompanion**2*cos(theta)**2-normCompanion**2+Rcomp**2)
68                dist = normCompanion*cos(theta)-root
69                call ray_tracer(primary, dir, xyzh, kappa, Rstar, tau, dists, rays_dim(i),
     dist)
70             else
71                call ray_tracer(primary, dir, xyzh, kappa, Rstar, tau, dists, rays_dim(i))
72             endif
73             listsOfTaus(:,i) = tau
74             listsOfDists(:,i) = dists
75          enddo
76          !$omp end parallel do
77
78          nsides = 2**(minOrder+refineLevel)
79          taus = 0.
80          !$omp parallel do private(index,vec)
81          do i = 1, npart
82             vec = xyzh(1:3,i)-primary
83             call vec2pix_nest(nsides, vec, index)
84             index = indices(index + 1)
85             call get_tau_on_ray(norm2(vec), listsOfTaus(:,index), listsOfDists(:,index),
     rays_dim(index), taus(i))
86          enddo
87          !$omp end parallel do
88
89       else
90          call get_all_tau_outwards_single(npart, primary, xyzh, kappa, &
91          Rstar, minOrder+refineLevel, 0, taus)
92       endif
93    end subroutine get_all_tau_adaptive
94
95    !--------------------------------------------------------------------------
96    !+
97    !  Return all the directions of the rays that need to be traced for the
98    !  adaptive ray-tracing scheme
99    !+
100   !  IN: npart:            The number of SPH particles
101   !  IN: primary:          The xyz coordinates of the primary star
102   !  IN: companion:        The xyz coordinates of the companion
103   !  IN: Rcomp:            The radius of the companion
104   !  IN: xyzh:             The array containing the particles position+smooting lenght
105   !  IN: minOrder:         The minimal order in which the rays are sampled
106   !  IN: refineLevel:      The amount of orders in which the rays can be
107   !                        sampled deeper
108   !  IN: refineScheme:     The refinement scheme used for adaptive ray selection
109   !+
110   !  OUT: rays:            A list containing the rays that need to be traced
111   !                        in the adaptive ray-tracing scheme
112   !  OUT: indices:         A list containing a link between the index in the
113   !                        deepest order and the rays in the adaptive ray-tracing scheme
114   !  OUT: nrays:           The number of rays after the ray selection
115   !+
```

```
116     !-----------------------------------------------------------------------
117     subroutine get_rays(npart, primary, companion, Rcomp, xyzh, minOrder, refineLevel,
         refineScheme, rays, indices, nrays)
118       integer, intent(in)  :: npart, minOrder, refineLevel, refineScheme
119       real, intent(in)     :: primary(3), companion(3), xyzh(:,:), Rcomp
120       real, allocatable, intent(out)    :: rays(:,:)
121       integer, allocatable, intent(out) :: indices(:)
122       integer, intent(out) :: nrays
123
124       real    :: theta, dist, phi, cosphi, sinphi
125       real, dimension(:,:), allocatable  :: circ
126       integer :: i, j, minNsides, minNrays, ind,n, maxOrder, max, distr(12*4**(minOrder+
         refineLevel))
127       integer, dimension(:,:), allocatable  :: distrs
128
129       maxOrder = minOrder+refineLevel
130       nrays = 12*4**(maxOrder)
131       allocate(rays(3, nrays))
132       allocate(indices(12*4**(maxOrder)))
133       rays = 0.
134       indices = 0
135
136       !If there is no refinement, just return the uniform ray distribution
137       minNsides = 2**minOrder
138       minNrays = 12*4**minOrder
139       if (refineLevel == 0) then
140          do i=1, minNrays
141             call pix2vec_nest(minNsides,i-1, rays(:,i))
142             indices(i) = i
143          enddo
144          return
145       endif
146
147       !Fill a list to have the number distribution in angular space
148       distr = 0
149       !$omp parallel do private(ind)
150       do i = 1, npart
151          call vec2pix_nest(2**maxOrder, xyzh(1:3, i)-primary, ind)
152          distr(ind+1) = distr(ind+1)+1
153       enddo
154       max = maxval(distr)
155
156       !Make sure the companion is described using the highest refinement
157       dist = norm2(primary-companion)
158       theta = asin(Rcomp/dist)
159       phi = atan2(companion(2)-primary(2),companion(1)-primary(1))
160       cosphi = cos(phi)
161       sinphi = sin(phi)
162       dist = dist*cos(theta)
163       n = int(theta*6*2**(minOrder+refineLevel))+4
164       allocate(circ(n,3))
165       do i=1, n !Define boundary of the companion
166          circ(i,1) = dist*cos(theta)
167          circ(i,2) = dist*sin(theta)*cos(twopi*i/n)
168          circ(i,3) = dist*sin(theta)*sin(twopi*i/n)
169          circ(i,:) = (/cosphi*circ(i,1) - sinphi*circ(i,2),sinphi*circ(i,1) + cosphi*circ
        (i,2), circ(i,3)/)
170       enddo
171       do i=1, n !Make sure the boundary is maximally refined
172          call vec2pix_nest(2**maxOrder,circ(i,:),ind)
173          distr(ind+1) = max
174       enddo
175
176       !Calculate the number distribution in all the orders needed
177       allocate(distrs(12*4**(minOrder+refineLevel),refineLevel+1))
178       distrs = 0
179       distrs(:,1) = distr
180       do i = 1, refineLevel
181          do j = 1, 12*4**(maxOrder-i)
182             distrs(j,i+1) = distrs(4*j,i)+distrs(4*j+1,i)+distrs(4*j+2,i)+distrs(4*j+3,i)
183          enddo
```

```
184       enddo
185       max = maxval(distrs(:,refineLevel+1))+1
186
187       !Fill the rays array walking through the orders
188       ind=1
189
190       ! refine half in each order
191       if (refineScheme == 1) then
192          do i=0, refineLevel-1
193             call merge_argsort(distrs(1:12*4**(minOrder+i),refineLevel-i+1), distr)
194             do j=1, 6*4**minOrder*2**(i)
195                call pix2vec_nest(2**(minOrder+i), distr(j)-1, rays(:,ind))
196                indices(4**(refineLevel-i)*(distr(j)-1)+1:4**(refineLevel-i)*distr(j)) =
    ind
197                ind=ind+1
198                distrs(4*(distr(j)-1)+1:4*(distr(j)), refineLevel-i) = max
199             enddo
200             do j = j+1, 12*4**(minOrder+i)
201                if (distrs(distr(j),refineLevel-i+1) == max) then
202                   distrs(4*(distr(j)-1)+1:4*(distr(j)), refineLevel-i) = max
203                endif
204             enddo
205          enddo
206
207       ! refine overdens regions in each order
208       else if (refineScheme == 2) then
209          do i=0, refineLevel-1
210             call merge_argsort(distrs(1:12*4**(minOrder+i),refineLevel-i+1), distr)
211             j=1
212             do while (distrs(distr(j),refineLevel-i+1)<npart/(12*4**(minOrder+i)))
213                call pix2vec_nest(2**(minOrder+i), distr(j)-1, rays(:,ind))
214                indices(4**(refineLevel-i)*(distr(j)-1)+1:4**(refineLevel-i)*distr(j)) =
    ind
215                ind=ind+1
216                distrs(4*(distr(j)-1)+1:4*(distr(j)), refineLevel-i) = max
217                j=j+1
218             enddo
219             do j = j, 12*4**(minOrder+i)
220                if (distrs(distr(j),refineLevel-i+1) == max) then
221                   distrs(4*(distr(j)-1)+1:4*(distr(j)), refineLevel-i) = max
222                endif
223             enddo
224          enddo
225       endif
226
227       do i=1, 12*4**maxOrder
228          if (distrs(i,1) .ne. max) then
229             call pix2vec_nest(2**maxOrder, i-1, rays(:,ind))
230             indices(i) = ind
231             ind=ind+1
232          endif
233       enddo
234       nrays = ind-1
235    end subroutine get_rays
236
237    !-----------------------------------------------------------------------
238    !+
239    !  Routine that returns the arguments of the sorted array
240    !  Source: https://github.com/Astrokiwi/simple_fortran_argsort/blob/master/sort_test.
    f90
241    !+
242    !-----------------------------------------------------------------------
243    subroutine merge_argsort(r,d)
244       integer, intent(in), dimension(:) :: r
245       integer, intent(out), dimension(size(r)) :: d
246
247       integer, dimension(size(r)) :: il
248
249       integer :: stepsize
250       integer :: i,j,n,left,k,ksize
251
```

```
252         n = size(r)
253
254         do i=1,n
255               d(i)=i
256         end do
257
258         if ( n==1 ) return
259
260         stepsize = 1
261         do while (stepsize<n)
262            do left=1,n-stepsize,stepsize*2
263               i = left
264               j = left+stepsize
265               ksize = min(stepsize*2,n-left+1)
266               k=1
267
268               do while ( i<left+stepsize .and. j<left+ksize )
269                  if ( r(d(i))<r(d(j)) ) then
270                        il(k)=d(i)
271                        i=i+1
272                        k=k+1
273                  else
274                        il(k)=d(j)
275                        j=j+1
276                        k=k+1
277                  endif
278               enddo
279
280               if ( i<left+stepsize ) then
281                  ! fill up remaining from left
282                  il(k:ksize) = d(i:left+stepsize-1)
283               else
284                  ! fill up remaining from right
285                  il(k:ksize) = d(j:left+ksize-1)
286               endif
287               d(left:left+ksize-1) = il(1:ksize)
288            end do
289            stepsize=stepsize*2
290         end do
291      end subroutine merge_argsort
292
293      !*********************************************************************!
294      !*************************    OUTWARDS    ****************************!
295      !*********************************************************************!
296
297      !---------------------------------------------------------------------
298      !+
299      !  Calculate the optical depth of each particle, using the uniform outwards
300      !  ray-tracing scheme
301      !+
302      !  IN: npart:           The number of SPH particles
303      !  IN: primary:         The xyz coordinates of the primary star
304      !  IN: xyzh:            The array containing the particles position+smooting lenght
305      !  IN: kappa:           The array containing the kappa of all SPH particles
306      !  IN: Rstar:           The radius of the star
307      !  IN: order:           The order in which the rays are sampled
308      !  IN: raypolation:     The interpolation scheme used for the ray interpolation
309      !+
310      !  OUT: taus:           The list of optical depths for each particle
311      !+
312      !  OPT: companion:      The location of the companion
313      !  OPT: Rcomp:          The radius of the primary star
314      !+
315      !---------------------------------------------------------------------
316      subroutine get_all_tau_outwards(npart, primary, xyzh, kappa, Rstar, order, raypolation
          , taus, companion, Rcomp)
317         integer, intent(in) :: npart, order, raypolation
318         real, intent(in)    :: primary(3), kappa(:), Rstar, xyzh(:,:)
319         real, optional      :: Rcomp, companion(3)
320         real, intent(out)   :: taus(:)
321
```

```
322       if (present(companion) .and. present(Rcomp)) then
323          call get_all_tau_outwards_companion(npart, primary, xyzh, kappa, Rstar,
      companion, Rcomp, order, raypolation, taus)
324       else
325          call get_all_tau_outwards_single(npart, primary, xyzh, kappa, Rstar, order,
      raypolation, taus)
326       endif
327    end subroutine get_all_tau_outwards
328
329    !-----------------------------------------------------------------------
330    !+
331    !  Calculates the optical depth to each SPH particle, using the uniform outwards
332    !  ray-tracing scheme for models containing a single star
333    !
334    !  Relies on healpix, for more information: https://healpix.sourceforge.io/
335    !+
336    !  IN: npart:           The number of SPH particles
337    !  IN: primary:         The xyz coordinates of the primary star
338    !  IN: xyzh:            The array containing the particles position+smooting lenght
339    !  IN: kappa:           The array containing the kappa of all SPH particles
340    !  IN: Rstar:           The radius of the primary star
341    !  IN: order:           The healpix order which is used for the uniform ray sampling
342    !  IN: raypolation:     The interpolation scheme used for the ray interpolation
343    !+
344    !  OUT: taus:           The array of optical depths to each SPH particle
345    !+
346    !-----------------------------------------------------------------------
347    subroutine get_all_tau_outwards_single(npart, primary, xyzh, kappa, Rstar, order,
      raypolation, tau)
348       use part, only : isdead_or_accreted
349       integer, intent(in) :: npart,order, raypolation
350       real, intent(in)    :: primary(3), kappa(:), Rstar, xyzh(:,:)
351       real, intent(out)   :: tau(:)
352
353       integer  :: i, nrays, nsides
354       real     :: ray_dir(3),part_dir(3)
355       real, dimension(:,:), allocatable  :: rays_dist, rays_tau
356       integer, dimension(:), allocatable :: rays_dim
357       integer, parameter :: ndim = 200
358
359       nrays = 12*4**order ! The number of rays traced given the healpix order
360       nsides = 2**order   ! The healpix nsides given the healpix order
361
362       allocate(rays_dist(ndim, nrays))
363       allocate(rays_tau(ndim, nrays))
364       allocate(rays_dim(nrays))
365
366       !-------------------------------------------------
367       ! CONSTRUCT the RAYS given the ORDER
368       ! and determine the optical depth along them
369       !-------------------------------------------------
370
371       !$omp parallel default(none) &
372       !$omp private(ray_dir) &
373       !$omp shared(nrays,nsides,primary,kappa,xyzh,Rstar,rays_dist,rays_tau,rays_dim)
374       !$omp do
375       do i = 1, nrays
376          !returns ray_dir, the unit vector identifying ray
377          !        (index i-1 becase healpix starts counting at index 0)
378          call pix2vec_nest(nsides, i-1, ray_dir)
379          !calculate the properties along the ray
380          call ray_tracer(primary,ray_dir,xyzh,kappa,Rstar,rays_tau(:,i),rays_dist(:,i),
      rays_dim(i))
381       enddo
382       !$omp enddo
383       !$omp end parallel
384
385
386       !-------------------------------------------------
387       ! DETERMINE the optical depth for each particle
388       ! using the values available on the rays
```

```
389       !--------------------------------------------------------
390
391       !$omp parallel default(none) &
392       !$omp private(part_dir) &
393       !$omp shared(npart,primary,nsides,xyzh,ray_dir,rays_dist,rays_tau,rays_dim,
      raypolation,tau)
394       !$omp do
395       do i = 1,npart
396          if (.not.isdead_or_accreted(xyzh(4,i))) then
397             part_dir = xyzh(1:3,i)-primary
398             call interpolate_tau(nsides, part_dir, rays_tau, rays_dist, rays_dim,
      raypolation, tau(i))
399          else
400             tau(i) = -99.
401          endif
402       enddo
403       !$omp enddo
404       !$omp end parallel
405    end subroutine get_all_tau_outwards_single
406
407    !-----------------------------------------------------------------------
408    !+
409    !  Calculates the optical depth to each SPH particle, using the uniform outwards
410    !  ray-tracing scheme for models containing primary star and a companion
411    !
412    !  Relies on healpix, for more information: https://healpix.sourceforge.io/
413    !+
414    !  IN: npart:           The number of SPH particles
415    !  IN: primary:         The xyz coordinates of the primary star
416    !  IN: xyzh:            The array containing the particles position+smooting lenght
417    !  IN: kappa:           The array containing the opacity of all the SPH particles
418    !  IN: Rstar:           The radius of the primary star
419    !  IN: companion:       The xyz coordinates of the companion
420    !  IN: Rcomp:           The radius of the companion
421    !  IN: order:           The healpix order which is used for the uniform ray sampling
422    !  IN: raypolation:     The interpolation scheme used for the ray interpolation
423    !+
424    !  OUT: tau:            The array of optical depths for each SPH particle
425    !+
426    !-----------------------------------------------------------------------
427    subroutine get_all_tau_outwards_companion(npart, primary, xyzh, kappa, Rstar,
      companion, Rcomp, order, raypolation, tau)
428       use part, only : isdead_or_accreted
429       integer, intent(in) :: npart, order, raypolation
430       real, intent(in)    :: primary(3), companion(3), kappa(:), Rstar, xyzh(:,:), Rcomp
431       real, intent(out)   :: tau(:)
432
433       integer  :: i, nrays, nsides
434       real     :: normCompanion,theta0,phi,cosphi,sinphi,theta,sep,root
435       real     :: ray_dir(3),part_dir(3),uvecCompanion(3)
436       real, dimension(:,:), allocatable  :: dirs
437       real, dimension(:,:), allocatable  :: rays_dist, rays_tau
438       integer, dimension(:), allocatable :: rays_dim
439       integer, parameter :: ndim = 200
440
441       nrays = 12*4**order ! The number of rays traced given the healpix order
442       nsides = 2**order   ! The healpix nsides given the healpix order
443
444       allocate(dirs(3, nrays))
445       allocate(rays_dist(ndim, nrays))
446       allocate(rays_tau(ndim, nrays))
447       allocate(rays_dim(nrays))
448
449       uvecCompanion = companion-primary
450       normCompanion = norm2(uvecCompanion)
451       uvecCompanion = uvecCompanion/normCompanion
452       theta0        = asin(Rcomp/normCompanion)
453       phi           = atan2(uvecCompanion(2),uvecCompanion(1))
454       cosphi        = cos(phi)
455       sinphi        = sin(phi)
456
```

```
457       !-----------------------------------------------
458       ! CONSTRUCT the RAYS given the ORDER
459       ! and determine the optical depth along them
460       !-----------------------------------------------
461
462       !$omp parallel default(none) &
463       !$omp private(ray_dir,theta,root,sep) &
464       !$omp shared(nrays,nsides,primary,kappa,xyzh,Rstar,Rcomp,rays_dist,rays_tau,
      rays_dim) &
465       !$omp shared(uvecCompanion,normCompanion,cosphi,sinphi,theta0)
466       !$omp do
467       do i = 1, nrays
468          !returns ray_dir, the unit vector identifying ray
469          !        (index i-1 becase healpix starts counting at index 0)
470          call pix2vec_nest(nsides, i-1, ray_dir)
471          !rotate ray vectors by an angle = phi so the main axis points to the companion
472          !This is because along the main axis (1,0,0) rays are distributed more
      uniformly
473          ray_dir = (/cosphi*ray_dir(1) - sinphi*ray_dir(2),sinphi*ray_dir(1) + cosphi*
      ray_dir(2), ray_dir(3)/)
474          theta   = acos(dot_product(uvecCompanion, ray_dir))
475          !the ray intersects the companion: only calculate tau up to the companion
476          if (theta < theta0) then
477             root  = sqrt(Rcomp**2-normCompanion**2*sin(theta)**2)
478             sep   = normCompanion*cos(theta)-root
479             call ray_tracer(primary,ray_dir,xyzh,kappa,Rstar,rays_tau(:,i),rays_dist(:,i)
      ,rays_dim(i), sep)
480          else
481             call ray_tracer(primary,ray_dir,xyzh,kappa,Rstar,rays_tau(:,i),rays_dist(:,i)
      ,rays_dim(i))
482          endif
483       enddo
484       !$omp enddo
485       !$omp end parallel
486
487       !-----------------------------------------------
488       ! DETERMINE the optical depth for each particle
489       ! using the values available on the rays
490       !-----------------------------------------------
491
492       !$omp parallel default(none) &
493       !$omp private(part_dir) &
494       !$omp shared(npart,primary,cosphi,sinphi,nsides,xyzh,ray_dir,rays_dist,rays_tau,
      rays_dim,raypolation,tau)
495       !$omp do
496       do i = 1, npart
497          if (.not.isdead_or_accreted(xyzh(4,i))) then
498             !vector joining the source to the particle
499             part_dir = xyzh(1:3,i)-primary
500             part_dir = (/cosphi*part_dir(1) + sinphi*part_dir(2),-sinphi*part_dir(1) +
      cosphi*part_dir(2), part_dir(3)/)
501             call interpolate_tau(nsides, part_dir, rays_tau, rays_dist, rays_dim,
      raypolation, tau(i))
502          else
503             tau(i) = -99.
504          endif
505       enddo
506       !$omp enddo
507       !$omp end parallel
508    end subroutine get_all_tau_outwards_companion
509
510    !-----------------------------------------------------------------------
511    !+
512    !  Calculate the optical depth of a particle.
513    !  Search for the four closest rays to a particle, perform four-point
514    !  interpolation of the optical depts from these rays. Weighted by the
515    !  inverse square of the perpendicular distance to the rays.
516    !
517    !  Relies on healpix, for more information: https://healpix.sourceforge.io/
518    !+
519    !  IN: nsides:          The healpix nsides of the simulation
```

```
520   !  IN: vec:              The vector from the primary to a point
521   !  IN: rays_tau:         2-dimensional array containing the cumulative optical
522   !                        depts along each ray
523   !  IN: rays_dist:        2-dimensional array containing the distances from the
524   !                        primary along each ray
525   !  IN: rays_dim:         The vector containing the number of points defined along
526   !                        each ray
527   !  IN: raypolation:      The interpolation scheme used for the ray interpolation
528   !+
529   !  OUT: tau:             The interpolated optical depth at the particle's location
530   !+
531   !-----------------------------------------------------------------------------
532   subroutine interpolate_tau(nsides, vec, rays_tau, rays_dist, rays_dim, raypolation,
       tau)
533     integer, intent(in) :: nsides, rays_dim(:), raypolation
534     real, intent(in)    :: vec(:), rays_tau(:,:), rays_dist(:,:)
535     real, intent(out)   :: tau
536
537     integer :: rayIndex, neighbours(8), nneigh, i, k
538     real    :: tautemp, ray(3), vectemp(3), weight, tempdist(8), distRay_sq, vec_norm2
539     logical :: mask(8)
540
541     ! 1 ray, no interpolation
542     if (raypolation==0) then
543        call vec2pix_nest(nsides, vec, rayIndex)
544        rayIndex = rayIndex + 1
545        call get_tau_on_ray(norm2(vec), rays_tau(:,rayIndex), rays_dist(:,rayIndex),
       rays_dim(rayIndex), tau)
546
547     ! 4 rays, linear interpolation
548     else if (raypolation==1) then
549        vec_norm2 = norm2(vec)
550        !returns rayIndex, the index of the ray vector that points to the particle
551        call vec2pix_nest(nsides, vec, rayIndex)
552        !returns ray(3), the unit vector identifying the ray with index number rayIndex
553        call pix2vec_nest(nsides, rayIndex, ray)
554        vectemp       = vec - vec_norm2*ray
555        distRay_sq    = norm2(vectemp)
556        call get_tau_on_ray(vec_norm2, rays_tau(:,rayIndex+1), rays_dist(:,rayIndex+1),
       rays_dim(rayIndex+1), tautemp)
557        if (distRay_sq > 0.) then
558           tau    = tautemp/distRay_sq
559           weight = 1./distRay_sq
560        else
561           ! the particle sits exactly on the ray, no need to get the neighbours
562           tau    = tautemp
563           return
564        endif
565
566        !returns the number nneigh and list of vectors neighbouring the ray number index
567        call neighbours_nest(nsides, rayIndex, neighbours, nneigh)
568        !for each neighbouring ray calculate its distance to the particle
569        do i=1,nneigh
570           call pix2vec_nest(nsides, neighbours(i), ray)
571           vectemp     = vec - vec_norm2*ray
572           tempdist(i) = norm2(vectemp)
573        enddo
574        neighbours       = neighbours+1
575        mask             = .true.
576        if (nneigh <8) mask(nneigh+1:8) = .false.
577        !take tau contribution from the 3 closest rays
578        do i=1,3
579           k       = minloc(tempdist,1,mask)
580           mask(k) = .false.
581           call get_tau_on_ray(vec_norm2, rays_tau(:,neighbours(k)), &
582                  rays_dist(:,neighbours(k)), rays_dim(neighbours(k)), tautemp)
583           tau    = tau + tautemp/tempdist(k)
584           weight = weight + 1./tempdist(k)
585        enddo
586        tau = tau / weight
587
```

```
588       ! 9 rays, linear interpolation
589       else if (raypolation==2) then
590          vec_norm2 = norm2(vec)
591          !returns rayIndex, the index of the ray vector that points to the particle
592          call vec2pix_nest(nsides, vec, rayIndex)
593          !returns ray(3), the unit vector identifying the ray with index number rayIndex
594          call pix2vec_nest(nsides, rayIndex, ray)
595          vectemp       = vec - vec_norm2*ray
596          distRay_sq    = norm2(vectemp)
597          call get_tau_on_ray(vec_norm2, rays_tau(:,rayIndex+1), rays_dist(:,rayIndex+1),
     rays_dim(rayIndex+1), tautemp)
598          if (distRay_sq > 0.) then
599             tau    = tautemp/distRay_sq
600             weight = 1./distRay_sq
601          else
602             ! the particle sits exactly on the ray, no need to get the neighbours
603             tau    = tautemp
604             return
605          endif
606
607          !returns the number nneigh and list of vectors neighbouring the ray number index
608          call neighbours_nest(nsides, rayIndex, neighbours, nneigh)
609          !for each neighbouring ray calculate its distance to the particle
610          do i=1,nneigh
611             call pix2vec_nest(nsides, neighbours(i), ray)
612             vectemp    = vec - vec_norm2*ray
613             tempdist(i) = norm2(vectemp)
614          enddo
615          neighbours       = neighbours+1
616          mask             = .true.
617          if (nneigh <8) mask(nneigh+1:8) = .false.
618          !take tau contribution from the 3 closest rays
619          do i=1,nneigh
620             k       = minloc(tempdist,1,mask)
621             mask(k) = .false.
622             call get_tau_on_ray(vec_norm2, rays_tau(:,neighbours(k)), &
623                    rays_dist(:,neighbours(k)), rays_dim(neighbours(k)), tautemp)
624             tau    = tau + tautemp/tempdist(k)
625             weight = weight + 1./tempdist(k)
626          enddo
627          tau = tau / weight
628
629       ! 4 rays, square interpolation
630       else if (raypolation==3) then
631          vec_norm2 = norm2(vec)
632          !returns rayIndex, the index of the ray vector that points to the particle
633          call vec2pix_nest(nsides, vec, rayIndex)
634          !returns ray(3), the unit vector identifying the ray with index number rayIndex
635          call pix2vec_nest(nsides, rayIndex, ray)
636          vectemp       = vec - vec_norm2*ray
637          distRay_sq    = dot_product(vectemp,vectemp)
638          call get_tau_on_ray(vec_norm2, rays_tau(:,rayIndex+1), rays_dist(:,rayIndex+1),
     rays_dim(rayIndex+1), tautemp)
639          if (distRay_sq > 0.) then
640             tau    = tautemp/distRay_sq
641             weight = 1./distRay_sq
642          else
643             ! the particle sits exactly on the ray, no need to get the neighbours
644             tau    = tautemp
645             return
646          endif
647
648          !returns the number nneigh and list of vectors neighbouring the ray number index
649          call neighbours_nest(nsides, rayIndex, neighbours, nneigh)
650          !for each neighbouring ray calculate its distance to the particle
651          do i=1,nneigh
652             call pix2vec_nest(nsides, neighbours(i), ray)
653             vectemp    = vec - vec_norm2*ray
654             tempdist(i) = dot_product(vectemp,vectemp)
655          enddo
656          neighbours       = neighbours+1
```

```
657           mask              = .true.
658           if (nneigh <8) mask(nneigh+1:8) = .false.
659           !take tau contribution from the 3 closest rays
660           do i=1,3
661              k       = minloc(tempdist,1,mask)
662              mask(k) = .false.
663              call get_tau_on_ray(vec_norm2, rays_tau(:,neighbours(k)), &
664                     rays_dist(:,neighbours(k)), rays_dim(neighbours(k)), tautemp)
665              tau     = tau + tautemp/tempdist(k)
666              weight = weight + 1./tempdist(k)
667           enddo
668           tau = tau / weight
669
670        ! 9 rays, square interpolation
671        else if (raypolation==4) then
672           vec_norm2 = norm2(vec)
673           !returns rayIndex, the index of the ray vector that points to the particle
674           call vec2pix_nest(nsides, vec, rayIndex)
675           !returns ray(3), the unit vector identifying the ray with index number rayIndex
676           call pix2vec_nest(nsides, rayIndex, ray)
677           vectemp       = vec - vec_norm2*ray
678           distRay_sq    = dot_product(vectemp,vectemp)
679           call get_tau_on_ray(vec_norm2, rays_tau(:,rayIndex+1), rays_dist(:,rayIndex+1),
    rays_dim(rayIndex+1), tautemp)
680           if (distRay_sq > 0.) then
681              tau     = tautemp/distRay_sq
682              weight = 1./distRay_sq
683           else
684              ! the particle sits exactly on the ray, no need to get the neighbours
685              tau     = tautemp
686              return
687           endif
688
689           !returns the number nneigh and list of vectors neighbouring the ray number index
690           call neighbours_nest(nsides, rayIndex, neighbours, nneigh)
691           !for each neighbouring ray calculate its distance to the particle
692           do i=1,nneigh
693              call pix2vec_nest(nsides, neighbours(i), ray)
694              vectemp      = vec - vec_norm2*ray
695              tempdist(i) = dot_product(vectemp,vectemp)
696           enddo
697           neighbours       = neighbours+1
698           mask             = .true.
699           if (nneigh <8) mask(nneigh+1:8) = .false.
700           !take tau contribution from the 3 closest rays
701           do i=1,nneigh
702              k       = minloc(tempdist,1,mask)
703              mask(k) = .false.
704              call get_tau_on_ray(vec_norm2, rays_tau(:,neighbours(k)), &
705                     rays_dist(:,neighbours(k)), rays_dim(neighbours(k)), tautemp)
706              tau     = tau + tautemp/tempdist(k)
707              weight = weight + 1./tempdist(k)
708           enddo
709           tau = tau / weight
710
711        ! 4 rays, cubed interpolation
712        else if (raypolation==5) then
713           vec_norm2 = norm2(vec)
714           !returns rayIndex, the index of the ray vector that points to the particle
715           call vec2pix_nest(nsides, vec, rayIndex)
716           !returns ray(3), the unit vector identifying the ray with index number rayIndex
717           call pix2vec_nest(nsides, rayIndex, ray)
718           vectemp       = vec - vec_norm2*ray
719           distRay_sq    = norm2(vectemp)**3
720           call get_tau_on_ray(vec_norm2, rays_tau(:,rayIndex+1), rays_dist(:,rayIndex+1),
    rays_dim(rayIndex+1), tautemp)
721           if (distRay_sq > 0.) then
722              tau     = tautemp/distRay_sq
723              weight = 1./distRay_sq
724           else
725              ! the particle sits exactly on the ray, no need to get the neighbours
```

```
726             tau     = tautemp
727             return
728          endif
729
730          !returns the number nneigh and list of vectors neighbouring the ray number index
731          call neighbours_nest(nsides, rayIndex, neighbours, nneigh)
732          !for each neighbouring ray calculate its distance to the particle
733          do i=1,nneigh
734             call pix2vec_nest(nsides, neighbours(i), ray)
735             vectemp      = vec - vec_norm2*ray
736             tempdist(i) = norm2(vectemp)**3
737          enddo
738          neighbours        = neighbours+1
739          mask              = .true.
740          if (nneigh <8) mask(nneigh+1:8) = .false.
741          !take tau contribution from the 3 closest rays
742          do i=1,3
743             k        = minloc(tempdist,1,mask)
744             mask(k) = .false.
745             call get_tau_on_ray(vec_norm2, rays_tau(:,neighbours(k)), &
746                    rays_dist(:,neighbours(k)), rays_dim(neighbours(k)), tautemp)
747             tau     = tau + tautemp/tempdist(k)
748             weight = weight + 1./tempdist(k)
749          enddo
750          tau = tau / weight
751
752       ! 9 rays, cubed interpolation
753       else if (raypolation==6) then
754          vec_norm2 = norm2(vec)
755          !returns rayIndex, the index of the ray vector that points to the particle
756          call vec2pix_nest(nsides, vec, rayIndex)
757          !returns ray(3), the unit vector identifying the ray with index number rayIndex
758          call pix2vec_nest(nsides, rayIndex, ray)
759          vectemp        = vec - vec_norm2*ray
760          distRay_sq     = norm2(vectemp)**3
761          call get_tau_on_ray(vec_norm2, rays_tau(:,rayIndex+1), rays_dist(:,rayIndex+1),
     rays_dim(rayIndex+1), tautemp)
762          if (distRay_sq > 0.) then
763             tau     = tautemp/distRay_sq
764             weight = 1./distRay_sq
765          else
766             ! the particle sits exactly on the ray, no need to get the neighbours
767             tau     = tautemp
768             return
769          endif
770
771          !returns the number nneigh and list of vectors neighbouring the ray number index
772          call neighbours_nest(nsides, rayIndex, neighbours, nneigh)
773          !for each neighbouring ray calculate its distance to the particle
774          do i=1,nneigh
775             call pix2vec_nest(nsides, neighbours(i), ray)
776             vectemp      = vec - vec_norm2*ray
777             tempdist(i) = norm2(vectemp)**3
778          enddo
779          neighbours        = neighbours+1
780          mask              = .true.
781          if (nneigh <8) mask(nneigh+1:8) = .false.
782          !take tau contribution from the 3 closest rays
783          do i=1,nneigh
784             k        = minloc(tempdist,1,mask)
785             mask(k) = .false.
786             call get_tau_on_ray(vec_norm2, rays_tau(:,neighbours(k)), &
787                    rays_dist(:,neighbours(k)), rays_dim(neighbours(k)), tautemp)
788             tau     = tau + tautemp/tempdist(k)
789             weight = weight + 1./tempdist(k)
790          enddo
791          tau = tau / weight
792       endif
793    end subroutine interpolate_tau
794
795
```

```
796   !-------------------------------------------------------------------------------
797   !+
798   !  Interpolation of the optical depth for an arbitrary point on the ray,
799   !  with a given distance to the starting point of the ray.
800   !+
801   !  IN: distance:         The distance from the staring point of the ray to a
802   !                        point on the ray
803   !  IN: tau_along_ray:    The vector of cumulative optical depths along the ray
804   !  IN: dist_along_ray:   The vector of distances from the primary along the ray
805   !  IN: len:              The length of listOfTau and listOfDist
806   !+
807   !  OUT: tau:             The optical depth to the given distance along the ray
808   !+
809   !-------------------------------------------------------------------------------
810   subroutine get_tau_on_ray(distance, tau_along_ray, dist_along_ray, len, tau)
811      real, intent(in)    :: distance, tau_along_ray(:), dist_along_ray(:)
812      integer, intent(in) :: len
813      real, intent(out)   :: tau
814
815      integer :: L, R, m ! left, right and middle index for binary search
816
817      if (distance .lt. dist_along_ray(1)) then
818         tau = 0.
819      else if (distance .gt. dist_along_ray(len)) then
820         tau = 99.
821      else
822         L = 2
823         R = len-1
824         !bysection search for the index of the closest ray location to the particle
825         do while (L < R)
826            m = (L + R)/2
827            if (dist_along_ray(m) > distance) then
828                  R = m
829            else
830                  L = m + 1
831            end if
832         enddo
833         !interpolate linearly ray properties to get the particle's optical depth
834         tau = tau_along_ray(L-1)+(tau_along_ray(L)-tau_along_ray(L-1))/ &
835               (dist_along_ray(L)-dist_along_ray(L-1))*(distance-dist_along_ray(L-1))
836      endif
837   end subroutine get_tau_on_ray
838
839   !-------------------------------------------------------------------------------
840   !+
841   !  Calculate the optical depth along a given ray
842   !+
843   !  IN: primary:          The location of the primary star
844   !  IN: ray:              The unit vector of the direction in which the
845   !                        optical depts will be calculated
846   !  IN: xyzh:             The array containing the particles position+smooting lenght
847   !  IN: kappa:            The array containing the particles opacity
848   !  IN: Rstar:            The radius of the primary star
849   !+
850   !  OUT: taus:            The distribution of optical depths throughout the ray
851   !  OUT: listOfDists:     The distribution of distances throughout the ray
852   !  OUT: len:             The length of tau_along_ray and dist_along_ray
853   !+
854   !  OPT: maxDistance:     The maximal distance the ray needs to be traced
855   !+
856   !-------------------------------------------------------------------------------
857   subroutine ray_tracer(primary, ray, xyzh, kappa, Rstar, tau_along_ray, dist_along_ray,
        len, maxDistance)
858      use linklist, only:getneigh_pos,ifirstincell,listneigh
859      use kernel,   only:radkern
860      use units, only:umass,udist
861      real, intent(in)      :: primary(3), ray(3), Rstar, xyzh(:,:), kappa(:)
862      real, optional        :: maxDistance
863      real, intent(out)     :: dist_along_ray(:), tau_along_ray(:)
864      integer, intent(out) :: len
865
```

```
866       integer, parameter :: maxcache = 0
867       real, allocatable  :: xyzcache(:,:)
868       real :: distance, h, dtaudr, previousdtaudr, nextdtaudr
869       integer :: nneigh, inext, i
870
871       distance = Rstar
872
873       h = Rstar/100.
874       inext=0
875       do while (inext==0)
876          h = h*2.
877          call getneigh_pos(primary+Rstar*ray,0.,h,3,listneigh,nneigh,xyzh,xyzcache,
      maxcache,ifirstincell)
878          call find_next(primary, ray, distance, xyzh, listneigh, inext, nneigh)
879       enddo
880       call calc_opacity(primary+Rstar*ray, xyzh, kappa, listneigh, nneigh, previousdtaudr
      )
881
882       i = 1
883       tau_along_ray(i)  = 0.
884       distance          = Rstar
885       dist_along_ray(i) = distance
886       do while (hasNext(inext,tau_along_ray(i),distance,maxDistance))
887          i = i + 1
888          call getneigh_pos(primary + distance*ray,0.,xyzh(4,inext)*radkern, &
889                            3,listneigh,nneigh,xyzh,xyzcache,maxcache,ifirstincell)
890          call calc_opacity(primary + distance*ray, xyzh, kappa, listneigh, nneigh,
      nextdtaudr)
891          dtaudr            = (nextdtaudr+previousdtaudr)/2
892          previousdtaudr    = nextdtaudr
893          tau_along_ray(i)  = tau_along_ray(i-1)+(distance-dist_along_ray(i-1))*dtaudr
894          dist_along_ray(i) = distance
895          call find_next(primary, ray, distance, xyzh, listneigh, inext,nneigh)
896       enddo
897       len = i
898       tau_along_ray = tau_along_ray*umass/(udist**2)
899    end subroutine ray_tracer
900
901    logical function hasNext(inext, tau, distance, maxDistance)
902       integer, intent(in) :: inext
903       real, intent(in)    :: distance, tau
904       real, optional      :: maxDistance
905       real, parameter :: tau_max = 99.
906       if (present(maxDistance)) then
907          hasNext = inext /= 0 .and. distance < maxDistance .and. tau < tau_max
908       else
909          hasNext = inext /= 0 .and. tau < tau_max
910       endif
911    end function hasNext
912
913    !*****************************************************************!
914    !*************************    INWARDS    *************************!
915    !*****************************************************************!
916
917    !-----------------------------------------------------------------------
918    !+
919    !  Calculate the optical depth of each particle, using the inwards ray-
920    !  tracing scheme
921    !+
922    !  IN: npart:           The number of SPH particles
923    !  IN: primary:         The xyz coordinates of the primary star
924    !  IN: xyzh:            The array containing the particles position+smooting lenght
925    !  IN: neighbors:       A list containing the indices of the neighbors of
926    !                       each particle
927    !  IN: kappa:           The array containing the opacity of all the SPH particles
928    !  IN: Rstar:           The radius of the primary star
929    !+
930    !  OUT: tau:            The array of optical depths for each SPH particle
931    !+
932    !  OPT: companion:      The location of the companion
933    !  OPT: R:              The radius of the companion
```

```
934     !+
935     !-----------------------------------------------------------------------------
936     subroutine get_all_tau_inwards(npart, primary, xyzh, neighbors, kappa, Rstar, tau,
         companion, R)
937        real, intent(in)    :: primary(3), kappa(:), Rstar, xyzh(:,:)
938        integer, intent(in) :: npart, neighbors(:,:)
939        real, optional      :: R, companion(3)
940        real, intent(out)   :: tau(:)
941
942        if (present(companion) .and. present(R)) then
943           call get_all_tau_inwards_companion(npart, primary, xyzh, neighbors, kappa, Rstar
         , companion, R, tau)
944        else
945           call get_all_tau_inwards_single(npart, primary, xyzh, neighbors, kappa, Rstar,
         tau)
946        endif
947     end subroutine get_all_tau_inwards
948
949     !-----------------------------------------------------------------------------
950     !+
951     !  Calculate the optical depth of each particle, using the inwards ray-
952     !  tracing scheme concerning only a single star
953     !+
954     !  IN: npart:           The number of SPH particles
955     !  IN: primary:         The xyz coordinates of the primary star
956     !  IN: xyzh:            The array containing the particles position+smooting lenght
957     !  IN: neighbors:       A list containing the indices of the neighbors of
958     !                       each particle
959     !  IN: kappa:           The array containing the opacity of all the SPH particles
960     !  IN: Rstar:           The radius of the primary star
961     !+
962     !  OUT: taus:           The list of optical depths for each particle
963     !+
964     !-----------------------------------------------------------------------------
965     subroutine get_all_tau_inwards_single(npart, primary, xyzh, neighbors, kappa, Rstar,
         tau)
966        real, intent(in)    :: primary(3), kappa(:), Rstar, xyzh(:,:)
967        integer, intent(in) :: npart, neighbors(:,:)
968        real, intent(out)   :: tau(:)
969
970        integer :: i
971
972        !$omp parallel do
973        do i = 1, npart
974           call get_tau_inwards(i, primary, xyzh, neighbors, kappa, Rstar, tau(i))
975        enddo
976        !$omp end parallel do
977     end subroutine get_all_tau_inwards_single
978
979     !-----------------------------------------------------------------------------
980     !+
981     !  Calculate the optical depth of each particle, using the inwards ray-
982     !  tracing scheme concerning a binary system
983     !+
984     !  IN: npart:           The number of SPH particles
985     !  IN: primary:         The xyz coordinates of the primary star
986     !  IN: xyzh:            The array containing the particles position+smooting lenght
987     !  IN: neighbors:       A list containing the indices of the neighbors of
988     !                       each particle
989     !  IN: kappa:           The array containing the opacity of all the SPH particles
990     !  IN: Rstar:           The radius of the primary star
991     !  IN: companion:       The xyz coordinates of the companion
992     !  IN: Rcomp:           The radius of the companion
993     !+
994     !  OUT: tau:            The array of optical depths for each SPH particle
995     !+
996     !-----------------------------------------------------------------------------
997     subroutine get_all_tau_inwards_companion(npart, primary, xyzh, neighbors, kappa, Rstar
         , companion, Rcomp, tau)
998        real, intent(in)    :: primary(3), companion(3), kappa(:), Rstar, xyzh(:,:), Rcomp
999        integer, intent(in) :: npart, neighbors(:,:)
```

```
1000        real, intent(out)    :: tau(:)
1001
1002        integer :: i
1003        real    :: normCompanion, theta0, uvecCompanion(3), norm, theta, root, norm0
1004
1005        uvecCompanion = companion-primary
1006        normCompanion = norm2(uvecCompanion)
1007        uvecCompanion = uvecCompanion/normCompanion
1008        theta0        = asin(Rcomp/normCompanion)
1009
1010        !$omp parallel do private(norm,theta,root,norm0)
1011        do i = 1, npart
1012           norm  = norm2(xyzh(1:3,i)-primary)
1013           theta = acos(dot_product(uvecCompanion, xyzh(1:3,i)-primary)/norm)
1014           if (theta < theta0) then
1015              root = sqrt(normCompanion**2*cos(theta)**2-normCompanion**2+Rcomp**2)
1016              norm0 = normCompanion*cos(theta)-root
1017              if (norm > norm0) then
1018                    tau(i) = 99.
1019              else
1020                 call get_tau_inwards(i, primary, xyzh, neighbors, kappa, Rstar, tau(i))
1021              endif
1022           else
1023              call get_tau_inwards(i, primary, xyzh, neighbors, kappa, Rstar, tau(i))
1024           endif
1025        enddo
1026        !$omp end parallel do
1027     end subroutine get_all_tau_inwards_companion
1028
1029     !-----------------------------------------------------------------------
1030     !+
1031     !  Calculate the optical depth for a given particle, using the inwards ray-
1032     !  tracing scheme
1033     !+
1034     !  IN: point:           The index of the point that needs to be calculated
1035     !  IN: primary:         The location of the primary star
1036     !  IN: xyzh:            The array containing the particles position+smooting lenght
1037     !  IN: neighbors:       A list containing the indices of the neighbors of
1038     !                       each particle
1039     !  IN: kappa:           The array containing the opacity of all the SPH particles
1040     !  IN: Rstar:           The radius of the star
1041     !+
1042     !  OUT: tau:           The list of optical depth of the given particle
1043     !+
1044     !-----------------------------------------------------------------------
1045     subroutine get_tau_inwards(point, primary, xyzh, neighbors, kappa, Rstar, tau)
1046        use linklist, only:getneigh_pos,ifirstincell,listneigh
1047        use kernel,   only:radkern
1048        use units, only:umass,udist
1049        real, intent(in)    :: primary(3), xyzh(:,:), kappa(:), Rstar
1050        integer, intent(in) :: point, neighbors(:,:)
1051        real, intent(out)   :: tau
1052
1053        integer :: i, next, previous, nneigh
1054        integer, parameter :: nmaxcache = 0
1055        real  :: xyzcache(0,nmaxcache)
1056        real    :: ray(3), nextDist, previousDist, maxDist, dtaudr, previousdtaudr,
     nextdtaudr
1057
1058        ray = primary - xyzh(1:3,point)
1059        maxDist = norm2(ray)
1060        ray = ray / maxDist
1061        maxDist=max(maxDist-Rstar,0.)
1062        next=point
1063        call getneigh_pos(xyzh(1:3,point),0.,xyzh(4,point)*radkern, &
1064                          3,listneigh,nneigh,xyzh,xyzcache,nmaxcache,ifirstincell)
1065        call calc_opacity(xyzh(1:3,point), xyzh, kappa, listneigh, nneigh, nextdtaudr)
1066        nextDist=0.
1067
1068        tau = 0.
1069        i=1
```

```
1070        do while (nextDist < maxDist .and. next /=0)
1071           i = i + 1
1072           previous = next
1073           previousDist = nextDist
1074           call find_next(xyzh(1:3,point), ray, nextDist, xyzh, neighbors(next,:), next)
1075           if (nextDist .gt. maxDist) then
1076                 nextDist = maxDist
1077           endif
1078           call getneigh_pos(xyzh(1:3,point) + nextDist*ray,0.,xyzh(4,previous)*radkern, &
1079                             3,listneigh,nneigh,xyzh,xyzcache,nmaxcache,ifirstincell)
1080           previousdtaudr=nextdtaudr
1081           call calc_opacity(xyzh(1:3,point) + nextDist*ray, xyzh, kappa, listneigh, nneigh
      , nextdtaudr)
1082           dtaudr = (nextdtaudr+previousdtaudr)/2
1083           tau = tau + (nextDist-previousDist)*dtaudr
1084        enddo
1085        !fix units for tau (kappa is in cgs while rho & r are in code units)
1086        tau = tau*umass/(udist**2)
1087     end subroutine get_tau_inwards
1088
1089     !*****************************************************************!
1090     !*************************   COMMON   ****************************!
1091     !*****************************************************************!
1092
1093     !-----------------------------------------------------------------------
1094     !+
1095     !  Find the next point on a ray
1096     !+
1097     !  IN: inpoint:          The coordinate of the initial point projected on the
1098     !                        ray for which the next point will be calculated
1099     !  IN: ray:              The unit vector of the direction in which the next
1100     !                        point will be calculated
1101     !  IN: xyzh:             The array containing the particles position+smoothing length
1102     !  IN: neighbors:        A list containing the indices of the neighbors of
1103     !                        the initial point
1104     !  IN: inext:            The index of the initial point
1105     !                        (this point will not be considered as possible next point)
1106     !+
1107     !  OPT: nneighin:        The amount of neighbors
1108     !+
1109     !  OUT: inext:           The index of the next point on the ray
1110     !+
1111     !-----------------------------------------------------------------------
1112     subroutine find_next(inpoint, ray, dist, xyzh, neighbors, inext, nneighin)
1113        integer, intent(in)    :: neighbors(:)
1114        real, intent(in)       :: xyzh(:,:), inpoint(:), ray(:)
1115        integer, intent(inout) :: inext
1116        real, intent(inout)    :: dist
1117        integer, optional      :: nneighin
1118
1119        real      :: trace_point(3), dmin, vec(3), tempdist, raydist
1120        real      :: nextdist
1121        integer   :: i, nneigh, prev
1122
1123        dmin = huge(0.)
1124        if (present(nneighin)) then
1125           nneigh = nneighin
1126        else
1127           nneigh = size(neighbors)
1128        endif
1129
1130        prev=inext
1131        inext=0
1132        nextDist=dist
1133        trace_point = inpoint + dist*ray
1134
1135        i = 1
1136        do while (i <= nneigh .and. neighbors(i) /= 0)
1137           if (neighbors(i) .ne. prev) then
1138              vec=xyzh(1:3,neighbors(i)) - trace_point
1139              tempdist = dot_product(vec,ray)
```

```
1140             if (tempdist>0.) then
1141                raydist = dot_product(vec,vec) - tempdist**2
1142                if (raydist < dmin) then
1143                   dmin = raydist
1144                   inext = neighbors(i)
1145                   nextdist = dist+tempdist
1146                end if
1147             end if
1148          endif
1149          i = i+1
1150       enddo
1151       dist=nextdist
1152    end subroutine find_next
1153
1154    !-----------------------------------------------------------------------
1155    !+
1156    !  Calculate the opacity in a given location
1157    !+
1158    !  IN: r0:              The location where the opacity will be calculated
1159    !  IN: xyzh:            The xyzh of all the particles
1160    !  IN: opacities:       The list of the opacities of the particles
1161    !  IN: neighbors:       A list containing the indices of the neighbors of
1162    !                       the initial point
1163    !  IN: nneigh:          The amount of neighbors
1164    !+
1165    !  OUT: dtaudr:         The local optical depth derivative at the given location
1166    !                       (inpoint)
1167    !+
1168    !-----------------------------------------------------------------------
1169    subroutine calc_opacity(r0, xyzh, opacities, neighbors, nneigh, dtaudr)
1170       use kernel,    only:cnormk,wkern
1171       use part,      only:hfact,rhoh,massoftype,igas
1172       real, intent(in)    :: r0(:), xyzh(:,:), opacities(:)
1173       integer, intent(in) :: neighbors(:), nneigh
1174       real, intent(out)   :: dtaudr
1175
1176       integer :: i
1177       real    :: q
1178
1179       dtaudr=0
1180       do i=1,nneigh
1181          q = norm2(r0 - xyzh(1:3,neighbors(i)))/xyzh(4,neighbors(i))
1182          dtaudr=dtaudr+wkern(q*q,q)*opacities(neighbors(i))*rhoh(xyzh(4,neighbors(i)),
     massoftype(igas))
1183       enddo
1184       dtaudr = dtaudr*cnormk/hfact**3
1185    end subroutine calc_opacity
1186 end module raytracer_all
```

Department of Physics and Astronomy
Celestijnenlaan 200d - bus 2412
33001 Leuven, Belgium
tel. + 32 16 32 71 24
www.kuleuven.be