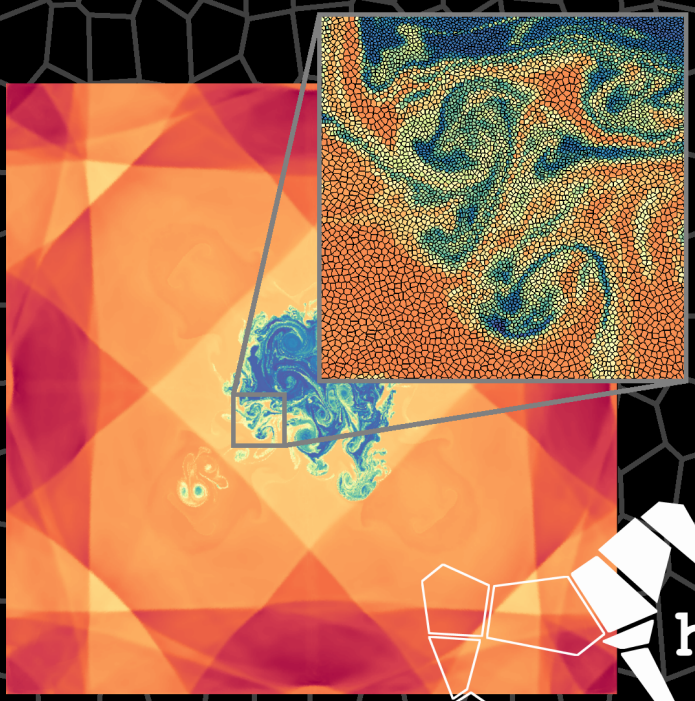
UNIVERSITEIT GENT

Universiteit Gent
Faculteit Wetenschappen
Vakgroep Fysica & Sterrenkunde

Advanced models for simulating dwarf galaxy formation and evolution

Geavanceerde modellen voor het simuleren van de vorming en evolutie van dwergsterrenstelsels



hadowfax

Bert Vandenbroucke

Proefschrift voorgedragen tot het behalen van de graad van
Doctor in de Wetenschappen: Sterrenkunde
Academiejaar 2015-2016

Promotor: Prof. Dr. Sven De Rijcke, Dr. Mina Koleva

Universiteit Gent
Faculteit Wetenschappen
Vakgroep Fysica & Sterrenkunde

Advanced models for simulating dwarf galaxy formation and evolution

Geavanceerde modellen voor het simuleren van de vorming en evolutie van dwergsterrenstelsels

Bert Vandenbroucke

Promotor: Prof. Dr. Sven De Rijcke, Dr. Mina Koleva

Dit werk kwam tot stand in het kader van een mandaat van het Bijzonder Onderzoeksfonds van de Universiteit Gent.

Figuur voorpagina: Dichtheidskleurenplot voor een 2D implosietest, met een zoom die het meebewegende discretizatierooster toont. De simulatie werd uitgevoerd met Shadowfax *(Chapter 4), en gebruikt technieken beschreven in Hoofdstuk 2 en Hoofdstuk 3.*

Proefschrift tot het behalen van de graad van
Doctor in de Wetenschappen: Sterrenkunde
Academiejaar 2015-2016

Een doctoraat is als een marathon. Een normaal examen of een normaal project is als een sprint, of hoogstens als een wedstrijd van 10km. Je kan de finish misschien net niet zien op het moment dat je vertrekt, maar je weet dat ze er is. Dus kan je van bij de start alles geven en dat kom je er wel. Je moet hoogstens een klein uurtje doorbijten. Bij een doctoraat is dat niet zo. Als je een nieuw onderzoeksproject aanvat, heb je meestal geen flauw idee waar het naartoe leidt. Je hebt geen flauw idee waar de finish is, hoe ver het nog is, hoe lang het nog zal duren... Soms weet je zelfs niet eens of er wel een finish is. Je wil van bij de start wel alles geven, maar je hebt geen flauw idee hoe je de komende beproeving moet indelen en of je er wel toe in staat zal zijn om ze tot een goed einde te brengen.

Gelukkig zijn er bij een doctoraat, net als bij een marathon, mensen die je begeleiden. En aan het begin van dit boekje, dat doet vermoeden dat ik nu toch eindelijk een finish in zicht heb, lijkt het mij dan ook een goed moment om die mensen even te bedanken. In de eerste plaats natuurlijk mijn promotor, Sven, mijn wetenschappelijke trainer en coach. Zonder hem was ik nooit aan dit doctoraat begonnen en zonder hem had ik ook geen inhoud gehad om dit boekje mee te vullen. En al had ik bij momenten liever gehad dat hij mij veel meer bij het handje had vastgehouden en mij stap voor stap gezegd had wat ik moest doen, ben ik toch heel blij dat hij me zo veel vrijheid heeft gegeven om te doen wat ik graag wilde doen en om mijn eigen ideeën uit te werken. Uiteindelijk is het niet de bedoeling dat een coach zelf mee loopt, maar dat hij zijn protegés leert lopen. Andere mensen hebben wel mee deze marathon beleefd, zij het in diverse fasen, en ook zij verdienen een bedanking, omdat dit de mensen zijn die ik doorheen die vier jaar in S9 mijn collega's heb mogen noemen. Eerst en vooral de vele collega's die voor mij gefinisht zijn: Joeri, Joachim, Gert, Annelies, Waad en Steven. Maar natuurlijk ook zij die later gestart zijn: Robbert, Sam, Christian en Marjorie. En zeker en vast zij die samen met mij gestart zijn en nu ook aan of vlakbij de finish zijn: Sébastien en Peter en iets verder in de toekomst Flor en Pieter. En laat ik ook vooral de collega post-docs niet vergeten: Mina, Tom, Jacopo, Gianfranco, Ilse, Aleksandr en Dukhang. En dan heb ik Maarten nog niet bedankt.

Tijdens een marathon staan er ook allerlei mensen aan de kant van de weg. Sommige om praktische redenen, zoals Inge, die het secretariaat van ons deel van de vakgroep op haar eentje draaiende houdt en Gerbrand, die ik herhaaldelijk heb moeten lastig vallen de laatste jaren als er weeral eens een harde schijf gecrasht was. Ook hen ben ik veel dank verschuldigd. Andere mensen zijn louter supporter, maar hun taak is niet minder belangrijk. Want als je de kaap van de 30km gerond hebt en alles pijn doet, dan is 42 echt nog ver en zijn hun voortdurende aanmoedigingen het enige dat je gaande houdt. Daarom ben ik mijn ouders heel dankbaar, omdat ik ondanks de verhuis naar Gent nog steeds bij hen kan thuisko-

men. En mijn zussen Sofie en Joke en broer Lowie, omdat ik bij hen altijd wel ontspanning vind als ik die nodig heb. Voorts zijn er nog de WiNA-vrienden uit Gent, die de herinnering aan mijn studententijd nog altijd levende weten te houden.

En dan zijn er nog een aantal speciale mensen die ik nog niet vermeld heb. Hen ben ik nog het meeste dankbaar, omdat zij weten dat metaforen nooit volledig kloppen en dat een doctoraat helemaal geen marathon is. Een marathon lopen vergt jaren voorbereiding en intensieve training, maar als je het juist aanpakt, valt de marathon zelf lopen heel goed mee. Een doctoraat is helemaal anders. Aan een doctoraat begin je vanuit het waanidee dat een doctoraat gewoon een langere versie van een thesis is. Met de stellige overtuiging dat die vijf jaar universitaire studies genoeg voorbereiding waren. Met veel motivatie en zonder enige tijd om na het behalen van een masterdiploma even op adem te komen. Mocht je op die manier een marathon aanvatten, dan zou je nooit de finish halen. En zo voelt een doctoraat ook, zo ongeveer halverwege. En dan helpen zelfs de meest enthousiaste supporters je geen meter meer vooruit.

Dat is het moment dat je echte vrienden nodig hebt. Kristof, die een nog veel zotter doctoraat dan ik heeft aangevat en toch exact hetzelfde leek mee te maken, afgaande op de vele gezamenlijke zaagsessies in diverse drinkgelegenheden (de beste pub van Schotland inclusief), meestal bij een ruime hoeveelheid gerste- of ander nat. Dries, die aanvankelijk als enige werkmens moest instaan voor ons beider doctoraten en de derde musketier was tijdens onze zotste reizen: IJsland, Oostenrijk en de legendarische uitstap naar Bouillon. En natuurlijk ook Kenny en Jiska en Jonathan en Heleen, zonder wie Schotland en Tsjechië nooit hetzelfde geweest zouden zijn. En mochten die laatste twee niet naar het westen verkast zijn, dan zouden ze ongetwijfeld ook vaste gast geweest zijn in Bahnhove.

En dan wil ik ook nog Iris bedanken. Omdat ze de grootste optimist is die ik ken en de beste muze die ik me maar kon wensen. Zien hoe zij iedere dag weer moet vechten tegen haar eigen lichaam om gewoon normaal te functioneren en er toch telkens weer in slaagt om de grootste glimlach op haar gezicht te toveren, duwt je met beide voeten op de grond. En doet je beseffen dat een doctoraat en alles dat er bij komt kijken helemaal niet zo belangrijk en helemaal niet zo zwaar is.

Wie ooit een marathon heeft gelopen, weet dat de laatste kilometer fantastisch is. Als je eindelijk de finish in zicht hebt, dan vlieg je en voel je je onoverwinnelijk. Zo is het ook met dit doctoraat. Nu het er eindelijk is, geniet ik van elke stap en kan ik maar geen genoeg krijgen van dit onderzoek. Ik hoop dan ook dat er nog ergens geld te vinden is om de komende jaren verder te doen. Ik heb ondertussen al twee keer een marathon gelopen. Maar ik ga het toch mooi bij dit ene doctoraat houden.

Mei 2016

Contents

0

Summary

Dwarf galaxies are the faintest and least massive inhabitants of the large scale Universe. They are hard to simulate in a cosmological context, since resolving them requires very high resolutions. For this reason, dwarf galaxies currently hold the key to falsifying the ΛCDM model of cosmology on small scales; a model that has proven to be quite successful on large scales.

There is currently a lot of tension between models and observations about what happens when a dwarf galaxy enters the sphere of influence of more massive galaxies, either as a satellite, or as a member of a large galaxy cluster. There is agreement on the fact that this influence changes the dwarf galaxy morphologically and dynamically, but it is not clear to what extent. Simulations of this interaction should help resolve these issues, but these simulations require high resolution models and state of the art simulation methods, that still need to be developed.

In this work, we focus on improving the current simulation model in two ways. First of all, we have developed new hydrodynamical integration methods that are better than the Smoothed Particle Hydrodynamics (SPH) scheme on which many current simulation models are based. To this end, we have studied some basic discretization methods, and efficient algorithms to work with them. We then used these discretization methods to construct hydrodynamical integration schemes: a moving mesh scheme that is based on an unstructured Voronoi mesh, and a mesh-free scheme that is based on a smoothed volume weighing of particle quantities. We show why these methods perform better than SPH, and illustrate the differences between these methods and methods that use a (refined) fixed mesh.

The second improvement of our model is an extension of the current sub-grid physics models used in galaxy simulations. We have adapted the equation of state of the gas in our model to take into account the multi phase character of the interstellar medium (ISM). We also implemented a cosmological ionising UV background (UVB) in our model, which affects the ionisation equilibrium in the ISM and hence has a significant impact on the gas cooling and the equation of

state of the fluid. The UVB furthermore acts as a heating term, which prevents diffuse gas from cooling and refuelling star formation. As a result, simulations that include a UVB behave very differently than simulations that do not include it. Star formation histories are limited to a single star formation peak, and the halo mass range that leads to dwarf galaxies with realistic stellar masses shifts to significantly higher masses.

To address these issues, we ran a large parameter study, aimed at adapting the sub-grid physics parameters to the presence of the UVB. We tried to adapt the strength and timing of the UVB itself, and the strength of the stellar feedback, but this did not improve the results. Only by including the metallicity dependent feedback of very low metallicity primordial stars (Pop III stars), were we able to suppress the initial star formation peak that would otherwise consume and expel the neutral gas in the simulated dwarf galaxies. Together with a more realistic treatment of gas accretion and halo growth through merger tree simulations, this enables us to simulate gas-rich dwarf galaxies with properties similar to those of observed dwarf galaxies.

The model improvements in this work clear the path for high resolution simulations of the interaction between dwarf galaxies and more massive galaxies.

0

Samenvatting – Summary in dutch

Dwergsterrenstelsels zijn de minst lumineuze en minst massieve bewoners van het grootschalige Universum. Ze zijn niet zo eenvoudig te simuleren in hun kosmologische context, aangezien dergelijke simulaties een heel hoge resolutie vereisen. Bijgevolg zijn dwergsterrenstelsels momenteel de belangrijkste objecten om het gangbare ΛCDM-kosmologiemodel, dat goed lijkt te werken op grote schaal, te testen op kleine schaal.

Modellen en observaties zijn het momenteel niet eens over wat er precies gebeurt wanneer een dwergsterrenstelsel in de invloedssfeer van een zwaarder sterrenstelsel terechtkomt. Dit gebeurt bijvoorbeeld met dwergsterrenstelsels die ingevangen worden als satelliet van een zwaarder sterrenstelsel, of met dwergsterrenstelsels die deel uitmaken van een grotere cluster van sterrenstelsels. Modellen en observaties zijn het erover eens dat deze invloed een effect heeft op de morfologische en dynamische evolutie van het dwergsterrenstelsel, maar het is helemaal niet duidelijk hoe sterk dit effect precies is. Simulaties van deze interactie kunnen ons helpen om hier meer vat op te krijgen, maar voor deze simulaties is een model nodig dat geschikt is voor de hoge resoluties die vereist zijn, en dat gebruikmaakt van een state of the art simulatiemethode. Beide moeten nog ontwikkeld worden.

In dit werk zullen we het huidige simulatiemodel op twee verschillende vlakken verbeteren. Eerst en vooral hebben we nieuwe hydrodynamische integratiemethoden ontwikkeld, die beter zijn dan de Smoothed Particle Hydrodynamics (SPH-)methode waarop veel van de huidige simulatiemodellen gebaseerd zijn. We hebben hiervoor een aantal eenvoudige discretizatiemethodes en de algoritmes om ermee te werken bestudeerd. Deze discretizatiemethodes hebben we dan gebruikt als basis voor twee nieuwe hydrodynamische integratieschema's: een meebewegend (moving mesh) schema dat gebaseerd is op een ongestructureerd, meebewegend Voronoi rooster, en een roostervrij (mesh-free) schema dat gebaseerd is op deeltjes waarvan de eigenschappen op een volumegewogen manier worden uitgemiddeld in de ruimte. We zullen aantonen waarom deze methodes

beter werken dan SPH en zullen de verschillen tussen deze methodes en methodes die gebruik maken van een (al dan niet verfijnd) vast rooster illustreren.

De tweede verbetering van ons model bestaat uit een uitbreiding van het model voor ongeresolveerde (sub-grid) fysica dat gebruikt wordt in simulaties van sterrenstelsels. We hebben de toestandsvergelijking van ons gas aangepast, zodat ze beter rekening houdt met de verschillende ionizatiefases in het interstellaire medium (ISM). Bovendien hebben we een kosmologische ionizerende UV-achtergrond (UVB) geïmplementeerd, die het ionizatie-evenwicht in het ISM beïnvloedt en op die manier een belangrijke invloed heeft op de koeling en de toestandsvergelijking van het gas. De UVB zorgt ook voor een verhitting van het gas, waardoor diffuus gas niet kan afkoelen om als brandstof voor late stervorming te dienen. Bijgevolg is er een groot verschil tussen simulaties met en zonder UVB. De stervormingsgeschiedenis met UVB wordt herleid tot een enkele stervormingspiek, en de minimale massa die een donkere-materiehalo moet hebben om een sterrenstelsel met een realistische stellaire massa te herbergen stijgt significant.

Om deze problemen op te lossen, hebben we een grote parameterstudie uitgevoerd, met als doel de parameters van het model voor ongeresolveerde fysica aanpassen aan de aanwezigheid van de UVB. We hebben geëxperimenteerd met het aanpassen van de sterkte en timing van de UVB zelf, en met het aanpassen van de sterkte van de stellaire feedback, maar dit leidde niet tot betere resultaten. We konden de initiële stervormingspiek die het neutrale gas in de gesimuleerde dwergsterrenstelsels zou opgebruiken en wegblazen enkel onderdrukken door het in rekening brengen van metalliciteitsafhankelijke feedback van primordiale sterren met een extreem lage metalliciteit (Pop III sterren). Als we dit model combineren met een realistischer model voor kosmologische gasaccretie en donkere-materiehalogroei via versmeltingsbomen, kunnen we gasrijke dwergsterrenstelsels simuleren die gelijkaardige eigenschappen hebben als geobserveerde dwergsterrenstelsels.

De verbeterde modellen uit dit werk maken het mogelijk om de interactie tussen dwergsterrenstelsels en zwaardere sterrenstelsels te simuleren op hoge resolutie.

1

Introduction

According to the ΛCDM model of cosmology, the Universe was formed during the Big Bang, 13.8 Gyr ago (Spergel *et al.*, 2007). The conditions in the very early Universe were very different from what they are today, and were governed by physical laws that nowadays are only accessible in huge particle accelerators. Nonetheless, observations of the Cosmic Microwave Background (CMB) provide very tight constraints on the matter contents of the Universe, and on the parameters that describe its expansion (Planck Collaboration, 2014).

The CMB itself consists of the UV radiation that was allowed to move freely when the atomic nuclei that formed during the Big Bang nucleosynthesis first combined with the free electrons inhabiting the Universe to form atoms, in a process that is misleadingly denoted as recombination, and which happened $\approx 378,000$ yr after the Big Bang, or at a redshift of $z \approx 1100$ (Zeldovich *et al.*, 1968). Due to cosmological redshifting, this radiation has since then cooled down to the microwave range of the radiation spectrum.

Before recombination, the Universe was a tightly coupled plasma. After the radiation decoupled from the baryonic matter, the Universe became dark, and all matter was predominantly affected by the force of gravity, until the formation of the first stars and galaxies. Hot UV radiation from these first structures then started ionising the neutral gas in the Universe again, in a process that is called reionization (Becker *et al.*, 2001). This ionising radiation constitutes a background radiation field that affects the formation of later structures.

The ΛCDM paradigm states that the Universe was initially filled with a mixture of matter, both baryonic matter and cold dark matter (CDM), and a mysterious dark energy, represented by a cosmological constant Λ. Structures in the Universe formed hierarchically, by the growth of initial overdense regions under the force of gravity. In an expanding Universe, the gravitational force and the expansion compete with each other, so that small overdensities at first grow slowly (Bond & Efstathiou, 1984). The CMB holds valuable information about the power spectrum of overdensities in the very early Universe, at a redshift of 1100, so that we know what the initial conditions for this hierarchical structure

formation were. This initial power spectrum can be safely analytically evolved by linear perturbation theory until a redshift of ≈ 100.

To further evolve the overdensities, large numerical simulations are needed (Davis *et al.*, 1985). The first large simulation of cosmic structure formation with enough resolution to compare with the observed Universe was carried out by Springel *et al.* (2005). They evolved a periodic box with a length of 500 Mpc containing 10,077,696,000 tracer particles under the force of gravity from redshift 127 to redshift 0. They showed that the initial density perturbations grow into massive halos in an overall web-like structure, called the *cosmic web*, in which massive clusters of halos are joined by filamentary structures, with large *voids* in between. The resulting abundances of massive halos are in excellent agreement with the observed abundances of luminous galaxies. This provides strong evidence for the ΛCDM model of cosmology.

Simulations since then have focused on refining this model, by using a higher resolution (Boylan-Kolchin *et al.*, 2009), and by including baryonic physics: gas cooling and heating, star formation, stellar and Active Galactic Nuclei (AGN) feedback,... (Vogelsberger *et al.*, 2014b; Schaye *et al.*, 2015). The aim of these simulations is to falsify ΛCDM on large scales and to turn it into a predictive theory that can make predictions for future observational campaigns, like ESA's upcoming Euclid mission (Scaramella *et al.*, 2014).

These large cosmological simulations lack the resolution to falsify ΛCDM on smaller scales. To this end, so called zoom simulations are used, which refine the resolution in a small region of the box, while still taking the relevant cosmological context into account (Governato *et al.*, 2010; Oñorbe *et al.*, 2015). These simulations require detailed (mostly *sub-grid*) models to accurately capture all relevant physical processes, and advanced numerical methods that are both accurate and computationally efficient.

There are currently a number of outstanding questions concerning small -scale predictions of ΛCDM that are claimed to contradict observations, like the low number of observed low-mass satellite galaxies of the Milky Way and Andromeda (the so called '*missing satellite* problem) (Wang *et al.*, 2012), the large number of observed massive satellite galaxies in the Local Group (the *too big to fail* problem) (Garrison-Kimmel *et al.*, 2014), and the form of the central density profile of galactic halos (the *cusp to core* problem) (de Blok, 2010).

Some authors use these claimed discrepancies to support alternative theories for dark matter (Vogelsberger *et al.*, 2014a) or even gravity (Rodrigues *et al.*, 2014). Solving these problems however requires the comparison of state of the art simulations and observations, which are close to the limit of what is currently feasible. Recent simulations suggest that these discrepancies can be overcome by including more baryonic physics and improved stellar feedback recipes in the models (Chan *et al.*, 2015; Dutton *et al.*, 2015).

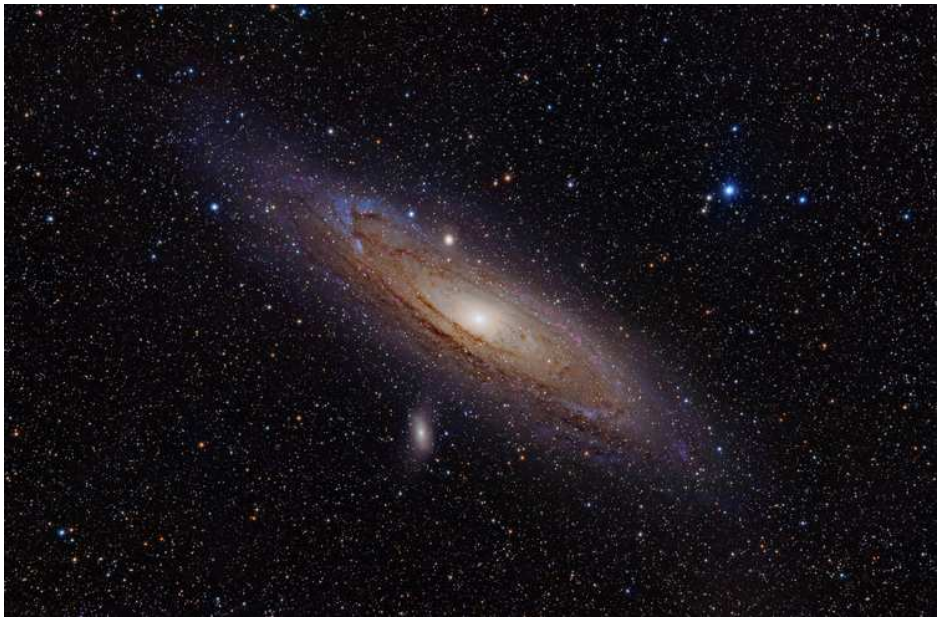


Figure 1.1: *"Andromeda Galaxy (with Hα)" by Adam Evans. Two satellite dwarf galaxies are also visible: M110 is the bright elliptical blob below Andromeda, M32 is the smaller round blob above and a bit to the left of the center of the galaxy.*

1.1 Dwarf galaxies

Dwarf galaxies are the faintest and smallest galaxies in the Universe, with typical magnitudes fainter than -20, and masses below $10^{10}\,M_{\odot}$. Unlike more massive galaxies like our own Milky Way, they host a relatively small number of stars ($\sim 10^9$) (Fig. 1.1). Furthermore, they have much smaller dust contents than larger galaxies (Walter *et al.*, 2007). Fig. 1.2 shows some properties of observed dwarf galaxies: stellar and neutral gas masses, half-light radii (the radius at which the integrated luminosity drops to half of its total value), and metallicities. Dwarf galaxies are generally metal-poor. The faintest dwarfs do not host observable neutral gas.

Like more massive galaxies, dwarf galaxies are generally subdivided in two types (Tolstoy *et al.*, 2009): early type dwarf elliptical and dwarf spheroidal galaxies that are gas-poor and have predominantly old stellar populations, and late type dwarf irregulars, which have ongoing star formation from a rich gas reservoir. Apart from these, there are also transition type dwarf galaxies that

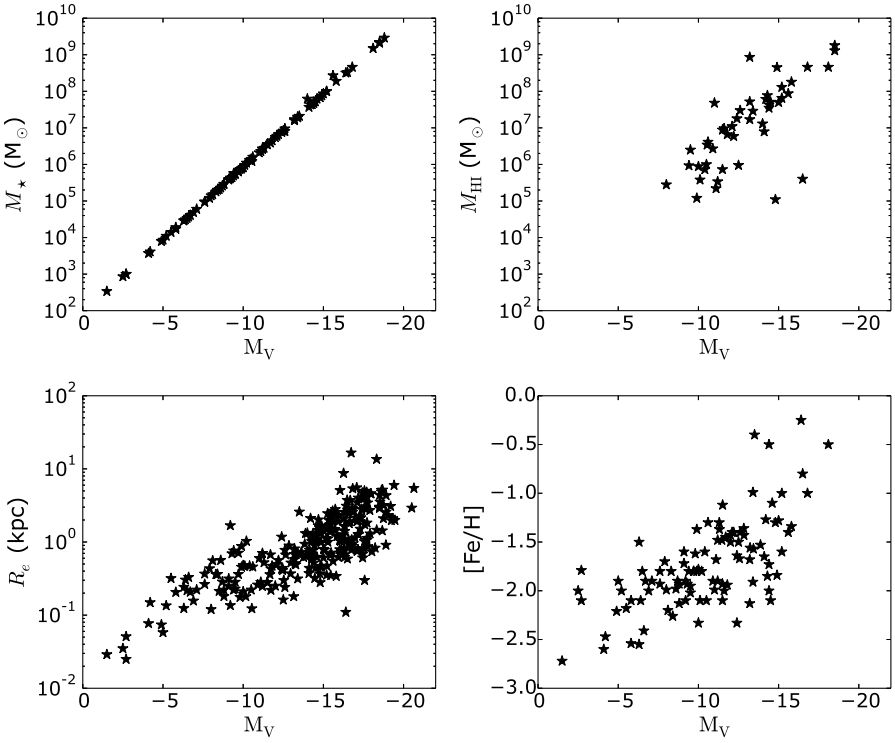


Figure 1.2: *Some properties of observed dwarf galaxies as a function of their V-band magnitude.* Top left: *stellar mass,* top right: *neutral gas mass,* bottom left: *half-light radius,* bottom right: *metallicity. These data were compiled from van Zee (2000); Geha et al. (2003); Grebel et al. (2003); Hunter & Elmegreen (2006); Dunn (2010); McConnachie (2012); McQuinn et al. (2013); Rhode et al. (2013); Tollerud et al. (2015).*

share properties of both early and late type galaxies, and more exotic types like ultra faint and ultra compact dwarf galaxies. Recently, tidal dwarf galaxies have gained a lot of interest (Sweet *et al.*, 2016). These galaxies are formed as a result of the tidal interaction between two massive galaxies, and should contain very little dark matter. This should have a detectable imprint on their rotation curves and hence provide a good test for dark matter models.

Late type dwarf galaxies are usually observed in isolation, i.e. far away from the influence of large galaxies or galaxy clusters, while early types are more common near the center of massive clusters and as satellites of more massive galaxies (Dressler, 1980; Recchi, 2014). This seems to suggest that, unlike what the naming suggests, early type dwarf galaxies were originally late type dwarf galaxies that somehow lost their gas, presumably due to external influences (Geha *et al.*, 2003; van Zee *et al.*, 2004; De Rijcke *et al.*, 2005, 2010; Lisker *et al.*, 2013).

To test this hypothesis, two approaches have been adopted. Large observational campaigns aim at gathering statistical data about dwarf galaxy types and their locations, trying to find a correlation between type and environment (Ryś *et al.*, 2014). In parallel, theoretical studies have tried to assess whether a late type dwarf galaxy can lose its gas due to external influences (Mayer, 2010). Different environmental effects have been implemented in numerical simulations, carrying ominous names like "galaxy harassment" (Smith *et al.*, 2015) and "tidal stripping" (Sales *et al.*, 2010). However, different simulations seem to lead to different results, indicating that the models need to be improved.

A good model of the interaction between a late type dwarf galaxy and a massive galaxy or cluster first of all requires the self-consistent modelling of the dwarf galaxy itself. If the modelled galaxy is to represent a late type dwarf in isolation, then we should first of all make sure the model produces such a galaxy in isolation, starting from the basic ingredients of a galaxy, i.e. cold dark matter and neutral gas. These models should include all relevant physical processes that could affect the evolution of the isolated galaxy, both internal (physics of the interstellar medium, star formation and stellar feedback), as external (mergers, a UV background). Only if such a model produces simulated dwarf galaxies that have the same properties as observed late type dwarf galaxies can we subject these models to the environmental effects of a larger galaxy or cluster and test the proposed evolutionary track from late to early type.

Secondly, a good interaction model needs to resolve the effects that could influence the late type dwarf galaxy once it enters the environment of interest. Some of these effects, like gravitational tidal forces, could affect the shape of the halo potential and require the modelling of a *live* potential with sufficient resolution by means of an N-body technique. Other effects, like ram-pressure stripping (see below), only affect the gas in the galaxy, but could potentially strip this gas completely from the halo, leaving the galaxy "red and dead". Resolving

these effects requires a hydrodynamical integration scheme that can handle the complex interface between the cold, dense gas in the dwarf galaxy, and the hot, diffuse environmental gas.

Changes in the halo potential will likely also affect the dynamics of the gas in the dwarf galaxy, leading to changes in star formation that could also affect the galaxy type. Modelling tidal forces without a self-consistent treatment of star formation could hence lead to wrong conclusions.

1.2 Dwarf galaxy formation

Like all structures in the Universe, dwarf galaxies formed hierarchically, i.e. by the merging of several smaller structures that all ultimately originated from the gravitational collapse of small density perturbations in the very early Universe (Springel *et al.*, 2005). Before recombination, only the cold dark matter cores of these perturbations were able to accrete more mass, as the baryonic plasma was kept homogeneous by radiation pressure (Silk, 1968). After recombination, the then neutral baryons started to slowly accrete onto these dark matter seed halos. As the central density of the baryons increased, pressure built up and impeded further accretion. However, as the baryonic gas was also able to cool as a result of internal radiative processes, very dense, cold gas clumps were able to form near the centers of more massive halos. As soon as these clumps become cold and dense enough, nothing can prevent them from further collapse, so that they start fragmenting into even denser clumps, at the center of which stars are formed (Krumholz, 2014).

Galaxy formation is hence mainly driven by the conversion of cold baryonic gas into stars at the centers of halos, on time scales that are set by radiative cooling processes. However, once stars have formed, they also affect the further evolution of the galaxy: young stars emit a lot of UV radiation that ionizes the surrounding interstellar medium (ISM), while massive stars are usually short-lived and explode as violent supernova explosions that also emit large amounts of energy into the ISM and enrich it with stellar material, which contains metals (elements heavier than ^{1}H and ^{4}He) (Marcolini *et al.*, 2006). This stellar feedback disperses the cold, dense gas in star forming regions and impedes further star formation. As stellar populations age, stellar feedback decreases again and subsequent star formation is enabled.

The conversion of gas into stars and the dispersion of this gas by stellar feedback will also be affected by the growth of the (dwarf) galaxy halo through mergers, as tidal interactions between merging halos can drive large outflows or fuel star formation bursts (Deason *et al.*, 2014; Verbeke *et al.*, 2014; Leaman *et al.*, 2015; Starkenburg *et al.*, 2016).

1.2.1 UV background

The UV radiation from young stars initially only ionizes small bubbles of ISM surrounding these stars, called Strömgren spheres (Strömgren, 1939). However, as the number of stars increases, more and more Strömgren spheres are formed, and different spheres stars to overlap and merge into larger spheres. As star formation is more violent inside more massive halos, these halos are quickly entirely surrounded by large Strömgren spheres that extend into the intergalactic medium (IGM). This process is further enhanced by AGN feedback in the centers of massive halos.

Galaxies are not homogeneously spread out throughout the Universe, but are also grouped into hierarchical clusters. Near the centers of these clusters, Strömgren spheres from different galaxies also start to overlap. The continuous star formation inside these spheres feeds their growth, so that in the end, the entire Universe is filled with merging Strömgren spheres (Alvarez *et al.*, 2009). By a redshift of 6 ($\sim 1\,\mathrm{Gyr}$ after the Big Bang), the entire Universe is reionized (Becker *et al.*, 2001), and neutral gas is only found in the disks of halos that are more massive than $\sim 10^8\,\mathrm{M}_\odot$ (Benítez-Llambay *et al.*, 2015).

As the entire Universe is reionized, there is no longer enough neutral gas to absorb the UV radiation that escapes from star forming galaxies. This radiation will hence roam the Universe, and constitutes an intense background radiation field, the UV background (UVB). The UVB reaches its peak strength at a redshift of ~ 2 ($\sim 3\,\mathrm{Gyr}$ after the Big Bang) (Faucher-Giguère *et al.*, 2009), and will have a large effect on the evolution of alread formed galaxies.

1.2.2 Galaxy environments

Massive galaxies do not only contain cold, neutral gas near their centers, but are also surrounded by a hot halo of ionized gas, which extends to large radii around the galaxy core. Temperatures in this hot halo are of the order of 10^6 K, while the typical density of $\sim 10^{-3}\,\mathrm{amu\,cm^{-3}}$ (Williams *et al.*, 2007). Dwarf galaxies do not contain such hot halos, as their gravitational potentials are to weak to hold on to this hot, diffuse gas.

When a dwarf galaxy enters the sphere of influence of a more massive galaxy, the cold gas in the dwarf will interact with the hot IGM surrounding the massive galaxy, in a process called ram-pressure stripping. As the cold gas cloud plunges into the hot surrounding IGM, the structure of the cloud is disrupted, and the resulting fragments are heated by the surrounding hot gas. However, this process is counteracted by the potential of the dwarf galaxy halo, which tries to hold on to the cold gas. It is therefore currently unclear how strong the effect of ram-pressure stripping will be, and if it could potentially strip the dwarf of all its neutral gas (Mayer, 2010).

The effect of galaxy environment is particularly strong near the centers of large galaxy clusters, where the galaxy density can reach values of 10^3 galaxies Mpc^{-1} (Dressler, 1980). Apart from the effects mentioned above, the combined UV radiation from all these galaxies will also cause these regions to reionize faster than less dense regions, which will increase the threshold mass for small halos to hold on to their neutral gas (Alvarez *et al.*, 2009). We do hence expect a different formation history for dwarf galaxies that formed near the centers of large galaxy clusters compared with dwarf galaxies that formed in isolation.

1.3 Dwarf galaxy models

The isolated dwarf galaxy models we will use in this work are based on the basic models of Valcke *et al.* (2008). They include a sub-grid model for gas cooling, star formation and stellar feedback, and lead to simulated galaxies that adhere to a number of observational scaling relations, within observational scatter.

Schroyen *et al.* (2011) improved these models by adding an initial rotation to the gas in the simulations, leading to less centrally concentrated star formation and more realistic star formation histories. Cloet-Osselaer *et al.* (2012) and Schroyen *et al.* (2013) refined the model even further by addressing the parameters that control the star formation and stellar feedback strength. They also improved the radiative gas cooling by first extending it below $10^4\,\mathrm{K}$, and then replacing it by a self-consistent three parameter cooling model. With that, the model includes all relevant internal processes that shape the dwarf galaxy.

Cloet-Osselaer *et al.* (2014) addressed the effect of halo mergers on the formation of the dwarf galaxies by means of merger tree simulations, which offer a computationally cheap alternative for cosmological zoom simulations. They showed that these mergers have an effect on the star formation history of the galaxy, and play a role in the conversion of a cuspy density profile into a cored profile, offering a possible solution to the cusp to core problem introduced above.

In this work, we will address the only non-environmental external effect left to complete our model: the cosmological UVB. Including the UVB not only requires the addition of gas heating to the sub-grid physics model, but also requires us to address the changes in the gas cooling and even in the gas physics that are associated with the change in ionisation equilibrium caused by the UVB (Vandenbroucke *et al.*, 2013; De Rijcke *et al.*, 2013). As we will show in Chapter 6, the effect of the UVB on simulated dwarf galaxies is rather dramatic, and this chapter will be entirely devoted to solving the issues that arise. The main conclusion will be that we need to incorporate the effect of primordial population III stars and their feedback into the simulations, to suppress an initial peak in the star formation that would otherwise lead to an excess in the stellar feedback that drives all neutral gas out of the galaxy. The ultimate model of isolated late

type dwarf galaxies, that takes into account all internal and external processes, was published by Verbeke *et al.* (2015).

1.4 Hydrodynamical instabilities

As Agertz *et al.* (2007) and Valcke *et al.* (2010) showed, the Smoothed Particle Hydrodynamics (SPH) scheme that forms the basis of our current model is incapable of resolving a number of hydrodynamical instabilities, that are important for the modelling of the interface between a cold, dense gas and a hot, diffuse gas. These issues are best illustrated by means of the "blob test", which models the interaction of a dense, spherical gas blob with a diffuse, hot gas stream in a wind-tunnel like setup, using different hydrodynamical integration schemes (Agertz *et al.*, 2007), see Chapter 3 and Fig. 3.8. Although no formal convergence is reached between any two methods due to the sensitivity of this idealised setup to method dependent numerical noise, there is a clear difference between grid based methods and SPH. While the grid models predict that the blob is completely disrupted by the hot, diffuse stream, the SPH blob only deforms, and stays overall intact.

This behaviour persists when higher SPH resolutions are used, so that it is not only a result of a higher accuracy of grid methods, but is really caused by a fundamental incapability of SPH to capture certain instabilities. We will come back to this problem in Chapter 3.

These fundamental problems make it impossible to use standard SPH for simulations of ram-pressure stripping, as these simulations are very similar to the blob test. If instead of a blob of gas, we would place a dwarf galaxy in the wind tunnel, SPH would predict the gas in the galaxy to stay were it is, irrespective of whether this is the correct physical behaviour in this situation.

We could of course use a grid technique to perform the hydrodynamical integration, because these techniques have no problems resolving instabilities. However there are two issues here. First of all, a practical issue: our entire model is based on an SPH code, with many concepts being tightly coupled to the concept of a particle-based integration scheme. Changing from an SPH scheme to a grid scheme requires an adaptation and possible recalibration of our entire model, which is non-trivial.

Secondly, a grid based method is not necessarily better than SPH due to other issues. As we will show in Chapter 3, methods that use a fixed grid experience problems when the fluid is moving at high velocity with respect to the grid. In principle, most of these problems should be solved by using an appropriately small integration time step, but this then has a major impact on the runtime of the simulations. Using a method that is Lagrangian in nature, i.e. where the integration "grid" (that consists of particles in the case of SPH) moves along with

the flow, in general can use larger time steps and is more efficient. For typical hydrodynamical test problems, this is generally not an issue, but for a full-fledged dwarf galaxy simulation with a large number of resolution elements and a large range in densities and velocities, we would rather use a Lagrangian method.

Luckily, there has been some advance in the field of astrophysical hydrodynamics since the confronting paper of Agertz *et al.* (2007). A number of authors have proposed ways to save SPH, by adding correction terms to its equations that should make it sensitive to instabilities (Price, 2008; Valcke *et al.*, 2010; Cha *et al.*, 2010; Read & Hayfield, 2012). A problem with most of these extra terms is that they can only be applied when necessary, since they are not physical. In other words, if we apply these terms in the entire fluid, the integration would be wrong everywhere, except in the regions where the instabilities are located. Applying the terms ad-hoc, when necessary, requires a *switch* that decides when to activate them. Designing a good switch has proven to be difficult (Read & Hayfield, 2012).

Other authors have abandoned the use of SPH, and have developed new Lagrangian techniques, that combine the Lagrangian nature of SPH with the accuracy of grid methods. Springel (2010) uses a moving unstructured mesh to discretize the fluid, while Hopkins (2015) uses the SPH formalism to estimate volumes, rather than densities for the SPH particles, in a so called *meshless* method. Both then use a finite volume method for the integration, which does not experience the problems traditional SPH has.

1.5 This work

In this work, we will build the foundations for future ram-pressure stripping simulations of late type dwarf galaxies. To this end, we will add the last ingredient, the UVB, to our model, and tune the model so that it produces realistic late type dwarf galaxies. We will also devote a considerable part of this work to an in-depth analysis of different hydrodynamical integration methods, and describe the implementation of the public moving mesh code Shadowfax. This work has resulted in Vandenbroucke *et al.* (2013); Verbeke *et al.* (2015); Vandenbroucke *et al.* (2016) and Vandenbroucke & De Rijcke (2016), while Shadowfax was also used for part of the analysis presented in Cloet-Osselaer *et al.* (2014).

2

Particles & grids

BEFORE we can start to even think about modelling physical processes like gravity, hydrodynamics and star formation, we need to find out how to represent the physical systems we are interested in on a computer. Although computers are orders of magnitude more powerful at computing than humans, they are at the same time incredibly bad at working with abstract concepts like gas clouds or dark matter distributions. In order to be able to do anything with those concepts on a computer, we will have to convert them into the small building blocks a computer can work with: bits. In this chapter, we will explain how this so called *discretization* of these concepts can be achieved. We will introduce different ways of discretizing the interstellar and intergalactic medium found in the literature, as well as ways of representing the enigmatic dark matter component we need in the simulations. This will lead us to a discussion about sampling of distributions, Poisson noise and how to reduce it, and will at the same time allow us to present some discretization algorithms in more detail, with a focus on our own work on Voronoi grids and evolving Voronoi grids.

To introduce the necessary concepts, we will have to make use of *densities* as a measure of how much matter (ordinary baryonic or dark) is present in some small subvolume of space. The precise definition of the hydrodynamical density in case of the interstellar medium can be found in Chapter 3; for now it suffices to see the density $\rho(\vec{x})$ as some function of space that gives the average amount of matter present around some coordinate of space, $\vec{x}$. The higher the density, the more matter present in a neighbourhood of that region. As an illustration, we will use the basic 2D and 3D density distribution shown in Fig. 2.1 throughout this and the next chapter. It is given by the simple expression

$$\rho(\vec{x}) = \begin{cases} 1 & |\vec{x} - \vec{o}| < 0.25, \\ 0.1 & 0.25 \leq |\vec{x} - \vec{o}|, \end{cases}$$

with $\vec{o} = (0.5, 0.5)$ in 2D and $\vec{o} = (0.5, 0.5, 0.5)$ in 3D.

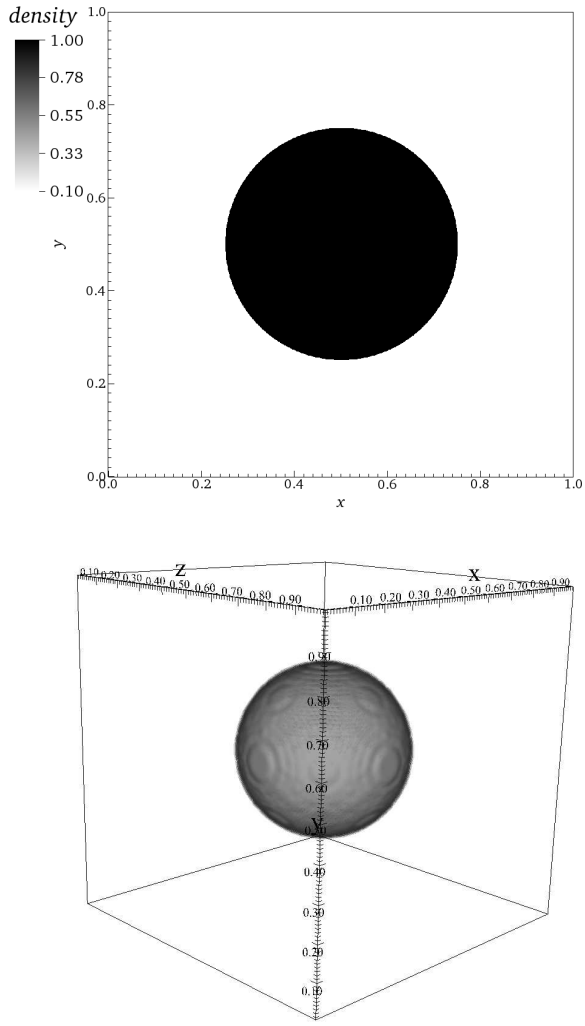


Figure 2.1: *The 2D and 3D density distribution we will be using throughout this chapter. The bottom plot shows the surface of equal density surrounding the high density sphere in the center.*

2.1 Grids

2.1.1 Cartesian grids

When showing the basic density distribution of Fig. 2.1, we silently already introduced the first and conceptually easiest way of discretizing a density distribution: by sampling it on a Cartesian grid. The picture consists of a 2D Cartesian grid of pixels and the colour of every pixel is set by the average value of the density in the space covered by that specific pixel.

This method can be very easily extended to 3D (although this is less suited for graphical representations). In practice, the discrete representation of the density distribution will then consist of at least two arrays (or some other array-like structure): (1) an array of coordinates in XD space (with $X = 1, 2, 3$ for most physical purposes), where each combination of X consecutive elements defines the position of the center or anchor of some XD cell in XD space, and (2) an array of average density values associated with the cells.

It is immediately clear that the representation will get better if the number of cells is increased, so that the number of cells is a good measure of the *resolution* of the representation. At the same time, it is also quite clear that it is much harder to increase the resolution in XD space if X is larger, since an increase of the total number of cells with a factor 2 will only increase the number of cells in one dimension with a factor $2^{\frac{1}{X}}$. If we want to have the same resolution increase in 3D as in 1D, we should multiply the number of cells in 3D by 8 for every doubling of the number of cells in 1D.

This last issue makes it very hard to use a simple Cartesian grid when discretizing systems with a high *dynamic range*, i.e. a system where some high density regions are embedded in a larger volume with a much lower density, e.g. the cosmic web. We clearly want to resolve the high density regions with enough resolution elements, since these are the places where the interesting processes like galaxy formation are taking place. However, if we would use the same high resolution to resolve the entire cosmic volume, we would end up with a huge number of cells, of which most will be almost empty and uninteresting. Suppose we ideally want every cell to have roughly the same average mass. Then cells in low density regions will be smaller than desired. To handle a *density contrast* (the ratio between high density and low density) of 10, we might easily be using 10 times too many cells in the low density region in 1D. In 3D, we will be using 1000 times too many cells for the low density region. In simulations of galaxy formation, the density contrast can easily be 10^7, which means we will be using a staggering factor of 10^{21} cells too many in the uninteresting regions. In practice, this makes it completely impossible to run this type of simulations with a fixed grid, since no computer can handle this number of cells.

One possibility to improve on this is to use *adaptive mesh refinement* (AMR),

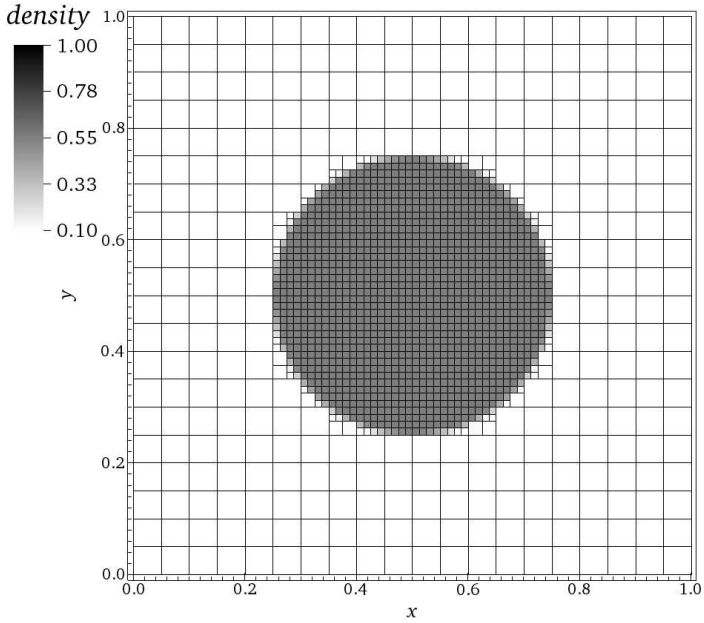


Figure 2.2: *An AMR representation of the 2D density distribution, refined to yield an almost constant mass in every cell. The low resolution region has* 20×20 *cells, in the high resolution region this is refined to* 80×80 *cells. Since the density is averaged over the cell, the distribution cannot be exactly represented around the edges of the disk. For clarity, the opacity of the colour plot is lowered to 50%.*

a technique whereby a cell can be split into smaller subcells if some cell splitting criterion is met (Berger & Colella, 1989). The easiest way to implement this would be to store an extra array, which for every cell gives the size of that cell, or tells us on which refinement level the cell is located. In practice, techniques that use an adaptive mesh (Teyssier, 2002; Keppens *et al.*, 2012) make use of more complex, but very efficient structures to manage the multi level grid. A very basic 2D AMR grid is shown in Fig. 2.2.

2.1.2 Static unstructured meshes

Another way to construct a grid that has smaller cells in regions of higher density, is by using *unstructured meshes*. An unstructured mesh is much harder to store

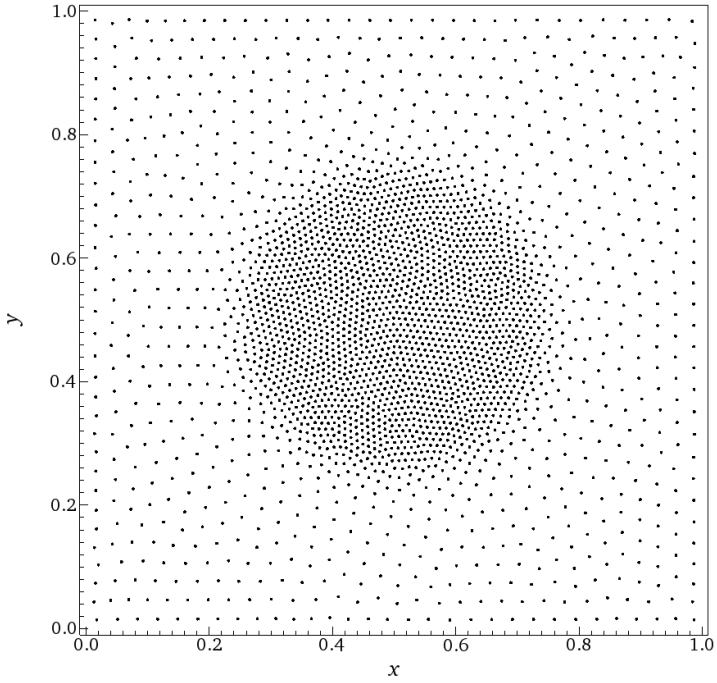


Figure 2.3: *A point representation of the 2D density distribution. The distribution was sampled using 2,780 points that were randomly sampled and then allowed to relax with techniques that will be discussed later. On average, there are 10 times more points in the high density region than in the low density region.*

and maintain, but it has the advantage that it is unrestricted: every cell can be split into two (or more) subcells irrespective of the dimension of the space. In a Cartesian grid in XD space, a cell can only be split in 2^X subcells. We hence have much more control on the number of cells and where to put them to get the best resolution when using an unstructured mesh.

There are many types of unstructured meshes that are used in various contexts (Mavriplis, 1997), but for our purposes there are two which will be important and will be introduced below: the Voronoi mesh and the Delaunay tessellation. They both start from a set of *mesh generators*: a set of points in XD space that are used to define the geometrical structure of the mesh. For a good mesh representation of the density distribution, we will require these generators to *sample* the density

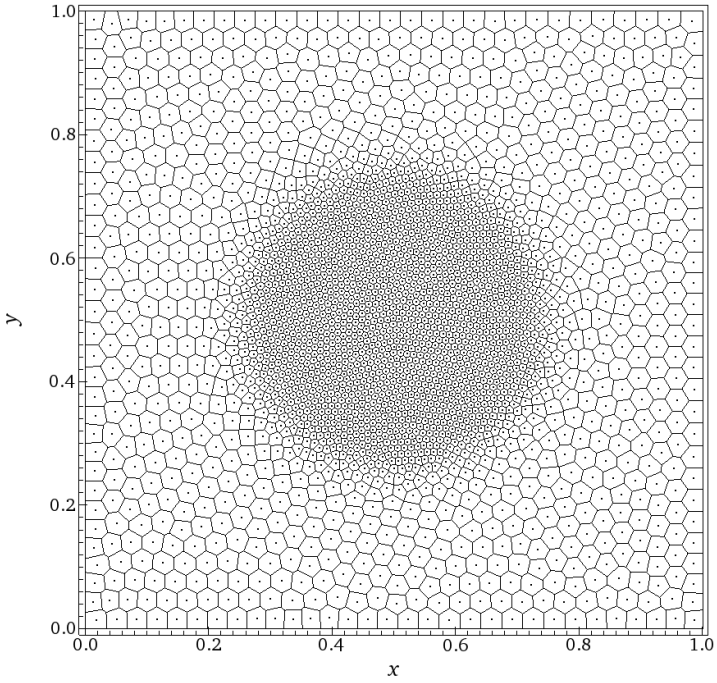


Figure 2.4: *Voronoi mesh for the set of generators based on the 2D density distribution. For every point in the set of generators, there is a corresponding Voronoi cell that contains the region of the box closest to that specific generator.*

profile. This means that we should have a higher probability of finding a mesh generator in a region with a higher density. We will come back to how to sample points in this way later; let us for now assume we have such a set of generators and hence have an array of XD coordinates corresponding to the coordinates of the generators, see Fig. 2.3.

Voronoi mesh

Given a large enclosing volume (typically a box, but other shapes are possible) and the set of mesh generators, the Voronoi mesh (or Voronoi diagram/grid/...) is defined as the collection of cells with the property that every cell contains the region of space that is closest to one of the generators (Dirichlet, 1850; Voronoi, 1908), which we will call the generator of that cell, see Fig. 2.4 and Fig. 2.5. The

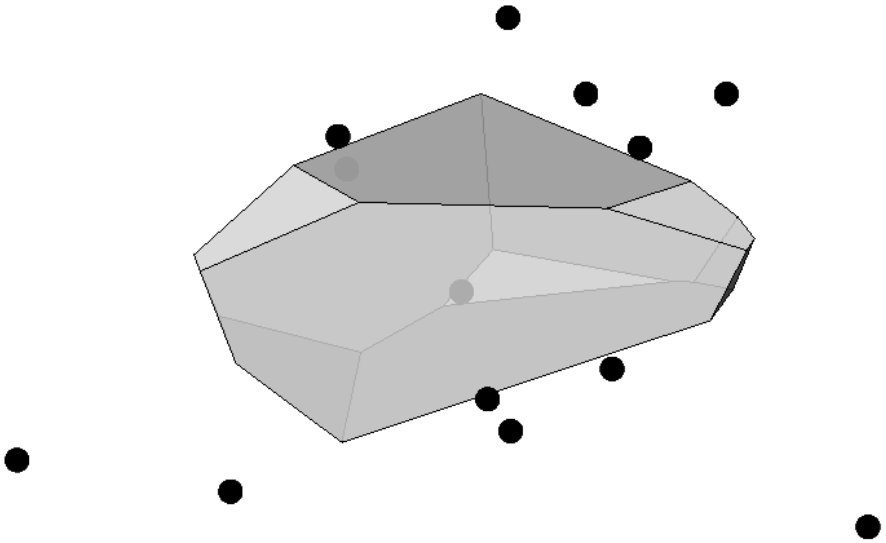


Figure 2.5: *A 3D Voronoi cell, together with its generators*

cell borders are hence formed by subspaces of points/lines/planes (depending on the dimension X of the space) that are equidistant to two of the generators.

The Voronoi mesh for a given box and a given set of mesh generators is unique. Furthermore, the Voronoi mesh has some very interesting properties. For our purposes, one of the most important properties is that a Voronoi cell will always be convex, i.e. it is always possible to connect any point inside the Voronoi cell with any other point inside the cell by using a line segment that is entirely contained inside the cell. A second important property is the way in which the Voronoi mesh changes when the generators are moved away from their original positions. If the movement of the generators is continuous, then the Voronoi grid will change in a continuous way: cells will grow or shrink, but will do so by a growing or shrinking of their boundaries in a continuous way (Reem, 2011). Generators that share a boundary between their Voronoi cells are *neighbours* and two cells can only become neighbours after a continuous movement of the mesh generators if a new boundary is created in between them with zero size that grows continuously. Two cells stop being neighbours when the common boundary between them continuously shrinks to zero size. This will turn out to be a very important property when we will define a hydrodynamical integration scheme based on a Voronoi mesh in Chapter 3.

Constructing and storing the Voronoi mesh in 1D is a trivial task, since in this case the boundary points will be the midpoints of the line segments between neighbouring generators. Generators are neighbours if there is no other generator in between them. For higher dimensions, things get more complicated. There exists an extensive body of literature on Voronoi meshes and how to construct and represent them in 2D and 3D (de Berg *et al.*, 2008). Mathematically, it can be shown that this is most efficiently done using Fortune's algorithm (Fortune, 1986). In 2D, this algorithm can be coded very elegantly in a so called *sweepline* algorithm with efficiency $O(N \log N)$. In 3D, algorithms get inevitably more complex and are less elegantly expressed. We will limit ourselves to one Voronoi construction algorithm in this work, which will be based on the dual Delaunay tessellation, and which is described in depth by Springel (2010). This algorithm has an efficiency of $O(N \log N + N^{d/2})$ in d dimensions (Edelsbrunner & Shah, 1996). In 2D, the algorithm is hence still $O(N \log N)$. In 3D, the second term will dominate the efficiency, yielding a worst case efficiency of $O(N^{3/2})$.

Delaunay tessellation

Given the same mesh generators used above, a triangulation of these points is a set of 2D triangles or 3D tetrahedra that have generators as vertices, so that every generator is respectively connected to at least 2 or 3 other generators through at least one triangle or tetrahedron. Of course, there are many ways to achieve this for a given set of generators. The Delaunay triangulation is then the specific set of triangulations that fulfils the *empty circumsphere* criterion: a triangle or tetrahedron is only valid if the circumcircle/circumsphere does not contain any other generator apart from the 3 or 4 generators that are its vertices (Delaunay, 1934). Generators on the border of the circumcircle of circumsphere are allowed, since otherwise it would be impossible to construct a Delaunay tessellation for generators placed on the vertices of a Cartesian grid, which is a perfectly acceptable representation of a homogeneous density (this is also the reason why we consider the specific *set* of triangulations fulfilling this criterion; the Delaunay tessellation is not unique for some sets of generators).

Delaunay tessellations have interesting properties as well. It can be shown that for a specific set of points, the Delaunay tessellation is the triangulation of those specific points that maximises the minimum angle occurring for all angles in all triangles. If we interpret the triangles or tetrahedra as being the cells of an unstructured mesh, then the union of these cells exactly fills the entire convex hull of the set of generators: the 2D or 3D polygon with generators as vertices that encompasses all generators, see Fig. 2.6.

For our purposes, the most important property of the Delaunay tessellation however is its geometrical link to the Voronoi mesh (de Berg *et al.*, 2008). Since the triangles or tetrahedra of the Delaunay tessellation have a unique circum-

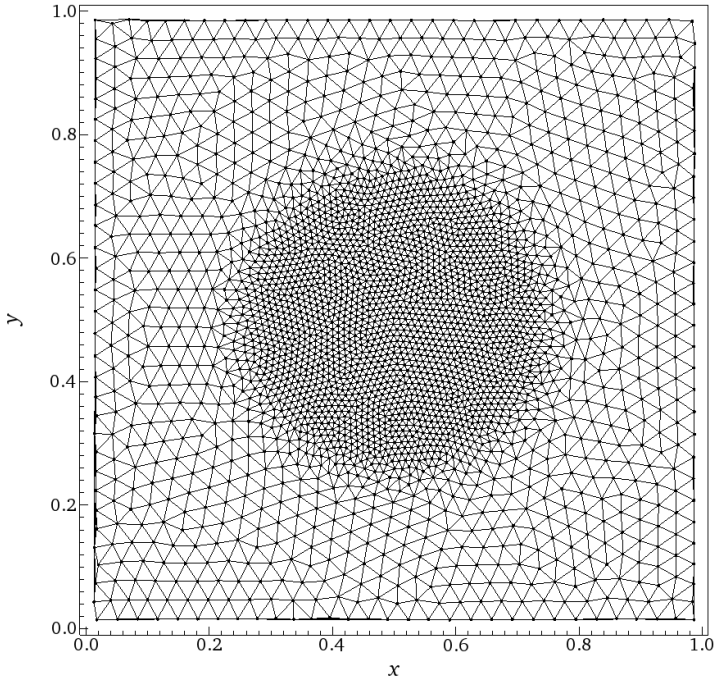


Figure 2.6: *The Delaunay tessellation for the set of generators sampling the 2D density distribution. Note that the tessellation does not fill the entire square box, but fills the convex hull of the generators.*

sphere (even in the case when the triangles/tetrahedra are not unique, since then two or more triangles/tetrahedra will share the same circumsphere), the midpoints of these circumspheres will also be unique. Furthermore, these midpoints are those points in space that are equidistant from 3 or 4 generators of the tessellation and hence correspond to vertices of cells of the Voronoi mesh for the same generators. The boundaries of the Voronoi cells can then be constructed by connecting these midpoints. We even know which midpoints to connect: the midpoints of the circumspheres of neighbouring triangles/tetrahedra in the tessellation. This also means that generators that are vertices of a triangle or tetrahedron of the tessellation will be neighbours in the Voronoi mesh.

The Delaunay tessellation is hence dual to the Voronoi mesh, in the sense that one can always be constructed from the other, since they encode similar

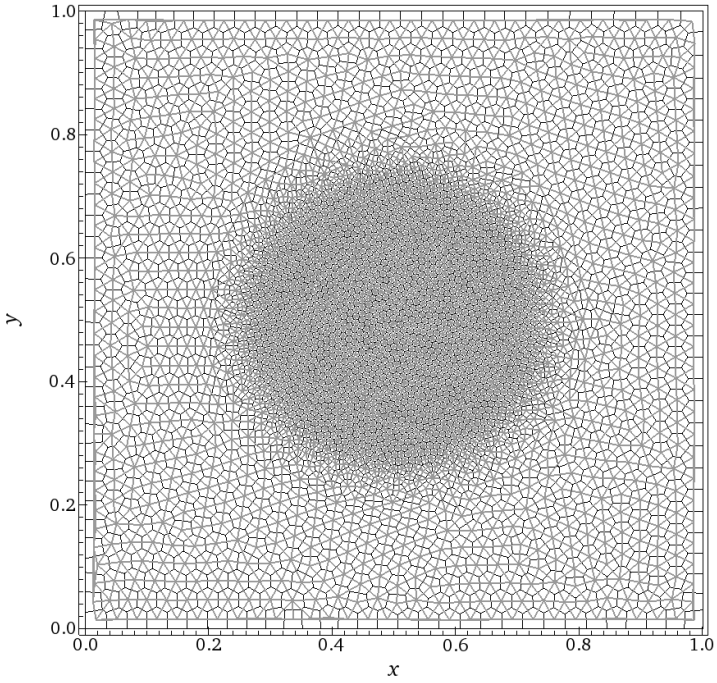


Figure 2.7: *The Voronoi mesh and the Delaunay tessellation for the set of generators that samples the 2D density distribution. The triangles of the tessellation connect the generators of the cells of the mesh that are neighbours. The vertices of the cells are the midpoints of the circumcircles of the triangles.*

information. To construct the Voronoi mesh, we need to know which generators are neighbours and we need the midpoints of the circumspheres of 3 or 4 generators that are *consecutive* neighbours in some sense (which will be made more clear later). In the Delaunay tessellation, this information is directly available through the triangles or tetrahedra, which explicitly specify the neighbour relations in terms of the connecting triangles or tetrahedra. The duality between the Voronoi mesh and the Delaunay tessellation is illustrated in Fig. 2.7.

Incremental construction

If we would have the Delaunay tessellation of the set of generators, it would hence be a trivial $O(N)$ task to construct the Voronoi mesh for the same set of gener-

ators. We are left with the task to construct the Delaunay tessellation. A wealth of algorithms is available to achieve this, and some important mathematical software libraries include triangulation algorithms. The conceptually most easy and stable algorithm is the incremental construction algorithm, for which we start with a valid tessellation, and then one-by-one add the generators and make sure the tessellation is valid again before proceeding with the next generator (Guibas *et al.*, 1992).

As a starting point, we will take a very large triangle or tetrahedron that encompasses the entire simulation box (and has non-generators as vertices), which is trivially a valid Delaunay tessellation. Adding a new generator then always proceeds in the same way, which however is very different for the 2D and 3D case. In both cases, we first need to find the triangle or tetrahedron that contains the new generator. To do this, we need a geometrical test to tell us whether a point is inside or outside a triangle or tetrahedron. We start with some first guess (usually generators are added in some clever order, so that a very good first guess is the last triangle or tetrahedron that was affected when adding the previous generator to the tessellation) and perform this test. If the new generator is outside the triangle or tetrahedron, we do a very quick test to find out at which side the line connecting the midpoint of the triangle/tetrahedron with the new generator leaves the triangle/tetrahedron and use the neighbour at that side as a new guess. This allows us to very efficiently find the triangle or tetrahedron that contains the new generator.

Of course, it can happen that the generator is on a boundary between multiple triangles or tetrahedra, in which case we will have a degeneracy. Most degeneracies can be solved by treating them as a special case, except for the fatal degeneracy where two generators happen to coincide, in the sense that their coordinates are exactly the same. We will always make sure that this cannot happen by requiring all generators to have distinct coordinates. Very relevant in this case is the occurrence of round off error that might skew the tests and make the algorithm unstable. We will discuss this in more depth below.

After the generator has been located inside the old tessellation, the triangle or tetrahedron containing it is split into several new triangles or tetrahedra. For each of those, we then have to check whether they fulfil the empty circumsphere criterion. Every faulty triangle or tetrahedron is then replaced and all new triangles or tetrahedra are checked until no faulty triangles or tetrahedra can be found any more. At that point, the tessellation is valid again and we can continue to add the next generator. Below, we will discuss this algorithm in more detail for the specific case of a 2D and 3D Delaunay tessellation.

2D In two dimensions, the triangle containing the new generator is split into three new triangles, as illustrated in Fig. 2.8. There is one degenerate case, in

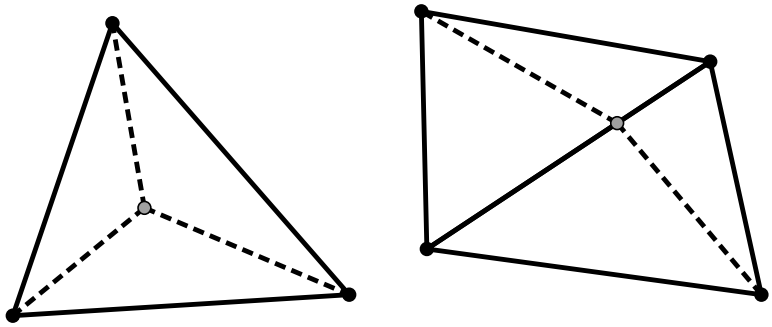


Figure 2.8: *In 2D, the triangle containing the new generator (gray) during the incremental construction is split into three new triangles (*left*). If the new generator lies on the common edge of two triangles, these two triangles are split in four triangles in total (*right*).*

which the new generator lies on the line segment separating two neighbouring triangles. In this case, we split the two triangles that share the line segment into four new triangles in total.

If one of the newly created triangles fails the empty circumcircle test, there will always be a second triangle that also fails the test: the triangle consisting of the two vertices of the faulty triangle that are not the new generator, and the other generator that lies inside the circumcircle. These two faulty triangles can be replaced by two new triangles by shuffling the vertices in a so called *face flip*, see Fig. 2.9.

3D In three dimensions, the tetrahedron containing the new generator is split into four new tetrahedra. We now have two degenerate cases: the case where the generator lies on the face between two neighbouring tetrahedra, and the case where it lies on the edge between in general n neighbouring tetrahedra. In the former case, the two tetrahedra that share the face are replaced by six new tetrahedra, in the latter case the n tetrahedra are replaced by $2n$ new tetrahedra, see Fig. 2.10

When a tetrahedron fails the empty circumsphere criterion, then just as in 2D, there will be at least one other faulty tetrahedron, formed by the three vertices that are not the newly added generator and the generator that lies inside the circumsphere. There are four possible solutions, depending on the behaviour of

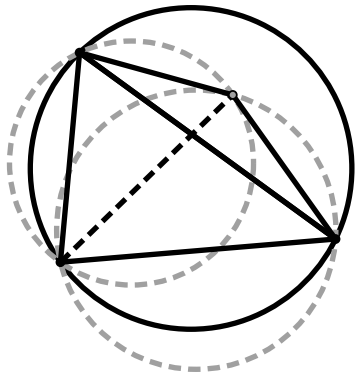


Figure 2.9: *When the generator (gray) opposite of the newly added generator in a neighbouring triangle lies inside the circumcircle of the new triangle (black), the two triangles are replaced by two new triangles, each containing the newly added generator and the opposite generator. This restores the empty circumcircle property for these triangles, as can be seen from the gray circles.*

the line that connects the new generator with the generator that lies inside the circumsphere.

This line will always intersect the common face of the two faulty tetrahedra. If the intersection point with this face lies inside the faulty tetrahedra (and hence inside the common triangle of the two tetrahedra), the two tetrahedra can be replaced by three new tetrahedra in a 2-to-3 flip (Fig. 2.11). If the line lies outside the triangle, then either there exists another tetrahedron that has the two vertices on the line and two vertices of the common triangle as vertices, or this tetrahedron does not exist. If it exists, we can replace this tetrahedron and the two faulty tetrahedra by two new tetrahedra in a 3-to-2 flip (Fig. 2.12). If it does not exist, we cannot solve this specific faulty tetrahedron, but the tessellation will be restored through a later flip in another tetrahedron.

The fourth case is the degenerate case where the intersection point of the line and the face lies on an edge of the common triangle (and hence a common edge of the two faulty tetrahedra). If this edge is shared by exactly four tetrahedra in total, then these four tetrahedra can be replaced by four new tetrahedra in a 4-to-4 flip (Fig. 2.13). If it is shared by more or less tetrahedra, then again we cannot solve this faulty tetrahedron and the tessellation will be restored by a flip in another tetrahedron.

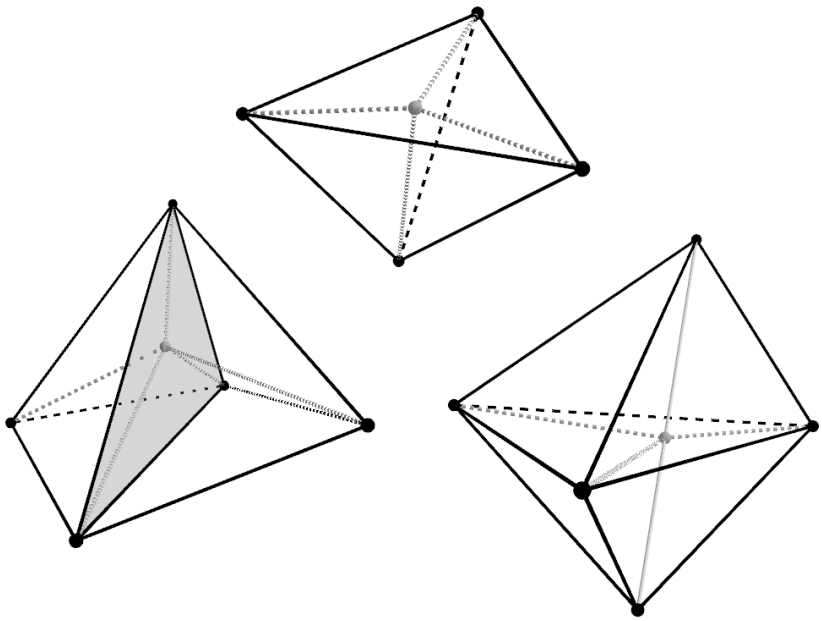


Figure 2.10: *The three possible cases for the insertion of a new generator in the 3D Delaunay tessellation.* Top: *normal case, the single tetrahedron containing the new generator (gray) is split in four new tetrahedra (dashed gray lines).* Left: *degenerate case where the new generator (gray) lies on the common face of two tetrahedra (gray face). The two tetrahedra are replaced by six new tetrahedra (dashed gray lines).* Right: *degenerate case where the new generator (gray) lies on the common edge (full gray line) of* n *(in this case* $n = 3$*) tetrahedra. The* n *tetrahedra are replaced by* $2n$ *new tetrahedra (dashed gray lines).*

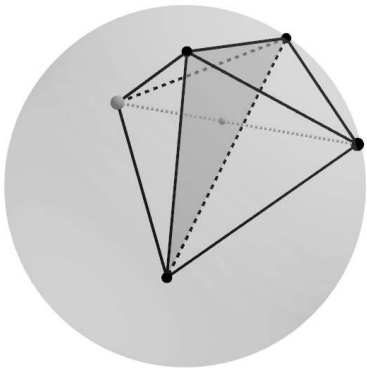


Figure 2.11: *When the generator (large gray point) opposite the newly added generator (large black point) lies inside the circumsphere of one of the newly added tetrahedra, and the line joining this generator and the newly added generator (dashed gray line) intersects the common face of both tetrahedra (darker gray) on the inside (small gray point), then the line segment between both generators becomes the common edge of three new tetrahedra that replace the two tetrahedra in a 2-to-3 flip.*

Round off error All operations described above are a combination of complex bookkeeping operations on the internal representation of the tessellation, and two geometric tests: a test to check whether a point lies *above*, *below* or *on* the line through two points or the plane through three points (an *orientation test*), and a test to check whether a point lies *inside*, *outside* or *on* the circle through three points or the sphere through four points (an *incircle/insphere test*).

Both tests can (both in 2D and 3D) be reduced to determining the sign of the determinant of a matrix involving the coordinates of the points involved, see Appendix A. If all calculations on a computer were *exact*, then all tests would always give the exact same answer and the algorithm would be stable. But computers are not exact: they represent numbers as a combination of a finite number of bits, 64 for most current systems. For integer arithmetics, the maximum number that can then be represented on a computer is equal to 2^{64} and numbers larger than that cannot be used (in fact, if we want to allow negative numbers, we need to reserve one bit for the sign and the maximum number would be 2^{63}).

Since integer arithmetics would be very limited for any relevant physical system (only supporting a numerical dynamic range of $\sim 10^{19}$), computers also support *floating point* arithmetics. A floating point value is a combination of

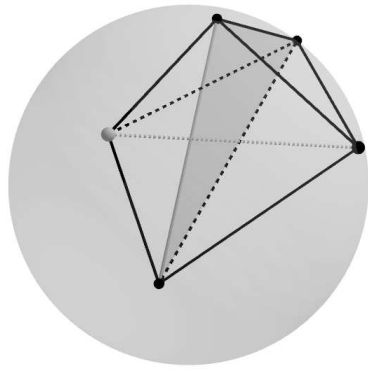


Figure 2.12: *When the generator (large gray point) opposite the newly added generator (large black point) lies inside the circumsphere of one of the newly added tetrahedra, and the line connecting both generators (dashed gray line) intersects the common face of both tetrahedra on the outside, then a 3-to-2 flip is possible if the segment between both generators is the edge of an existing tetrahedron. In this case, we can remove the segment that lies entirely inside the three tetrahedra (full gray line), effectively converting the three tetrahedra into two new tetrahedra.*

the form $m \times 2^e$, where m is the *mantissa* and e the *exponent*. In most representations, the exponent is an integer as defined above. The mantissa is a binary number with a fixed point after the first bit, e.g. 1.0001 for a 5-bit mantissa. The IEEE standard (IEEE, 2008) specifies that a *double precision* floating point has a 53-bit mantissa (including sign bit) and a 11-bit exponent. This is the floating point value type we will use in this work. It has a much larger numerical dynamic range of $\sim 10^{616}$.

Since there is no physical system with a numerical dynamic range that even approximately resembles this number, the finite character of double precision floating points is no longer an issue. What is an issue, is the fact that operations involving multiple floating points are not necessarily exact. To add two floating points, we need their exponents to be the same, which might mean we have to shift the mantissa of one of them, which causes a round off error. If we multiply two floating point values, we add the exponents and multiply the mantissas. If the resulting mantissa is too large, we can scale up the resulting exponent, but we will still loose precision at the other end of the mantissa. Most modern architectures are capable of using extended precision during the calculations on the CPU, which means adding floating points might be less prone to round off

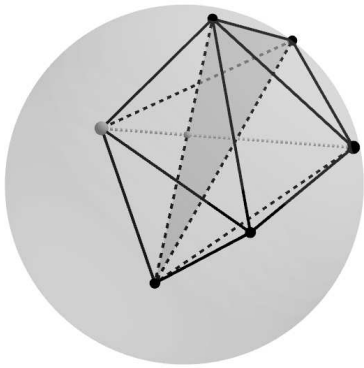


Figure 2.13: *When the generator (large gray point) opposite the newly added generator (large black point) lies inside the circumsphere of one of the newly added tetrahedra, and the line through both generators (gray dashed line) intersects the common face of both tetrahedra on an edge (small gray point) of the tetrahedra, a 4-to-4 flip is possible if this edge is shared by exactly four tetrahedra. In this case, the edge is replaced by the segment connecting the generators, effectively replacing the four tetrahedra by four new tetrahedra.*

then explained above (in fact, the IEEE standard sets strict limits on the amount of round off error that the basic arithmetic operations are allowed to generate). However, at the end of the calculation, the resulting float is put back in memory and rounded again to 64 bits. If we want the result to still be a floating point value with the same specifications as the operands, then every operation involving floating points is susceptible to round off error (Shewchuk, 1997).

The coordinates that are used as input values for the geometrical tests will be floating point values. And to calculate the sign of the determinants, we have to add, subtract and multiply these values. This will involve round off errors, which inevitably means that in some special cases the result of the geometrical test might be wrong: if the determinant is very close to zero, then maybe some intermediate result during the determinant calculation might have been a little bit too large due to round off, resulting in a determinant that is a bit too large and that, if we would calculate it exactly, would have a different sign.

This in itself is not a problem for any algorithm, since computers are deterministic. If the sign of the determinant is wrong, it will always be wrong for that specific set of input values. Whether or not a point is inside or outside a sphere does not really matter for the final Voronoi mesh, since the neighbour relations

that might be wrong will correspond to very small faces that will not have any influence on the physics of the problem. We might then just as well consider the wrong result to be correct.

The issue is that determinants for tests with different input values might be inconsistent. Consider the example of the first step in the 2D incremental construction algorithm, when we need to find the triangle containing the next generator we want to add. To test whether the generator lies inside some triangle, we need to perform three orientation tests, one for every side of the triangle. If all tests indicate that the point is above that specific side, then the generator is inside the triangle. Suppose now the generator is very close to one of the sides, so that the orientation test for that side incorrectly signals the generator to lie below that side. We incorrectly assume the generator lies outside the triangle and continue the search algorithm in the neighbouring triangle that shares that side with the first triangle. Again, we test the three sides, but since in this case the order of the vertices of the triangle is different, we now correctly find that the generator again lies below the side (but now from the point of view of the neighbouring triangle). Hence we continue the search in the neighbouring triangle that shares the side, which unfortunately turns out to be the initial triangle again! We are hence stuck in an infinite loop, jumping back and forth between the two triangles.

To prevent this from happening, we have to make sure that geometrical tests are consistent: they should give the same answer, irrespective of the order in which the values are passed on to the test. While this might seem an obvious thing to do, it turns out this is actually not so easy. The most straightforward way to do this turns out to be by requiring the tests to not only be consistent, but to also be correct. This requires us to use extended precision arithmetics, as is discussed in Appendix A.

2.1.3 Evolving unstructured meshes

In Chapter 3, we will use the Voronoi mesh as the basic discretization for a *moving mesh* hydrodynamical integration method, where we make use of the remarkable property of the Voronoi mesh that it evolves continuously when the generators move continuously, as discussed above. This integration scheme will require us to maintain a Voronoi mesh for a large number of generators, whereby the generators are allowed to move in between integration steps (but the movements will be *small* – we will discuss what we mean with small below).

Since we have an algorithm to construct a Voronoi mesh for a given set of generators, the easiest way to maintain this mesh would be to rebuild the entire mesh for every time step, using the latest coordinates for the mesh generators. This is the method used in Springel (2010) and Duffell & MacFadyen (2011), and

is also the default method used in our own moving mesh code SHADOWFAX (see Chapter 4).

Intuitively, this feels unnecessarily expensive, since the differences in the mesh between two consecutive time steps will be small. Some cells will gain neighbours, some will loose neighbours, and a considerable part of the cells might just change shape without changing neighbour relations. The latter means that the underlying Delaunay tessellation will not change for these specific cells, since this tessellation encodes the neighbour information. If we rebuild the entire mesh, we throw away all information from the previous time step, while a considerable part of this information actually does not change. Duffell & MacFadyen (2011) use information from the previous time step when reconstructing the mesh, but they do not conserve this information.

We experimented with a novel *mesh evolution* algorithm, which makes better use of the information from a previous time step to evolve rather than rebuild the mesh. The algorithm starts with a valid Voronoi mesh (obtained using the incremental construction algorithm), and then restores this mesh after the mesh generators have been allowed to move (if the movement is small enough). This algorithm effectively yields a valid Voronoi mesh at every time step, while at the same time being faster than a total reconstruction. The algorithms for 2D and 3D are a bit different, and will be discussed below. We will focus on the main steps of the algorithm, the specific solution for various substeps is discussed in more detail in Appendix B, and is subject of Vandenbroucke & De Rijcke (*in preparation*).

2D

In 2D, cells can lose or gain neighbours through one single process involving four neighbouring cells, which we will call a *face flip*, analogous to the face flip in the incremental Delaunay construction algorithm. After a movement of one or more generators, a non-neighbour of a cell enters the circumcircle through the cell generator and two of the cell neighbours (that are also neighbours of the non-neighbour), thereby invalidating the Delaunay triangle that connects the generators of these cells. The face associated with the neighbour relation between the two mutual neighbours thereby flips orientation with respect to the cell generators, which leads to two faces of these cells intersecting, see Fig. 2.14. To solve this, it suffices to (a) remove the flipped face and the associated neighbour relation for the two mutual neighbours, and (b) insert a new neighbour relation and associated face in between the original cell and the non-neighbour. The net effect is that the face between the mutual neighbours flips to a new face between the original cell and the non-neighbour.

The easiest place to detect the face flip is in the cells that loose the face, for they have all three other generators as a direct neighbour, so that we only need

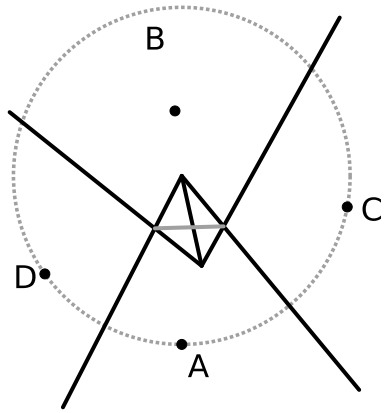


Figure 2.14: *When generator B enters the circle through generator A and two consecutive mutual neighbours of A and B, C and D, the face between C and D flips orientation, causing an overlap of the cells of A and B. The mesh can be restored by removing the face between C and D, and inserting a new face (gray line) in between A and B.*

to check neighbours of cells. The face flip is then detected by explicitly checking the empty circumcircle criterion for all triples of consecutive neighbours. Since a new face can only be created if at the same time another face is removed, this suffices to cover all cases.

To construct a mesh evolution algorithm, we first need to store the information from the existing (valid) Voronoi mesh in such a way that we can easily distinguish between the geometry of the mesh, set by the actual positions of the vertices, and the abstract notion of the connectivity, which tells us which cells are neighbours of each other. We opt for a scheme in which every cell explicitly stores a list of neighbouring cells, ordered counterclockwise, whereby we store a reference to the generator of the neighbouring cell, rather than an actual generator position. We can then keep this information when the generator positions change and reconstruct the mesh after the movement. This is achieved by calculating the midpoints of the circumcircles through a cell generator and two consecutive neighbours of the cell, which are the vertices of the Voronoi cell.

Notice that this procedure will always work, as long as the neighbouring information stored in the cells is consistent. Consistent means that cells should always be mutual neighbours, and that when two cells share two neighbours, the ordering of these neighbours is consistent in both cells (i.e. if A comes before B in

cell C, then B should come before A in cell D). Having a consistent connectivity however does not mean the Voronoi mesh constructed from it is valid. This is only the case if the connectivity satisfies the empty circumcircle criterion. There are many possible triangulations of a set of generators, but only one (or some for degenerate generator sets) corresponds to the Delaunay tessellation which is the dual of the Voronoi mesh.

The procedure is now very straightforward: we move all mesh generators, but keep the connectivity from the previous time step. For every cell, we loop over its ordered list of neighbours and check the empty circumcircle criterion for all triples of consecutive neighbours. If a triple fails the test, we remove the faulty neighbour from the neighbour list, and add a new face in between the other two neighbours. The new face is then added to some secondary test stack, so that all faces are tested at least once. If no more tests need to be performed, the Voronoi mesh is valid again.

To see that this is indeed the case, consider again the underlying Delaunay tessellation. After a movement of the generators, the structure we obtain using the connectivity from the Delaunay tessellation before the movement, will result in a triangulation of the generators. This triangulation will however no longer correspond to the Delaunay tessellation. The operations described above are nothing else than a series of face flips in this triangulation. It is a property of a general 2D triangulation that it is always possible to convert it to the corresponding Delaunay tessellation by means of a finite series of such face flips (de Berg *et al.*, 2008). This property does not hold for triangulations in 3D, which will give us some trouble below.

Before that, we need to say something about degenerate cases, since these tend to complicate most geometrical algorithms. In this case however, there are no problems. The only degenerate case that could occur is when a generator and a triple of consecutive neighbours are cocircular. In practice, this means that the face in between the generator and the middle neighbour will have shrunk to zero length. Likewise, the non existing face between the other two neighbours would have zero length. Since zero length faces are uninteresting to us, we do not have to treat this case. The only condition is that we need to be absolutely sure that the cocircularity of the generators comes out irrespective of the order in which the generators are tested, so that the zero length face is consistently present in both neighbours. This can be realised using the exact arithmetics described above and in Appendix A.

3D

In 3D, unlike in 2D, face insertion and removal are separate processes. Since every vertex of the Voronoi mesh is now determined by four mesh generators,

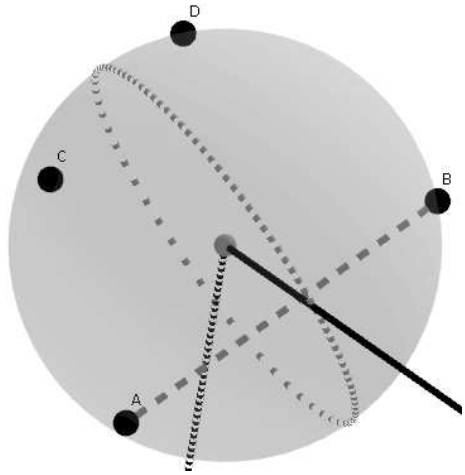


Figure 2.15: *The vertices of the 3D Voronoi face between generator A and generator B are the midpoints of the spheres through A, B and two consecutive face neighbours (C and D) of the face. The generators C and D are both also real neighbours of A and B.*

deviations will involve five generators. Representing the mesh internally in a useful format also becomes harder. We will discuss this issue first.

Suppose we want to represent the connectivity of the mesh by means of some neighbour list for every Voronoi cell, as in 2D. Since every face now has the same geometric structure as the whole cell in 2D, we need multiple lists: one for every face. For every real neighbour of the Voronoi cell, we need an ordered list of what we will call *face neighbours*, so that the vertices of the Voronoi cell are now the midpoints of the circumspheres through the mesh generator, the neighbour associated with the face, and two consecutive face neighbours in the face neighbour list for that neighbour, see Fig. 2.15. A face neighbour should always be a real neighbour of the cell as well, and can be face neighbour for multiple faces of the cell.

Inserting and removing a face will now not only require us to add or remove neighbour relations between cells, it will also cause changes in the face neighbours. In principle, every face starts out as a very small triangle, having only three face neighbours (we will soon encounter a degenerate case which does not). When new neighbours are added to the cell, this number can grow, allowing the face to have the more general polynomial size we usually associate with Voronoi cells. Likewise, a face will only be removed when it has a triangular shape again, which

means it first has to go through a phase of successive face neighbour removals.

Having a consistent connection now has a slightly different meaning. We still require neighbour relations to be mutual, but since the neighbours themselves need no longer be ordered, the requirement that neighbours a consistent ordering in between neighbouring cells is dropped. We however now have the extra requirement that face neighbour relations between neighbouring faces of the same cell should be mutual.

We can again use the connectivity from a previous step to construct an initial guess for the Voronoi mesh after a movement of the generators, and then try to restore the mesh by a series of operations we will lay out below. However, since these operations will still correspond to flips in the underlying triangulation, this algorithm will only be successful if it would be possible to convert an arbitrary triangulation of a set of 3D points into a valid Delaunay tessellation for that set of points by means of flipping operations. We saw above that a 3D triangulation does not have this property.

This means it is impossible to first update all generator positions, and then restore the mesh, at least theoretically (in some cases, this algorithm does work, but we can never know if it actually did without doing a very expensive check). However, we can do something similar if we move the generators *one at a time*. As shown above, the Voronoi mesh will always change continuously when the generators change continuously. The problem with the algorithm as laid out above is that the actual generator movements are not continuous, but correspond to time discretized continuous jumps of the generator positions. We can however make the movement continuous by only moving one generator and if necessary even subdividing this generators movement in smaller substeps. Then the only mesh changing processes will indeed be face insertions and removals, which can be handled.

What now are the mesh restoring operations we mentioned? First of all, there is the insertion of a new face. If the mutual neighbour of three neighbouring cells enters the circumsphere through the three generators and another mutual neighbour of the three cells, we need to create a new triangular face in between the two mutual neighbours, which will have the three neighbouring cells as face neighbours, see Fig. 2.16. The face insertion can only be detected in the three mutual neighbours, as a violation of the empty circumsphere criterion for three consecutive face neighbours in the faces between these neighbours. From the point of view of these faces, we have a similar situation as for the 2D face flips, since now a common edge in these faces will flip its orientation with respect to the general direction of the face (we will need to be more clear about this below, especially for the case of faces with three and four vertices). To restore these faces, we have to remove the flipped edge, which corresponds to removing a corresponding face neighbour from the lists of those faces.

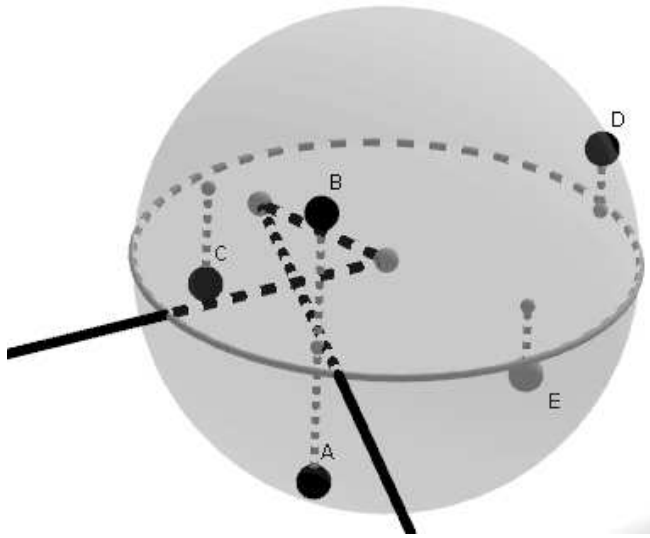


Figure 2.16: *When generator E enters the sphere through neighbours A and B, and face neighbours C and D, the face segment in between the vertices mapped out by ABCD and ABDE respectively, flips. The face can be restored by creating a new face (and neighbour relation) between C and E. The same segment will also flip in the faces between A and D, and B and D.*

When an edge flips in a face with only three edges, the entire face flips, i.e. all edges change orientation, so that we get a situation similar to the flipped face in 2D (where neighbouring faces intersect and it seems as if part of the cell is literally split off from the rest of the cell). In this case, we need to remove this face and we actually have a face removal. Since the face removal is the opposite process of the face insertion, we also have to add a new edge in the faces in between three neighbours that have the two cells as mutual neighbours (and that will of course be the face neighbours of the flipped face).

The processes described above correspond to the 2-to-3 and 3-to-2 flips we encountered in the incremental Delaunay construction algorithm. We also encountered a degenerate 4-to-4 flip, which we will also have to deal with here. If two edges flip in a face with only four edges, then we can also consider the entire face to have flipped. However, instead of just removing the face, we now also have to insert a new face (with four edges) in between the two face neighbours that correspond to the two flipped edges. This also requires some face neighbour removals and insertions, which are explained in more detail in Appendix B.

The fact that face insertion and removal are separate processes, and the existence of a degenerate face flip, make the evolution algorithm a lot more complicated. For every mesh generator and every neighbour, we have to subject all triples of consecutive face neighbours to the empty circumsphere test. We then need to keep track of the total number of flipped edges. If this number is too large, we cannot restore the mesh and we have to refine the movement. If the number has some specific value corresponding to one of the three cases, we restore the mesh and continue. Since face insertions are not detectable in the cells that gain the new face, we also have to include the neighbours of the cells in this procedure.

Small movements

The algorithms presented above crucially depend on the movement of the generators being small, since only then it does make sense to try to reconstruct the cells using information from the previous time step. We did however not quantify the size of the movements that yields a stable algorithm. Furthermore, we need to find out what happens when movements are too large: will the algorithm simply crash or deadlock, or will it produce a faulty updated mesh? In the former case, we can always revert to a reconstruction of the entire mesh and continue the simulation. In the latter case, we might end up using a geometrically incorrect mesh, and it is not safe to use the algorithm.

For the 2D algorithm, it is easy to show that we can never end up with an incorrect mesh, since every wrong neighbour relation inevitably causes a flipped face in some cells, which will always be detected. Neither can the algorithm deadlock, since an arbitrary 2D triangulation can always be converted into a Delaunay tessellation by means of a finite number of flips (de Berg *et al.*, 2008).

This means that the algorithm can only crash. By running the algorithm on a large set of randomly moving generators, we found one situation where this happens: when the generator of a cell geometrically falls outside of its reconstructed cell. This happens either when the movement of the cell is larger than the size of the cell, or when the generator is close to the cell boundaries before the movement. Either way, detecting if a point lies inside or outside a polygon is a well known problem in computational geometry, and there exist very fast algorithms to do it. We use a technique based on the *winding number* of the cell (Hormann & Agathos, 2001). After the movement and before the restoration phase, we check the winding number for every cell and crash the algorithm whenever a generator is detected to be outside its cell.

In 3D, the important flipping property that turns out to be crucial for the stability of the 2D algorithm no longer holds. It is therefore perfectly possible for the algorithm to produce an invalid mesh or deadlock. Experimentally, we were able to assess that in these cases, the algorithm crashes somewhere in most of

Table 2.1: *Number of successful consecutive random generator movements before the evolution algorithm crashes.*

2D		3D	
fraction	iterations	fraction	iterations
0.5	2	0.25	15
0.25	5	0.125	32
0.125	19	0.0625	100

the cases, so that it is still possible to revert to the old algorithm. However, it is also possible that the algorithm is unsafe. To prevent this from ever happening, we explicitly limit the maximal movement of a generator to a fraction of the cell size, by subdividing the generator movement in smaller movements if it is too large. By explicitly checking the mesh validity after every update for a large test problem, we were able to finetune this fraction to a value of 10%, which is small enough to be unconditionally safe, but large enough to still be fast.

To check when generator movements are small enough for the algorithm to work efficiently, we set up a test problem in which the generators are moved completely randomly, with the size of the random movements limited to some fraction of the cell size. After restoring the mesh, we then repeated this procedure until the algorithm crashed. In 2D this happens when a generator lies outside its cell after the movement, in 3D the algorithm crashes when a case cannot be resolved correctly.

As initial condition for the test, we use a uniform periodic box with unit length, in which 10,000 generators are placed by drawing random coordinates from a uniform distribution. The set of generators is then allowed to relax using 100 iterations of Lloyd's algorithm (see below). This results in an initial mesh with an average cell volume of $(1.000 \pm 0.073) \times 10^{-4}$, and an average cell size of $(5.600 \pm 0.021) \times 10^{-3}$ in 2D, and an average cell volume of $(1.000 \pm 0.047) \times 10^{-6}$ and cell size of $(2.900 \pm 0.045) \times 10^{-2}$ in 3D.

We quantify the random mesh movement as a fraction of the average cell size, and list the number of iterations of the random displacement algorithm before the restoration algorithm crashes (with a maximum of 100 iterations) in Table 2.1. Since the random movements of the generators correspond to a random walk in 2 and 3 dimensions, we can estimate the average size of the generator movement before the algorithm crashes as respectively the square and cubic root of the number of iterations, multiplied with the amplitude of the random movements. We then see that this average movement size before a crash goes down with decreasing amplitude. This is to be expected, as every random movement will distort the mesh, and crashes are caused by heavy distortions of the mesh. However, the average movement of a single generator is not really meaningful

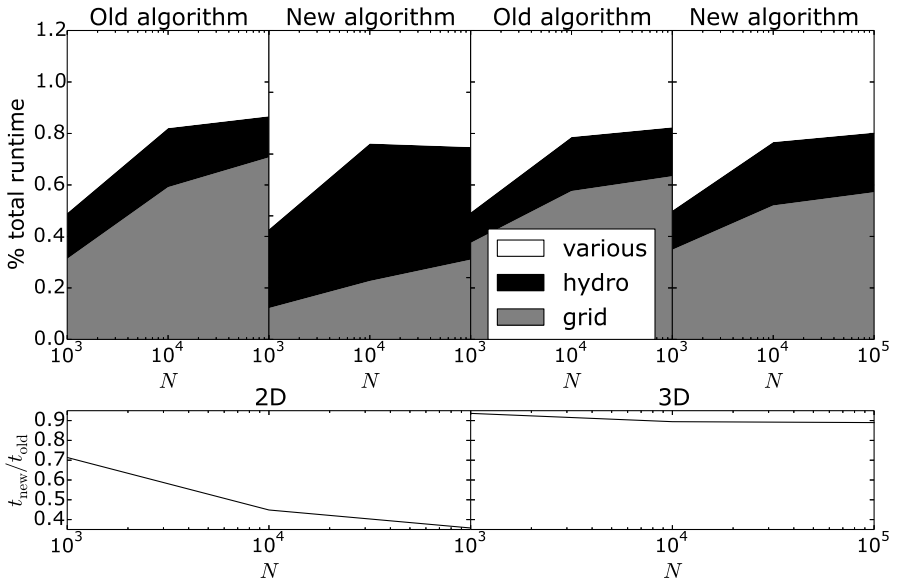


Figure 2.17: *Fraction of the total run time of a* SHADOWFAX *simulations spent in grid construction and hydrodynamical integration routines, and speed up obtained using the new algorithm, as a function of the number of cells N for a simple spherical overdensity test (see Chapter 4).*

in this case, as the fatal distortions are likely caused by outliers: neighbouring generators that consistently move away from each other during a (small) number of consecutive random steps. For a small number of iterations, their contribution can be better estimated by simply multiplying the number of iterations with the amplitude of the random movements. Moreover, in hydrodynamical applications, the cell movement will contain correction terms to keep cells more regular. These terms will keep the mesh more regular over a large number of iterations. They can however not guarantee regularity over a small number of iterations.

Using this heuristic, we see that the 2D algorithm is not stable for displacement fractions of 0.5 and 0.25, but is stable enough for a displacement fraction of 0.125, or approximately 10% of the average cell size. In 3D, this is the fraction that is explicitly used to subdivide the generator movement. As a result, the algorithm is stable enough for all fractions in this case for our simple test.

Efficiency

Fig. 2.17 shows timing results for a number of simulations using the old and the new mesh evolution algorithm for a hydrodynamical test problem that will be introduced in Chapter 3 and discussed in more detail in Chapter 4. The tests for the old and the new algorithm are identical, except for the way in which the mesh is evolved: for the old algorithm, the entire mesh is discarded and reconstructed from the new positions of the generators, while for the new algorithm, we use the advanced evolution algorithm discussed above.

In 2D, there is a clear speedup of more than a factor two, and this factor increases with increasing cell number, indicating that the mesh evolution algorithm scales better with the number of cells. The speedup in 3D is less profound due to the higher complexity of the algorithm, but there we also notice a slightly better scaling behaviour.

2.2 Particles

In the previous section, we saw that grids come in two flavours: a computationally cheap version that is static and relies on a power of two hierarchy for cell refinement, and a much more adaptive unstructured version that requires a lot of resources to construct and maintain.

In this section, we will focus on another method to represent a density distribution: by means of particles. In fact, we already shortly touched upon this subject when introducing the set of generators in the previous section (Fig. 2.3). Recall that these generators were sampled from the density distribution, so that the number of generators in the high density region was higher than in the low density region, the number ratio being equal or close to the density contrast of the distribution. The general idea would then be to assign some mass to the generators, so that we can retrieve the density in some subregion of the box by dividing the total mass of all the generators in that subregion by its volume. If we require the mass of all generators to be the same, then the generator positions indeed sample the density distribution. Since the generators now no longer generate anything, we will just call them *particles*.

Given the particles, we then have to find some computational way to arrive at the densities, which should closely relate to the intuitive description given above. In fact, multiple methods exist; we will focus on two of them next.

2.2.1 Smoothed Particle Hydrodynamics

The first method to assign a notion of density to a volume filled with a discrete set of particles, is by viewing the particles as being some smooth *blobs* instead of

discrete points in space. The mass contained in the particle is then not localised at a single point in space, but is spread out in some spherical volume around the particle coordinates. The blobs of multiple particles can (and will) overlap, so that the density in any point in space can be found by summing up the contributions of all particles that contribute some of their mass to that particular point. To make this more concrete, we need to find a mathematical way to spread out the mass of a particle, and we need to define an algorithm to compute a consistent density from this.

When the resulting density estimate is used as a basis for a hydrodynamical integration scheme, this technique is called Smoothed Particle Hydrodynamics (SPH), see also Chapter 3. Many reviews of this technique can be found in literature, and we will base our short description on that of Price (2012).

Smoothing kernels

In practice, we will define the density distribution associated with one of the particles (particle i) to be

$$\rho_i(\vec{x}) = m_i W(|\vec{x} - \vec{x}_i|, h_i), \tag{2.1}$$

with m_i and $\vec{x}_i$ the mass and coordinates of particle i. $W(|\vec{x} - \vec{x}_i|, h_i)$ is the *smoothing kernel*, i.e. the function that mathematically represents the smoothing procedure, and that depends on some parameter h_i (the *smoothing length*), which sets the size of the smooth blob.

If we want ρ_i to be a density, then it is immediately clear that W should be a reciprocal volume. In order to correspond to a smoothing procedure as described above, we will also require it to be a decreasing function of the distance $|\vec{x} - \vec{x}_i|$. Finally, we will also require it to have *compact support*, meaning that W drops to zero for some cut off radius, and stays zero outside this spherical region. The last requirement is necessary to make the method computationally feasible, since we do not want to add contributions from all particles for all points in space.

Given these restrictions, there is still a large freedom in choosing the smoothing kernel. The most commonly used kernel is the *cubic spline* (Monaghan & Lattanzio, 1985), given by (Springel, 2005):

$$W(|\vec{x} - \vec{x}_i|, h_i) = \frac{8}{\pi h_i^3} \begin{cases} 1 - 6\left(\frac{|\vec{x}-\vec{x}_i|}{h_i}\right)^2 + 6\left(\frac{|\vec{x}-\vec{x}_i|}{h_i}\right)^3 & 0 \leq |\vec{x} - \vec{x}_i| \leq \frac{1}{2}h_i \\ 2\left(1 - \frac{|\vec{x}-\vec{x}_i|}{h_i}\right)^3 & \frac{1}{2}h_i \leq |\vec{x} - \vec{x}_i| \leq h_i \\ 0 & h_i < |\vec{x} - \vec{x}_i| \end{cases}.$$

The precise choice of kernel function is not important for the density estimation procedure below, although it might affect hydrodynamical integration

schemes based upon it. The cubic spline e.g. leads to a vanishing hydrodynamical force for particles that come very close together, which can lead to particles clustering together, the so called *clumping instability* (Read *et al.*, 2010). These problems can be overcome by using more advanced kernel functions.

Density iteration

Given (2.1), it is possible to obtain the local density for any point in space by simply summing up the contributions of all particles whose kernel overlaps with that particular point. In practice, we will only be interested in obtaining density estimates for the particle positions themselves, given by

$$\rho_i = \sum_j m_j W(|\vec{x}_i - \vec{x}_j|, h_j).$$

There is some complexity involved in this simple equation, that we will now try to explain in more detail.

First of all, there is the choice of smoothing length in the kernel function above. If we strictly follow the procedure described above, then we should take this to be the smoothing length of the other particles, h_j. This corresponds to a so called *scatter* operation, in which each particle contributes a fraction of its mass to the surrounding region. If we would replace h_j by h_i in the above equation, we end up with a *gather* operation, in which each particle gathers a fraction of the mass of its neighbouring particles to constitute its own density. If all particles would have equal smoothing lengths, this distinction would disappear. In general, neighbouring particles will have similar smoothing lengths, so that both operations will yield similar results.

Second, there is the choice of parameters. The density defined above depends upon two completely independent parameters: the particle mass and the smoothing length. The former is a measure for the amount of matter that is represented by the smallest resolution element, and we will choose it to be the same for all particles, so that we have a good idea of the global resolution of the method. The latter sets the local spatial resolution, which is the quantity that we want to adapt to the local density field, and will hence treat as a free parameter. The smoothing length is closely related to the number of neighbours for a given particle and will hence also determine the computational cost of the method. We want it to be small enough to make the method cheap, but large enough for the method to be smooth. In practice, we will try to fix the number of neighbours for a given particle to some value.

This requires an iterative procedure in which the densities and number of neighbours are calculated using some first guess for the smoothing length. Assuming the density is spread out over the volume of a sphere with radius the

smoothing length, we can then update the smoothing length guess until the number of neighbours is close to the desired number. Since this requires a large number of *neighbour search* operations, in which we try to find the particles inside a sphere with some radius around a given set of coordinates, we will always use the gather form of the density equation. To speed up convergence, we will allow some flexibility in the number of neighbours.

Modern SPH implementations do even better by using a *weighted number of neighbours* (Hopkins, 2013), in which the contribution of a single neighbour to the total number of neighbours for a particle is also weighted with the smoothing kernel. The number of neighbours is then no longer an integer, but rather a continuous function of space. This way, the iteration always converges after just a few iterations, irrespective of the initial guess for the smoothing lengths.

Since the method by construction smooths out large density jumps, it is impossible to exactly represent the example density distribution using SPH, as illustrated in Fig. 2.18.

2.2.2 Meshless Volumes

The method above provides an easy to program and scalable way to obtain consistent density estimates for a set of particles. It is however very poor in terms of volume partitioning (Hopkins, 2015). Assuming constant particle mass, the contribution of a single particle to the density at some point in space will correspond to some fraction of the *volume* associated with the particle being assigned to that point in space. If we were to assign a real volume to every particle, then we would of course require the sum of the volumes of all particles to be equal to the total volume of the simulation box. Furthermore, we would require the same to hold for any arbitrary subvolume of the total simulation box: the volume of this region as sampled by summing the contributions from all neighbouring particles in some points of the region should correspond to the real volume of that subvolume of the box. The standard SPH density estimate fails on both points, as illustrated in Fig. 2.19.

This is not really a problem if we are only interested in estimating densities, but it leads to problems if we want to use the density estimates as a basis for a finite volume hydrodynamical integration method (see Chapter 3), where such a volume partitioning of space is essential. What we actually want is a method that is as computationally cheap as the SPH density estimate, but partitions space in a way similar to the unstructured meshes introduced in the previous section.

An additional disadvantage of sampling the density from a discrete set of points is that sampling noise will affect the density distribution itself. If we would use the same particles as generators of a Voronoi mesh, this sampling noise completely disappears from the density distribution, and is completely absorbed

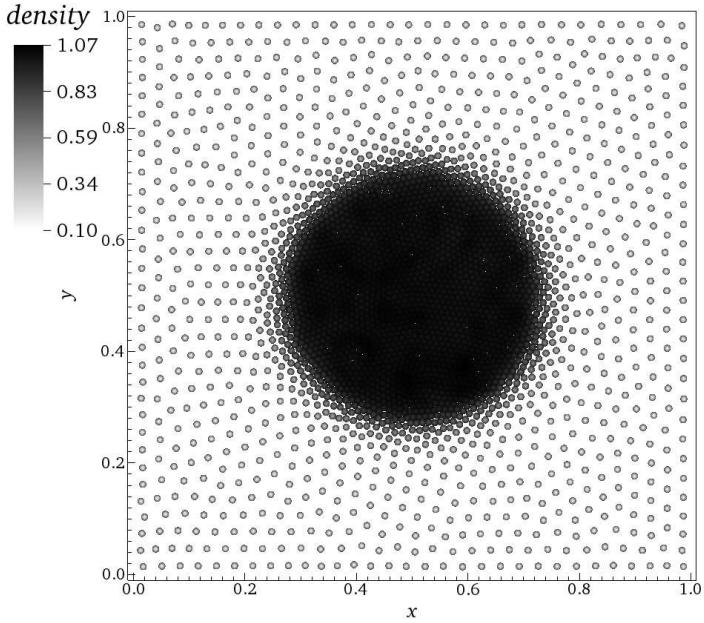


Figure 2.18: *The 2D density distribution as sampled using Smoothed Particle Hydrodynamics. The particles are coloured according to their density. In regions of constant density, the density is recovered reasonably well, although it can be slightly lower or higher due to noise on the particle distribution. Around the edges of the high density region, the density is smoothed out over a layer that is a few particles thick.*

by the cells having slightly different sizes. Noise in the particle distribution should translate to noise in the geometrical properties of this distribution, like volumes, and not in the hydrodynamical properties, like the densities.

The key is to treat the volume in some small region of the simulation box as known. We are not interested in obtaining a volume for this small region, but rather in dividing this volume over the neighbouring particles, so that some fraction ψ_i of the local volume is assigned to particle i (Ivanova *et al.*, 2013; Hopkins, 2015):

$$\psi_i(\vec{x}) = \frac{1}{A(\vec{x})} W(|\vec{x} - \vec{x}_i|, h_i), \tag{2.2}$$

where $A(\vec{x})$ is a normalisation function that makes sure that the total fraction at

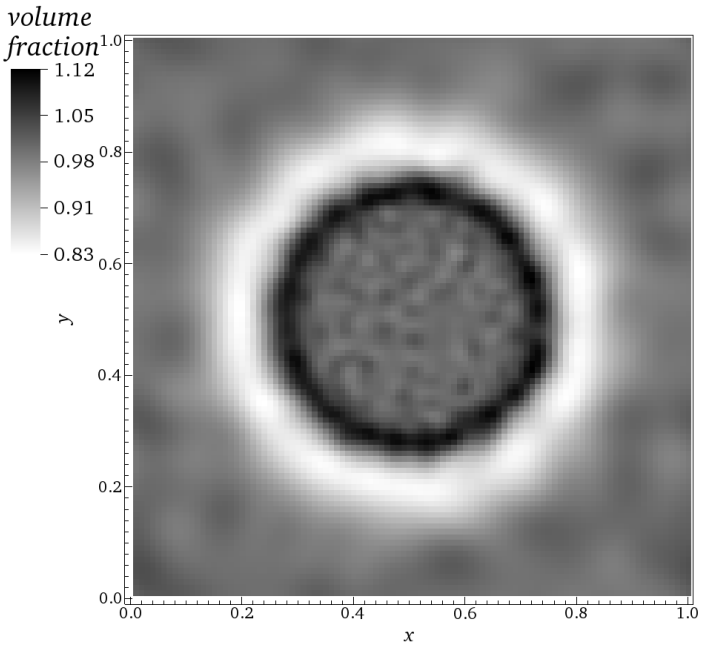


Figure 2.19: *The volume fraction resulting from the SPH density estimation, sampled in* 100×100 *pixels. Around the interface between high and low density region, the volume is oversampled in some pixels, and undersampled in others. As a direct result, the sum of the particle volumes is smaller than the actual volume of the box.*

every point in space is always 1:

$$A(\vec{x}) = \sum_i W(|\vec{x} - \vec{x}_i|, h_i).$$

Since this procedure is also valid at the particle positions themselves, we can then define the volume of particle i to be

$$V_i = \frac{1}{\sum_j W(|\vec{x}_i - \vec{x}_j|, h_i)}.$$

The density of the particle is then calculated from this volume and the mass of the particle.

Unlike the SPH density estimate, this method provides a volume partitioning that indeed sums up to the total volume of the box, and that per construction

has the correct volume for every subvolume of the box. We will see in Chapter 3 that we can also define meshless surface areas for the particles.

2.2.3 Stars

The typical size of a main sequence star like our Sun is $\approx 7 \times 10^8\,\mathrm{m}$ (Emilio *et al.*, 2012). More massive stars (like the red supergiant Betelgeuse) can be up to a factor 1,000 larger (Smith *et al.*, 2009), while most stars will have similar or smaller sizes.

Our Milky Way has an estimated stellar mass of $\sim 6 \times 10^{10}\,\mathrm{M}_\odot$ (McMillan, 2011), non-uniformly distributed in a sphere with a radius of at least 25 kpc (Xu *et al.*, 2015) (with most of these stars residing in the disc of the Milky Way). Assuming all stars have the mass of our Sun, we can estimate the average distance between stars to be of the order of a few pc, where $1\,\mathrm{pc} = 3.0857 \times 10^{16}\,\mathrm{m}$, and hence easily a factor 10^5 larger than the sizes of individual stars. When studying stars in the context of galaxy evolution, treating the stars as being particles with all of their mass located at a single point in space is hence a perfectly acceptable approximation. This will automatically lead us to the treatment of N-body problems. However, we will also show that this treatment is not really suited for simulating the stellar contents of a complete galaxy, so that extra approximations will have to be made.

N-body problems

An N-body problem is the problem of solving the equations of motion for the movement of N point masses that only interact through the force of (Newtonian) gravity. For $N = 2$, the problem has an exact analytic solution, e.g. the elliptical orbit of Earth around the Sun. For all higher values of N, an exact solution is impossible, so that we have to resort to numerical methods to solve it.

Their is a vast body of literature on this topic alone, both to obtain very high accuracy results for systems with small N (e.g. the orbits of the planets in our solar system) (Blanes *et al.*, 2013), and to obtain less accurate solutions for systems with very high N (Heggie & Hut, 2003). The topic has even led to the development of special purpose hardware to calculate the inverse square distance between two points more efficiently (Makino, 2002).

For relatively high N, there are two important approaches. The first approach is to calculate the gravitational interactions between the particles directly, so called *direct summation*. For a system of N particles, there are $N(N-1)/2$ gravitational interactions that need to be calculated, so the runtime of these algorithms scales very badly ($\sim N^2$) when N gets larger. State of the art direct summation codes therefore use very fast algorithms implemented on relatively

cheap Graphics Processing Units (GPUs), that can perform the large amount of calculations in a highly parallelized way (Portegies Zwart *et al.*, 2007).

Direct summation is used when a high accuracy for the gravitational forces is required, to obtain a very accurate solution for the particle movements. Even then, the solution can easily diverge from the real physical solution due to the chaotic nature of the Newtonian force equation: arbitrary small numerical errors can grow unboundedly under repeated application of the equations of motion, so that the solution at any given time depends on the exact order in which numerical operations are performed during the simulation, and different orders can give rise to largely different solutions. It is generally acknowledged that only the properties of an ensemble of such systems can be trusted (Boekholt & Portegies Zwart, 2015).

When we are less interested in the real physical solution and the movements of single stars, but rather want to describe the dynamics of a stellar body as a whole, we can use more approximate methods to calculate the gravitational forces. A very important example is the algorithm of Barnes & Hut (1986). The idea is to put the particles in a hierarchical structure, a *tree*. Every particle will then correspond to a *leaf* of this tree (an example of a tree is shown in Fig. 4.4). Particles that are close together have leaves that are children of the same *node* of the tree. When we want to calculate the gravitational forces for a particle, we start at the top of the tree and calculate the distance between the particle and every node on the highest level. If this distance is large enough, we use the center of mass and the total mass of all child leaves of that node rather than the individual particles to approximate the gravitational force. If the distance is not large enough, the node is *split* and the procedure is repeated at a lower level. For leaves close to the particle, we still go all the way down in the tree and calculate the forces directly, but for particles on large distances an approximation is used.

By using a carefully selected *tree opening criterion*, we can gain an enormous speed up of the calculation (with $O(N \log N)$ rather than $O(N^2)$ scaling), while still having a sufficiently accurate gravitational force for all particles. Barnes & Hut (1986) used a simple geometrical tree opening criterion that is based on the distance between the particle and the node, and the node size. We will however use a more advanced relative tree opening criterion that is also based on the magnitude of the gravitational acceleration during the previous integration time step (Springel, 2005).

Single Stellar Populations

Even with the approximate methods discussed above, it is still almost impossible to simulate the stellar contents of a galaxy on a single star basis (but see Bédorf *et al.*, 2014). This can be easily appreciated when considering the typical mass of a star, $\sim 1\,\mathrm{M}_\odot$, and the typical stellar mass of a dwarf galaxy, $\sim 10^8\,\mathrm{M}_\odot$. If we

would want to simulate the $\sim 10^{10}$ stars (taking into account the large number of stars that have a mass lower than our Sun), we would need an awfully large N.

The problem gets even more complicated if we also want to take into account the effect of the stars on the interstellar medium of the galaxy. Massive stars emit a lot of high energetic UV radiation during their lifetime, which will heat up the surrounding gas. They are short-lived and when the fuel for the fusion reactions that keeps them from gravitationally collapsing is exhausted, they explode as violent supernova type II explosions (SNII) that put even more energy and matter into the surrounding gas. When we want to take into account these *stellar feedback* effects, we hence need to keep track of the age and mass of the star (and its *metal content*, see Chapter 5). This means we need a very large variety of different star types, masses, etc.

To overcome this complexity and to reduce the number of stellar particles we need to integrate over, we will use Single Stellar Populations (SSPs) rather than single stars to simulate the stars in our galaxies (Vazdekis *et al.*, 2012). An SSP is a population of stars that is born in the same part of the interstellar medium, and roughly at the same time. Given that we consider a population that is large enough, it has statistical properties which are better constrained than the properties of individual stars. We know for example what mass fraction of the SSP consists of massive stars that give SNII feedback, or of intermediate mass stars in binary systems that will give rise to SNIa explosions.

Using this approximation, we can of course no longer resolve the trajectories of individual stars. However, since we are only interested in the dynamics of the stellar body as a whole, this is not a problem.

2.2.4 Dark matter

Dark matter is the common name for all matter that only interacts gravitationally, and of which we generally have no idea what it physically consists of. We will assume that it consists of small particles that are orders of magnitude smaller than the typical distance scales in our galaxies, so that we will treat it as a *collisionless* fluid.

To discretize this density distribution, we could use the same techniques we already discussed for the gas. However, since the fluid is collisionless and only interacts gravitationally, the requirements for this discretization are more relaxed than in the case of the gas, where we need a discretization that can be used as a basis for a hydrodynamical integration scheme. As we will see in Chapter 3, the equations of hydrodynamics are derived from the Boltzmann equation. This equation is generally valid for any type of fluid, also for a collisionless fluid. For a collisional fluid, a number of approximations are necessary to convert the

Boltzmann equation to the Euler equations that form the basis of hydrodynamics. For a collisionless fluid, these approximations are unnecessary, and we can solve the Boltzmann equation directly, by using characteristic solutions. To this end, we represent the collisionless fluid by N mass elements (particles). Solving the Boltzmann equation then reduces to solving the gravitational interactions for this N-body system (Dehnen & Read, 2011).

Gravitational softening

Treating the N particles as being point masses can lead to serious efficiency problems. This is because of the $1/r^2$ behaviour of gravity: if two point masses come very close together, their mutual gravitational force becomes very large. This leads to high relative velocities, which means a large number of very small time steps is required to integrate this system. This while the other $N-2$ particles all behave nicely and can be integrated using a lot larger time step.

Since we are not interested in the detailed behaviour of individual particles, and since the particles themselves have no real physical meaning, since they only sample the underlying density distribution of the collisionless fluid, we will suppress these close encounters by means of *gravitational softening*. The idea is to treat the particles as fuzzy blobs of mass, rather than real point masses, so that the gravitational force between two particles takes a slightly different form (Dyer & Ip, 1993). For particles that are reasonably separated in space, the force is still the ordinary Newtonian force of gravity, while for particles close together, the force takes a non-diverging form.

This means we have two parameters setting the resolution of an N-body simulation: the particle mass and the *softening length*. The latter is the generic size of the mass blob represented by a particle. It might be clear from the above that the softening length should be chosen carefully; a small softening length leads to high accuracy, but also has a considerable computational cost. A larger softening length is more efficient, but less accurate. Some methods therefore use variable softening lengths that adapt to the local particle density (Price & Monaghan, 2007) in an SPH-like way. We found that this approach itself is quite expensive, since the iteration required to obtain the softening length is less well behaved than the equivalent SPH iteration due to the collisionless character of the N-body system. We hence will always use a fixed softening length, which will also tell us something about the physical scales up to which the simulations resolve the relevant physics.

When incorporating self-gravity in a hydrodynamical fluid, we will also use an N-body method to calculate the gravitational forces. Depending on the method, the particles will then either be the SPH particles or the generators of the unstructured mesh. The forces will also be softened in this case. In principle, the softening length could then be chosen based on the smoothing length of the SPH

particle or the size of the unstructured mesh cell. However, since we will also treat mixtures of gas and dark matter, we will choose a fixed softening length in this case as well.

Finally, for the stellar particles, which represent SSPs, we will also soften the gravitational force. This makes sense, since we saw above that even the stellar particles rather sample a stellar density distribution than represent real stars.

Sampling noise

To sample N particles from a density distribution (either a dark matter density distribution or an ordinary gas density distribution), we use a Monte Carlo sampling technique called *rejection sampling*. We generate a random position by appropriately combining three independent uniform random numbers, and then determine the theoretical value of the distribution at that position. We then generate a fourth uniform random number, which will determine whether the position can be accepted or is rejected. The latter happens when the random number (in the range [0,1]) is larger than the ratio of the theoretical value of the distribution and the maximal value of the distribution over the whole domain. This way, positions in regions with a high density are more likely to be accepted than those in regions with a lower density, so that statistically the generated positions will sample the distribution.

This method is not very efficient, since for distributions that span a significant range in magnitudes a very large number of positions will be rejected, while we need to evaluate the density distribution for every generated position. The method can be made more efficient by dividing the domain of the distribution in bins width precalculated weights, that allow for a more efficient rejection of invalid positions. It is also possible to sample positions in a skewed way, so that a generated position is more likely to lie in a high density region of the distribution. We will not go into the details here.

A potential problem with this way of sampling is *Poisson noise*, most clearly illustrated in the case of SPH. We saw that SPH works well for particles that are homogeneously distributed in space, so that the minimal inter-particle spacing is close to constant for all particles. If the particles are less homogeneously distributed, the local density as calculated from the SPH iteration can deviate significantly from the desired density. These denser regions will end up having a higher pressure than the surrounding regions, which will affect the hydrodynamics. Similarly, overdensities in the sampled dark matter distribution might grow and form artificial substructures, which is also not desired.

We clearly do not want Poisson noise to dominate the sampled distribution. This means we either need to find a better way to sample distributions, or a good way to reduce the noise on a randomly sampled distribution.

The former can be achieved in various ways (Diehl *et al.*, 2015). It is for example quite easy to generate a perfectly homogeneous particle distribution in a box by putting the particles on the positions of the vertices of a Cartesian grid. This however leads to *special directions* in the particle distribution, since the distribution will have a lot of symmetries. A less symmetric distribution is obtained if the particles are spread out homogeneously along a space-filling curve (see Chapter 4). However, in this case the local density can still be quite noisy, so that in practice this method needs to be combined with a noise reducing method as well.

Noise reduction can be achieved by using a relaxation method that smooths out the initial overdensities. This can be done in a physical way, or in a purely geometrical way. In the case of SPH for example, the initial overdensities will have a higher pressure than the surrounding region, so that the hydrodynamics itself will wash them out if the fluid is evolved adiabatically. If we want to sample a completely homogeneous distribution, it hence suffices to integrate it through time until an equilibrium pressure is reached (Vandenbroucke *et al.*, 2013). The same can be achieved using the force of gravity with its sign reversed, so that particles repel rather than attract each other. The resulting *gravitational glasses* are commonly used as initial conditions for large cosmological simulations (Springel *et al.*, 2005).

Geometrically, we can use the Voronoi mesh introduced above to smooth out Poisson noise. When we iteratively move the generator of a Voronoi cell to the centroid of the cell and recalculate the mesh, then the mesh very quickly relaxes towards a *centroidal* Voronoi mesh, where the generator and the centroid of a cell are very close together (Springel, 2010). This effectively corresponds to a homogeneous particle distribution. This algorithm, known as Lloyd's algorithm, can only be proven to converge in 1D (Lloyd, 1982). However, we also obtained good convergence in 2D and reasonable convergence in 3D during our own experiments with this technique. This is illustrated in Fig. 2.20, which shows the standard deviation on the cell volumes and the average cell size as a function of the number of Lloyd iterations for a uniformly sampled unit box. Both in 2D and 3D, the standard deviation drops significantly during the first ~ 20 iterations, while the average cell size converges to the radius of a sphere with the same volume as the average cell volume (denoted as r_{ideal}). It is interesting to notice that this algorithm very quickly washes out small density perturbations, while it takes a lot more iterations to wash out large scale perturbations. It is therefore ideally suited to wash out Poisson noise in non-homogeneous distributions without affecting the large scale density profile of the distribution too much.

To sample the particles for the 2D distribution used throughout this chapter, we first sampled two uniform homogeneous distributions with respectively 1,000 and 10,000 particles by using rejection sampling (although in the homogeneous

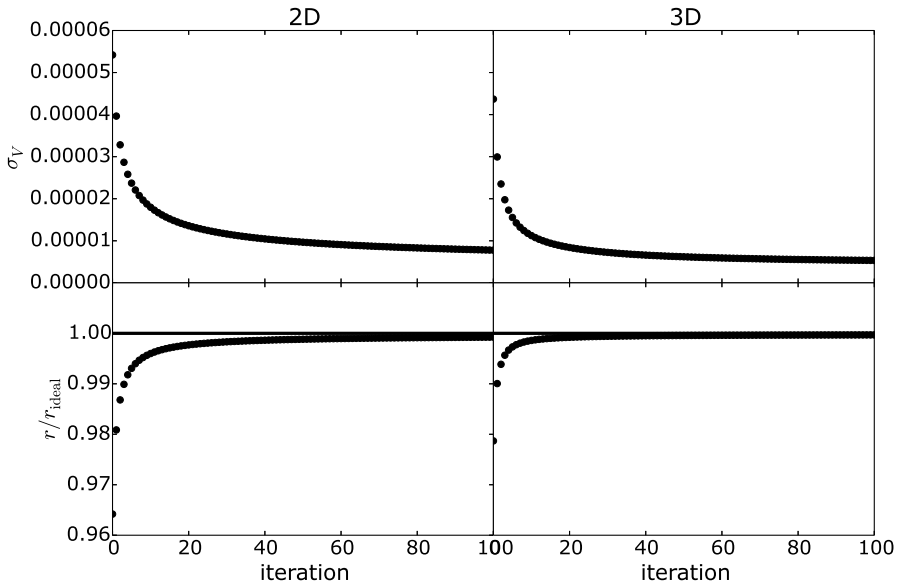


Figure 2.20: *Standard deviation on the average cell size (*top*) and ratio of the average cell size to the radius of a sphere with the average cell volume (*bottom*) as a function of the number of Lloyd iterations. Both the 2D and the 3D test regularize a uniformly sampled distribution in a unit box, using 10,000 particles.*

density case there is no actual rejection and all positions are accepted). These distributions were then relaxed using 1,000 iterations of Lloyd's algorithm. Then, they were combined by cutting out a circular region from the low density distribution, and pasting in a circular region of the same size cut out of the high density distribution. To wash out noise around the interface between high and low density region, we locally applied Lloyd's algorithm for ten more iterations. This finally yielded a distribution that is smooth enough for SPH, and still has the required density contrast.

3

Hydrodynamics

In the previous chapter, we discussed different ways of discretizing space, so that we can represent a fluid (gas, dark matter or even stars) as a discrete set of elements on a computer. In this chapter, we will focus on one particular type of fluid: the interstellar or intergalactic gas. We will first introduce the basic equations of hydrodynamics, and then discuss several ways of solving these on a discrete representation of the gas. At the end of the chapter, we will also discuss the coupling of other physical processes to the equations of hydrodynamics, more specifically gravity, and gas cooling and heating. More advanced models that also take into account the composition of the interstellar gas are discussed in Chapter 5.

Most of the theory on the Euler equations presented in this chapter is based on the excellent book by Toro (2009). We only discuss those aspects that are of direct use to the numerical methods described in this work, and try to sketch the general picture without going into too much detail. The interested reader is encouraged to read Toro (2009) for more information. Descriptions of different hydrodynamical integration techniques come from literature. The results shown in this chapter were obtained using a number of publicly available simulation codes, as well as our own moving-mesh code Shadowfax (Chapter 4) and our own mesh-free implementation in the simulation code swift.

3.1 The Euler equations

3.1.1 Derivation

Macroscopic quantities

A physical gas is a fluid consisting of a huge amount of microscopically small particles. In the case of the interstellar or intergalactic medium, these particles are mostly atoms, possibly ionised. Very cold interstellar gas can also contain molecules and even small dust grains (Draine, 2011). The physical interactions

between and within these particles will govern the large scale behaviour of the fluid. It is however impossible to model these interactions for even a limited number of these particles, let alone for the huge amount of particles that is present in even small parts of the interstellar volume. Classical hydrodynamics therefore resorts to a macroscopic description of these processes, which necessarily includes approximations.

As a first approximation, we will treat the gas as a continuous fluid, rather than as a huge set of discrete particles (Huang, 1987). This can be done by introducing macroscopic *volume averaged* quantities to express the properties of the gas. The first and most obvious of these quantities is the density ρ, which measures the amount of mass M within a certain volume V:

$$\rho = \frac{M}{V}.$$

However, rather than calculating a density for the whole volume under consideration, we are interested in treating the density as a function of space. We would like to define a function that assigns a specific density value to every point in space, so that the total mass of the system is recovered when integrating this function over the entire space. From the perspective of the microscopic particles constituting space, this does not make sense: either the point in space corresponds to a microscopic particle and its local density is non-zero, or it does not and its local density is zero. We can however introduce the concept of a *volume element* $\mathrm{d}V = \mathrm{d}\vec{x}$, which is a very small region around the point with coordinates $\vec{x}$. Provided this volume element is still large enough to contain a reasonable number of microscopic particles, this allows for the consistent definition of a local density, which will be a smooth function of space. The approximation then lies in treating this volume element as if it were an infinitesimal volume element as used in classical calculus. Physically, this only makes sense if the spatial domain is much larger than the length scales on which the microscopic particles interact, which happens to be the case for the interstellar or intergalactic medium.

Once we have a macroscopic density function $\rho(\vec{x})$ defined for every point in space, it is not so hard to use the same sort of reasoning to define a fluid velocity function $\vec{v}(\vec{x})$, which gives the average flow velocity for every point in space. This average flow velocity then corresponds to the average velocity of all microscopic particles within the same volume element $\mathrm{d}\vec{x}$. This notion of flow velocity does not discriminate between flows with different variances in microscopic particle velocities, so that a flow in which the particles are moving with large velocities in all directions is indistinguishable from a flow where all particles move coherently with the average velocity if the average velocity happens to be the same. We will encounter a way to discriminate between these scenarios later on in this chapter.

At this point, it is useful to extend our view from the 3D space to the 6D *phase space*, which is the space of all coordinates, extended with the space of all linear

momenta $\vec{p}$. Every microscopic particle will correspond to a point in the 6D phase space; the first three coordinates give its position in 3D space, while the last three coordinates give its linear momentum. Just as for 3D space, we can also define a volume element $\mathrm{d}\vec{x}\mathrm{d}\vec{p}$ in 6D phase space, which is now occupied by all particles with coordinates within a small volume around the point with coordinates $\vec{x}$, and with linear momenta in a small range around the value $\vec{p}$. Following the same procedure as above, we can now introduce a continuous function $f(\vec{x},\vec{p})$ in 6D phase space, which represents the number density of microscopic particles within a volume element of phase space.

As the microscopic particles are moving, their positions in the phase space will change, so that the number density function $f(\vec{x},\vec{p})$ will also depend on the time t: $f(\vec{x},\vec{p},t)$. By multiplying this function with the phase space volume element, we obtain the number of particles within this phase space volume element at a given time t.

Boltzmann equation

Suppose that the system described above is indeed moving through space. In the absence of external forces and assuming the microscopic particles do not collide, the number of particles within a phase space volume element should stay constant when the system is evolved for a time interval $\mathrm{d}t$:

$$f(\vec{x},\vec{p},t)\mathrm{d}\vec{x}\mathrm{d}\vec{p} = f(\vec{x}+\frac{\vec{p}}{m}\mathrm{d}t,\vec{p},t+\mathrm{d}t)\mathrm{d}\vec{x}\mathrm{d}\vec{p},$$

where m is the mass of a microscopic particle.

We can develop the right hand side of this equation in a Taylor series:

$$f(\vec{x}+\frac{\vec{p}}{m}\mathrm{d}t,\vec{p},t+\mathrm{d}t)\mathrm{d}\vec{x}\mathrm{d}\vec{p} = f(\vec{x},\vec{p},t)\mathrm{d}\vec{x}\mathrm{d}\vec{p} + \vec{\nabla}f(\vec{x},\vec{p},t).\frac{\vec{p}}{m}\mathrm{d}\vec{x}\mathrm{d}\vec{p}\mathrm{d}t + \frac{\partial}{\partial t}f(\vec{x},\vec{p},t)\mathrm{d}\vec{x}\mathrm{d}\vec{p}\mathrm{d}t + O(\mathrm{d}t^2),$$

which immediately gives us the Boltzmann equation:

$$\vec{\nabla}f(\vec{x},\vec{p},t).\frac{\vec{p}}{m} + \frac{\partial}{\partial t}f(\vec{x},\vec{p},t) = 0.$$

We will derive the equations of hydrodynamics by multiplying this equation with so called *moments* of the linear momentum $\vec{p}$, which are just polynomial expressions in the variable $\vec{p}$. The Euler equations can then be retrieved by integrating these equations out over the entire linear momentum space (Huang, 1987). Since we explicitly neglected particle collisions in the above derivation, we should only consider moments that stay constant under particle collisions

when integrating over the entire phase space. The zeroth, first and second order moments m, $\vec{p}$ and $|\vec{p}|^2/m$ satisfy this condition, corresponding to the total mass, linear momentum and kinetic energy (up to a factor 1/2) of the system.

Continuity equation

Multiplying with the zeroth order moment and integrating over linear momentum space yields

$$\int \left[\vec{\nabla} f(\vec{x}, \vec{p}, t).\vec{p} + m \frac{\partial}{\partial t} f(\vec{x}, \vec{p}, t) \right] \mathrm{d}\vec{p} = 0.$$

The second term within the integral sign can be rewritten easily by realising that the particle mass m and the linear momentum integral both are independent of time, so that the partial time derivative can be brought outside of the integral sign. What is left after integration is then nothing else than the number density of particles within a volume element in 3D position space, multiplied with there respective masses. This is what we defined to be the density $\rho(\vec{x}, t)$.

More specifically, if

$$\rho(\vec{x}, t) = \int m f(\vec{x}, \vec{p}, t) \mathrm{d}\vec{p},$$

then we can define the average value of the quantity $X(\vec{x}, \vec{p}, t)$ to be

$$\langle X \rangle (\vec{x}, t) = \frac{1}{\rho(\vec{x}, t)} \int X(\vec{x}, \vec{p}, t) m f(\vec{x}, \vec{p}, t) \mathrm{d}\vec{p}.$$

For the first term, it is also possible to bring the gradient outside of the integral sign, since it only affects the spatial coordinate $\vec{x}$. We then are left with

$$\int m f(\vec{x}, \vec{p}, t) \frac{\vec{p}}{m} \mathrm{d}\vec{p} = \int m f(\vec{x}, \vec{p}, t) \vec{v}(\vec{x}, \vec{p}, t) \mathrm{d}\vec{p}$$

Using the definition of the average of a quantity, we rewrite this as

$$\rho(\vec{x}, t) \langle \vec{v} \rangle (\vec{x}, t)$$

Putting both terms together again, we get the continuity equation:

$$\frac{\partial}{\partial t} \rho(\vec{x}, t) + \vec{\nabla}. \left(\rho(\vec{x}, t) \langle \vec{v} \rangle (\vec{x}, t) \right) = 0.$$

This equation expresses the constancy of mass, since it links all transport of mass and consequent local density change to the spatial movement of the corresponding mass elements.

Momentum equation

For the first order moment, we get

$$\int \left[\vec{p}\left(\vec{\nabla} f(\vec{x},\vec{p},t).\frac{\vec{p}}{m}\right) + \vec{p}\frac{\partial}{\partial t} f(\vec{x},\vec{p},t)\right] \mathrm{d}\vec{p} = 0.$$

The second term of this equation is again easily rewritten by bringing the time derivative outside of the integral sign. What is left is exactly the same term involving the average linear momentum density we encountered before.

The first term is somewhat more involved and we will tackle it on a component basis:

$$\int p_i \left(\vec{\nabla} f(\vec{x},\vec{p},t).\frac{\vec{p}}{m}\right) \mathrm{d}\vec{p} = \vec{\nabla}. \int v_i(\vec{x},\vec{p},t) m f(\vec{x},\vec{p},t) \vec{v}(\vec{x},\vec{p},t) \mathrm{d}\vec{p} = \vec{\nabla}. \left(\rho(\vec{x},t)\langle v_i \vec{v}\rangle(\vec{x},t)\right),$$

where we introduced the components v_i and p_i $(i = 0, 1, 2)$ of $\vec{v}$ and $\vec{p}$ and again made use of the definition of the average of a quantity.

To give more meaning to this term, we will split up the particle velocity in two parts: a part corresponding to the average particle velocity and an extra part, corresponding to the random movement of the particle with respect to this average flow velocity, $\vec{v}(\vec{x},\vec{p},t) = \langle\vec{v}\rangle(\vec{x},t) + \vec{v}_{\mathrm{rand}}(\vec{x},\vec{p},t)$. This will allow us to discriminate between the two scenarios mentioned before in which the average flow velocity is the same, but the individual particle velocities are very different. It trivially follows that

$$\begin{aligned}\langle v_i(\vec{x},\vec{p},t)\vec{v}(\vec{x},\vec{p},t)\rangle = {} & \langle v_i\rangle(\vec{x},t)\langle\vec{v}\rangle(\vec{x},t) + \langle v_{i,\mathrm{rand}}\rangle(\vec{x},t)\langle\vec{v}\rangle(\vec{x},t) \\ & + \langle v_i\rangle(\vec{x},t)\langle\vec{v}_{\mathrm{rand}}\rangle(\vec{x},t) + \langle v_{i,\mathrm{rand}}\vec{v}_{\mathrm{rand}}\rangle(\vec{x},t).\end{aligned}$$

The second and third term in this equation are zero, since the average of the random velocities is zero. The last term describes the flow of the ith component of the velocity. If we multiply it with the density, we end up with a quantity describing a net force per unit area. We therefore introduce the *pressure tensor*:

$$\overrightarrow{\overrightarrow{P}}(\vec{x},t) = \rho(\vec{x},t)\langle\vec{v}_{\mathrm{rand}}\vec{v}_{\mathrm{rand}}\rangle(\vec{x},t).$$

This makes physically sense, since a flow with large random velocities of the microscopic particles on top of the average flow velocity will have a much higher temperature and pressure.

At this point, we will make a second approximation and assume an *isotropic* medium, in which the pressure is independent of the direction. This means the

components of the pressure tensor are given by $P_{ij} = P\delta_{ij}$, with δ_{ij} the Kronecker delta, which is 1 if i and j are equal and 0 otherwise, and $P(\vec{x},t)$ the local pressure of the fluid.

Putting everything together, we end up with the momentum equation:

$$\vec{\nabla}.\left(\rho(\vec{x},t)\langle\vec{v}\rangle(\vec{x},t)\langle\vec{v}\rangle(\vec{x},t)\right) + \vec{\nabla}P(\vec{x},t) + \frac{\partial}{\partial t}\left(\rho(\vec{x},t)\langle\vec{v}\rangle(\vec{x},t)\right) = 0.$$

This equation links the local change of the linear momentum density to the transport of linear momentum due to the average flow and the extra momentum transfer due to the internal movement of the microscopic particles within the volume element.

Energy equation

The second order moment gives

$$\int \left[\frac{|\vec{p}|^2}{m}\left(\vec{\nabla}f(\vec{x},\vec{p},t).\frac{\vec{p}}{m}\right) + \frac{|\vec{p}|^2}{m}\frac{\partial}{\partial t}f(\vec{x},\vec{p},t)\right]\mathrm{d}\vec{p} = 0.$$

After having brought the time derivative outside of the integral sign as usual, the second term can be rewritten as

$$\int \frac{|\vec{p}|^2}{m}\frac{\partial}{\partial t}f(\vec{x},\vec{p},t)\mathrm{d}\vec{p} = \int |\vec{v}(\vec{x},\vec{p},t)|^2 m f(\vec{x},\vec{p},t)\mathrm{d}\vec{p} = \rho(\vec{x},t)\langle|\vec{v}|^2\rangle(\vec{x},t).$$

Again splitting up the particle velocity in an average and a random part, we obtain:

$$\langle|\vec{v}(\vec{x},\vec{p},t)|^2\rangle = |\langle\vec{v}\rangle(\vec{x},t)|^2 + 2\langle\vec{v}_{\mathrm{rand}}\rangle(\vec{x},t).\langle\vec{v}\rangle(\vec{x},t) + \langle|\vec{v}_{\mathrm{rand}}|^2\rangle(\vec{x},t).$$

The second term vanishes since the average random velocity is zero. The last term can be interpreted as the average energy per unit mass $u(\vec{x},t) = \frac{1}{2}\langle|\vec{v}_{\mathrm{rand}}|^2\rangle(\vec{x},t)$, which again expresses the fact that a flow with large random velocities will be physically different from a flow where all particles move with the average flow velocity. It has to be noted that by adopting this definition of the energy per unit mass, we automatically fixed the *equation of state* of the gas, since

$$\langle|\vec{v}_{\mathrm{rand}}|^2\rangle(\vec{x},t) = \sum_i \langle v_{i,\mathrm{rand}} v_{i,\mathrm{rand}}\rangle(\vec{x},t) = \frac{1}{\rho(\vec{x},t)}\sum_i P_{ii}(\vec{x},t) = \frac{3P(\vec{x},t)}{\rho(\vec{x},t)},$$

which means $P(\vec{x},t) = \frac{2}{3}\rho(\vec{x},t)u(\vec{x},t)$.

It is common practice to assume a somewhat more general equation of state of the form

$$P(\vec{x},t) = (\gamma-1)\rho(\vec{x},t)u(\vec{x},t),$$

so that a more general definition of the thermal energy, which takes other forms of internal energy into account apart from the kinetic degrees of freedom, reads

$$u(\vec{x},t) = \frac{2}{3}\frac{\langle|\vec{v}_{\text{rand}}|^2\rangle(\vec{x},t)}{(\gamma-1)}.$$

The first definition is then recovered for the specific case of a monatomic gas, for which $\gamma = \frac{5}{3}$. For simplicity, we will assume a monatomic gas for now.

The second term in the integral then reads

$$\frac{\partial}{\partial t}\left(\rho(\vec{x},t)\langle|\vec{v}|^2\rangle(\vec{x},t)\right) = \frac{\partial}{\partial t}\left(\rho(\vec{x},t)|\langle\vec{v}\rangle|^2(\vec{x},t) + 2\rho(\vec{x},t)u(\vec{x},t)\right) = 2\frac{\partial}{\partial t}\left(\rho(\vec{x},t)e(\vec{x},t)\right),$$

where we introduced the *total energy per unit mass* $e(\vec{x},t)$, which is the sum of the internal and kinetic energy per unit mass.

The first term of the integral yields (after bringing the gradient outside of the integral)

$$\int \frac{|\vec{p}|^2}{m}\left(\vec{\nabla}f(\vec{x},\vec{p},t).\frac{\vec{p}}{m}\right)\mathrm{d}\vec{p} = \int |\vec{v}(\vec{x},\vec{p},t)|^2 m f(\vec{x},\vec{p},t)\vec{v}(\vec{x},\vec{p},t)\mathrm{d}\vec{p} = \rho(\vec{x},t)\langle|\vec{v}|^2\vec{v}\rangle(\vec{x},t).$$

Working out the averaged triple product is preferably done using component notation:

$$\begin{aligned}
\langle|\vec{v}|^2 v_j\rangle(\vec{x},t) &= \sum_i \langle v_i v_i v_j\rangle(\vec{x},t) \\
&= \sum_i \Big(\langle v_i\rangle^2(\vec{x},t)\langle v_j\rangle(\vec{x},t) + \frac{2}{\rho(\vec{x},t)}\langle v_i\rangle(\vec{x},t)P_{ij} \\
&\qquad + \langle v_{i,\text{rand}}\rangle^2(\vec{x},t)\langle v_j\rangle(\vec{x},t) + \langle v_{i,\text{rand}}^2 v_{j,\text{rand}}\rangle(\vec{x},t)\Big) \\
&= |\langle\vec{v}\rangle(\vec{x},t)|^2\langle v_j\rangle(\vec{x},t) + \frac{2}{\rho(\vec{x},t)}P\langle v_j\rangle \\
&\qquad + 2u(\vec{x},t)\langle v_j\rangle(\vec{x},t) + \langle|\vec{v}_{\text{rand}}|^2 v_{j,\text{rand}}\rangle(\vec{x},t).
\end{aligned}$$

The last term expresses the transport of thermal energy by means of the random velocities of the particles. It can be seen as a measurement of the heat conductivity of the flow and we will define the heat conductivity vector as

$$\vec{h}(\vec{x},t) = \frac{1}{2}\rho(\vec{x},t)\langle|\vec{v}_{\text{rand}}|^2\vec{v}_{\text{rand}}\rangle(\vec{x},t).$$

As the heat conductivity is an anti-symmetric function of $\vec{v}_{\text{rand}}$, the average of this quantity is zero for an isotropic medium, so that this term vanishes.

Putting everything together, we finally obtain the energy equation:

$$\frac{\partial}{\partial t}\left(\rho(\vec{x},t)e(\vec{x},t)\right) + \vec{\nabla}.\left(\rho(\vec{x},t)e(\vec{x},t)\langle\vec{v}\rangle(\vec{x},t) + P(\vec{x},t)\langle\vec{v}\rangle(\vec{x},t)\right) = 0.$$

Summary

From now on, we will regard the macroscopic quantities $\rho(\vec{x},t)$, $\langle\vec{v}\rangle(\vec{x},t)$, $P(\vec{x},t)$ and $e(\vec{x},t)$ (and the associated $u(\vec{x},t)$) as hydrodynamical variables rather than functions, and we will drop the function parameters and average signs to simplify the notations. The three equations derived above then become

$$\begin{aligned}
\frac{\partial\rho}{\partial t} + \vec{\nabla}.\left(\rho\vec{v}\right) &= 0 \\
\frac{\partial\left(\rho\vec{v}\right)}{\partial t} + \vec{\nabla}.\left(\rho\vec{v}\vec{v}\right) + \vec{\nabla}P &= 0 \\
\frac{\partial}{\partial t}\left(\rho e\right) + \vec{\nabla}.\left(\rho e\vec{v} + P\vec{v}\right) &= 0.
\end{aligned}$$

We now have five equations for six hydrodynamic variables (e and u counting as one variable). To be able to solve this system of equations, we hence need an extra equation that links two of the six remaining variables without introducing new variables. This will be the same equation of state we already encountered, which links the thermal energy u to the pressure P:

$$P = (\gamma - 1)\rho u,$$

where we will now assume the more general form described above. It turns out that the equations derived using a monatomic gas are also valid for a gas with a different adiabatic index, as long as the equation of state retains this more general form.

It is also instructive to introduce a temperature T, as this makes it easier to interpret the thermal energy physically. The thermal energy and the temperature are related through

$$u = \frac{1}{(\gamma-1)}\frac{kT}{m},$$

where m is the generic microscopic particle mass we encountered before. k is the Boltzmann constant, $k = 1.38064852 \times 10^{-23}$ J/K. The equation of state can similarly be rewritten as

$$P = \frac{\rho}{m}kT.$$

Finally, it is also instructive to rewrite the Euler equations in the compact form (Toro, 2009; Springel, 2010)

$$\frac{\partial U}{\partial t} + \vec{\nabla}.\vec{F} = 0,$$

with

$$U = \begin{pmatrix} \rho \\ \rho\vec{v} \\ \rho e \end{pmatrix}$$

and

$$\vec{F} = \begin{pmatrix} \rho\vec{v} \\ \rho\vec{v}\vec{v} + P\overrightarrow{1} \\ (\rho e + P)\,\vec{v} \end{pmatrix},$$

and where $\overrightarrow{1}$ represents the unit tensor. This form of the equations will be particularly helpful when introducing finite volume methods.

Related equations

For the hydrodynamical integration methods we will introduce later in the chapter, we will also use different forms of the equations above. We start by rewriting the momentum equation by using the continuity equation:

$$\vec{v}\frac{\partial \rho}{\partial t} + \rho\frac{\partial \vec{v}}{\partial t} + \rho\vec{v}.\vec{\nabla}\left(\vec{v}\right) + \vec{\nabla}.\left(\rho\vec{v}\right)\vec{v} + \vec{\nabla}P = \rho\frac{\partial \vec{v}}{\partial t} + \rho\vec{v}.\vec{\nabla}\left(\vec{v}\right) + \vec{\nabla}P = 0 \quad (3.1)$$

Using the definition of the total energy per unit mass,

$$e = \frac{1}{2}|\vec{v}|^2 + u,$$

we can rewrite the energy equation in terms of the thermal energy:

$$\begin{aligned}
\frac{\partial}{\partial t}\left(\rho u + \frac{1}{2}\rho|\vec{v}|^2\right) + \vec{\nabla}.\left(\rho u\vec{v} + \frac{1}{2}\rho|\vec{v}|^2\vec{v} + P\vec{v}\right) = \\
\frac{\partial}{\partial t}\left(\rho u\right) + \frac{1}{2}\rho\vec{v}.\frac{\partial}{\partial t}\left(\vec{v}\right) + \frac{1}{2}\vec{v}.\frac{\partial}{\partial t}\left(\rho\vec{v}\right) + \vec{\nabla}.\left(\rho u\vec{v}\right) \\
+ \frac{1}{2}\rho\vec{v}.\vec{\nabla}\vec{v}.\vec{v} + \frac{1}{2}\vec{\nabla}.\left(\rho\vec{v}\vec{v}\right).\vec{v} + P\vec{\nabla}.\vec{v} + \vec{\nabla}P.\vec{v} = \\
\frac{\partial}{\partial t}\left(\rho u\right) + \frac{1}{2}\vec{v}.\frac{\partial}{\partial t}\left(\rho\vec{v}\right) + \vec{\nabla}.\left(\rho u\vec{v}\right) \\
+ \frac{1}{2}\vec{\nabla}.\left(\rho\vec{v}\vec{v}\right).\vec{v} + P\vec{\nabla}.\vec{v} + \frac{1}{2}\vec{\nabla}P.\vec{v} = \\
\frac{\partial}{\partial t}\left(\rho u\right) + \vec{\nabla}.\left(\rho u\vec{v}\right) + P\vec{\nabla}.\vec{v} = 0,
\end{aligned}$$

where for the first step we used the momentum equation, and for the second step we used both the continuity equation and the momentum equation. This can be simplified to read

$$\rho\frac{\partial u}{\partial t} + u\frac{\partial \rho}{\partial t} + u\vec{\nabla}.(\rho\vec{v}) + \rho\vec{v}.\vec{\nabla}u + P\vec{\nabla}.\vec{v} = \rho\frac{\partial u}{\partial t} + \rho\vec{v}.\vec{\nabla}u + P\vec{\nabla}.\vec{v} = 0. \quad (3.2)$$

The equations above give the change of velocity and thermal energy directly, rather than in the conserved form and they will be used e.g. for the formulation of the Smoothed Particle Hydrodynamics (SPH) technique.

In this light, it is also useful to introduce an *entropic function* (Springel, 2005)

$$A = \frac{P}{\rho^\gamma},$$

so that

$$u = \frac{A}{\gamma - 1}\rho^{\gamma-1}.$$

If we plug this into the energy equation, we get (Springel & Hernquist, 2002)

$$\rho\frac{\partial}{\partial t}\left(\frac{A}{\gamma-1}\rho^{\gamma-1}\right) + \rho\vec{v}.\vec{\nabla}\left(\frac{A}{\gamma-1}\rho^{\gamma-1}\right) + P\vec{\nabla}.\vec{v} =$$
$$\frac{\rho^\gamma}{\gamma-1}\frac{\partial A}{\partial t} + A\rho^{\gamma-1}\frac{\partial \rho}{\partial t} + \frac{\rho^\gamma\vec{v}}{\gamma-1}.\vec{\nabla}A + A\rho^{\gamma-1}\vec{v}.\vec{\nabla}\rho + A\rho^\gamma\vec{\nabla}.\vec{v} =$$
$$\frac{\rho^\gamma}{\gamma-1}\frac{\partial A}{\partial t} + \frac{\rho^\gamma\vec{v}}{\gamma-1}.\vec{\nabla}A = 0.$$

If we define the *co-moving time derivative* $\frac{\mathrm{d}}{\mathrm{d}t} = \frac{\partial}{\partial t} + \vec{v}.\vec{\nabla}$, we see that this effectively means A is a conserved quantity. Since entropy is conserved in the absence of discontinuities, the name entropic function is hence justified. Note that A is not the entropy itself, but a monotonic function of it.

3.1.2 Physical solutions

Before trying to solve the Euler equations in a numerical integration scheme, it is instructive to first explore the possible physical solutions to this problem. This will not only allow us to introduce the Riemann problem, which will be of great interest for some numerical integration schemes, but it will also give us the necessary background to understand why some numerical integration schemes fail for specific hydrodynamical problems.

Characteristic waves

Let us start by considering the very basic partial differential equation

$$\frac{\partial y}{\partial t} + a\frac{\partial y}{\partial x} = 0,$$

with a a constant, and where $y(x,t)$ is a general two dimensional function of a position coordinate x and time coordinate t. A characteristic $y'(t) = y(x(t),t)$ corresponds to the one dimensional solution of this equation along a characteristic curve $x(t)$ in two dimensional space (consisting of time and a single spatial coordinate). This characteristic curve has the specific property

$$\frac{\mathrm{d}x}{\mathrm{d}t} = a,$$

so that

$$\frac{\mathrm{d}y'}{\mathrm{d}t} = \frac{\partial y}{\partial t} + a\frac{\partial y}{\partial x} = 0.$$

Suppose we have an initial condition $y(x,0) = y_0(x)$. The condition $\frac{\mathrm{d}y'}{\mathrm{d}t} = 0$ then implies that $y'(t)$ stays constant along a characteristic curve with general equation

$$x(t) = x_0 + at.$$

In other words,

$$y(x,t) = y_0(x_0) = y_0(x - at).$$

The solution to the general partial differential equation is hence reduced to a linear translation of the initial condition with the characteristic speed.

Linear systems of partial differential equations

Now consider the system of linear partial differential equations below:

$$\frac{\partial Y}{\partial t} + A\frac{\partial Y}{\partial x} = 0,$$

where Y now is a m-dimensional vector and A is a $m \times m$ matrix.

We will call this system of equations hyperbolic if the matrix A has m real eigenvalues and m corresponding eigenvectors. It follows that A can be written in diagonal form

$$A = K\Lambda K^{-1},$$

where the columns of the matrix K are the right eigenvectors K^i:

$$AK^i = \lambda_i K^i$$

with λ_i the corresponding eigenvalues, that are the diagonal elements of the diagonal matrix Λ.

We can introduce the new vector of *characteristic* variables Z:

$$Z = K^{-1}Y,$$

which allows us to rewrite the system of equations as

$$K\frac{\partial Z}{\partial t} + AK\frac{\partial Z}{\partial x} = 0$$

or

$$\frac{\partial Z}{\partial t} + \Lambda\frac{\partial Z}{\partial x} = 0.$$

This last equation corresponds to the much simpler system of m decoupled partial differential equations of the form

$$\frac{\partial Z_i}{\partial t} + \lambda_i\frac{\partial Z_i}{\partial x} = 0,$$

with $i = 1...m$.

Given an initial value $Y_0(x)$ for the m variables Y, it is hence possible to obtain the solution $Y(x,t)$ at a later time t, by diagonalizing the system of equations. Once we have the matrix K, it is straightforward to convert $Y_0(x)$ to $Z_0(x)$. The m solutions $Z_i(x,t)$ are given by the characteristic speeds λ_i. It is again straightforward to convert these solutions back to the required solutions $Y(x,t)$.

Conservation laws

As we have seen, the Euler equations are not linear and have the more general form

$$\frac{\partial U}{\partial t} + \frac{\partial F(U)}{\partial x} = 0,$$

where we only consider the one dimensional Euler equations for now. Or we could just think of this system as a system of m *conservation laws*: non linear partial differential equations of the specific form above.

We can calculate the Jacobian matrix of this system, given by

$$A(U) = \frac{\partial F}{\partial U}.$$

The system of conservation laws is said to be hyperbolic if this matrix has real eigenvalues $\lambda_i(U)$ and m linearly independent eigenvectors $K^i(U)$. It might be clear by now that the general solution of a conservation law will depend upon a diagonalization of this Jacobian matrix and a more detailed study of the resulting m decoupled partial differential equations, although the actual solution will be a lot more involved. It will be very instructive however to keep this idea in mind and explore some of its consequences.

Wave solutions for conservation laws

The single conservation law

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0$$

can be rewritten as

$$\frac{\partial u}{\partial t} + \lambda(u)\frac{\partial u}{\partial x} = 0,$$

where $\lambda(u) = \frac{\mathrm{d}f(u)}{\mathrm{d}u}$.

By introducing a characteristic curve $x(t)$ for which

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \lambda(u),$$

we still find that the solution $u'(t) = u(x(t), t)$ is constant along this characteristic curve. The characteristic speed $\lambda(u)$ however is now also function of the solution u itself, so that this does no longer allow us to solve the partial differential equation.

In the case where $\lambda(u)$ is constant, we found that the general solution is just a linear translation of the initial condition. If $\lambda(u)$ is not constant, distortions will be produced during the translation of the initial condition that will affect the general solution. Some characteristics will move faster than others, eventually leading to a flattening of the initial condition profile in regions where characteristics move away from each other (expansive regions) and a steepening of the profile where characteristics move towards each other (compressive region). The latter effect, also called wave steepening, will eventually lead to crossing of characteristics and wave breaking. At this point, a discontinuous solution u is possible.

In expansive regions, only continuous solutions of u make physical sense. The waves corresponding to these solutions are called *rarefaction waves* and are completely described by the conservation law.

A discontinuity of a variable u in a region of excessive wave steepening is what we will call a *shock wave*. Since the solution u is no longer continuous, we can also no longer use the conservation law to solve for u. It is however possible to still learn something about the solution by looking at the integral form of the conservation law.

Provided the flux function $f(u)$ is continuous in a range $[x_\mathrm{L}, x_\mathrm{R}]$, we can integrate the conservation law over this entire range:

$$\int_{x_\mathrm{L}}^{x_\mathrm{R}} \frac{\partial u}{\partial t}\mathrm{d}x = f(u(x_\mathrm{L}, t)) - f(u(x_\mathrm{R}, t)).$$

If we choose the range so that it is independent of time, we can also bring the partial time derivative outside of the integral sign and convert it to a total time derivative. We will now enforce this integral form, even for the case where u is discontinuous at a point $s(t)$:

$$f(u(x_{\mathrm{L}},t)) - f(u(x_{\mathrm{R}},t)) = \frac{\mathrm{d}}{\mathrm{d}t}\int_{x_{\mathrm{L}}}^{s(t)} u(x,t)\mathrm{d}x + \frac{\mathrm{d}}{\mathrm{d}t}\int_{s(t)}^{x_{\mathrm{R}}} u(x,t)\mathrm{d}x.$$

The integrals on the right hand side can be expanded using the general formula (Flanders, 1973)

$$\frac{\mathrm{d}}{\mathrm{d}y}\int_{x_1(y)}^{x_2(y)} f(x,y)\mathrm{d}x = \int_{x_1(y)}^{x_2(y)} \frac{\partial f}{\partial y}\mathrm{d}x + f(x_2,y)\frac{\mathrm{d}x_2}{\mathrm{d}y} - f(x_1,y)\frac{\mathrm{d}x_1}{\mathrm{d}y},$$

to yield

$$f(u(x_{\mathrm{L}},t)) - f(u(x_{\mathrm{R}},t)) = (u(s_{\mathrm{L}},t) - u(s_{\mathrm{R}}))S + \int_{x_{\mathrm{L}}}^{s(t)} \frac{\partial u}{\partial t}\mathrm{d}x + \int_{s(t)}^{x_{\mathrm{R}}} \frac{\partial u}{\partial t}\mathrm{d}x,$$

where $u(s_{\mathrm{L}},t)$ and $u(s_{\mathrm{R}},t)$ are the limits of u when approaching the discontinuity from the left and from the right respectively. $S = \frac{\mathrm{d}s}{\mathrm{d}t}$ is the speed of the discontinuity. If we put $x_{\mathrm{L}} = s(t) - \varepsilon$ and $x_{\mathrm{R}} = s(t) + \varepsilon$ ($\varepsilon > 0$) and take the limit $\varepsilon \to 0$, the two integrals vanish, since their integrand should be bounded for physical reasons.

We are left with the following condition, linking the jump $\Delta u = u(s_{\mathrm{R}},t) - u(s_{\mathrm{L}},t)$ across the discontinuity to the jump in the fluxes $\Delta f = f(u(s_{\mathrm{R}},t)) - f(u(s_{\mathrm{L}},t))$:

$$\Delta f = S\Delta u.$$

This condition is called the Rankine-Hugoniot condition (after Rankine (1870) and Hugoniot (1887)) and we will use it to solve the Euler equations across discontinuities.

Wave solutions for the Euler equations

For the specific case of the 1D Euler equations, the Jacobian matrix is given by

$$A(U) = \begin{pmatrix} 0 & 1 & 0 \\ -\frac{1}{2}(\gamma-3)v^2 & (3-\gamma)v & \gamma-1 \\ -\gamma v e + (\gamma-1)v^3 & \gamma e - \frac{3}{2}(\gamma-1)v^2 & \gamma v \end{pmatrix}.$$

This matrix can be diagonalized to read

$$\Lambda(U) = \begin{pmatrix} v - c_s & 0 & 0 \\ 0 & v & 0 \\ 0 & 0 & v + c_s \end{pmatrix},$$

where we have introduced the *sound speed* $c_s = \sqrt{\frac{\gamma P}{\rho}}$, and the corresponding eigenvectors are given by the columns of

$$K(U) = \begin{pmatrix} 1 & 1 & 1 \\ v - c_s & v & u + v_s \\ e + \frac{P}{\rho} - vc_s & \frac{1}{2}v^2 & e + \frac{P}{\rho} + vc_s \end{pmatrix}.$$

We will hence always have three wave components for the solution of the 1D Euler equations in some region of space: a component that moves with the local fluid velocity, and two components that move with the sound speed relative to this component. We will call these components the *left*, *middle* and *right* waves, as they will always occur in this specific spatial order.

Across the waves, we have a set of ordinary differential equations, also known as *Generalised Riemann invariants* (Whitham, 2011). For the left wave, these are

$$\frac{\mathrm{d}\rho}{1} = \frac{\mathrm{d}(\rho v)}{v - c_s} = \frac{\mathrm{d}(\rho e)}{e + \frac{P}{\rho} - vc_s}.$$

For the middle wave, we have

$$\frac{\mathrm{d}\rho}{1} = \frac{\mathrm{d}(\rho v)}{v} = \frac{\mathrm{d}(\rho e)}{\frac{1}{2}v^2},$$

and for the right wave

$$\frac{\mathrm{d}\rho}{1} = \frac{\mathrm{d}(\rho v)}{v + c_s} = \frac{\mathrm{d}(\rho e)}{e + \frac{P}{\rho} + vc_s}.$$

For the middle wave, we immediately find

$$v\mathrm{d}\rho = v\mathrm{d}\rho + \rho\mathrm{d}v,$$

which means that v is constant across the middle wave. Using this, we also find

$$\mathrm{d}e = \mathrm{d}\left(\frac{1}{2}v^2 + \frac{P}{(\gamma-1)\rho}\right) = \frac{1}{(\gamma-1)\rho}\mathrm{d}P - \frac{P}{(\gamma-1)\rho^2}\mathrm{d}\rho,$$

which we can use to work out the second Riemann invariant for the middle wave:

$$\frac{1}{2}v^2\mathrm{d}\rho = e\mathrm{d}\rho + \rho\mathrm{d}e = \left(\frac{1}{2}v^2 + \frac{P}{(\gamma-1)\rho}\right)\mathrm{d}\rho + \frac{1}{(\gamma-1)}\mathrm{d}P - \frac{P}{(\gamma-1)\rho}\mathrm{d}\rho,$$

which learns us that P is also constant across the middle wave. Since both v and P do not change across the middle wave, the only change allowed is a change in

ρ. We will therefore call the physical solution associated to the middle wave a *contact discontinuity*.

The Riemann invariants for the left and right wave cannot be manipulated to yield similar relations. The first Riemann invariant even reduces to

$$v\mathrm{d}\rho \pm c_s\mathrm{d}\rho = v\mathrm{d}\rho + \rho\mathrm{d}v,$$

which means any change in ρ across the left or right wave will always lead to a change in v across the same wave for a non-trivial sound speed. The left and right waves hence cannot correspond to a contact discontinuity.

There are two options left for these waves, corresponding to the two wave structures introduced above: a shock wave and a rarefaction wave. The former corresponds to a discontinuity in all three hydrodynamical quantities, while for the second all three quantities are continuous. As indicated above, the Euler equations themselves cannot be used across a shock wave (nor can the derived Generalised Riemann Invariants), so that we will have to use the Rankine-Hugoniot conditions in this case to link the quantities to the left and to the right of the shock wave.

Primitive variables

Across a discontinuity, the variables v and P are constant. It hence makes sense to consider the hydrodynamical equations with these quantities as variables, instead of the quantities ρv and ρe. We already rewrote the momentum and energy equations accordingly, and the same can be done with the continuity equation, to yield the following system of equations (again in 1D):

$$\begin{aligned} \frac{\partial \rho}{\partial t} + v\frac{\partial \rho}{\partial x} + \rho\frac{\partial v}{\partial x} &= 0 \\ \frac{\partial v}{\partial t} + v\frac{\partial v}{\partial x} + \frac{1}{\rho}\frac{\partial P}{\partial x} &= 0 \\ \frac{\partial P}{\partial t} + \gamma P\frac{\partial v}{\partial x} + v\frac{\partial P}{\partial x} &= 0. \end{aligned} \tag{3.3}$$

This system has the general form

$$\frac{\partial W}{\partial t} + A(W)\frac{\partial W}{\partial x} = 0,$$

with

$$A(W) = \begin{pmatrix} v & \rho & 0 \\ 0 & v & \frac{1}{\rho} \\ 0 & \gamma P & v \end{pmatrix}.$$

The variables $W = \begin{pmatrix} \rho \\ v \\ P \end{pmatrix}$ are called the *primitive variables*.

The matrix $A(W)$ (unsurprisingly) has the same eigenvalues as the Jacobian matrix for the conservative Euler equations, but now the matrix of eigenvectors reads

$$K(W) = \begin{pmatrix} 1 & 1 & 1 \\ -\frac{c_s}{\rho} & 0 & \frac{c_s}{\rho} \\ c_s^2 & 0 & c_s^2 \end{pmatrix}.$$

The second eigenvector immediately expresses the constancy of v and P across the middle wave.

If we replace the energy equation by the entropy equation (using the entropic function A introduced above), we can use ρ, v and A as primitive variables. The matrix $A(W)$ then reads

$$A(W) = \begin{pmatrix} v & \rho & 0 \\ \frac{c_s^2}{\rho} & v & \frac{1}{\rho}\frac{\partial P}{\partial A} \\ 0 & 0 & v \end{pmatrix}.$$

The eigenvalues of this matrix are again the same as above, with

$$K(W) = \begin{pmatrix} 1 & -\frac{\partial P}{\partial A} & 1 \\ -\frac{c_s}{\rho} & 0 & \frac{c_s}{\rho} \\ 0 & c_s^2 & 0 \end{pmatrix}.$$

In the case where the primitive variables change continuously, $\frac{\partial P}{\partial A} = \rho^\gamma$, and the Generalised Riemann Invariants for the left and right (rarefaction) wave immediately show that A is constant across these waves.

Using this information, we can also deduce

$$\pm c_s \mathrm{d}\rho = \pm\sqrt{\gamma\rho^{\gamma-1}A}\mathrm{d}\rho = \rho \mathrm{d}v,$$

or

$$\pm\sqrt{\gamma\rho^{\gamma-1}A}\mathrm{d}\rho - \rho\mathrm{d}v = 0.$$

Since A is constant, we can integrate this:

$$\pm\int \sqrt{\gamma A}\rho^{\frac{\gamma-3}{2}}\mathrm{d}\rho - v = \text{constant},$$

which learns us that the following quantities are also constant across the left and right rarefaction wave:

$$v \pm \frac{2c_s}{\gamma - 1}.$$

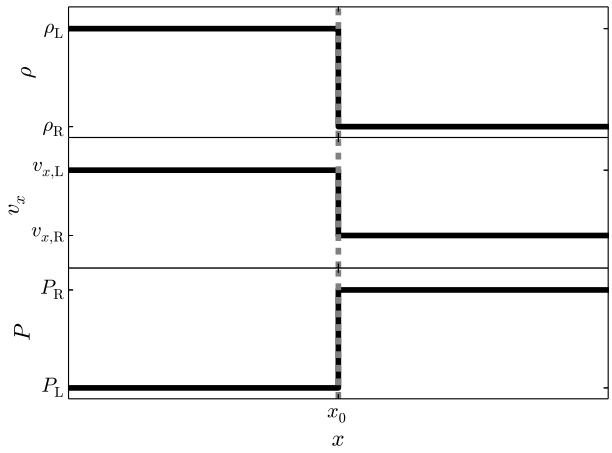


Figure 3.1: *The general 1D Riemann problem. The primitive variables at the left and the right of $x = x_0$ are given at time $t = 0$. We want to know what the values of the primitive variables are at an arbitrary later time t.*

3.1.3 The Riemann problem

Theory

Suppose we have a two state hydrodynamical setup as depicted in Fig. 3.1. At some time $t = 0$, the hydrodynamic variables $W = (\rho, \vec{v}, P)$ are given by

$$W(\vec{x}) = \begin{cases} W_\mathrm{L} & x < x_0 \\ W_\mathrm{R} & x > x_0, \end{cases}$$

where W_L and W_R represent constant vectors of primitive variables.

The Riemann problem is the general problem of finding the hydrodynamical solution for the primitive variables at position $x = x_0$ and at all later times $t > 0$. We will start by solving the 1D Riemann problem and then generalise this solution to the 3D case.

From the general discussion on physical solutions of the Euler equations, we already know that the solution will consists of four different regions, separated by the three elementary wave characteristics. We also know that the middle characteristic wave will always be a contact discontinuity, across which v and P are constant. The left and right wave can either be a continuous rarefaction wave, or a discontinuous shock wave.

Using this knowledge, we can connect the density, velocity and pressure at the left side with that on the right side, using the Rankine-Hugoniot conditions and Riemann invariants discussed above. This leads to the following equation for the pressure P_* and the velocity v_* associated with the middle wave (Toro, 2009):

$$\begin{aligned} f(P_*, W_\mathrm{L}, W_\mathrm{R}) &= 0 \\ v_* &= \frac{1}{2}(v_\mathrm{L} + v_\mathrm{R}) + \frac{1}{2}(f_\mathrm{R}(P_*, W_\mathrm{R}) - f_\mathrm{L}(P_*, W_\mathrm{L})), \end{aligned}$$

where $f(P, W_\mathrm{L}, W_\mathrm{R})$ is given by

$$\begin{aligned} f(P, W_\mathrm{L}, W_\mathrm{R}) &= f_\mathrm{L}(P, W_\mathrm{L}) + f_\mathrm{R}(P, W_\mathrm{R}) + v_\mathrm{R} - v_\mathrm{L} \\ f_\mathrm{X}(P, W_\mathrm{X}) &= \begin{cases} (P - P_\mathrm{X})\sqrt{\frac{A_\mathrm{X}}{P + B_\mathrm{X}}} & P > P_\mathrm{X} \\ \frac{2c_{s,\mathrm{X}}}{(\gamma-1)}\left[\left(\frac{P}{P_\mathrm{X}}\right)^{\frac{\gamma-1}{2\gamma}} - 1\right] & P \leq P_\mathrm{X} \end{cases} \\ A_\mathrm{X} &= \frac{2}{(\gamma+1)\rho_\mathrm{X}} \\ B_\mathrm{X} &= \frac{(\gamma-1)}{(\gamma+1)} P_\mathrm{X}, \end{aligned}$$

with $\mathrm{X} = \mathrm{L}, \mathrm{R}$. The condition $P > P_\mathrm{X}$ corresponds to a shock wave, $P \leq P_\mathrm{X}$ to a rarefaction wave.

Iterative solution

Given the complex form of the function $f(P, W_\mathrm{L}, W_\mathrm{R})$, finding the root of this function is a non-trivial task, which needs to be done numerically using a root finding algorithm.

Toro (2009) recommends using a Newton-Raphson method (Thijssen, 1999), which makes use of the first derivative of the function:

$$\begin{aligned} f'(P, W_\mathrm{L}, W_\mathrm{R}) &= f'_\mathrm{L}(P, W_\mathrm{L}) + f'_\mathrm{R}(P, W_\mathrm{R}) \\ f'_\mathrm{X}(P, W_\mathrm{X}) &= \begin{cases} \left(1 - \frac{P - P_\mathrm{X}}{2(B_\mathrm{X} + P)}\right)\sqrt{\frac{A_\mathrm{X}}{B_\mathrm{X} + P}} & P > P_\mathrm{X} \\ \frac{1}{\rho_\mathrm{X} c_{s,\mathrm{X}}}\left(\frac{P}{P_\mathrm{X}}\right)^{\frac{-(\gamma+1)}{2\gamma}} & P \leq P_\mathrm{X}. \end{cases} \end{aligned}$$

This method is quite fast, but can end up getting stuck in an endless loop for some initial values. We therefore combined it with a more robust root finding algorithm: Brent's method (Brent, 1973). This method requires an initial guess for an interval containing the root, which we do not normally have at our disposal.

We know that $f(0, W_\mathrm{L}, W_\mathrm{R}) < 0$, but it is hard to predict a value for P where the function is positive. We therefore use Newton-Raphson as long as the current guess for P results in a negative function value, and switch to the more robust Brent when we find a positive function value to use as upper limit for the interval.

To obtain fast convergence, a good initial guess for the pressure P_* is important. To obtain a good guess, we can make use of approximations in the wave structure of the solution. If the two outer waves for example are both rarefaction waves, then the pressure for the middle wave is given by an analytic expression (Toro, 2009):

$$P_{*,\mathrm{TR}} = \left(\frac{c_{s,\mathrm{L}} + c_{s,\mathrm{R}} - \frac{1}{2}(\gamma - 1)(v_\mathrm{R} - v_\mathrm{L})}{c_{s,\mathrm{L}} P_\mathrm{L}^{\frac{-(\gamma-1)}{2\gamma}} + c_{s,\mathrm{R}} P_\mathrm{R}^{\frac{-(\gamma-1)}{2\gamma}}} \right)^{\frac{2\gamma}{\gamma-1}}.$$

Similarly, if the two outer waves are shock waves, the pressure for the middle wave is approximately given by (Toro, 2009)

$$P_{*,\mathrm{TS}} = \frac{g_\mathrm{L}(\hat{P}) P_\mathrm{L} + g_\mathrm{R}(\hat{P}) P_\mathrm{R} + v_\mathrm{L} - v_\mathrm{R}}{g_\mathrm{L}(\hat{P}) P_\mathrm{L} + g_\mathrm{R}(\hat{P}) P_\mathrm{R}}$$
$$g_\mathrm{X}(P) = \sqrt{\frac{A_\mathrm{X}}{P + B_\mathrm{X}}},$$

with $\hat{P}$ a suitable guess for the pressure, that can be obtained using a linearized version of the Euler equations:

$$P_{*,\mathrm{lin}} = \frac{1}{2}(P_\mathrm{L} + P_\mathrm{R}) - \frac{1}{8}(v_\mathrm{R} - v_\mathrm{L})(\rho_\mathrm{L} + \rho_\mathrm{R})(c_{s,\mathrm{L}} + c_{s,\mathrm{R}}).$$

In practice, we will always start by calculating the linearized guess $P_{*,\mathrm{lin}}$, since it is the cheapest to calculate. If this pressure lies in between the pressure for the left and right states, and if the pressure contrast between the states is less than two, we will use this guess as starting point for the Newton-Raphson. If this guess is smaller than the minimum pressure for left and right states, we use the two rarefaction wave approximation $P_{*,\mathrm{TR}}$. If it is larger, we use the two shock wave approximation $P_{*,\mathrm{TS}}$, using $P_{*,\mathrm{lin}}$ as initial guess.

Since some of the approximations above can lead to negative pressures, we make sure the initial guess for the pressure is positive by enforcing some very small positive value for the pressure if necessary.

Approximate solution

We will see below that the solution of the Riemann problem is at the core of finite volume hydrodynamical integration methods, so that these methods crucially

depend on the fast and accurate solution of a large number of Riemann problems. The iterative procedure sketched above can be computationally very expensive, leading to a potential bottleneck for these methods. For this reason, a number of approximate solutions for the Riemann problem have been developed, which can obtain quite accurate results at constant computational cost.

The most simple approximate solution is the two rarefaction wave solution we already encountered above. In this case, we assume that both outer waves are rarefaction waves, and we immediately get P_*. This specific wave solution will occur when the primitive variables for left and right states are similar in magnitude, so that this type of solution is related to smooth, continuous flow, which is the type of flow most often encountered in astrophysical situations.

A two shock wave approximation is less useful, since it does not lead to a direct analytic expression for the pressure of the middle wave. Furthermore, shock waves usually occur in very specific regions of the flow, so that the approximation is generally not valid for the bulk of the flow.

Even better approximations to the Riemann problem can be found by assuming a two- or a three-wave model that is then directly used to calculate inter-cell fluxes that can be used in a finite volume method (see below). These Harten-Lax-van Leer (HLL) and Harten-Lax-van Leer Contact ($HLLC$) approximate Riemann solvers (Toro, 2009) do not actually give a solution to the whole Riemann problem, but are nonetheless very useful. We will not discuss them in more detail, as they were not used for our work.

Sampling the solution

Once we know the pressure and velocity for the middle wave, the solution is completely fixed, since this also fixes the wave form of the outer waves. To obtain the solution at position x and at time t, we need to sample the solution, to find out in which region of the solution we are. This is illustrated in the flowchart in Fig. 3.2.

To find out at which side of the middle contact wave we are, we have to compare the generic speed $S = \frac{x}{t}$ with the velocity of the middle wave, v_*. If it is smaller, we are at the left of the contact wave and need to sample the left wave. If it is larger, we sample the right wave instead.

To find out if the left or right wave is a shock wave, we compare the pressure of the middle wave with the pressure of the left or right state. If this pressure is larger, we have a shock wave. If it is smaller, we have a rarefaction wave.

At last, to find out at which side of the left or right wave we are, we need to compare S with the signal velocities of the appropriate wave. For a shock wave,

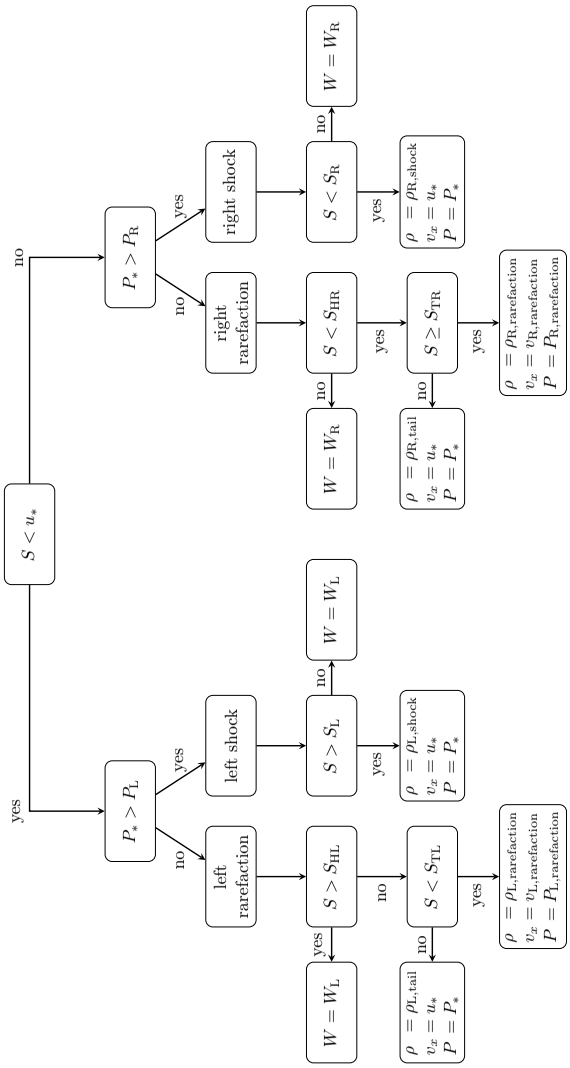


Figure 3.2: *Graphical representation of the sampling procedure for the Riemann solution once the pressure P_* and velocity v_* in the middle region are known. The solution is sampled at position x and time t and has associated wave speed $S = \frac{x}{t}$.*

the shock velocity is given by (Toro, 2009)

$$S_{\mathrm{X}} = v_{x,\mathrm{X}} \pm c_{s,\mathrm{X}} \sqrt{\frac{\gamma+1}{2\gamma}\frac{P_*}{P_{\mathrm{X}}} + \frac{\gamma-1}{2\gamma}}, \tag{3.4}$$

where the positive sign corresponds to a right wave (X = R) and the negative sign to a left wave (X = L). If S is in between the velocity of the middle wave and the shock velocity, the solution for the density is given by

$$\rho_{\mathrm{X,shock}} = \rho_{\mathrm{X}} \left(\frac{\frac{P_*}{P_{\mathrm{X}}} + \frac{\gamma-1}{\gamma+1}}{\frac{\gamma-1}{\gamma+1}\frac{P_*}{P_{\mathrm{L}}} + 1} \right),$$

while the velocity and pressure are given by v_* and P_*.

For a rarefaction wave, there are two signal velocities: the velocity of the head of the wave, which separates the left or right state from the left or right wave, and the velocity of the tail of the wave, which separates the left or right wave from the middle wave. The former is given by

$$S_{\mathrm{HX}} = v_{\mathrm{X}} \pm c_{s,\mathrm{X}},$$

and the latter by

$$S_{\mathrm{TX}} = v_* \pm c_{s,\mathrm{X}} \left(\frac{P_*}{P_{\mathrm{X}}} \right)^{\frac{\gamma-1}{2\gamma}},$$

where again in both cases the positive sign corresponds to the right wave and vice versa.

In between head and tail of the wave, the density, velocity and pressure are given by

$$\begin{aligned}
\rho_{\mathrm{X,rarefaction}} &= \rho_{\mathrm{X}} \left(\frac{2}{\gamma+1} \pm \frac{\gamma-1}{(\gamma+1)c_{s,\mathrm{X}}} \left(v_{\mathrm{X}} - \frac{x}{t} \right) \right)^{\frac{2}{\gamma-1}} \\
v_{\mathrm{X,rarefaction}} &= \frac{2}{\gamma+1} \left(\pm c_{s,\mathrm{X}} + \frac{\gamma-1}{2} v_{\mathrm{X}} + \frac{x}{t} \right) \\
P_{\mathrm{X,rarefaction}} &= P_{\mathrm{X}} \left(\frac{2}{\gamma+1} \pm \frac{\gamma-1}{(\gamma+1)c_{s,\mathrm{X}}} \left(v_{\mathrm{X}} - \frac{x}{t} \right) \right)^{\frac{2\gamma}{\gamma-1}},
\end{aligned}$$

where this time the positive sign corresponds to the left wave.

In between the tail of the wave and the middle wave, the velocity and pressure again correspond to v_* and P_*, while the density is given by

$$\rho_{\mathrm{X,tail}} = \rho_{\mathrm{X}} \left(\frac{P_*}{P_{\mathrm{X}}} \right)^{\frac{1}{\gamma}}.$$

Vacuum

In everything we did above, we silently assumed that the density had a positive value everywhere. However, we will encounter cases where the density effectively becomes zero, a situation which is called *vacuum*. This for example happens when we simulate isolated galaxies as a spherical cloud of gas embedded in a spherical dark matter potential with open boundaries, so that outside of the gas cloud we have a vacuum. If there is no matter, concepts as velocity, energy and pressure become meaningless, and we can no longer use the Euler equations to describe the hydrodynamics. This means the wave form of the solution of a Riemann problem involving vacuum will be different than the ordinary three wave structure.

Physical common sense dictates that the hydrodynamical variables are all just zero in vacuum, so that we do not require anything special to treat the vacuum itself. We have to take care however in cases where the vacuum is adjacent to a region with non-zero density. We distinguish three different cases: the two trivial cases where one of both states in the Riemann problem is a vacuum, and the somewhat more involved case were vacuum is created as part of the solution to the Riemann problem. The latter happens when the following *vacuum condition* is satisfied (Toro, 2009):

$$\frac{2c_{s,\mathrm{L}}}{\gamma-1}+\frac{2c_{s,\mathrm{R}}}{\gamma-1}\leq v_{\mathrm{R}}-v_{\mathrm{L}}.$$

This can clearly only happen if there is a large velocity contrast in a flow with very small sound speeds.

In all cases, the vacuum region is separated from the region with non-zero density by a wave front (or two wave fronts if vacuum is generated) with velocity

$$S_{\mathrm{X,vacuum}}=v_{\mathrm{X}}\pm\frac{2c_{s,\mathrm{X}}}{\gamma-1},$$

where the positive sign corresponds to a left vacuum wave. It can be proven that a vacuum can never be adjacent to a shock wave, so that the left or right state is separated from the vacuum front by a rarefaction wave. For the cases with left or right vacuum state, the wave solution hence consists of only two waves, while for the vacuum generation case it consists of four: a left rarefaction wave, a left vacuum front, a right vacuum front, and a right rarefaction wave.

To sample the vacuum solution, we can then proceed as before, but now only using the velocity of the head of the rarefaction wave and the velocity of the vacuum front. If the speed $S=\frac{x}{t}$ is in the vacuum region, all primitive variables are simply zero. If it is in the region between the head of the rarefaction wave and the vacuum front, the primitive variables are given by $\rho_{\mathrm{X,rarefaction}}$, $v_{\mathrm{X,rarefaction}}$, and $P_{\mathrm{X,rarefaction}}$.

Multiple dimensions

Until now, we have always considered the one dimensional Riemann problem. For most practical purposes however, we will need to consider the Riemann problem in two or three dimensions, which means the velocity v will be a vector quantity $\vec{v}$. Furthermore, the partial derivatives with respect to the spatial coordinate x will be replaced by gradient and divergence operations, which will make the Euler equations a lot more complex.

By an appropriate rotation of the coordinate system, it is always possible to align the x-axis with the direction in which the change from left to right state occurs, so that we can formulate the Riemann problem in *split multidimensional form* (Toro, 2009). The 3D Euler equations then take the following form (since derivatives in the y- and z-direction vanish):

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = 0,$$

with

$$F = \begin{pmatrix} \rho v_x \\ \rho v_x^2 + P \\ \rho v_x v_y \\ \rho v_x v_z \\ (\rho e + P) v_x \end{pmatrix}.$$

There is an interesting symmetry in the way in which the y- and z-component of the velocity are treated in these equations, which is also reflected in the eigenvalues of the Jacobian matrix for this system. These are still given by $v_x - c_s$, v_x and $v_x + c_s$, but now the middle eigenvalue has multiplicity three, which means the middle contact wave will consist of three contact waves that coincide. The two extra contact waves correspond to simple jumps in the tangential velocity components, while the solution for the other three waves is as before.

We can hence solve the multidimensional Riemann problem very easily by converting it into a 1D Riemann problem and treating the tangential velocity components as *passively advected quantities*, which are all quantities q that satisfy the equation

$$\frac{\partial q}{\partial t} + \vec{v}.\vec{\nabla} q = 0,$$

or, using the continuity equation,

$$\frac{\partial (\rho p)}{\partial t} + \vec{\nabla}.\,(\rho p \vec{v}) = 0.$$

For these quantities, the solution is given by a simple jump from left to right value across the central contact discontinuity.

3.2 Finite volume methods

In the previous section, we derived the Euler equations and saw that the physical solutions to these equations are given by rarefaction waves, contact discontinuities and shock waves. In this and the next section, we will discuss how we can solve these equations numerically for a discretized density distribution. In Chapter 2, we introduced various possible discretizations for the density distribution of a gas, and the different hydrodynamical integration methods will be intimately linked with these different discretizations. We will start with grid based discretizations.

3.2.1 Fixed grid

A grid based discretization consists of cells, which have a geometrical surface area or volume. This quantity can be used to convert the primitive variables to *conserved variables*. These naturally pop up when we integrate the Euler equations in conservative form over the volume V_i of the cell (Springel, 2010):

$$\frac{\partial}{\partial t}\int_{V_i} U \mathrm{d}V = \frac{\partial}{\partial t}\int_{V_i}\begin{pmatrix}\rho\\ \rho\vec{v}\\ \rho e\end{pmatrix}\mathrm{d}V = \frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix}m_i\\ \vec{p}_i\\ E_i\end{pmatrix} = \frac{\mathrm{d}}{\mathrm{d}t}Q_i = -\int_{V_i}\vec{\nabla}.\vec{F}(U)\mathrm{d}V,$$

where we introduced the *mass* m_i, *linear momentum* $\vec{p}_i$ and *total energy* E_i of the cell.

The volume integral on the right hand side of this equation can be converted into a surface integral:

$$\frac{\mathrm{d}}{\mathrm{d}t}Q_i = -\int_S \vec{F}(U).\mathrm{d}\vec{n},$$

where S is the total surface area of the boundaries of the cell, and $\vec{n}$ is a normal vector to the boundary of the cell. We now see why this formulation of the Euler equations is called the conservative form, as these equations link the time evolution of conserved variables to a flux through the boundary of the cell. It is very important to note that the above conversion from a volume integral to a surface integral is mathematically only allowed if the fluxes and their first derivatives are continuous. This is not the case if the physical solution for the hydrodynamical variables corresponds to a contact discontinuity or a shock wave, since then the Euler equations are no longer valid. By doing the conversion, we hence exclude these solutions inside the cell.

Since in practical grids a cell will be bordered by a finite number of geometrical faces, we can rewrite the equation above in discrete form as

$$\frac{\mathrm{d}}{\mathrm{d}t}Q_i = -\sum_j A_{ij}F_{ij}, \tag{3.5}$$

where we introduced the *face averaged flux*

$$F_{ij} = \frac{1}{A_{ij}} \int_{A_{ij}} \vec{F}(U).\mathrm{d}\vec{A}_{ij},$$

and where $\vec{A}_{ij}$ is the surface area of the geometrical face between cells i and j.

This equation is the cornerstone of so called *finite volume* methods. It expresses the change of the conserved variables of a cell as a sum of fluxes through the boundaries of the cell. The outward flux for one cell will correspond to an inward flux for one of its neighbouring cells; the hydrodynamical integration is hence completely governed by a *flux exchange* between cells. Since all changes in the conserved variables are given by a symmetric exchange between cells, the total sum for these values for the entire system will be constant, so that globally, these variables are indeed *manifestly* conserved.

To actually solve the Euler equations, we still need to determine appropriate fluxes. These fluxes should take into account the actual physical solutions to the Euler equations, especially if we also want to allow discontinuities or shock waves, which were excluded in the interior of the cell by the conversion of the volume integral above. We will start with a simple first order method and then extend it to second order in space and time.

First order: Godunov fluxes

In the previous section, we encountered the Riemann problem, and showed that its solutions include the three physical wave structure solutions to the Euler equations. The only necessary ingredients of a general Riemann problem are two states, separated by some region in space. It is hence possible to formulate a Riemann problem in between two neighbour cells of the grid, taking their primitive variables as the two states and the face between them as the separating region. We saw that we can always reduce the multidimensional Riemann problem to the one dimensional equivalent by means of an appropriate rotation, which in this case corresponds to a rotation that aligns the x-axis with the normal of the face between the two cells.

This is the main idea of the flux estimation of Godunov (1959): we use the primitive variables of the cells as input to a Riemann solver (approximate or exact) and calculate fluxes based on the solution to the Riemann problem. Since this solution automatically includes rarefaction waves, shock waves and contact discontinuities, we also include these possible solutions in the flux estimation.

The method is illustrated in Fig. 3.3 for the spherical overdensity problem that was introduced in Chapter 2. To this end, we introduce a pressure contrast,

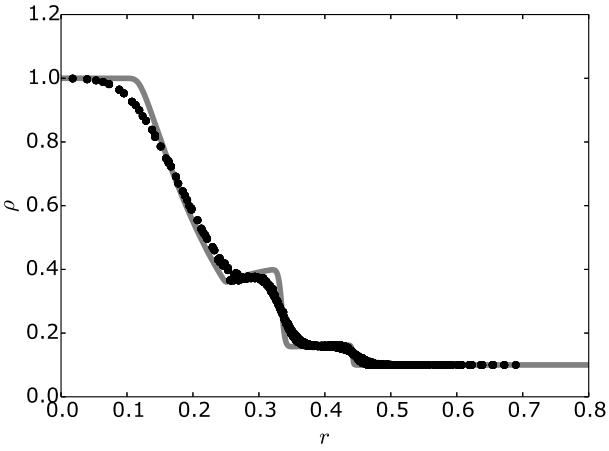


Figure 3.3: *The first order finite volume result for the spherical overdensity test in 2D, obtained using the AMR code* MPI-AMRVAC *on a* 40×40 *cell grid with 2 levels of refinement, and at time* $t = 0.1$*. The gray line represents the high resolution solution to the equivalent 1D problem, which acts as a reference solution. The black dots are the 2D simulation results.*

given by

$$P(\vec{x}) = \begin{cases} 1 & |\vec{x} - \vec{o}| < 0.25, \\ 0.1 & 0.25 \leq |\vec{x} - \vec{o}|, \end{cases}$$

which is the same as the density contrast.

Due to the pressure contrast, the dense region, which is initially at rest, will start to expand into the low density region. The radial density profile reflects the solution of a typical Riemann problem, and consists of a inward travelling rarefaction wave, a central contact discontinuity and an outward travelling shock wave. These are illustrated at time $t = 0.1$.

Although the setup has a very modest resolution, these features are clearly visible. To do better, we can increase the number of cells, but this only slowly increases the resolution of the solution. The reason for this is that the method described here is only *first order* in time and space. This means that the error we make by discretizing the Euler equations on a grid scales linearly with the cell size. Similarly, the error we make by discretizing time in finite steps scales linearly with the time step size. To reduce the overall error by a factor 2, we hence need to double the number of cells per dimension (which means using 8

times more cells in 3D!) and divide the time step size by a factor 2. We can do better than this by using a second order method.

Second order: gradients

To extend the first order Godunov method to second order in space, we need to take into account extra spatial information. Before, we treated every cell of the grid as being an independent entity, having its own conserved variables and derived primitive variables. However, if we go back to the original density distribution, we see that the cells are not independent, but together constitute a discrete representation of that density distribution. If two neighbouring cells have different density values, then this is likely caused by a local density gradient of the underlying distribution, rather than a discrete contact discontinuity between the two cells. To better represent the distribution in space, we can hence take into account these gradients and *reconstruct* the density profile inside the cell.

We need to modify the first order method as follows. Just as before, we still use the volume of the cells to convert the conserved variables to primitive variables. We then use these primitive variables to estimate local gradients for the primitive variables: for every cell, we calculate the linear gradients of the primitive variables between the cell and its neighbouring cells and combine them into a single gradient for every primitive variable. Before solving the Riemann problem at the face between two neighbouring cells, we reconstruct the primitive variables at the position of the face by linearly interpolating from the midpoint of the cell using the gradients of the cells. These reconstructed values are then used as input for the Riemann problem.

To also obtain second order accuracy in time, we need to take include a time dependence into this reconstruction step. To this end, we will also integrate the primitive variables forward in time for half a time step, using the Euler equations in primitive form (3.3). The idea is that the primitive variables will have evolved for half a time step when they arrive at the face if they have to arrive at the midpoint of the neighbouring cell at the full time step. The Euler equations in primitive form link the time evolution of the primitive variables to their gradients, which we already need for the spatial reconstruction.

The result for the spherical overdensity test using this second order MUSCL-Hancock method (van Leer, 1979) is shown in Fig. 3.4. The different features are now more clearly visible and the resolution increases a lot faster when increasing the number of cells.

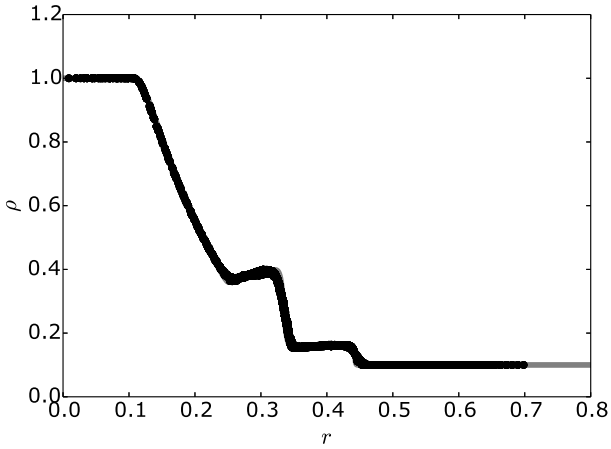


Figure 3.4: *The second order result for the spherical overdensity test, obtained using the AMR code* MPI-AMRVAC *using a grid of* 40×40 *cells with 2 levels of refinement, and at time* $t = 0.1$*. The gray line represents the high resolution 1D reference solution, the black dots are the 2D simulation results.*

Time stepping

To solve (3.5) numerically, we rewrite it as

$$\mathrm{d}Q_i = -\sum_j A_{ij} F_{ij} \mathrm{d}t.$$

We can hence calculate changes in the conserved quantities by applying the fluxes for these quantities during some discrete time step $\mathrm{d}t$. It is clear that the accuracy of the integration method will depend crucially on the size of the time step $\mathrm{d}t$: if the time step is too large, the discretization error will be very large as well, and we cannot expect the solution to be accurate. If the time step is too small, we might take a lot more steps than actually required, each leading to small numerical round off error. The accumulation of this round off error will then also lead to a less accurate result. Ideally, we hence want to use a time step that is small enough to sufficiently suppress discretization error, but not much smaller than that to limit the total number of steps.

For simple differential equations of the form

$$\frac{\partial y}{\partial t} + a\frac{\partial y}{\partial x} = 0$$

there exists a mathematical framework to find a time step that is small enough to suppress discretization error, known as von Neumann stability analysis (Isaacson & Keller, 1994; Toro, 2009). The idea is to look at how the discretization error due to a single step behaves when applying the differential equation itself again. If the discretization error grows under the differential equation, then the integration is unstable. If on the other hand the discretization error itself shrinks under the differential equation, then small errors will be automatically suppressed, and the integration is stable.

The stability of an integration method is usually expressed through the dimensionless *Courant number*, for the differential equation above given by

$$C = \frac{a}{\frac{\mathrm{d}x}{\mathrm{d}t}},$$

with $\mathrm{d}t$ the time step and $\mathrm{d}x$ the size of a (one dimensional) cell. The quantity in the denominator corresponds to a *grid speed*, i.e. the maximal speed that can be handled by the integration scheme. As we have seen before, a corresponds to the *wave speed* of the solution. It can be shown that a simple *upwind* integration scheme of the form

$$y_i^{n+1} = y_i^n - \frac{a}{\mathrm{d}x}\left(y_i^n - y_{i-1}^n\right)\mathrm{d}t,$$

where y_i^n corresponds to the value of y in cell i at time t_n, is stable if the Courant number lies in the range $[0, 1]$. The name upwind comes from the fact that we base the spatial discretization in between brackets on the sign of the wave speed a: for a positive value, the net flow will be from left to right, and we take the spatial discrete derivative by subtracting the value of y in the cell to the left from that of the cell i.

For the Euler equations, there are multiple waves associated with the general solution (we encountered this general solution while discussing the Riemann problem), which will generally have different associated wave speeds. However, the integration scheme above can still be seen as an upwind scheme if the flux between neighbouring cells is only based on the values of the primitive variables of these two cells. We can hence still define a Courant number (now called the Courant-Friedrichs-Lewy or CFL number after Courant *et al.* (1928)):

$$C_{\mathrm{CFL}} = \frac{S_{\max}}{\frac{\mathrm{d}x}{\mathrm{d}t}},$$

where $S_{\max}$ represents the maximal wave speed for the cell under consideration. Since obtaining this maximal wave speed requires the solution of the local Riemann problem, this speed is usually approximated by

$$S_{\max} \approx |\vec{v}| + c_s.$$

For multidimensional cells, we should in principle also take the wave speeds in different directions into account, and we should use a better generic cell size estimate $\mathrm{d}x$. However, in practice we will still use the CFL number above, and use a somewhat more strict stability condition, $0 \leq C_{\mathrm{CFL}} \leq 0.5$. This turns out to be both fast and accurate.

The CFL criterion applies on a cell-by-cell basis, meaning that the time step for different cells can be very different. Since we usually do not want to integrate cells with a time step that is orders of magnitude too small (since this is computationally expensive and leads to the accumulation of round off error), we will in practice use individual time steps. To this end, the total simulation time is mapped to an integer timeline, and cell time steps are restricted to power of two subdivisions of the total integer simulation time. To keep the integration manifestly conservative, we have to make sure all fluxes are exchanged symmetrically. If a cell that is *active* at some time interacts with an *inactive* cell, then we still exchange flux with the inactive cell, but with a time step that is equal to the (smaller) time step of the active cell (Springel, 2010). As input for the Riemann problem we use the primitive variables for the inactive cell that were calculated at the last time the cell was active. When the cell is active again, it will have exchanged fluxes through all of its faces for a time step that is equal to the total time step of the cell, and only then it is possible to update the primitive variables using the new values of the conserved variables.

Limiters

The first order Godunov method for the pure Euler equations without external terms is unconditionally stable, given that an appropriate time step criterion is used. This means that the method will never produce unphysical results were the mass or energy for a cell becomes negative. The reason for this is that the Riemann problem formulated using the cell values as inputs can never produce results that lead to fluxes that are too large.

The second order MUSCL-Hancock method does no longer satisfy this unconditional stability. By locally reconstructing the primitive variables, we can end up with input states that have higher densities than physically possible for a cell, so that the resulting fluxes might carry away more mass or energy than available in the cell. Furthermore, the use of gradients can introduce new extreme values for the primitive variables, which will in turn lead to wave patterns in the solution that were not part of the initial setup.

To solve these issues, we need to include *limiters* in the scheme (Toro, 2009). These limiters exist in various flavours. At the one hand, it is possible to limit the cell gradients, in which case we have *slope limiters*. On the other hand, it is also possible to limit the fluxes, which is done using *flux limiters*.

Slope limiters themselves also come in two varieties: cell wide slope limiters and per face limiters. The former limit the gradients on a cell wide basis before the reconstruction step, by ensuring that the gradients never lead to reconstructed values that are larger than the primitive variables in one of the neighbouring cells, for any of the faces (Springel, 2010). The advantage is that they treat the entire cell on the same level. Per face limiters limit the reconstructed values at the face without affecting the gradients themselves and are hence face dependent. A good per face slope limiter makes sure that no new wave structures are introduced in the solution by performing the reconstruction step, so that when the first order scheme would result in a rarefaction wave, the second order scheme does so as well and does not result in e.g. a shock wave (Hopkins, 2015).

The strength of a specific finite volume method is completely determined by the interplay between the Riemann solver and the slope limiter. If an approximate Riemann solver is used that does not handle shocks (e.g. the two rarefaction wave Riemann solver), there is no risk of introducing unphysical shocks and the slope limiter can be quite soft. This in turn leads to less numerical dissipation, but also means the method will be bad at resolving shock waves. An exact Riemann solver will produce shocks and requires a more stringent slope limiter, leading to an overall better behaviour close to shock waves, but also to more dissipation.

Flux limiters are the last chance to prevent a cell from loosing more mass or energy than it has, and act directly on the flux between cells (M. Schaller, personal communication, February 2015). They are about as artificial as resetting the mass or energy in a cell to some small value whenever it becomes negative, but at least they preserve the manifest mass and energy conserving character of the method by acting on the flux between cells rather than individual cell quantities. There are many possible flux limiters, and we opted to use a geometrical one that makes sure that the flux is never larger than some fraction of the total cell mass or energy, which corresponds to the ratio of the surface area of the active face to the total surface area of the cell (multiplied with the ratio of the smallest and the largest time step of the two cells if individual time stepping is used). This way the mass or energy can never become negative, even if the outflux for all faces is maximal. For consistency, we limit all five fluxes with the same factor.

3.2.2 Moving mesh

The discussion about finite volume methods above was completely independent of the geometry of the grid (although the examples used a regular Cartesian grid). This illustrates the fact that a finite volume method can be defined on just about any type of grid structure, as long as it has a notion of cells with an associated notion of volume, and every cell has a list of neighbouring cells that share faces with some notion of a location in space and a surface area. A Voronoi mesh

satisfies all of these criteria.

Unstructured meshes can be advantageous for stationary problems with some clear geometry, as they make it possible to adapt the cell geometry to the problem at hand. They however become even more interesting if we consider the case where the mesh is allowed to move. In Chapter 2, we saw that the Voronoi mesh changes in a very continuous way when the generators of the mesh change: faces between neighbouring cells grow or shrink linearly, but they only appear or disappear by continuously passing through a phase of zero length. This means that the flux between neighbouring cells would change linearly as well if the generators are allowed to move, which makes it possible to define a consistent integration scheme based on such a moving mesh. By coupling the movement of the mesh to the movement of the fluid itself, we can hence make sure that the mesh keeps being adapted to the problem at hand, even if this problem is highly dynamical.

Of course, a finite volume method on such a moving mesh would only be meaningful if the method does not depend on the underlying mesh: the integration can be better if the cells are allowed to move, but the solution can not in any way depend directly on the movement of the cells. When we exchange the fluxes, we hence need to take the movement of the cells into account, by rewriting the fluxes as (Springel, 2010)

$$\vec{F} = \begin{pmatrix} \rho(\vec{v} - \vec{w}) \\ \rho\vec{v}(\vec{v} - \vec{w}) + P\overrightarrow{1} \\ \rho e(\vec{v} - \vec{w}) + P\vec{v} \end{pmatrix},$$

where $\vec{w}$ is the velocity of the face, while $\vec{v}$ is still the fluid velocity with respect to a reference frame fixed to the simulation box. This takes the geometrical flux $-\vec{w}U$ due to the movement of the face into account .

We could in principle still solve the Riemann problem in the reference frame fixed to the box, but most of the power of a moving mesh method comes from the fact that we now can do it in the rest frame of the face as well. To this end, we deboost the fluid velocities for the left and right states before solving the Riemann problem by simply subtracting the face velocity, and boost the solution afterwards by adding the face velocity again. This leads to a much higher accuracy when the fluid has a high bulk velocity with respect to the simulation box, since this bulk velocity will completely cancel out of the Riemann problem.

Generator velocities

Ideally, we would like to set the velocities of the mesh generators to the local fluid velocity inside the corresponding Voronoi cell, so that the cell exactly follows the flow. This can however lead to very irregular cell shapes in regions with complex

flow behaviour, which is undesired (especially if an evolving Voronoi mesh is used, see Chapter 2). By using Voronoi cells as the discrete representations of a continuous distribution, we silently assume that the cell quantities are sampled at the position of the centroid of the cell. However, for part of the algorithm, we treat them as if the quantities are located at the position of the generators instead. The method will only be accurate if these positions are not too far apart, which means the cells need to be regular enough. Vogelsberger *et al.* (2012) further note that the gradient reconstruction works best if the cells are to some extent *spherical*, i.e. there should not be an order of magnitude difference between the distance from the centroid to the cell boundary in one direction with respect to another direction.

To make sure the mesh stays regular enough, we add an extra correction term to the generator velocity. This correction term is inspired by Lloyd's algorithm and is directed towards the actual centroid of the current cell. It only switches on if the distance between the generator position and the cell centroid is large, and is given by (Springel, 2010)

$$\vec{v}_{\mathrm{cor}} = \begin{cases} 0 & |\vec{s}-\vec{x}| < 0.9\eta R \\ c_s \left(\frac{\vec{s}-\vec{x}}{|\vec{s}-\vec{x}|}\right) \left(\frac{|\vec{s}-\vec{x}|-0.9\eta R}{0.2\eta R}\right) & 0.9\eta R \leq |\vec{s}-\vec{x}| < 1.1\eta R \\ c_s \left(\frac{\vec{s}-\vec{x}}{|\vec{s}-\vec{x}|}\right) & 1.1\eta R \leq |\vec{s}-\vec{x}|, \end{cases}$$

where $\vec{x}$ is the position of the generator of the cell, $\vec{s}$ is the position of the centroid of the cell, η is a parameter (we adopt the value $\eta = 0.25$), and $R = \sqrt[3]{\frac{3V}{4\pi}}$ is the generic size of the cell, which corresponds to the radius of a sphere with the same volume V.

Face midpoints and velocities

Calculating the midpoints of the faces of the cells is a purely geometrical task, which can be achieved by storing the vertex positions in some fixed order (clockwise or counterclockwise) around the line joining the generators of the cells that share the face. The midpoint is then the surface area weighted average of the midpoints of the triangles formed by the first vertex and two consecutive other vertices of the face. This calculation can easily be done together with the calculation of the total surface area of the face.

To obtain the velocity of the face, we have to combine the velocities of the two generators appropriately. If $\vec{x}_{ij}$ is the position of the midpoint of the face, then the velocity of the face is given by (Springel, 2010)

$$\vec{v}_{ij} = \frac{\vec{w}_i + \vec{w}_j}{2} + \left(\frac{(\vec{w}_i - \vec{w}_j).(\vec{x}_{ij} - \frac{\vec{x}_i + \vec{x}_j}{2})}{|\vec{x}_j - \vec{x}_i|} \right) \left(\frac{\vec{x}_j - \vec{x}_i}{|\vec{x}_j - \vec{x}_i|} \right).$$

Time stepping

For unstructured meshes, it is still possible to use a CFL-like time step criterion, as introduced for fixed grids above. If the mesh is allowed to move along with the flow, we can even relax the maximal wave speed estimate to only include the sound speed and the relative motion of the fluid with respect to the movement of the cell. This will lead to time steps that are somewhat larger than for the equivalent fixed grid integration, especially in fluids with large bulk velocities.

Saitoh & Makino (2009) pointed out that the use of a standard CFL criterion in combination with individual time steps can lead to severe problems in the vicinity of strong shocks. For a shock wave, the speed is given by (3.4), which can be significantly larger than the sound speed. Furthermore, shock waves arise very locally, so that they can only be detected in a few cells, but affect many more cells. To this end, Springel (2010) proposes using a tree based time step criterion that also takes into account the relative velocity of the cell with respect to other cells in the vicinity. If a shock is detected, then we make sure the individual time step for all cells that can be affected by this shock is small enough to resolve it.

This tree based criterion is computationally very expensive. We will hence only use it when absolutely necessary, for example when handling the Sedov-Taylor blast wave test which we will encounter in Chapter 5.

Results

The result of the spherical overdensity test using a moving mesh hydrodynamical integration method is shown in Fig. 3.5. The accuracy of the solution is comparable to that of the second order AMR solution, but uses less resolution elements. Computationally, the moving mesh method is a lot more expensive, so that in most practical cases it is still cheaper to use a fixed grid with a higher resolution to obtain the same level of accuracy. For the spherical overdensity test, the second order AMR simulation on a grid with an effective resolution of 160×160 cells in the most refined regions took 3.082 seconds to finish. The same simulation using a moving mesh code with only 2,780 cells took 2.516 seconds on the same hardware.

The situation is different for flows with a high bulk velocity with respect to the simulation box. Fig. 3.6 shows the example of a shearing layers test (Lecoanet *et al.*, 2015; Vandenbroucke & De Rijcke, 2016), in which two layers with different densities are in pressure equilibrium. The setup is made highly unstable by introducing a large shearing velocity in one or both layers, so that any small velocity in the direction perpendicular to the boundary layer will grow exponentially to form a so called *Kelvin-Helmholtz instability*.

This instability will arise at all scales, so that in practice the wavelength of the instabilities that will dominate the simulation depends on the resolution of

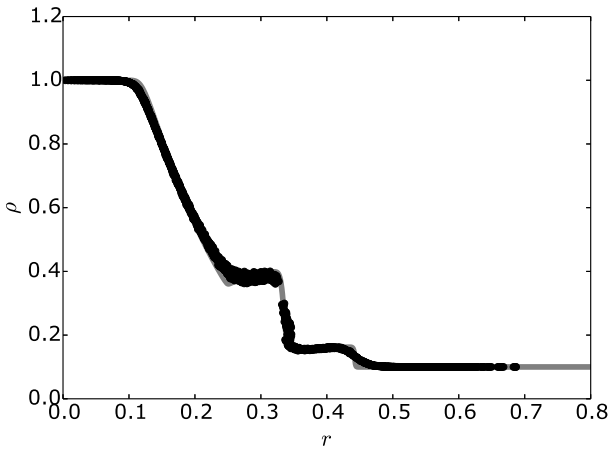


Figure 3.5: *The second order moving mesh result for the spherical overdensity test, obtained using the moving mesh code* Shadowfax *using the generator distribution from Chapter 2 with 2,780 generators, and at time* $t = 0.1$*. The gray line represents the high resolution 1D reference solution, the black dots are the 2D simulation results.*

the simulation. To make the test resolution independent, we introduce a linear transition layer in between the two layers that suppresses all instabilities below a certain threshold wavelength (Chandrasekhar, 1961; Hendrix & Keppens, 2014). Since this will also suppress the numerical noise that seeds the instability, we need to seed the instability artificially, by introducing a small velocity perturbation in the direction perpendicular to the interface. The thickness of the transition layer also sets a maximally unstable wavelength mode. If we seed this particular wavelength, then we are certain that it will dominate the simulation and we can compare between methods.

With the normal setup, both a fixed grid and a moving mesh succeed in resolving the exponential growth of the instability. However, if we add an arbitrary (but large) extra velocity to the entire fluid, so that both layers are moving supersonically and the relative shear velocity is much smaller than the bulk velocity of the flow, we see that fixed grid methods have difficulties resolving the instabilities, while the result is basically unchanged for the moving mesh code. More importantly, the time step for the fixed grid integration is significantly smaller in this case, to be able to accommodate the high flow velocity. For the moving mesh it is exactly the same, since the cells automatically follow the high flow velocity.

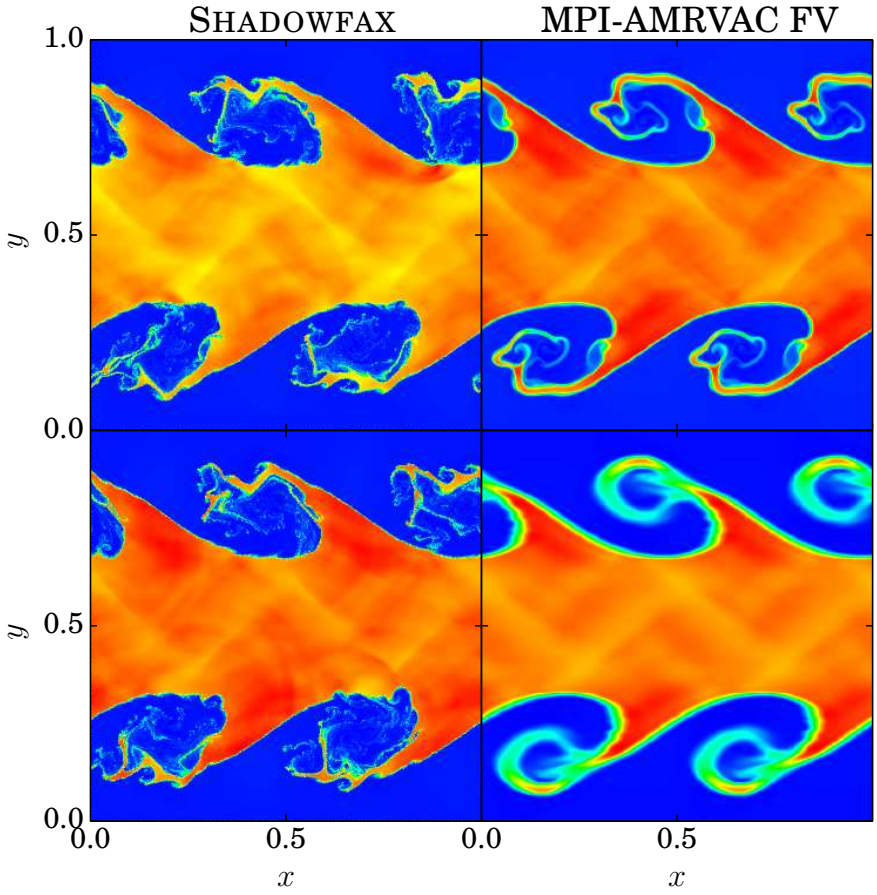


Figure 3.6: *Density colour plot for the shearing layers test at time* $t = 1.5$. *The left column shows simulations run with* SHADOWFAX, *the right column shows simulations run with* MPI-AMRVAC. *Both use a* 400×400 *grid which is (initially) Cartesian. The top row shows simulations without a bulk velocity, the bottom row shows simulatons where the entire fluid has a velocity* $v_{bulk} = 100$ *with respect to the simulation box. This clearly affects the* MPI-AMRVAC *result.*

For a high enough bulk velocity, the run time of a moving mesh simulation will hence always be smaller than for a fixed grid simulation.

3.3 Smoothed Particle Hydrodynamics

In Chapter 2, we introduced the smoothing kernel and showed how we can consistently estimate the density for a set of particles using an iterative procedure. Smoothed Particle Hydrodynamics (SPH) is the hydrodynamical integration scheme based on this density estimate. To this end, we will rewrite the other primitive hydrodynamical quantities in terms of the smoothing kernel as well, so that every particle gets an associated velocity, thermal energy, and pressure next to its density:

$$\begin{aligned}\vec{v}(\vec{x}) &= \sum_i \vec{v}_i \frac{m_i}{\rho_i} W(|\vec{x} - \vec{x}_i|, h_i) \\ u(\vec{x}) &= \sum_i u_i \frac{m_i}{\rho_i} W(|\vec{x} - \vec{x}_i|, h_i) \\ P(\vec{x}) &= \sum_i P_i \frac{m_i}{\rho_i} W(|\vec{x} - \vec{x}_i|, h_i).\end{aligned}$$

The particle quantities will still be time-dependent, but their position dependence now is absorbed by the smoothing kernel. The gradient or divergence of these quantities can then be expressed as a simple particle sum as well (Price, 2012):

$$\vec{\nabla} X(\vec{x}) = \sum_i X_i \frac{m_i}{\rho_i} \vec{\nabla} W(|\vec{x} - \vec{x}_i|, h_i).$$

If we want the density estimate to hold at all times, the particle positions should be adapted in between time steps. This can be achieved by moving the particles with their local fluid velocity, so that the continuity equation is trivially satisfied. This corresponds to a so called *Lagrangian* point of view, which can formally be obtained by rewriting the Euler equations so that all partial time derivatives are replaced by total Lagrangian time derivatives: $\frac{\mathrm{d}}{\mathrm{d}t} = \frac{\partial}{\partial t} + \vec{v}.\vec{\nabla}$.

3.3.1 Equations

Applying the procedure described above, we can rewrite the momentum equation (3.1) as

$$\frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = -\frac{1}{\rho}\vec{\nabla} P.$$

Evaluating this equation at the position of particle i, and inserting the kernel estimates for the velocity, density and pressure, we get

$$\frac{\mathrm{d}\vec{v}_i}{\mathrm{d}t} = -\frac{1}{\rho_i}\sum_j P_j \frac{m_j}{\rho_j}\vec{\nabla}W(|\vec{x}_i - \vec{x}_j|, h_j),$$

which can be used to integrate the particle velocity in time. The right hand side of this equation then corresponds to a hydrodynamical acceleration, which can be treated just like the gravitational acceleration in an N-body simulation.

The expression above is however not symmetric in i and j. This is problematic, since this also means that the momentum transfer from particle i to particle j will differ from the momentum transfer from particle j to particle i, which violates momentum conservation. We can symmetrize the momentum equation by noting that

$$\vec{\nabla}\left(\frac{P}{\rho}\right) = \frac{\vec{\nabla}P}{\rho} - \frac{P}{\rho^2}\vec{\nabla}\rho.$$

This leads to a more symmetric velocity equation:

$$\frac{\mathrm{d}\vec{v}_i}{\mathrm{d}t} = -\sum_j \left(\frac{P_j}{\rho_j^2} + \frac{P_i}{\rho_i^2}\right) m_j \vec{\nabla}W(|\vec{x}_i - \vec{x}_j|, h_j).$$

Note that, for strict momentum conservation, we also need to use a symmetrized form of the smoothing kernel, which is done in most SPH codes.

The energy equation (3.2) in Lagrangian form is given by

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -\frac{P}{\rho}\vec{\nabla}.\vec{v}.$$

This leads to the following equation for the thermal energy of an SPH particle:

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = -\frac{P_i}{\rho_i}\sum_j \vec{v}_j.\frac{m_j}{\rho_j}\vec{\nabla}W(|\vec{x}_i - \vec{x}_j|, h_j).$$

This equation is not a very good Lagrangian equation, since it depends on the absolute velocity v_j of the neighbouring particle. We can rewrite it using

$$\vec{\nabla}.\vec{v} = \frac{1}{\rho}\rho\vec{\nabla}.\vec{v} = \frac{1}{\rho}\left(\vec{\nabla}.(\rho\vec{v}) - \vec{v}.\vec{\nabla}\rho\right),$$

to read

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = -\frac{P_i}{\rho_i^2}\sum_j (\vec{v}_j - \vec{v}_i).m_j\vec{\nabla}W(|\vec{x}_i - \vec{x}_j|, h_j).$$

The relative velocity of particle j with respect to particle i will usually be a lot smaller than its absolute velocity, allowing a much more accurate integration of the thermal energy, especially for fluids with a high bulk velocity and low thermal energy.

When using variable smoothing lengths, as we will always do, we also have to add small correction terms to the momentum and energy equation to take the variation of the smoothing length into account (Springel & Hernquist, 2002; Price, 2012).

Springel & Hernquist (2002) show that the equations of SPH can be derived in a fully conservative form by starting from a Lagrangian formalism, in which variable smoothing lengths are assumed by construction. Such a scheme can be shown to conserve both energy and entropy (which is not the case for the equations above) if not the thermal energy, but the entropic function A is integrated. This is the approach adopted by the public version of the SPH code GADGET2 (Springel, 2005). We will however use the thermal energy to accommodate for the use of a more generalised equation of state, see Chapter 5.

3.3.2 Problems

The above equations lead to a conceptually very simple integration scheme, whereby the hydrodynamics is done during two *neighbour loops*, whereby we loop over all particles and for every particle find the particles that lie within its smoothing length. During the first loop, the densities for the particles are calculated using the iterative procedure described in Chapter 2. These densities are then used during the second loop to calculate the hydrodynamical acceleration and to update the thermal energy, using the equations above. This algorithm fits perfectly within the kick-drift-kick formalism of a symplectic leapfrog integrator, which has good energy conserving properties. In Fig. 3.7, we show the result of the spherical overdensity test obtained using SPH.

Compared with other methods, the solution is significantly less accurate, but this is mainly due to the effective resolution of SPH being significantly lower: where the moving mesh simulation uses all 2,780 cells, everything is smeared out over 20 neighbours in the (2D) SPH simulation. The effective resolution is hence only of the order $\sim$ 100 cells. Nonetheless, the overall density profile is quite good, and there is even a small bump around the location of the shock wave. The central contact discontinuity is completely absent however.

Computationally, SPH performs better than its competitors: the spherical overdensity test only takes 1.29 seconds.

Some fundamental problems exist with the standard formulation of SPH. Agertz *et al.* (2007) compared a number of popular hydrodynamical solvers on a set of benchmark problems. They found a striking disagreement between finite

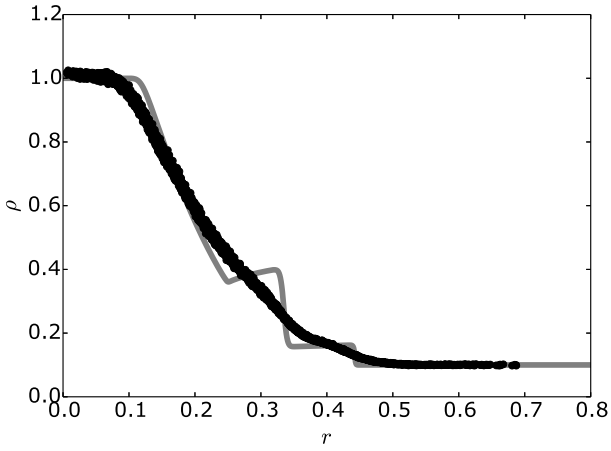


Figure 3.7: *SPH result for the spherical overdensity test in 2D at time $t = 0.1$, obtained using the SPH code* GADGET2. *The initial particle distribution used is the same as for the moving mesh simulation, which contains 2,780 particles. The black dots are the simulation results, the gray line corresponds to the high resolution 1D reference solution.*

volume methods and SPH on problems involving a strong discontinuity in the density. SPH performs particularly bad for problems involving Kelvin-Helmholtz and Rayleigh-Taylor instabilities, as illustrated in Fig. 3.8, which shows the interaction of a shock wave with a high density blob: while the finite volume solution shows a complex interface with multiple instabilities, the SPH solution contains no instabilities, and just shows a deformed blob.

Even more striking is that the SPH solution does not improve when the resolution of the SPH simulation is increased. While increasing the resolution of the finite volume simulations leads to a higher level of detail in the instabilities, SPH fails to reproduce any physical instability, independent of the resolution employed. The problems are hence not caused by a poor effective resolution, but point to a fundamental inability of SPH to resolve particular physical solutions of the Euler equations. This will be discussed in more detail below.

To solve these problems, some ad hoc fixes have been proposed, involving artificial viscosity and conductivity terms in the velocity and thermal energy equation (Price, 2008; Valcke *et al.*, 2010; Read & Hayfield, 2012; Schaye *et al.*, 2015; Yamamoto *et al.*, 2015). The problem with these terms is that they should

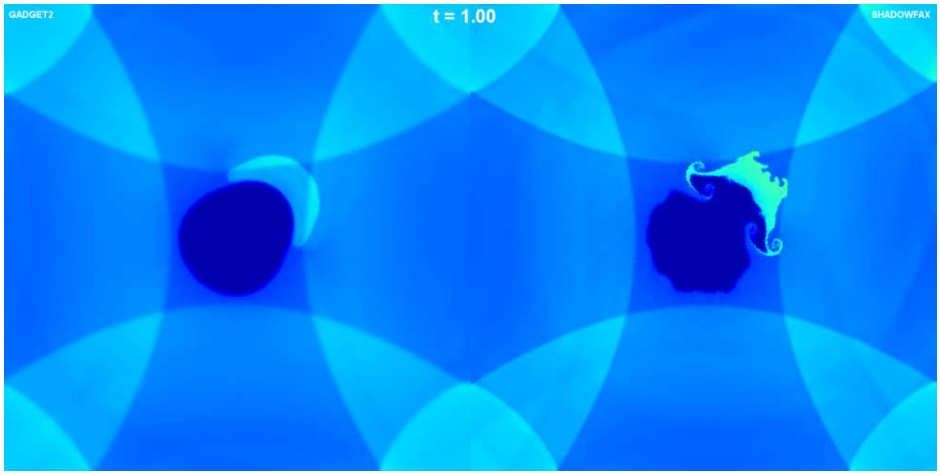


Figure 3.8: *Density profile for the 2D interaction of a shock wave with a spherical high density blob at time $t = 1$. Left: solution obtained using the SPH code* GADGET2*, right: solution obtained using the moving mesh code* SHADOWFAX*.*

only be applied when necessary. If there is a large density contrast, we need artificial terms to retrieve the missing instabilities, but we do not want these terms to affect the (correct) solution in regions with fairly regular flow. Artificial terms are hence always combined with appropriate *switches* that should detect the occurrence of discontinuities and activate correction terms accordingly. This is far from trivial, and usually involves a significant extra computational cost.

3.4 Mesh-free finite volume methods

Compared with a finite volume method, SPH clearly suffers some fundamental problems. It has however the advantage of being a *Lagrangian* method, a property which it only shares with moving mesh finite volume methods. Compared with a moving mesh method, SPH is computationally a lot cheaper, since we do not need to construct and maintain a complex geometrical structure. This is the reason that many astrophysical simulators still use SPH methods with ad hoc fixes rather than using a moving mesh method.

Part of the problems with SPH stem from the poor volume partitioning properties of SPH, which we already encountered in Chapter 2. Since the particles sample the simulation box in an inconsistent way, the local resolution is not always what we expect it to be, especially around discontinuities. More funda-

mentally, SPH is incomplete. When discussing the physical solutions to the Euler equations, we showed that there are three such solutions: rarefaction waves, contact discontinuities and shock waves. Only rarefaction waves are mathematical solutions to the Euler equations themselves, while the solutions involving discontinuities have to be treated using Riemann invariants and the Rankine-Hugoniot conditions. Since the SPH equations are derived using only the Euler equations, we completely miss the possibility of discontinuous solutions.

To put the missing solutions back into the method, we could try to base the hydrodynamical forces on the solution of a Riemann problem somehow. This is the method Cha *et al.* (2010) explore. This method however still suffers from the poor volume partitioning of regular SPH, and is hence nothing more than a computationally expensive fixed SPH version.

What we actually want, is a method with good volume partitioning properties, so that the method becomes independent from the particle distribution in a sense, and preferably also a method that is similar to a finite volume method, since we then have manifest conservation of conserved quantities. The meshless volumes we introduced in Chapter 2 offer exactly this. They however only define a notion of volume. We will see below how we can also derive some notion of surface area, which will allow us to construct an effective mesh-free finite volume method.

3.4.1 Weak solutions of the Euler equations

We can integrate the Euler equations in conservative form over the entire simulation volume V and over some time interval T, and multiply them with an arbitrary test function $\phi(\vec{x}, t)$, which has compact support. This means that $\phi(\vec{x}, t)$ has some non-trivial value at position $\vec{x}$ and time t, but drops to zero away from this position and time, so that it is certainly zero at the boundaries of the simulation box and the endpoints of the time interval:

$$\int_T \int_V \phi(\vec{x}, t) \left(\frac{\partial U}{\partial t} + \vec{\nabla}.\vec{F}(U) \right) \mathrm{d}V \mathrm{d}t = 0.$$

Due to the spatial and temporal coordinates being completely independent, we can switch around the integral signs as desired and arrive at the following

relations:

$$\begin{aligned}\int_T \int_V \frac{\partial}{\partial t}\left(\phi(\vec{x},t)U\right)\mathrm{d}V\mathrm{d}t &= \int_T \int_V \phi(\vec{x},t)\frac{\partial U}{\partial t}\mathrm{d}V\mathrm{d}t \\ &\quad + \int_T \int_V U\frac{\partial}{\partial t}\phi(\vec{x},t)\mathrm{d}V\mathrm{d}t \\ \int_T \int_V \vec{\nabla}.\left(\phi(\vec{x},t)\vec{F}(U)\right)\mathrm{d}V\mathrm{d}t &= \int_T \int_V \phi(\vec{x},t)\vec{\nabla}.\vec{F}(U)\mathrm{d}V\mathrm{d}t \\ &\quad + \int_T \int_V \vec{F}(U).\vec{\nabla}\phi(\vec{x},t)\mathrm{d}V\mathrm{d}t.\end{aligned}$$

The integrals on the left hand side trivially vanish because of the test function having compact support. This means we can rewrite the Euler equations as

$$\int_T \int_V \left(U\frac{\partial}{\partial t}\phi(\vec{x},t) + \vec{F}(U).\vec{\nabla}\phi(\vec{x},t)\right)\mathrm{d}V\mathrm{d}t = 0. \tag{3.6}$$

The solutions $\phi(\vec{x},t)$ are called *weak solutions* of the Euler equations. It is interesting to note that the weak solutions are still valid in regions of space were the hydrodynamical variables or the fluxes are discontinuous.

In Chapter 2, we divided up the volume of every small region in space over a number of neighbouring particles, using the fractions defined in (2.2). Similarly, we can divide up the value for a function $f(\vec{x})$ over the discrete set of neighbouring particles, using the same fractions:

$$f(\vec{x}) = \sum_i f_i\psi_i(\vec{x}),$$

for which we have

$$\vec{\nabla}f(\vec{x}) = \sum_i f_i\vec{\nabla}\psi_i(\vec{x}).$$

If we integrate out the function $f(\vec{x})$ over the entire volume of the simulation box, we get

$$\int_V f(\vec{x})\mathrm{d}V = \int_V \sum_i f_i\psi_i(\vec{x})\mathrm{d}V = \sum_i f_iV_i,$$

where we defined the volume of particle i to be the integral of its volume fraction over the entire volume of the simulation box:

$$V_i = \int_V \psi_i(\vec{x})\mathrm{d}V.$$

Without loss of generality, we can expand the weak solution $\phi(\vec{x})$ in the second term of (3.6) (Ivanova *et al.*, 2013):

$$\int_T \int_V \vec{F}(U).\vec{\nabla}\phi(\vec{x},t)\mathrm{d}V\mathrm{d}t = \int_T \int_V \vec{F}(U).\sum_i \phi_i(t)\vec{\nabla}\psi_i(\vec{x})\mathrm{d}V\mathrm{d}t.$$

We know that $\sum_i \psi_i(\vec{x}) = 1$ (and hence also $\vec{\nabla}\sum_i \psi_i(\vec{x}) = 0$), so that we can rewrite this as

$$\begin{aligned}
&\int_T \int_V \vec{F}(U).\vec{\nabla}\phi(\vec{x},t)\mathrm{d}V\mathrm{d}t \\
&\quad = \int_T \int_V \vec{F}(U).\sum_{i,j} \psi_i(\vec{x})\left(\phi_j(t) - \phi_i(t)\right)\vec{\nabla}\psi_j(\vec{x})\mathrm{d}V\mathrm{d}t \\
&\quad = \int_T \int_V \vec{F}(U).\sum_{i,j} \phi_i(t)\left(\psi_j(\vec{x})\vec{\nabla}\psi_i(\vec{x}) - \psi_i(\vec{x})\vec{\nabla}\psi_j(\vec{x})\right)\mathrm{d}V\mathrm{d}t \\
&\quad = \int_T \sum_{i,j} \phi_i(t) \int_V \vec{F}(U).\mathrm{d}\vec{\Sigma}_{ij},
\end{aligned}$$

where we introduced a *generalised surface area*

$$\vec{A}_{ij} = \int_V \mathrm{d}\vec{\Sigma}_{ij} = \int_V \left(\psi_j(\vec{x})\vec{\nabla}\psi_i(\vec{x}) - \psi_i(\vec{x})\vec{\nabla}\psi_j(\vec{x})\right)\mathrm{d}V.$$

We can know approximate the volume integral by a simple point quadrature, to obtain

$$\int_T \int_V \vec{F}(U).\vec{\nabla}\phi(\vec{x},t)\mathrm{d}V\mathrm{d}t \approx \int_T \sum_{i,j} \phi_i \vec{F}_{ij}(U).\vec{A}_{ij}\mathrm{d}t,$$

where $\vec{F}_{ij}(U)$ is the flux through the abstract *interface* between particle i and j, similar to the flux through the face of a cell in a finite volume method.

The first term of (3.6) can be similarly approximated by

$$\begin{aligned}
\int_T \int_V U\frac{\partial}{\partial t}\phi(\vec{x},t)\mathrm{d}V\mathrm{d}t &= \int_T \int_V U\frac{\partial}{\partial t}\phi(\vec{x},t)\sum_i \psi_i(\vec{x})\mathrm{d}V\mathrm{d}t \\
&\approx \int_T \sum_i \frac{\mathrm{d}\phi_i(t)}{\mathrm{d}t}U_i \int_V \psi_i(\vec{x})\mathrm{d}V\mathrm{d}t \\
&= \int_T \sum_i \frac{\mathrm{d}\phi_i(\vec{x})}{\mathrm{d}t}U_i V_i \mathrm{d}t.
\end{aligned}$$

Since the weak solution has compact support, we can use integration by parts to rearrange the time derivative:

$$\int_T \int_V U\frac{\partial}{\partial t}\phi(\vec{x},t)\mathrm{d}V\mathrm{d}t = -\int_T \sum_i \frac{\mathrm{d}}{\mathrm{d}t}(U_i V_i)\phi_i(t)\mathrm{d}t.$$

Putting both terms together again (and noting the anti-symmetry of $\vec{A}_{ij} = -\vec{A}_{ji}$), we obtain

$$\int_T \sum_i \phi_i(t) \left(\frac{\mathrm{d}}{\mathrm{d}t}(U_i V_i) + \sum_j \vec{F}_{ij}.\vec{A}_{ji} \right) \mathrm{d}t = 0.$$

This equation should hold for every arbitrary set of weak solutions $\phi_i(t)$ and for all times, which means the factor in between brackets should be zero as well. We arrive at the following meshless equivalent of the finite volume flux equation (Hopkins, 2015):

$$\frac{\mathrm{d}}{\mathrm{d}t}(U_i V_i) + \sum_j \vec{F}_{ij}.\vec{A}_{ji} = 0$$

3.4.2 Gradient estimation

If we want to calculate the generalised surface areas derived above, or want to be able to do a second order gradient reconstruction as for the grid based finite volume methods, we need to be able to estimate gradients on the particle based discretization.

A first approximate way to do this is by using the SPH gradients derived by Price (2012). However, due to the poor volume partitioning properties of SPH, these estimates will be highly dependent on the positions of the particles, which can make them very susceptible to Poisson noise.

To overcome this drawback, Hopkins (2015) suggests using second order accurate, spatial configuration independent locally-centred least-squares matrix gradient estimates. These express the αth component ($\alpha = x, y, z$) of the gradient of the function $f(\vec{x})$ at the position of particle i as

$$\left(\vec{\nabla} f\right)_i^\alpha = \sum_j (f_j - f_i) \bar{\psi}^\alpha{}_j(\vec{x}_i),$$

with

$$\bar{\psi}^\alpha{}_j(\vec{x}_i) = \sum_\beta B_i^{\alpha\beta} (\vec{x}_j - \vec{x}_i)^\beta \psi_j(\vec{x}_i),$$

and the 3×3-matrix $B_i = E_i^{-1}$, with

$$E_i^{\alpha\beta} = \sum_j (\vec{x}_j - \vec{x}_i)^\alpha (\vec{x}_j - \vec{x}_i)^\beta \psi_j(\vec{x}_i).$$

The matrix E_i can be calculated together with the volumes during a first neighbour iteration. Gradients for the primitive variables are then calculated

during a second neighbour iteration, while the flux exchange happens during a third neighbour iteration. This scheme hence requires one neighbour iteration more than SPH.

3.4.3 Interface position and velocity

The second order gradient reconstruction step requires a spatial extrapolation of the primitive variables at the position of the particle to the position of the interface between two particles. However, since this interface is more an abstract notion of an oriented surface area that arises when identifying the equations of finite volume hydrodynamics with the meshless hydrodynamical equations, rather than a real geometrical surface, such a position does not actually exist.

We could adopt a position similar to that of the face of a Voronoi cell, by just using the midpoint of the line segment joining the two neighbouring points. Hopkins (2015) suggests weighing this midpoint by the smoothing length of the particles:

$$\vec{x}_{ij} = \vec{x}_i + \frac{h_i}{h_i + h_j}(\vec{x}_j - \vec{x}_i).$$

Similarly, it is no longer possible to define a geometrical velocity for the abstract interface. We therefore just use a linear interpolation from the particle velocities to the position of the interface:

$$\vec{v}_{ij} = \vec{v}_i + (\vec{v}_j - \vec{v}_i)\left(\frac{(\vec{x}_{ij} - \vec{x}_i).(\vec{x}_j - \vec{x}_i)}{|\vec{x}_j - \vec{x}_i|^2}\right).$$

3.4.4 Results

The result for the spherical overdensity test is shown in Fig. 3.9. This result is just slightly more accurate than the SPH result, but takes 1.89 seconds to calculate. Again, the effective resolution in this simulation is no more than ~ 100 cells, explaining the overall bad accuracy.

3.5 Sources and sinks

Until now, we only considered the Euler equations in the absence of sources and sinks. This means that the movement of the fluid, described by the momentum equation, was purely due to pressure gradients in the fluid. Likewise, the only heating and cooling influencing the energy of the system through the energy equation, was due to the adiabatic contraction and expansion of the fluid.

In real hydrodynamical applications, we would like to include other external sources and sinks as well. In an astrophysical context, the force of gravity is a

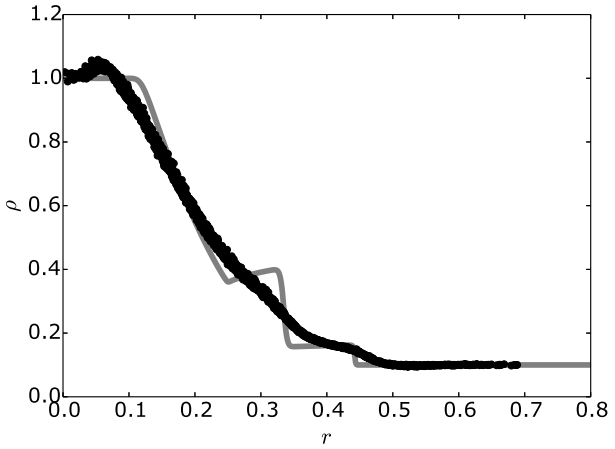


Figure 3.9: *Result of the 2D spherical overdensity test at time $t = 0.1$, obtained using the mesh-free finite volume code* GIZMO*, using the initial particle distribution from Chapter 2, containing 2,780 particles. The black dots are the simulation results, the gray line is the high resolution 1D reference solution.*

good example. We distinguish external gravitational forces due to some (dark matter) potential, and the self-gravity the fluid exerts on itself. The gas in the interstellar medium will loose energy due to radiative cooling, and is heated by the UV light emitted by young stars or coming from the cosmic UV background.

In this section, we will discuss the inclusion of these external sources and sinks in the Euler equations, and discuss how they couple to the existing hydrodynamical integration schemes. Since external forces also exert work on the fluid, they couple to both the momentum and the energy equation. Cooling and heating only influence the latter, and we will discuss them first.

3.5.1 Cooling and heating

The inclusion of a cooling and heating term is most easily done in the thermal energy equation (3.2). The cooling and heating is usually expressed as an intensive property of the gas, i.e. an energy per unit volume. Since the thermal energy is an energy per unit mass, we arrive at the following adapted energy equation:

$$\frac{\partial u}{\partial t} + \left(\vec{v}.\vec{\nabla}\right) u + \frac{P}{\rho}\vec{\nabla}.\vec{v} = \frac{1}{\rho}\left(\mathcal{H} - \mathcal{L}\right),$$

where $\mathcal{H}$ is the heating and $\mathcal{L}$ the cooling.

Returning to the conservation law form of the energy equation, we get

$$\frac{\partial}{\partial t}\left(\rho e\right) + \vec{\nabla}.\left(\rho e\vec{v} + P\vec{v}\right) = \mathcal{H} - \mathcal{L}.$$

The cooling and heating can also be coupled to the entropy equation:

$$\frac{\partial A}{\partial t} + \vec{v}.\vec{\nabla}A = \frac{\gamma - 1}{\rho^{\gamma}}\left(\mathcal{H} - \mathcal{L}\right),$$

which is the form used in standard GADGET2 SPH (Springel & Hernquist, 2002; Springel, 2005).

Any radiative cooling function $\mathcal{L}$ will depend on the temperature of the gas, and hence its thermal energy, so that in all practical applications the cooling vanishes when the thermal energy tends to zero. In principle, the loss of energy can hence never become larger than the thermal energy, so that the total energy can never become zero due to the cooling.

However, in practical integration schemes the cooling is applied as part of the overall integration scheme, using the integration time step deduced from the primitive quantities. Since the classic time step criteria do not take into account cooling and heating explicitly, this might lead to serious over-cooling, effectively leading to unphysical negative energies. To prevent this from happening, care has to be taken when applying the cooling term. We therefore subcycle the integration time step, so that the cooling is applied over smaller time steps, leading to an overall cooling that more closely follows the temperature dependence of the cooling function. In some cases, the net sum of cooling and heating will balance out for a given temperature, meaning that during the time step an equilibrium temperature is reached. This is also taken into account when applying the cooling.

3.5.2 Gravity

The gravitational acceleration due to a gravitational potential Φ is given by

$$a_{\text{grav}} = -\vec{\nabla}\Phi.$$

This can be straightforwardly coupled to the momentum equation:

$$\frac{\partial\left(\rho\vec{v}\right)}{\partial t} + \vec{\nabla}.\left(\rho\vec{v}\vec{v}\right) + \vec{\nabla}P = -\rho\vec{\nabla}\Phi.$$

The gravitational force hence acts as a simple extra term in the momentum equation. For SPH, taking into account gravity then boils down to updating the

particle velocities with the sum of the hydrodynamical and gravitational acceleration, which can be done with minor modifications to the integration scheme. This explains the popularity of SPH in simulations of galaxy formation and evolution.

If the energy is integrated by means of the total energy equation, rather than the thermal energy equation, we need to take into account the change of the kinetic energy due to the change in gravitational potential. The work done by the gravitational force to move a mass m, moving at a velocity $\vec{v}$, over a distance $\mathrm{d}\vec{x}$ during a time interval $\mathrm{d}t$ is given by

$$W_{\mathrm{grav}} = \vec{F}_{\mathrm{grav}}.\mathrm{d}\vec{x} = -m\vec{\nabla}\Phi.\vec{v}\mathrm{d}t.$$

So that the change in energy for the fluid in the volume element $\mathrm{d}V$ will be given by $-\rho\vec{v}.\vec{\nabla}\Phi$:

$$\frac{\partial}{\partial t}\left(\rho e\right) + \vec{\nabla}.\left(\rho e\vec{v} + P\vec{v}\right) = -\rho\vec{v}.\vec{\nabla}\Phi.$$

This is the energy equation we need to integrate in a finite volume method in the presence of a gravitational potential.

As Springel (2010) notes, there are serious issues with energy conservation when just treating the gravitational work as an extra term in the energy equation. By using the expression above, we implicitly assume that the entire mass of a cell of the (co-moving) grid moves with velocity $\vec{v}$. However, we know from the hydrodynamical integration scheme that this is not the case: part of the mass of the cell will leave the cell during the time step due to the mass flux from the cell to its neighbouring cells. This mass flux will contribute a small fraction of gravitational work to the energy contents of the cell. The gravitational energy flux for a cell with mass m_i, moving with velocity $\vec{w}_i$ is hence given by

$$\Delta E_{i,\mathrm{grav}} = -m_i\vec{\nabla}\Phi.\vec{w}_i\Delta t - \frac{1}{2}\sum_j \Delta m_{ij}\left(\vec{r}_i - \vec{r}_j\right).\vec{\nabla}\Phi,$$

where Δm_{ij} is the mass flux from cell i to cell j, and the sum extends over all neighbouring cells. Note that in this scheme, we explicitly replace the fluid velocity $\vec{v}_i$ by the cell velocity $\vec{w}_i$, since it is the actual movement of the cell that causes the gravitational work, not the movement of the fluid.

Just as in the case of cooling and heating, applying the extra term in the energy equation can cause the energy to become negative. However, in this case there is no clear physical way to prevent this from happening, since the gravitational work does not depend directly on the energy of the system. In very cold fluids with large bulk velocities, the gravitational work might cause a severe over-flux of energy, which can only be handled by using an appropriate gravitational time step criterion combined with the usual hydrodynamical time step criterion, and by including a strict flux limiter.

3.6 Overview

In this chapter, we introduced the Euler equations and their solutions, and laid out the basics for four different hydrodynamical integration schemes: a finite volume AMR scheme using a (refined) Cartesian grid, a finite volume moving mesh method, classical SPH and a mesh-free finite volume method. All have specific advantages and disadvantages, but all solve the same equations, so that they should in principle yield similar results. Here, we list the most important differences and try to indicate when a particular method is preferred.

Note that the list of methods above is non-exhaustive: there are other methods to solve the Euler equations that were not mentioned in this chapter. Examples are finite difference methods (Smith, 1985), and finite element methods (Zienkiewicz *et al.*, 2005), which are often used as an alternative for the finite volume method on a Cartesian grid.

3.6.1 Eulerian versus Lagrangian

In general, Eulerian methods are computationally cheaper than Lagrangian methods for the same number of discretization elements, since the underlying geometrical structures are independent of time. This makes these methods more efficient in obtaining a high resolution result for a given problem.

For problems involving a high dynamical range in space, AMR is needed to keep Eulerian methods efficient. This leads to problems when the high density region moves with respect to the fixed grid, since then the refined region has to move along.

Eulerian methods are also at a disadvantage when the fluid as a whole moves with respect to the fixed grid, since they are not Galilean invariant. This becomes especially apparent when the fluid moves at high Mach numbers, with the Mach number being defined as the ratio $M = v/c_s$. Not all Eulerian methods can handle these cases, and those that can need to adapt their time step accordingly, making them in most cases less efficient than Lagrangian methods.

For high resolution cosmological simulations that include baryonic physics (Vogelsberger *et al.*, 2014b; Schaye *et al.*, 2015), Eulerian methods cannot be used, since the combination of a high dynamic range and high Mach number movement of the halos (Mach numbers of 300 and more) makes them essentially useless. Since many simulations of isolated halos are zoom simulations based on these cosmological simulations, Lagrangian methods are commonly used in this type of simulations as well.

Eulerian cosmological simulations are used however when the grid is more important than the resolution inside the halos. To study cosmic voids for example, a high resolution is needed in regions with little to no matter, which is impossible with a Lagrangian method (Kreckel *et al.*, 2011).

For other types of simulations, Eulerian methods usually perform better than Lagrangian ones, both computationally and accuracy-wise. It is also easier to extend them with extra equations, like the equations of Magneto-Hydrodynamics (MHD). In problems involving self-gravity, Lagrangian methods are however preferred, since these methods couple more naturally to N-body methods.

3.6.2 Grid versus particles

Classical SPH is clearly at a disadvantage with respect to the other methods, since it is incapable of resolving discontinuities. However, in simulations of galaxy formation and evolution, these discontinuities play no role, and the only effect of using improved versions of SPH turns out to be somewhat better mixing properties at high resolution (Schaller *et al.*, 2015). We will therefore still use classical SPH for the dwarf galaxy simulations in Chapter 6.

When including other equations, like those of MHD, SPH is usually abandoned in favour of grid based methods (but see Price, 2012). For galaxy simulations including MHD, other Lagrangian methods are used (Pakmor *et al.*, 2011; Hopkins & Raives, 2016).

Simulations of ram-pressure stripping crucially depend on resolving the discontinuities at the interface between the cold, dense interstellar medium in the dwarf galaxy and the hot, diffuse intergalactic medium surrounding the massive galaxy or cluster. For these simulations, classical SPH cannot be used. For future simulations of this effect, we will hence use either a moving mesh or a mesh-free approach.

4

Shadowfax

In Chapter 2 and Chapter 3, the details of a hydrodynamical integration scheme based on an unstructured Voronoi mesh were given. I have implemented the relevant algorithms in the simulation code Shadowfax, which can be used to evolve a mixture of gas, subject to the laws of hydrodynamics and gravity, and any collisionless fluid only subject to gravity, such as cold dark matter or stars. We will now discuss this simulation code in more detail.

Shadowfax is written in C++ and makes ample use of the object oriented capabilities of the language. It is parallelized for use on distributed memory systems by means of MPI[1]. Some of its features make use of the open source Boost C++ libraries[2] to extend basic C++ functionality, e.g. to read in parameters from a .ini file or to use extended precision integer arithmetics (see Appendix A). Optionally, it is also possible to expose some of the functionality of the code to Python scripts through the Boost Python library[3].

To allow for a user friendly compilation process, we make use of the automatic build file generation system CMake[4]. The program consists of a main simulation program, consisting of the binary programs `shadowfax2d` and `shadowfax3d`, and up to three auxiliary programs to generate initial condition files, apply Lloyd's algorithm to obtain the centroids of a Voronoi mesh, and generate .vtk files to plot the Voronoi mesh in a visualisation program (called respectively `icmakerXd`, `lloydXd` and `vtkmakerXd`, with `X` equal to 2 or 3, depending on the number of dimensions). To speed up the compilation process, common parts of these programs are precompiled as static libraries that are then linked into the appropriate program.

Input and output of (possible very large) initial condition and snapshot files is done using the HDF5 library[5], in a format that is compatible with the formats used in the public simulation codes Gadget2, gizmo, and swift. This makes it

[1] `http://www.mpi-forum.org`
[2] `http://www.boost.org`
[3] `http://www.boost.org/doc/libs/release/libs/python/doc/index.html`
[4] `https://cmake.org`
[5] `https://www.hdfgroup.org/HDF5`

extremely easy to compare all these codes on the same test problems. The same format was also used for large simulation projects, e.g. the EAGLE simulation (Schaye *et al.*, 2015). Other advantages of HDF5 are the existence of a toolset to read and edit these files (both using the command line or the interactive `hdfview` program[6]), and the existence of user-friendly python bindings (the `h5py` package[7]).

Shadowfax is free software, and is publicly available[8] under the GNU Affero General Public Licence[9]. For more details, see Vandenbroucke & De Rijcke (2016).

In this chapter, we will give an overview of the program structure, and highlight some of its main features. We will also discuss ways to visualise Shadowfax output, and demonstrate the capabilities of the code on a number of test problems.

4.1 Program structure

The program execution can be roughly divided into two phases: program initialisation, and the execution of the actual simulation. Program initialisation consists of the setup of the initial condition, and the initialisation of the computational resources for the simulation. The former has been strictly separated from the main simulation program by the use of *initial condition files*. These have the same structure as the snapshot files that are written during the course of the simulation, but specify the primitive hydrodynamical variables at the beginning of the simulation. The main simulation program is then nothing more than a program that reads in the initial condition, and writes out snapshots.

4.1.1 Initial condition generation

To set up the initial conditions for the simulation, we need to create an initial condition file. This file should contain the coordinate positions of the cell generators, and the values of the primitive variables for the cells. It also assigns a unique long integer ID to each generator, so that we can relate generators in between different snapshots. If dark matter or other types of particles are to be used in the simulation, their positions, masses, IDs and velocities should also be specified in the initial condition file. Apart from this, the initial condition file should also specify the dimensions of the simulation box, the total number of particles for each type (currently Shadowfax supports gas particles (cells) and

[6] `https://www.hdfgroup.org/products/java/hdfview`
[7] `http://www.h5py.org`
[8] `https://github.com/AstroUGent/shadowfax`
[9] `http://www.gnu.org/licenses`

cold dark matter particles), and the type of boundary conditions used for the box.

Creating initial condition files can be done in various ways. First of all, it is possible to use a snapshot from a previous simulation as initial condition file, since the format of these files is identical. The snapshot format of SHADOWFAX is also equal to that of GADGET2 (type 3), GIZMO and SWIFT, so that we can also use output from one of these codes as input for a SHADOWFAX simulation. When writing an initial condition from scratch, it is possible to either use the `icmakerXd` program, or use a Python script and the `h5py` library.

`icmaker2d` and `icmaker3d`

`icmakerXd` is a program that uses SHADOWFAX functionality to create initial condition files. There are two modes of operation: the first uses hardcoded methods to specify the values of the primitive variables and the positions of the cell generators for some specific set of problems, while the second mode of operation allows the user to specify the hydrodynamical *regions* of the initial condition in a syntax that was based on that used for the AMR code RAMSES[10] (Teyssier, 2002). The program allows sampling Cartesian grids as well as random unstructured meshes, and allows to specify the number of cells (although this number is restricted to a specific set of possible values for a Cartesian grid).

To suppress Poisson noise in randomly sampled generator distributions, the SHADOWFAX Voronoi mesh is used to regularise the initial mesh using Lloyd's algorithm. By default, 10 iterations are used for both 2D and 3D initial conditions when using a random unstructured mesh. When using the second mode of operation, random sampling is biased towards regions with high density, to obtain particle distributions that represent the underlying density distribution (see Chapter 2).

For the region syntax of the second mode of operation, we conceive the simulation box as a combination of geometrical building blocks. For every building block, we specify a geometrical shape and values for the primitive variables inside the region. Building blocks can overlap, in which case the values for the last region in the list are used for the overlap region. The regions are defined in a .xml file.

The file for generating the 2D spherical overdensity test used in Chapter 3 is given in Listing 4.1. We first specify the dimensions of the simulation box, by specifying an origin and a height, width, and depth (the latter only for 3D initial conditions). We then need to specify the type of boundary conditions used, in this case reflective boundaries. We then proceed to specify two regions: the low density region that fills the entire simulation box, and the high density spherical

[10] `http://www.itp.uzh.ch/~teyssier/ramses/RAMSES.html`

```
<?xml version="1.0"?>
<box>
  <height>1.0</height>
  <width>1.0</width>
  <origin>
    <x>0.5</x><y>0.5</y><z>0.5</z>
  </origin>
  <boundary>reflective</boundary>
  <regions>
    <region>
      <height>1.0</height>
      <width>1.0</width>
      <exponent>10.0</exponent>
      <origin>
        <x>0.5</x><y>0.5</y><z>0.5</z>
      </origin>
      <hydro>
        <rho>0.1</rho><vx>0.</vx><vy>0.</vy><p>0.1</p>
      </hydro>
    </region>
    <region>
      <height>0.5</height>
      <width>0.5</width>
      <exponent>2.0</exponent>
      <origin>
        <x>0.5</x><y>0.5</y><z>0.5</z>
      </origin>
      <hydro>
        <rho>1.</rho><vx>0.</vx><vy>0.</vy><p>1.</p>
      </hydro>
    </region>
  </regions>
</box>
```

Listing 4.1: *Example .xml file for the 2D spherical overdensity test.*

region in the center. By first specifying the former, we make sure that the latter is used in the spherical overlap region between the two.

Each region has an origin and dimensions as well. However, there now is also an extra parameter, the *exponent* of the region. This specifies how distances should be calculated inside the region. The general formula for the distance between the point with coordinates $\vec{x}$ and the origin $\vec{o}$ in a region with exponent e and dimensions $\vec{d}$ is given by.

$$\mathrm{dist}(\vec{x},\vec{o}) = \left[\left(\frac{x_x - o_x}{0.5d_x}\right)^e + \left(\frac{x_y - o_y}{0.5d_y}\right)^e + \left(\frac{x_z - o_z}{0.5d_z}\right)^e\right]^{\frac{1}{e}}.$$

To find out if a point lies inside the region, we calculate the distance between the origin and the point, using this formula. If the distance is smaller or equal than 1, the point is said to lie inside the region.

The exponent hence sets the shape of the region. If we consider only 2D and assume all components of the dimensions $\vec{d}$ to be the same, then $e = 1$ corresponds to a diamond, and $e = 2$ corresponds to a disc. e very large will lead to a very small $1/e$, which will lead to all points inside the square with dimensions $\vec{d}$ to be accepted, so that we end up with a square region. Due to numerical precision, $e = 10$ is large enough to define a square region. If the components of $\vec{d}$ are not the same, rectangle based shapes are obtained.

This simple syntax only allows for geometrical regions with constant primitive variables. To also allow more complex initial conditions, we added support for mathematical expressions, using addition, subtraction, multiplication and division, and the special functions `cos`, `sin`, `tan`, `sinh`, `cosh`, `tanh`, `log`, `exp`, `acos`, `asin`, `atan`, `log10`, `sqrt` and `cbrt`. We also support the use of brackets, the mathematical constant `pi`, and the coordinates `x`, `y`, `z` and `r`. The latter is always calculated with respect to the origin of the region (using the ordinary definition $r = \sqrt{(x_x - o_x)^2 + (x_y - o_y)^2 + (x_z - o_z)^2}$), while the former are absolute. To support this kind of syntax, we make use of Boost Spirit[11], a versatile regular expression parsing library.

To bias the random sampling towards dense regions in complex setups using non-constant regions, we numerically integrate the density function over each region, to obtain relative weights for the different regions. Sampling inside a single region is then done using a rejection sampling technique, as discussed in Chapter 2.

Python scripting

Since the initial condition files are written in HDF5 format, it is possible to create them directly using the `h5py` library. An example creating the initial condition

[11] `http://www.boost.org/doc/libs/release/libs/spirit/doc/html/index.html`

```
import h5py
import numpy as np

ncell = 40
npart = ncell**2
rhoIC = [0.1, 1.]
pIC = [0.1, 1.]
rsphere = 0.25
boxL = 1.

coords = np.zeros((npart, 3))
v      = np.zeros((npart, 3))
m      = np.zeros((npart, 1))
rho    = np.zeros((npart, 1))
h      = np.zeros((npart, 1))
u      = np.zeros((npart, 1))
ids    = np.zeros((npart, 1), dtype='L')

dx = boxL/ncell

idx = 0
for i in range(ncell):
  for j in range(ncell):
    coords[idx,0] = (i+0.5)*dx
    coords[idx,1] = (j+0.5)*dx
    v[idx,0] = 0.
    v[idx,1] = 0.
    v[idx,2] = 0.
    r2 = (coords[idx,0]-0.5)**2 + (coords[idx,1]-0.5)**2
    if r2 < rsphere**2:
      rho[idx] = rhoIC[1]
      P = pIC[1]
    else:
      rho[idx] = rhoIC[0]
      P = pIC[0]
    u[idx] = 1.5*P/rho[idx]
    ids[idx] = idx
    idx += 1
```

Listing 4.2: *Example .py file for the 2D spherical overdensity test: setting up the variables.*

```
file = h5py.File("Overdensity.hdf5", 'w')

grp = file.create_group("/Header")
grp.attrs["BoxSize"] = boxL
grp.attrs["NumPart_Total"] =  [npart, 0, 0, 0, 0, 0]
grp.attrs["NumPart_Total_HighWord"] = [0, 0, 0, 0, 0, 0]
grp.attrs["NumPart_ThisFile"] = [npart, 0, 0, 0, 0, 0]
grp.attrs["Time"] = 0.0
grp.attrs["NumFilesPerSnapshot"] = 1
grp.attrs["MassTable"] = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
grp.attrs["Flag_Entropy_ICs"] = [False, False, False,
                                 False, False, False]

grp = file.create_group("/RuntimePars")
grp.attrs["PeriodicBoundariesOn"] = 1

grp = file.create_group("/Units")
grp.attrs["Unit␣current␣in␣cgs␣(U_I)"] = 1.
grp.attrs["Unit␣length␣in␣cgs␣(U_L)"] = 100.
grp.attrs["Unit␣mass␣in␣cgs␣(U_M)"] = 1000.
grp.attrs["Unit␣temperature␣in␣cgs␣(U_T)"] = 1.
grp.attrs["Unit␣time␣in␣cgs␣(U_t)"] = 1.

grp = file.create_group("/PartType0")
ds = grp.create_dataset('Coordinates', (npart, 3), 'd')
ds[()] = coords
ds = grp.create_dataset('Velocities', (npart, 3), 'f')
ds[()] = v
ds = grp.create_dataset('Masses', (npart,1), 'f')
ds[()] = m
ds = grp.create_dataset('Density', (npart,1), 'f')
ds[()] = rho
ds = grp.create_dataset('SmoothingLength', (npart,1), 'f')
ds[()] = h
ds = grp.create_dataset('InternalEnergy', (npart,1), 'f')
ds[()] = u
ds = grp.create_dataset('ParticleIDs', (npart, 1), 'L')
ds[()] = ids

file.close()
```

Listing 4.3: *Example .py file for the 2D spherical overdensity test: writing the HDF5 file.*

for the 2D spherical overdensity test (on a Cartesian grid) is given in Listing 4.2 and Listing 4.3.

The advantage of this method is that it is very easy to set up complex initial conditions, as it is possible to use the full strength of Python in specifying coordinates and primitive variables. `h5py` communicates well with `numpy`[12], a very powerful library for numerical calculations.

However, the disadvantage is that Python has no native support for Voronoi meshes (there is limited support for 2D Voronoi meshes using `scipy`[13]), meaning we cannot easily use Lloyd's algorithm to regularise random distributions.

4.1.2 Program initialisation

The main `shadowfaxXd` program takes a single command line parameter, specifying the name of a *parameter file*, in .ini format, containing the runtime parameters for the simulation. These parameters control the input and output of the simulation, the time integration, the accuracy of the integration... A complete overview of these parameters and their default values is given in the example .ini file in Listing 4.4 and Listing 4.5. Most of these parameters have a default value, also indicated in the table. The only exception is the maximal simulation time, which needs to be set in order for the program to run.

When the program starts, the parameter file is read in. We then locate the initial condition file (indicated in the parameter file) and read in the particle data. If the program is run in parallel, only part of the data is read in on each individual node, so that the memory usage on a single node is limited. The particles are then redistributed over the nodes as part of the initial domain decomposition and tree construction.

Once the particle data is loaded, the simulation time line is initialised. We then construct the initial Voronoi mesh, and use it to convert the primitive variables from the initial condition file to the conserved variables which will be integrated. The initial Voronoi mesh is also used to set the initial velocities of the mesh generators and to set the variables necessary to calculate the initial time steps.

Once conserved variables have been calculated and the initial time steps are set, we can start the main program loop.

4.1.3 Main program loop

The main program loop combines a leapfrog scheme for the collisionless particles with a simple kick-drift integration scheme for the hydrodynamics. It is imple-

[12] `http://www.numpy.org`
[13] `http://www.scipy.org`

```
[Time]
; Total simulation time
MaxTime = -
; Use a global time step?
GlobalTimestep = false
; Maximum size of the particle time step
MaxTimeStep = <MaxTime>
; Minimum size of the particle time step
MinTimeStep = <MaxTime/(2^60)>

[Snapshots]
; Prefix for snapshot name
BaseName = snapshot
; Time between subsequent snapshots
SnapTime = <0.1*MaxTime>
; Index number of first snapshot that is effectively
; written out
FirstSnap = 0
; Directory in which snapshots, log-files and restart
; files will be stored
OutputDir = .
; Type of the snapshot files (Gadget/Shadowfax)
Type = Gadget
; Does every node write a local snapshot?
PerNodeOutput = false

[IC]
; Name of the initial condition file
FileName = <BaseName><FirstSnap>.hdf5
; Type of the initial condition file (Gadget/Shadowfax)
Type = Gadget

[RiemannSolver]
; Type of Riemann solver used (Exact/TRRS)
Type = Exact
; Tolerance used for Newton-Raphson iteration
Tolerance = 1.e-8
; Deprecated parameter used in exact Riemann solver
CutOff = -5.
; Courant-Friedrichs-Lewy parameter for time step criterion
CFL = 0.4
```

Listing 4.4: *Example .ini file with default values, part 1.*

```
[Hydro]
; Adiabatic index for the gas
Gamma = <5/3>

[Gravity]
; Use gravity?
Gravity = true
; Softening length used for particles with a fixed
; softening length
Softening = 0.03
; Eta factor used in gravitational time step criterion
Eta = <0.05/2.8>
; Alpha factor used in relative tree opening criterion
Alpha = 0.005

[Voronoi]
; Deprecated parameter used during grid construction
Tolerance = 1.e-9

[Tree]
; Side of the grid on which the Ewald correction to the
; gravitational force or mesh movement is precomputed
EwaldSize = 64
; Parameter alpha determining the cutoff between short
; range and long range in Ewald's method
EwaldAlpha = 2.

[Memory]
; Maximum size of the MPI-buffer in memory
MaximumSize = 1 GB

[Code]
; Computational time interval after which to write a
; restart file, in seconds
RestartTime = 3600.

[Units]
; Units used during the simulation (SI/CGS/galactic)
InternalUnits = SI
; Units used in snapshots (SI/CGS/galactic)
OutputUnits = SI
```

Listing 4.5: *Example .ini file with default values, part 2.*

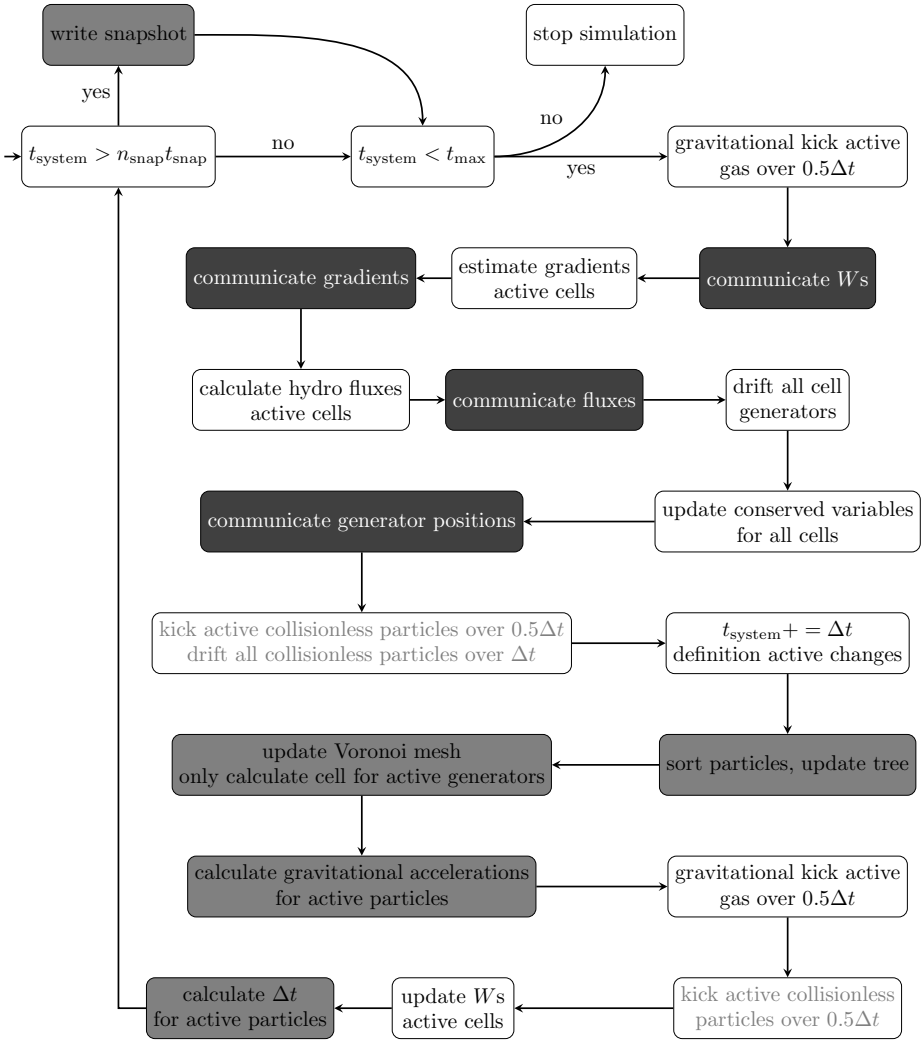


Figure 4.1: *Flowchart of the main* SHADOWFAX *simulation loop. The small arrow at the top left indicates the entrance point of the loop. Dark gray boxes indicate communication steps that are only necessary in parallel runs. Light gray boxes indicate steps that involve communication in parallel runs, but are also necessary in serial runs.*

mented as a loop, with the system time as loop variable. The structure of the loop is depicted in Fig. 4.1. At the end of an iteration, the system time is compared with the maximum simulation time parameter. If the system time equals the maximum simulation time, the loop is ended and the simulation stops. Together with this check, we also check if a snapshot needs to be written. To this purpose, we store the integer counter of the last snapshot that was written (initialised with the value of the `FirstSnap` parameter). When the time interval between subsequent snapshots (the `SnapTime` parameter) multiplied with this counter becomes smaller or equal to the current system time, a new snapshot needs to be written.

Note that this simple scheme does not produce snapshots at system times that are not covered by the integration scheme. In other words, suppose the entire simulation consists of 8 time steps between 0 and 1: 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875 and 1 (which takes into account the power of two subdivision of the maximum simulation time, see Chapter 3). Suppose now that we want 5 snapshots, spaced 0.2 in time. Then we will get snapshots at times 0.25, 0.5, 0.625, 0.875 and 1. For this particular example, the actual snapshot interval is very different from the desired 0.2, but in most cases, the deviations will be a lot smaller.

Note also that the scheme can produce less snapshots than expected. If in the example above the snapshot interval is set to 0.1, we will get a snapshot at every system time, but we will only have 8 snapshots in total, and not the expected 10.

It should be possible to resolve the problems above by either adapting the maximal system time step to make sure that enough snapshots are produced (this does not solve the first problem), or by drifting the primitive variables and positions to the actual time of the snapshot when necessary. The former affects the integration directly and is not desired. The second is currently not implemented, but might be in the future. It however requires a significant adaptation of the time line implementation.

4.1.4 Restarting

Large simulations run in parallel on a large number of nodes on special purpose machines, and run for a long time (up to months). Since these large machines are used intensively by many different users, and since large machines are inevitably vulnerable to hardware failure, there is a large probability that the simulations will be interrupted multiple times during the course of a run. It is in principle possible to restart an interrupted run from the last snapshot that was written (remember that initial condition files have the same format as snapshot files), but this requires adapting the parameter file, and can lead to numerical error, since snapshots do not necessarily contain the correct values for all the cells at

the time they are written (if individual time steps are used).

To allow for easy restarting of a run, we periodically write specific restart files. These are simple binary dumps of the memory as it is, which can be read in very efficiently to restart the run as if it was never interrupted. For this to work, it is of course not possible to change the computational resources of the simulation when restarting, and it is also not advised to change the code itself. We therefore implemented strict checks on the hardware layout and the code version and compilation time when reading and writing restart files. Some changes will lead to warnings, while most will lead to errors that make it impossible to restart the run.

To restart a run, the restart file rather than the parameter file is given as command line argument to the `shadowfaxXd` program. This file is then used in a separate restart (rather than initialisation) routine to re-initialise the run. When this is done, the main simulation loop is entered again and the simulation resumes as before.

4.2 Special features

4.2.1 Domain decomposition and tree construction

Domain decomposition is the subdivision of the simulation box into smaller regions that are then assigned to different processes in a distributed memory parallel environment. This is done to limit the memory usage of a single process. During the parallel computation, a process will only perform a part of the computation, which will only require part of the data, so that it would be unwise to store all data for every process. However, it can of course happen that a single piece of data is needed in the computation on multiple processes, so that simply splitting up the data will not work either. Nonetheless, this is the approach we will take: apart from some very general data which are stored on every process, all particle data is split up over the different computing processes. If the computation on process A requires data stored on process B, then this is signalled and the relevant data is *communicated* from B to A.

Since communication inevitably leads to extra work compared to a non-parallel simulation, and since on many systems communication between process A and B requires some form of synchronisation between A and B (which might require one of them to interrupt its computations and *wait* for the other), we would like to minimise the amount of communication. Otherwise, communication will require a considerable fraction of the simulation runtime, which will decrease the performance gain we get by doing the simulation in parallel.

To limit the amount of communication, we need to identify the parts of the simulation which are communication intensive, and we need to adapt the domain

decomposition. For the latter, it is important to realise that communication will occur mainly at the boundaries between domains: the finite volume method exchanges fluxes between neighbouring cells and if these neighbouring cells are on different processes, this implies communication. Similarly, the calculation of gravitational forces is done directly for particles that are close together (in space), while for particles far apart, approximate methods involving only globally stored data are used. We hence need to minimise the *surface area* of the domain boundaries, while at the same time maximising the domain *volume*.

The best surface to volume ratios are obtained for domains that have the form of Voronoi cells (Steinberg *et al.*, 2015), but computing and maintaining these domains is hard. We will therefore use a domain decomposition consisting of large cubic blocks, which could for example be obtained by recursively splitting the simulation box into eight smaller boxes. This approach only works if the number of processes is a power of two, since only then is it possible to divide such blocks equally over the processes. Furthermore, this approach also only works if the data are uniformly distributed in the simulation box, since only then will blocks with equal sizes correspond to equal computational loads.

Since none of these requirements is generally satisfied, we will use a more flexible domain decomposition, based on *space-filling curves*. A space-filling curve is a 1D curve that winds through 2D or 3D space in a continuous way, without ever intersecting itself, in such a way that it fills the entire space (up to some level). They have the property that two points that are close together on the curve, are also close together in the 2D or 3D space (but points close together in 2D or 3D space are not necessarily close together on the curve). An example of the space-filling Hilbert curve is shown in Fig. 4.2 and Fig. 4.3.

Space-filling curves are obtained by converting the coordinates of a point in 2D or 3D space to an integer key, which corresponds to the 1D coordinate of the same point along the curve. If we calculate the keys for a set of points and then sort the set according to these keys, then the points are in space-filling order. We can then split up the space-filling curve in pieces by selecting key ranges and assigning these to different processes. Since points that are close together on the space-filling curve are close together in space, these pieces will consist of block-like structures as defined above, but with somewhat more complex structures. However, the domains will still have a large, cohesive volume and a relatively small surface area.

Using these space-filling curves has two extra advantages. First of all, there exist very efficient parallel sorting algorithms to sort sets of points (or particles in our case) across processes (Siebert & Wolf, 2010). This allows for a very efficient domain decomposition. It is possible to assign weights to the particles (for example based on the computational cost of the particle during the previous time step), so that the space-filling curve can be split up based on the computational

weight rather than just the number of particles.

Secondly, there is a correspondence between the levels of the space-filling curve, and the levels of the octree used for calculating the gravitational accelerations and to perform neighbour searches (Springel, 2005). To see this correspondence, we need to explain how an octree is constructed and how the space-filling Hilbert keys are calculated.

Note that a geometrical domain decomposition as described above is not the most efficient way to divide up a parallel computation. Gonnet (2014) advocates the use of a work-based domain decomposition, which uses a graph of the task dependencies to minimise the amount of communication. This however only works well for algorithms that have clearly defined tasks and dependencies, which is not the case for the current version of SHADOWFAX.

Space-filling curve

To calculate a 64-bit Hilbert key, we first need to convert the 2 or 3 coordinates of the particle to 32- or 21-bit integers. This can be done by mapping the domain of the simulation in every dimension to an integer domain with a 32- or 21-bit precision. Once we have the integer coordinates, the key is obtained by *interleaving* parts of the key. This works as follows. We start by taking the first digits (being the most significant bit) of the integer coordinates for every dimension, and combining them into a single 2- or 3-bit number. If these bits are labelled b_x, b_y and b_z, the number is e.g. given by $b_x * 4 + b_y * 2 + b_z$, which is the number obtained by putting b_x on the position of the most significant bit, and so on.

The 2- or 3-bit number is called the level 1 key of the set of coordinates, and it tells us exactly in which of the four or eight parts of the simulation box the coordinates lie if we would subdivide the box in smaller copies of the box. With every one of these four or eight boxes, we associate a *rotation angle*. Before continuing to the next level, we rotate the integer coordinates over this angle. We also set the highest bits of the (empty) Hilbert key to the level 1 key of the coordinates.

After the rotation has been performed, we continue by taking the second digits of the integer coordinates. These are again converted to a 2- or 3-bit number, the level 2 key. This again tells us in which one of the four or eight subboxes of the level 1 box we reside, and associates a rotation to this box. We then again rotate the coordinates and add the level 2 key to the next empty bits in the Hilbert key. This procedure is repeated until the last (least significant bits) of the coordinates have been processed. We then end up with a 64- or 63-bit Hilbert key (we loose a bit in 3D). Examples of two level Hilbert curves are shown in Fig. 4.2 and Fig. 4.3.

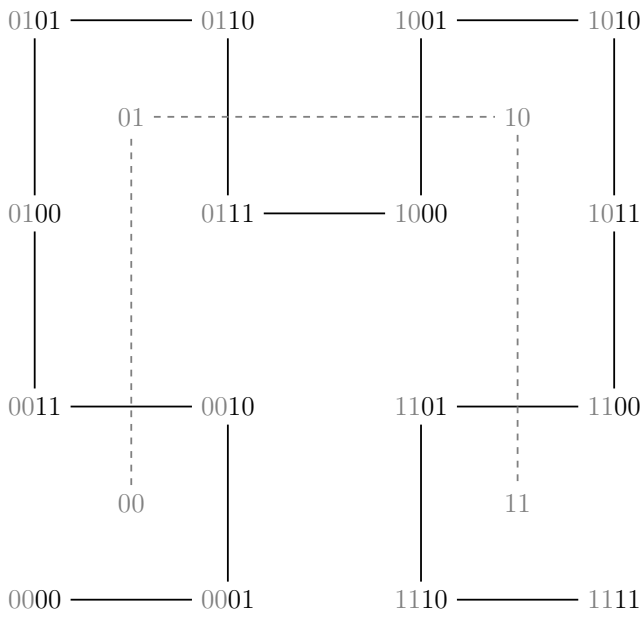


Figure 4.2: *2D space-filling Hilbert curve. The black lines represent the level 2 curve, derived from the gray level 1 curve by applying a block specific rotation to a copy of the level 1 curve. The labels are a binary representation of the corresponding Hilbert key values.*

The rotations that are performed when going from one level to the next make sure that the curve always stays connected, so that the length of the line segment in between two consecutive points on the curve is always the same. If we would not perform these rotations, we end up with a *Morton* space-filling curve, which has similar properties, but has a large jump in between high level boxes.

The rotations can be expressed by relatively easy binary operations on the integer coordinates, so that the Hilbert key calculation is computationally cheap. It is however possible to speed up the calculation even more by introducing a look-up table (Jin & Mellor-Crummey, 2005). To this end, we store the 8 or 12 (2D and 3D respectively) possible orientations of the level 1 U-shape that is replicated on each level. For every orientation, we store the key on that level as a function of the Morton key on that level (obtained by simply interleaving the 2 or 3 coordinate contributions on that level), together with the index of the orientation on the next level. To calculate the key, we need only walk trough the table and add the level keys.

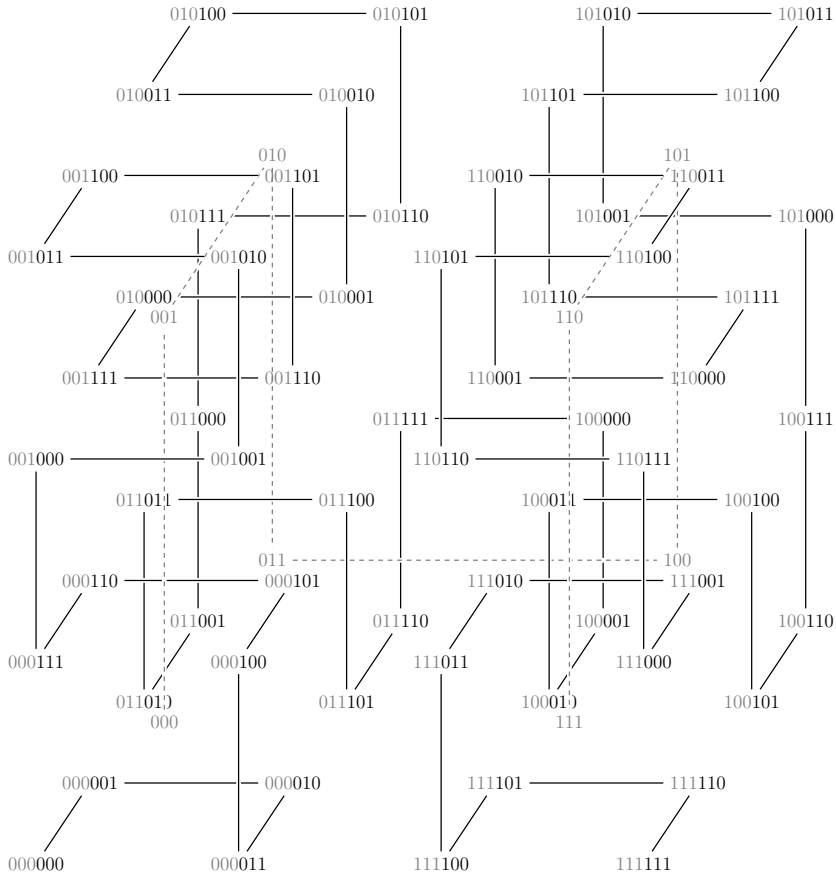


Figure 4.3: *3D space-filling Hilbert curve. The black line represent the level 2 curve, derived from the gray level 1 curve by applying a block specific rotation to a copy of the level 1 curve. The labels are a binary representation of the corresponding Hilbert keys.*

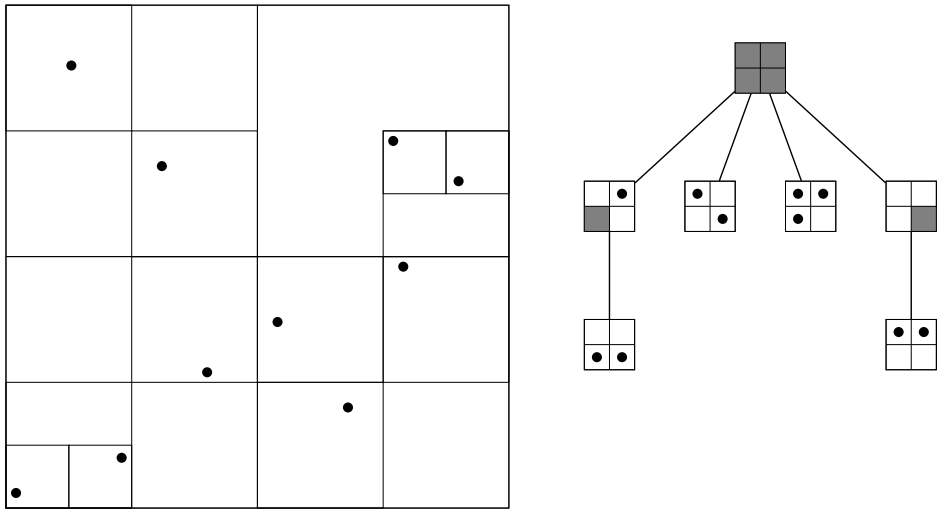


Figure 4.4: *2D particle distribution and corresponding octree.*

Octree

The octree is a complex structure, consisting of *nodes* and *leaves*. During tree construction, a node is a simple (rectangular) box with a center or anchor, and two or three side lengths. It also holds a list of four or eight children, which are themselves either nodes or leaves. A leaf corresponds to a single particle of the tree. Examples of 2D and 3D octrees are shown in Fig. 4.4 and Fig. 4.5.

The tree itself needs to be nothing more than a pointer or a reference to the highest level node of the tree, which we call the *root*. Initially, this root will be empty. During *incremental tree construction*, particles are added one-by-one to the tree, and the tree structure is adjusted accordingly.

To add a particle, we first need to figure out to which one of the four or eight children of the root node it belongs, based on the particle coordinates and the dimensions of the root node. If the corresponding entry in the list of children is empty (which it is initially), we add a new leaf pointing to the particle to that entry. If the entry corresponds to a leaf, we need to replace the leaf by a new node, to which we then add both the particle in the old leaf, and the new particle. If the entry corresponds to a node, we add the particle to the node, which means starting over the whole process for that node. We need to go recursively deeper into the tree until the particle can be stored in a new leaf of the tree.

After the incremental construction is finished, we restructure the tree, so that every node contains two pointers: one to the next node of the tree that should be

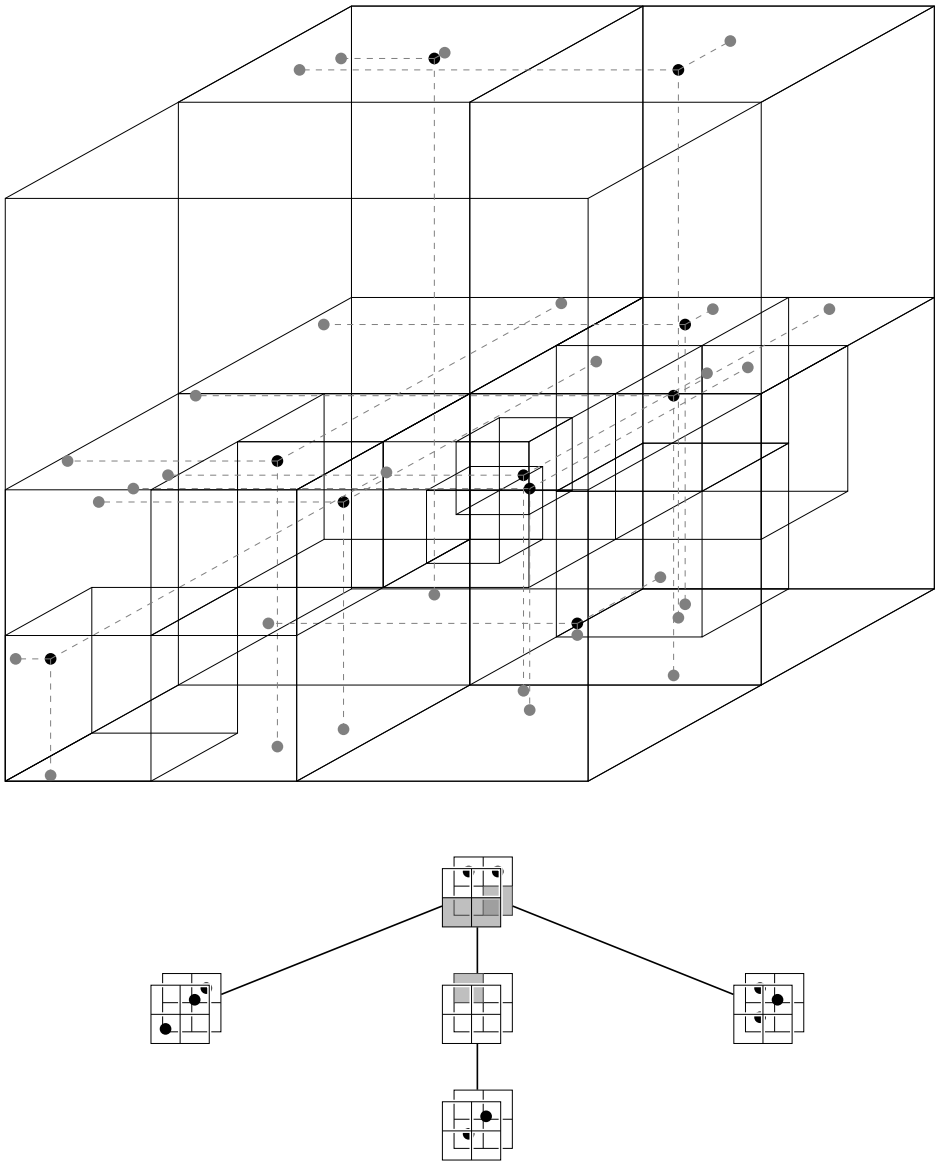


Figure 4.5: *3D particle distribution and corresponding octree.*

handled if the node were to be discarded (which is either a *sibling* of the current node, or the sibling of one of its ancestors), and one to the first child of the node that should be handled if the node is opened. A leaf contains a single pointer, pointing to the sibling of the leaf that is next in the tree walk. Walking through the tree then boils down to following these pointers. The memory that is freed up by no longer storing the 3 or 7 other children of a node is used to store general node properties, like its center of mass and total mass.

Key-based octree

The levels of the tree and the levels of the Hilbert key are actually the same. While constructing the Hilbert key, we noted that the level 1 key tells us in which one of the four or eight boxes the particle lies. These boxes are nothing else than the child nodes of the root node of the tree. The level 1 key hence tells us exactly in which subnode of the root the particle resides. If we go down a level, then the level 2 key will do exactly the same, but then for the children of that specific node (and taking into account the appropriate rotation). The correspondence goes all the way down to the least significant bits of the key.

This means we do not need to use the particle coordinates themselves to figure out what child of a specific node to test during tree construction. We rather just look at the corresponding part of the key, which is much easier. Furthermore, this method also eliminates problems that could arise due to round off error when particles are very close to the boundaries of one of the high level nodes.

We could go even further and use the keys directly to build up the tree in a bottom-up rather than a top-down manner (Sundar *et al.*, 2008), but this was found to be less efficient than using the incremental construction algorithm.

Note that this procedure does limit the maximal depth of the tree, since the length of the key is limited to 32 levels in 2D and to 21 levels in 3D. However, in practice, particle trees that deep were not encountered during any of our tests.

The correspondence between key and tree levels has advantages for parallel tree construction as well. In a parallel environment, each process holds a valid global tree, which contains all the nodes on the highest level. On lower levels however, the tree is not necessarily locally complete, since the particles stored in the leaves are not all on the same process. To represent the information that is missing in the local memory, we make use of *pseudo nodes*, which store the global properties of the nodes they represent, and a reference to the process that holds the underlying data. If during some tree walk the pseudo node needs to be opened, we need to communicate.

Since every domain in the domain decomposition corresponds to a given range in Hilbert keys, and since this range in keys corresponds to a range of tree nodes, it is not so difficult to find out which nodes are local on a given process and which ones are not. We can hence easily add the pseudo nodes to the tree by simply

adding the keys of these nodes. To make sure the global tree is not too deep (since we need to represent it on all processes), we can round the key ranges to a given level precision. This at the same time guarantees that the domains will be relatively large blocks.

When we need to retrieve pseudo node information from the process that holds the original node, we also just use the key of the pseudonode to address this information, and we do not need complex data structures to keep track of the location in memory of data on other processes.

4.2.2 Template tree walks

The point of a particle tree as discussed above is to speed up computations that require the spatial distance between particles. A good example is a *neighbour search* algorithm, which consists of finding all particles within some radius R around a point with coordinates $\vec{x}$ (these coordinates do not necessarily need to correspond to the coordinates of a particle).

We start by finding out which child nodes of the root node of the tree overlap with the search region, i.e. either contain the entire sphere with origin $\vec{x}$ and radius R, or part of it. Each of these nodes is then traversed in turn, and the same procedure is repeated for its children. When one of the children inside the search region is a leaf, then the corresponding particle is added to the list of neighbours.

This algorithm is orders of magnitude faster than a naive direct search algorithm, in which we traverse all particles and for each particle calculate the distance to $\vec{x}$ and compare that directly with R. The speed up comes from the fact that we can very quickly discard large portions of the particle set by discarding high level nodes.

Other types of tree walks exist, for example for the calculation of the gravitational acceleration. In this case, high level nodes are not discarded, but are used to approximate the true gravitational acceleration using the total mass and the center of mass of the entire node rather than the individual masses and positions of the underlying particles. Apart from this, the overall structure of these tree walks is very similar to that of the neighbour search algorithm.

We exploit the similarities between tree walks to define a general *template* parallel tree walk algorithm, which makes use of C++ templates. Every tree walk is represented by a specific C++ class, the members of which are the variables that are used during the tree walk. The class interface then defines a number of methods that are used during the tree walk: a method that decides whether or not to open a specific node of the tree, a method that is called when a node is not opened, and a method that is called when a leaf of the tree is encountered. There is also a method that is called when a pseudo node of the tree is encountered, and

that decides if the tree walk should be communicated to another process. The interface then also defines two subclasses that are used to export the tree walk to another process, and to import the results back to the original process. At the end of the tree walk, a corresponding method of the tree walk object is called that finalises the result of the tree walk and updates the appropriate particle data.

The advantage of this abstraction is that we only need to implement one tree walk routine, which uses a template tree walk object, without having to think about the details of the tree walk itself. Furthermore, the actual communications have been completely taken out of the tree walk object and are limited to this general tree walk routine. To make sure the communications are correct, we need only make sure the method that decides whether or not to open a pseudonode is correct, and we need to correctly initialise a non-local tree walk based on the appropriate export object (and update the local results using the imported results from another process).

Another advantage of using templates is the compile time polymorphism they offer. It is perfectly possible to define an actual abstract tree walk class from which all concrete tree walks inherit, using classical (runtime) polymorphism. However, this means every time a method of the tree walk object is called, we first need to call the corresponding method of the abstract tree walk class, which then is translated *at runtime* into a call to the actual underlying tree walk. This extra function call completely disappears when we use templates, since then the compiler translates the single general tree walk routine to specific tree walk routines for every template tree walk class, directly calling the appropriate methods of the specific tree walk.

We verified that the parallel template tree walk for the gravitational accelerations implemented in Shadowfax is comparable in efficiency with that of Gadget2, while at the same time being a lot easier to understand and adapt in the code.

4.2.3 Voronoi mesh

The Voronoi mesh is an important part of Shadowfax if gas is present in the simulation, since it sets the volumes that are needed to convert conserved variables to primitive variables, defines the faces through which the fluxes are exchanged, and sets the positions of the centroids that are used to interpolate primitive variables and steer the motion of the mesh.

On the other hand, we do not want to build the entire code around the Voronoi mesh, since we would like to be able to use other types of grid as well, or use different mesh construction/evolution algorithms. The mesh and the hydrodynamical integration should be well separated in order to test one without depending on the correctness of the other.

The Voronoi mesh is hence represented by a general class that only implements the general geometrical information we actually want to extract from the mesh. We feed the gas particles to the mesh by means of an associated index and then use that index to retrieve the volumes or the centroids. Ideally, we would also like to have a general way to extract the face information (together with the information of which particles are neighbours), but for the moment this is still done separately for each grid type, so that we only define a method to calculate the fluxes, which then sets the fluxes for the particles in an example of bad object-oriented programming.

We currently support one fully functional mesh: a Voronoi mesh that is reconstructed every time step using the incrementally constructed Delaunay tessellation. Apart from that, we also support the mesh evolution algorithm discussed in Chapter 2, but this only works for 2D and some 3D cases (see Appendix B). To check the hydrodynamical integration, we also implemented a fixed Cartesian grid, which however is not fully functional.

Further decoupling the hydrodynamical integration and the geometrical structure of the mesh should greatly improve the code, but is far from trivial.

4.2.4 Riemann solvers

The Riemann problem introduced in Chapter 3 can be solved exactly using an iterative exact Riemann solver. Solving the Riemann problem is at the core of a finite volume method and for some grid types even makes up most of the computational cost, so that using an exact solver can become very expensive. To this end, a number of approximate Riemann solvers have been developed as well, which offer a good approximation to the exact solution without iteration.

To be able to switch between different Riemann solvers, we defined an abstract Riemann solver interface, which defines a single method that solves the Riemann problem for a given left and right state. Implementations of this interface are generated by a *factory* class, which takes the name of a Riemann solver as input. This name can then be specified as a parameter in the parameterfile. As explained when discussing the template tree walks, this runtime polymorphism inevitably brings along an extra function call, redirecting the abstract method call to the specific implementation. However, in this case the slow down does not outweigh the ability to change the type of Riemann solver as a parameter, especially since the number of Riemann solver method calls will be significantly lower than the number of function calls in a typical tree walk.

For the moment, we implemented two types of Riemann solvers: an exact Riemann solver and a TRRS solver (see Chapter 3). If we want to implement more precise approximate solvers, like the HLL or HLLC solvers (Toro, 2009), we will need to refactor the code, as these approximate solvers directly output the

fluxes rather than the solution to the Riemann problem. This is left for future work.

Note that the Riemann solver class also contains methods to convert primitive variables into conserved variables (given a volume) and the other way around, and methods to calculate a sound speed for a set of primitive variables. This makes it possible to implement support for advanced equations of state (see Chapter 5) by simply writing a new Riemann solver implementation. Support for this is not perfect yet, and still requires parts of the code to be refactored.

4.2.5 Units

Units have historically been a cause for concern to many simulators, as they link the simulation to the real physical world. For a computer, all calculations only involve numbers, both as input and as output, and these numbers do not represent anything. If we want to interpret the calculations however, we can choose to interpret the variables that are involved as distances or time intervals, and we can e.g. multiply masses with accelerations to get forces. The computer is unaware of these interpretations.

At the end of the simulation (or in every snapshot), the simulation program produces some values, and it is up to the simulator to find out to which physical quantities they belong. More importantly, it is up to the simulator to figure out which physical units these values have. This is not at all difficult, as long as we realise that these units will be a function of the values that are put in the simulation at the start.

We identify three different stages of the program where units are involved: at the start, during the simulation and at the end. As long as all equations involved during the simulation are physically correct, in the sense that the units at one side of the equation are equal to the units at the other side, the units during the simulation are irrelevant. The only thing that matters then are the units that are put in and come out of the simulation. If no explicit conversions happen, both will be equal to the units during the simulation. If we set the initial coordinates of the particles in metres, then the coordinates of the particles in the snapshots will be in metres as well.

To make sure we do set units at the start and the end of the simulation, we explicitly specify them in both the initial condition file and the snapshot files. These contain a HDF5 group with 5 attributes, specifying the values of the length, mass, time, temperature and current unit in CGS units[14]. All quantities in the file are then assumed to be specified in these units (or derived units). If no units

[14] CGS stands for Centimetres, Grammes, Seconds; the five basic units in this system are cm, g, s, K and A.

are specified, SHADOWFAX assumes SI units[15].

When reading in the initial condition, all quantities are converted to the simulation units, which might or might not be the same as the input units. Similarly, when writing a snapshot file, all quantities are again converted to the output units. Both the simulation units and the output units are specified as parameters in the parameter file, the default value being SI units. The input units need not be specified as parameters, since they should be present in the initial condition file.

The only other things that have units are the quantities specified as parameters in the parameter file, for which we adopt the convention that they use the simulation units that are also specified in the parameter file, and the physical constants that are used in the simulation.

In the current version of SHADOWFAX, there is only one physical constant: the gravitational constant $G = 6.67408 \times 10^{-11}$ m^3/kg/s^2, which is used to calculate the gravitational accelerations. This constant also has units, and we need to make sure its value is converted to the simulation units.

To make working with units easy, we implemented a unit class, with three member variables. These are the quantity which the unit represents, the value of the unit in SI units, and an optional name for the unit (which was used in an old snapshot format). The quantity needs to be composed of the five basic quantities by means of multiplications and divisions, and for convenience we also require the expression to be ordered with first the multiplications and then the divisions, with the factors being in alphabetical order.

Two different units are compatible if their quantities are the same (which means the expressions are exactly the same if both adhere to the standard outlined above). Values having compatible units can be converted into each other by using a unit converter class, which can be constructed from two compatible units with different SI values.

Physical constants are represented by a class holding a value and an associated unit. When the value of the physical constant is requested, we need to specify the desired unit for the quantity, and the value is converted accordingly. The converted value is stored in a separate class, so that it does not need to be recalculated every time it is needed.

4.3 Visualisation using VisIt

Many of the figures in this work were produced using `matplotlib`[16], a Python plotting library. Since SHADOWFAX snapshot files can be read into Python using

[15] Système Internationale, with as basic units m, kg, s, K and A

[16] `http://matplotlib.org`

`h5py`, this is a good match. However, since Python is an *interpreted* language, it is known to be slow when processing large data sets. There is also no direct interface between Python's visualisation capabilities and graphics processing units (GPUs), so that making high quality 3D figures in Python is very difficult.

For this kind of figures, we use VisIt[17], an open source, interactive visualisation tool, based on the powerful Visualization Toolkit (VTK)[18]. VisIt makes use of all graphical capabilities of the system, including the GPU, and even supports stereographic output for 3D screens. Apart from that, it also has a powerful (although poorly documented) Python interface.

To read in snapshots in VisIt, we developed our own database reader plugin, called SWIZMO. It reads in the default HDF5 snapshot format and stores it as a VTK point mesh, where the data is represented as being located on the position of the cell generators (or the particle positions if the snapshot comes from an SPH simulation).

To plot the actual Voronoi grid corresponding to a snapshot, we wrote the auxiliary `vtkmakerXd` program, which reads in the generator positions and recalculates the corresponding Voronoi mesh. This is then outputted as an unstructured mesh in the .vtk file format, which can be read by VisIt.

4.4 Test suite

The public version of SHADOWFAX contains a number of basic test problems to validate the code, which are gathered in a test suite. These problems are meant to be run as a general code check after any significant change to the code, and hence focus more on efficiency than on accuracy. We will give an overview in this section, to demonstrate the capabilities of SHADOWFAX. We will focus on accuracy in the next section, where we compare SHADOWFAX with a number of other codes.

We will not discuss test problems that only check the proper working of the code itself, like the `restarttest`, which is used to verify that SHADOWFAX correctly restarts from restart files.

4.4.1 Spherical overdensity

This test was already discussed in Chapter 2 and Chapter 3, and is a 2D or 3D generalisation of one of the Riemann solver tests from Toro (2009). The version included in the test suite uses 10,000 cells in 2D, and 100,000 cells in 3D. The results are shown in Fig. 4.6 at time $t = 0.1$, together with the 1D reference solution.

[17] `https://wci.llnl.gov/simulation/computer-codes/visit`
[18] `http://www.vtk.org`

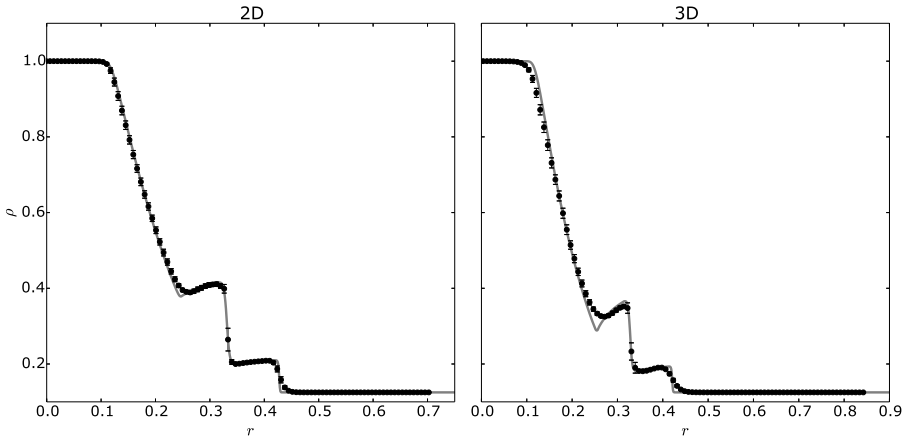


Figure 4.6: *Radial density profile for the spherical overdensity test at time $t = 0.1$. The black dots are the simulation results, the full gray line is the 1D reference solution. To limit the number of data points, the simulation results have been binned, the standard deviation of the density within the bins is indicated by the error flags.*

4.4.2 Gresho vortex

A known problem with Eulerian integration methods is a bad local conservation of angular momentum. These methods typically generate a lot of numerical diffusion in the velocity of the fluid, which leads to angular momentum being smeared out over a number of neighbouring cells. Lagrangian methods should not suffer this problem, as the resolution elements move along with the flow and angular momentum is naturally conserved.

A good test for local angular momentum conservation is the Gresho vortex test. For this test, a vortex in hydrostatic equilibrium is evolved for some time. The problem consists of a box with unit length in 2D and dimensions $1\times1\times1/3$ in 3D, containing a gas with constant unit density. Inside the box, a 2D azimuthal velocity profile of the form (Springel, 2010)

$$v_\phi(r) = \begin{cases} 5r & 0 \leq r < 0.2 \\ 2 - 5r & 0.2 \leq r < 0.4 \\ 0 & 0.4 \leq r, \end{cases}$$

with $r = \sqrt{(x-0.5)^2 + (y-0.5)^2}$ both in 2D and 3D, is balanced by a pressure

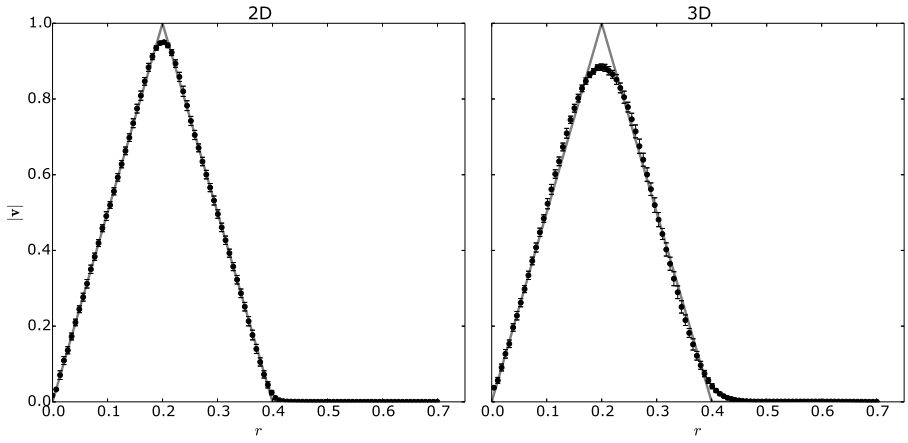


Figure 4.7: *Radial velocity profile for the Gresho vortex test at time $t = 3$. Both the 2D and 3D simulations use 10,000 uniformly sampled cells. The black dots represent the binned simulations results, with the error flags indicating the standard deviation on the values within the bins. The full gray line is the initial velocity profile.*

profile of the form

$$p(r) = \begin{cases} 5 + \frac{25}{2}r^2 & 0 \leq r < 0.2 \\ 9 + \frac{25}{2}r^2 - 20r + 4\log\left(\frac{r}{0.2}\right) & 0.2 \leq r < 0.4 \\ 3 + 4\log(2) & 0.4 \leq r. \end{cases}$$

Since the setup is in hydrostatic equilibrium, the result should be independent of time. To test this, we evolve the simulation to time $t = 3$, and compare the velocity profile with the initial profile in Fig. 4.7. Apart from a loss of precision around the peak of the velocity profile, the overall agreement is good.

4.4.3 Sedov-Taylor blast wave

A crucial aspect for simulations of galaxy formation and evolution is the capability to resolve strong shock waves, as e.g. caused by stars going supernova. These events deposit large amounts of energy into a small volume of the interstellar medium, causing a rapid heating and expansion of the surrounding gas. A good test for this situation is the Sedov-Taylor blast wave test, which we will also encounter in Chapter 5.

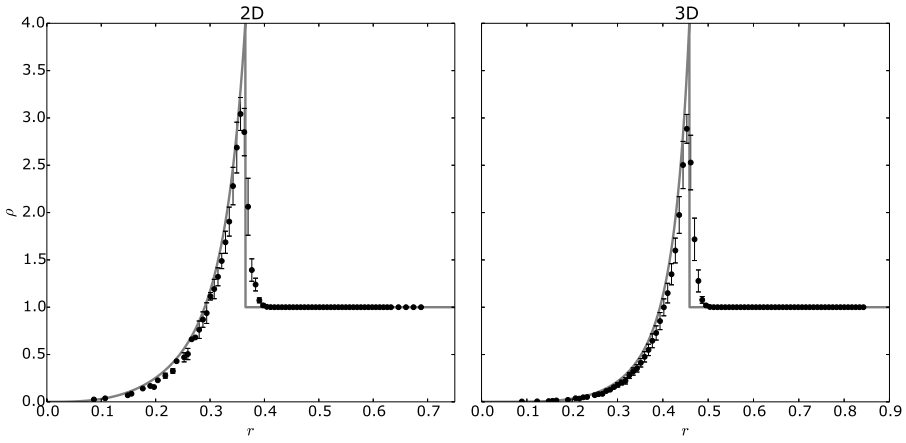


Figure 4.8: *Density profile for the Sedov-Taylor blast wave test at time $t = 0.1$. Both the 2D and 3D version use an initially Cartesian grid with 45 cells in every dimension. The black dots represent the binned simulation results, the error flags indicate the standard deviation on the density values within the bins. The full gray line is the analytic solution of Sedov (1977).*

For this test, we set up a box with unit length, in which a cold medium with unit density and pressure 10^{-6} is in rest. At the start of the simulation, we set the pressure in the central cell to a much higher value, corresponding to an effective energy input of 1. This initiates a strong explosion with a self-similar shock profile with a known analytic solution (Sedov, 1977). The 2D and 3D results are shown at time $t = 0.1$ in Fig. 4.8, together with the analytic solution.

To accurately capture the explosion, we need to make sure that the central cells are kept regular enough at the start of the simulation, since any irregularities will lead to asymmetries in the solution. Furthermore, we need to adopt an appropriate time step criterion that detects the shock in cells surrounding the central cell. Finally, we also need an accurate Riemann solver that can handle vacuum generating conditions at the faces of the central cell. This test hence can be used to verify these crucial parts of the code.

4.4.4 N-body test

Apart from a hydrodynamical integrator, SHADOWFAX also contains an N-body solver that is used to evolve the collisionless component and to calculate the

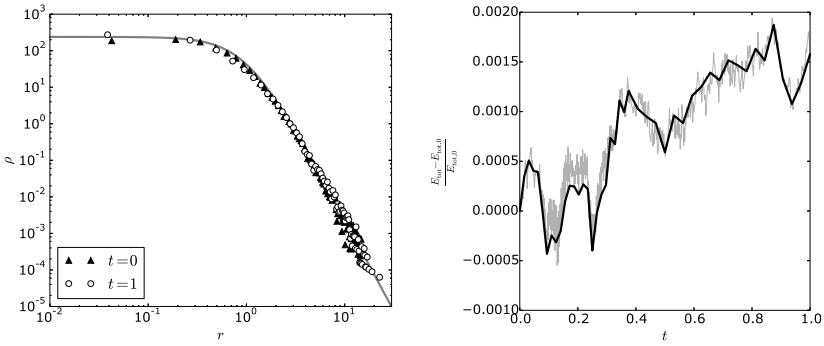


Figure 4.9: *Results of the N-body test using 10,648 cold dark matter particles.* Left: *Density profile at the start and end of the simulation, calculated by summing the masses of all particles within spherical shells. The full gray line represents the theoretical density profile from which the initial condition was sampled using rejection sampling.* Right: *Relative energy error as a function of time. The gray line shows the energy error for every system step, the black line shows the energy error at the system steps when all particles were active.*

gravitational accelerations for the gas. We limit use of this N-body solver to 3D.

To test the N-body solver itself, we evolve a collisionless Plummer sphere (Plummer, 1911) with mass 1000 and scale parameter 1 to time $t = 1$. We set the gravitational constant $G = 1$ for this problem, so that $t = 1$ corresponds to ~ 10 dynamical times. The density profile should stay constant during this time. We use a fixed gravitational softening length of 0.03.

The initial and resulting density profile are shown in Fig. 4.9, together with the relative energy error throughout the simulation. We see that the density profile indeed is stable, and that the total energy is quite accurately preserved. Very similar results were obtained using GADGET2 on the same initial conditions.

4.4.5 Evrard collapse

To test the coupling of gravity and hydrodynamics, we study the collapse of a cold self-gravitating sphere with unit mass and unit radius, which has an initial density profile of the form

$$\rho(r) = \begin{cases} \frac{1}{2\pi(r+0.001)} & r \leq 1 \\ 0 & 1 < r, \end{cases}$$

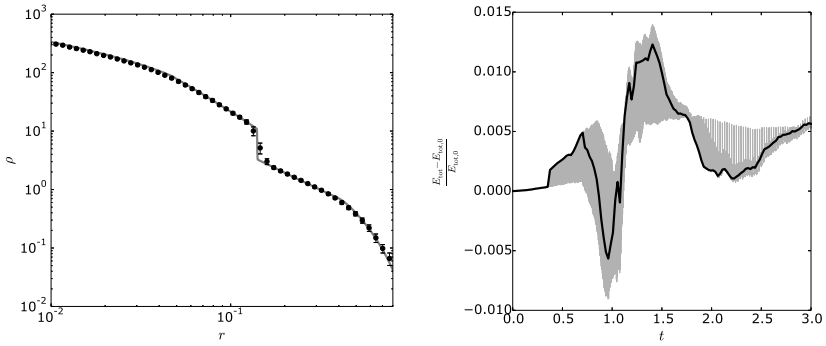


Figure 4.10: *Results for the Evrard test using 20,000 cells.* Left: *density profile at time* $t = 0.81$. *The black dots are the binned simulation results, the error bars indicate the standard deviation on the density values within the bins. The full gray line is the solution of an equivalent 1D problem.* Right: *relative energy error. The gray line represents the energy error at all system steps, the black line is the energy error at the steps when all particles were active.*

with $r = \sqrt{x^2 + y^2 + z^2}$ in this case. The entire sphere is given a constant low thermal energy by imposing a pressure profile of the form

$$p(r) = \begin{cases} \frac{0.05}{3\pi(r+0.001)} & r \leq 1 \\ 0 & 1 < r. \end{cases}$$

This setup is very similar to the setup introduced by Evrard (1988). The sphere will first collapse under its own gravity, causing a conversion of gravitational potential energy into kinetic energy. While the central density rises, this kinetic energy is converted into thermal energy, causing an outward travelling shock wave that virializes the sphere. The density profile of the sphere at time $t = 0.81$ is shown in Fig. 4.10, together with the solution of an equivalent 1D problem. We also show the relative energy error as a function of time to time $t = 3$, when the sphere has virialized.

The conversion of gravitational potential energy into first kinetic and later thermal energy is a good test for the gravity implementation. Another challenging aspect of this test is the treatment of the vacuum boundary of the cloud, which requires an appropriate Riemann solver and a good flux limiter.

4.5 Comparison with other methods

In Chapter 3, we already mentioned some of the advantages of a moving mesh scheme over more traditional methods. In this section, we will quantify the differences between a moving mesh scheme and other methods by comparing them on a number of test problems. The initial conditions for these tests are available online[19], so that anyone can repeat these tests and use them to compare with other codes.

4.5.1 Kelvin-Helmholtz instabilities

Kelvin-Helmholtz instabilities arise when two layers of fluid shear against each other, so that small velocity perturbations in the direction perpendicular to the shear direction grow exponentially to form wave-like structures. As already mentioned in Chapter 3, Kelvin-Helmholtz instabilities arise at all scales, so that we need to suppress small scale instabilities to be able to compare different methods. This can be done by the introduction of a transition layer in between the shearing layers.

Since Kelvin-Helmholtz instabilities mix up fluid from the two layers, resolving them is essential when e.g. studying the elemental composition of the interstellar medium. Agertz *et al.* (2007) showed that classical SPH does not resolve Kelvin-Helmholtz instabilities at all, meaning that codes like GADGET2 have very bad mixing properties. Eulerian codes generally have much better mixing properties. We will therefore compare SHADOWFAX with MPI-AMRVAC, a public grid based AMR code[20](Keppens *et al.*, 2012).

We will focus on two different aspects: the mixing of two layers with different densities late in the simulation, and the early linear exponential growth of the instability. Both will be studied in 2D only.

Mixing

The first setup consists of a periodic box with unit length, in which the fluid is given a density profile of the form

$$\rho(x,y) = \begin{cases} 1 & y < 0.25 \\ 10 & 0.25 \leq y \leq 0.75 \\ 1 & 0.75 < y. \end{cases}$$

[19] `http://www.dwarfs.ugent.be/shadowfax/`
[20] `http://homes.esat.kuleuven.be/~keppens/`

The x component of the velocity is given by

$$v_x = \begin{cases} -0.5 & y \leq 0.25 - d \\ -0.5 + \frac{y+d-0.25}{2d} & 0.25 - d < y < 0.25 + d \\ 0.5 & 0.25 + d \leq y \leq 0.75 - d \\ 0.5 - \frac{y+d-0.25}{2d} & 0.75 - d < y < 0.75 + d \\ -0.5 & 0.75 + d \leq y, \end{cases} \tag{4.1}$$

where we introduced a middle layer with thickness $d = 0.025$. The y component of the velocity is given by

$$v_y = A \sin(4\pi x) \left(\mathrm{e}^{-\frac{(y-0.25)^2}{2\sigma^2}} + \mathrm{e}^{-\frac{(y-0.75)^2}{2\sigma^2}} \right),$$

and seeds an instability with wavelength half a box length and amplitude $A = 0.1$ in a small layer with thickness $\sigma = 0.00125$ around the two interfaces between high density and low density layers. The pressure is set to $P = 2.5$ in the entire box. We use the same initial condition consisting of a Cartesian grid with 100×100 cells for both the MPI-AMRVAC and SHADOWFAX simulations. Of course, this grid will deform during the SHADOWFAX simulation, while it is static for the MPI-AMRVAC simulation.

We use two different modes of MPI-AMRVAC for this test: a mode using a conservative finite difference scheme with global Lax-Friedrich splitting and a fifth order spatial reconstruction (FD), and a finite volume scheme with a HLLC Riemann solver (FV). Both schemes use a fourth order accurate Runge-Kutta time integration scheme.

As shown in the top row of Fig. 4.11, both the SHADOWFAX simulation and the MPI-AMRVAC simulations lead to similar results at time $t = 1.5$. All three simulations develop clear instabilities at around the same time. The non-linear evolution of these instabilities is a bit different, but this is to be expected.

As a variant of the test, we also ran the three simulations using the same setup, but with a large bulk velocity $v_{\mathrm{bulk}} = 100$ added in the x direction, corresponding to a Mach number of 155 in the high density layer. This does not change the physical problem, so that a properly Galilean invariant code should yield the same results as for the setup without bulk velocity. As can be seen from the bottom row of Fig. 4.11, this is indeed the case for the SHADOWFAX simulation. The MPI-AMRVAC simulations are clearly affected by the bulk velocity, with the FV result being smeared out, and the FD simulation not yielding any result at all. Increasing the resolution does not help improving the FV result much, as was already shown in Fig. 3.6.

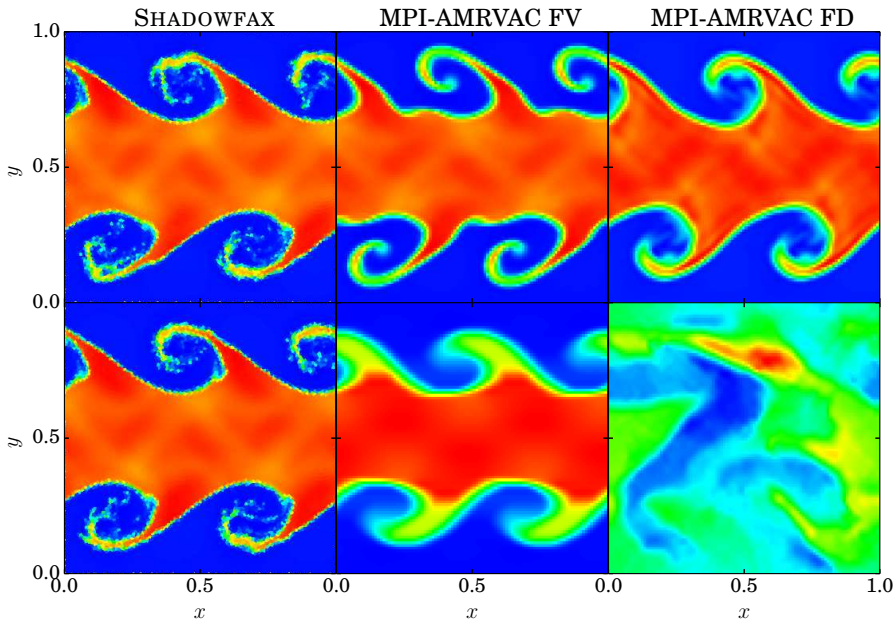


Figure 4.11: *Density colour plots for the shearing layers test at time $t = 1.5$, using a 100×100 Cartesian initial condition. The individual cells are shown, which explains the irregular shapes in the* SHADOWFAX *results. The top row shows the result for the normal setup, the bottom row corresponds to a setup with an extra bulk velocity $v_{bulk} = 100$ added to the x component of the velocity.*

Linear growth rate

If we use a setup without density contrast, the initial exponential growth of the instability will only depend on the wavelength of the instability and the thickness of the middle layer, so that simulations of this phase should converge to a consistent growth rate, irrespective of the resolution or method that is used (Hendrix & Keppens, 2014).

To test this, we set up a variant of the test above whereby the initial density is set to unity in the entire box. The y component of the velocity is now given by

$$v_y = B \sin(4\pi x) \left(\mathrm{e}^{-\frac{(y-0.25)^2}{32d^2}} + \mathrm{e}^{-\frac{(y-0.75)^2}{32d^2}} \right),$$

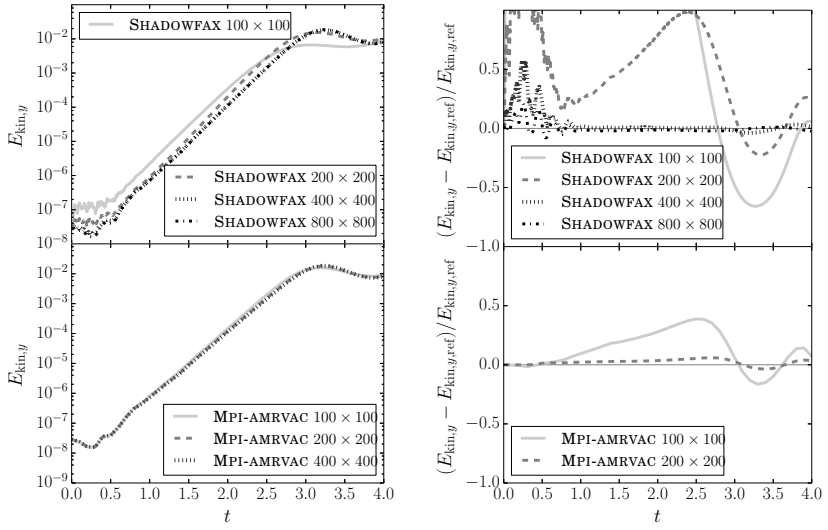


Figure 4.12: *Convergence of the linear exponential growth rate of the Kelvin-Helmholtz instability for the shearing layers tests without density contrast.* Left: *kinetic energy in the y direction as a function of time.* Right: *relative difference between the kinetic energy in the y direction for the* 400×400 MPI-AMRVAC *simulation and that for the other simulations as a function of time.*

with $B = 0.0005$ and $d = 0.0317$. The same d is now also used as thickness of the middle layer in (4.1). The other quantities have the same values as before.

To quantify the growth of the instability, we track the total kinetic energy in the y direction. This energy is initially set by the seed velocity, and will grow exponentially as kinetic energy in the x direction is converted into extra kinetic energy in the y direction by the growing instability. The results are shown in Fig. 4.12. The high resolution SHADOWFAX simulations are clearly in agreement with the high resolution MPI-AMRVAC results, but the convergence seems to be slower. To eliminate differences caused by slightly different onsets of the growth of the instability, we also fitted a simple exponential curve to the linear exponential part of the curves in the left panel of Fig. 4.12. The slopes are given in Table 4.1. Again, we see that convergence is slower for the SHADOWFAX simulations. Furthermore, both methods converge to slightly different slopes.

Table 4.1: *Slope of an exponential fit to the kinetic energy in the y direction in the time interval [1.5,2.5] for the shearing layers tests without density contrast*

Simulation	Slope
SHADOWFAX 100×100	4.61
SHADOWFAX 200×200	5.43
SHADOWFAX 400×400	5.10
SHADOWFAX 800×800	5.10
MPI-AMRVAC 100×100	5.28
MPI-AMRVAC 200×200	5.14
MPI-AMRVAC 400×400	5.14

4.5.2 Sod shock

In Chapter 3 we introduced mesh-free methods as another alternative for SPH, that uses a finite volume method similar to that implemented in SHADOWFAX, but using kernel-based volume estimates. We will compare SHADOWFAX with our own mesh-free hydrodynamics implementation in the simulation code SWIFT[21] (SWIFT GIZMO), and the standard SPH implementation in SWIFT (SWIFT SPH).

We will focus on a high resolution version of the Sod shock test, which consists of a $1 \times 0.125 \times 0.125$ periodic cuboid containing a fluid with density

$$\rho(x) = \begin{cases} 1 & x \leq 0.5 \\ 0.25 & 0.5 < x \end{cases}$$

and pressure

$$p(x) = \begin{cases} 1 & x \leq 0.5 \\ 0.1795 & 0.5 < x \end{cases}$$

that is initially at rest. The results are evolved to time $t = 0.12$, and should equal the results of the corresponding Riemann problem, which consists of a left rarefaction wave, a central contact discontinuity and a right shock wave.

We generated a single initial condition file containing two gravitational glasses that were pasted together to yield the correct density contrast when using SPH. The resulting setup has 1,024,128 particles and was used for all three simulations.

The results and the analytic solution are shown in Fig. 4.13. Overall, the SHADOWFAX result is more accurate than the other two results. Both SWIFT results are comparable, although the SWIFT SPH result has more noise on the density, and systematically underestimates the density in most of the box. This

[21] `https://gitlab.cosma.dur.ac.uk/swift/swiftsim/tree/gizmo`

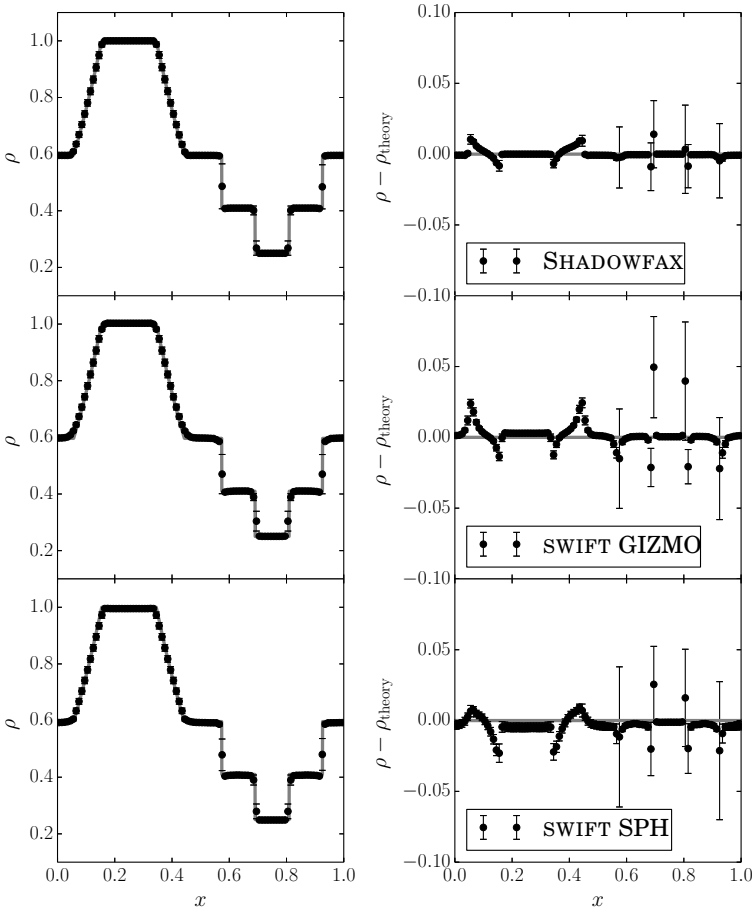


Figure 4.13: *Results of the Sod shock test at time $t = 0.12$.* Left: *density profile,* right: *difference between the density and the theoretical density given by the solution of the equivalent Riemann problem. The black dots represent the binned simulation results, the error flags indicate the standard deviation on the values within the bins. The full gray lines represent the analytic solution of the equivalent Riemann problem.*

is a direct result of the way in which the density is calculated in SPH. To quantify the accuracy, we calculated χ^2 values for all three simulations by summing the quadratic differences between the effective densities and the theoretical result. This yielded $\chi^2 = 29.96$ for SHADOWFAX, $\chi^2 = 106.71$ for SWIFT GIZMO, and $\chi^2 = 123.54$ for SWIFT SPH.

4.5.3 Noh test

Noh (1987) proposed a strong shock test that is very challenging and has a known analytic solution. The setup consists of a reflective box with unit length, in which a fluid with unit density and a very small thermal energy of 1×10^{-5} is enclosed. At time $t = 0$, we set the radial velocity everywhere to -1, with the radius $r = \sqrt{x^2 + y^2}$. We will restrict ourselves to the 2D problem, although the problem can be solved in 3D as well.

Since the fluid collapses on the origin, a strong outward shock wave develops, with very high Mach number. The velocity of the shock front is given by $v_{\text{shock}} = 1/3$, and the density profile by

$$\rho(r,t) = \begin{cases} 16 & r \leq v_{\text{shock}} t \\ 1 + \frac{t}{r} & v_{\text{shock}} t < r. \end{cases}$$

We compare SHADOWFAX with MPI-AMRVAC FV using a low resolution 400×400, and a high resolution 1600×1600 Cartesian setup. The radial flow of the fluid will create low density cavities at the boundaries of the simulation box, which are normally compensated by using inflow boundaries. However, SHADOWFAX does not support this type of boundary condition, so that we will restrict the simulation to $t = 0.5$, before the shock interacts with these cavities.

The resulting density profiles are shown in Fig. 4.14. Overall, both methods resolve the shock quite well, although MPI-AMRVAC overestimates the central density. We also calculated χ^2 values in this case, by summing the quadratic differences between the simulated densities and the analytic solution. For the low resolution simulations, this yielded $\chi^2 = 1.78 \times 10^4$ for SHADOWFAX and $\chi^2 = 2.87 \times 10^5$ for MPI-AMRVAC. For the high resolution simulations, we get $\chi^2 = 1.55 \times 10^5$ for SHADOWFAX, and $\chi^2 = 4.54 \times 10^6$ for MPI-AMRVAC. The high values for the MPI-AMRVAC results are likely caused by the low density cavities, since the results in these regions will necessarily be wrong and since these regions will contain a lot more fixed MPI-AMRVAC cells than co-moving SHADOWFAX cells.

4.5.4 Implosion test

The implosion test of Liska & Wendroff (2003) is another challenging test problem. In a 2D periodic box with dimensions 0.6×0.6, a fluid with unit density

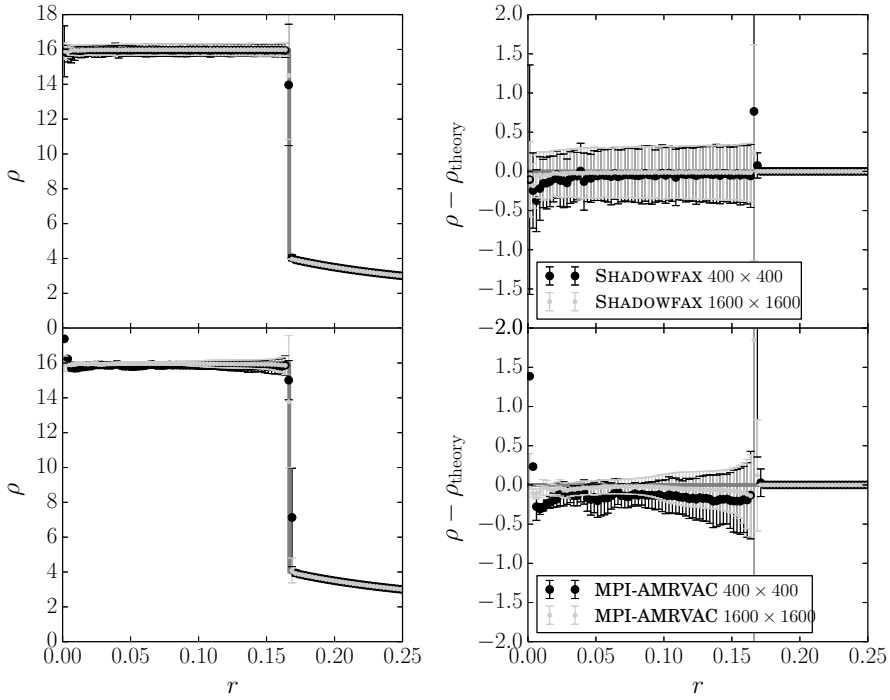


Figure 4.14: *Results of the Noh test at time $t = 0.5$.* Left: *density profile,* right: *difference between density and analytic solution. The black and gray dots represent the binned simulation values, the error flags indicate the standard deviation on the values inside the bins. The full gray line is the analytic solution.*

and pressure is placed at rest. In the center of the box, we cut out a rhombus with width 0.3, in which the density is set to $\rho = 0.125$, and the pressure to $P = 0.14$. We use an initial Cartesian grid of 800×800 cells, and evolve it until time $t = 0.75$ using SHADOWFAX and MPI-AMRVAC FV. As in Liska & Wendroff (2003), we set the adiabatic index $\gamma = 1.4$ for this test.

The high density region will initially implode into the low density region, and cause a large scale shock wave, that will travel back and forth through the simulation box. Meanwhile, small scale Rayleigh-Taylor and Kelvin-Helmholtz instabilities will arise at the interface between the two regions, and will interact with the large scale shock wave. These instabilities are seeded by numerical noise

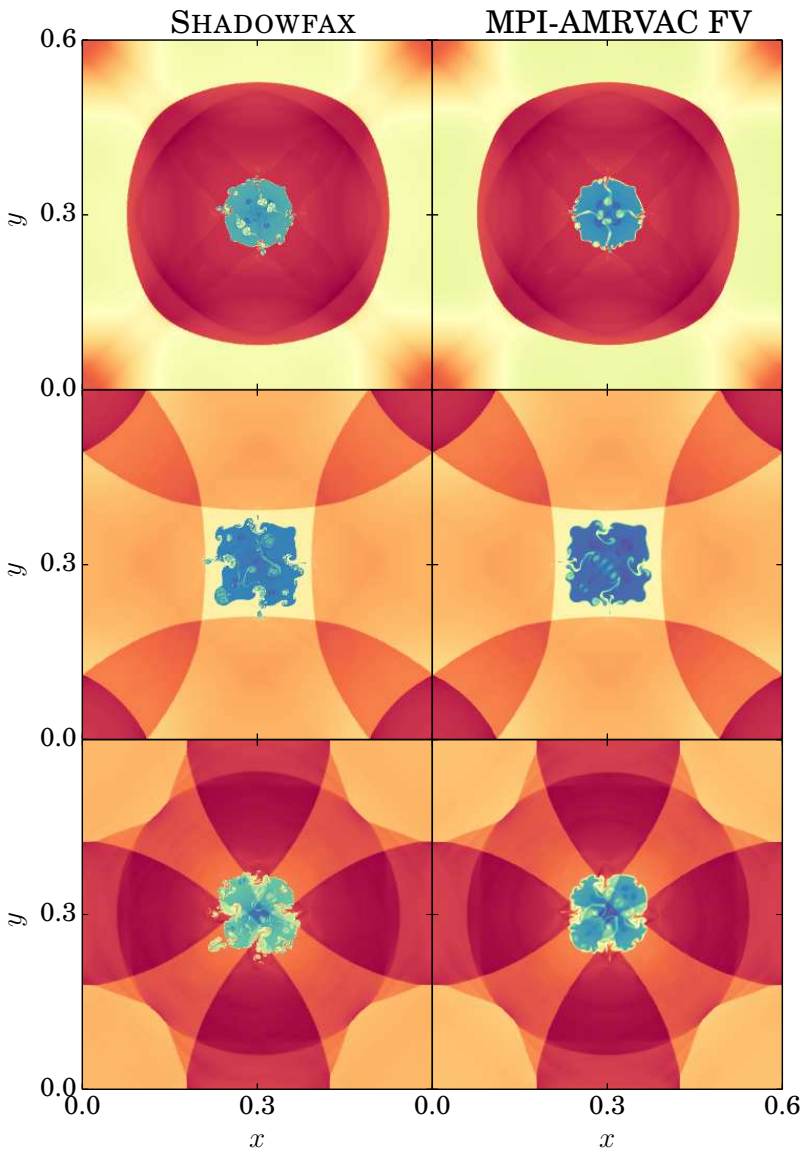


Figure 4.15: *Density colour plot for the implosion test at times* $t = 0.25$ *(*top*),* $t = 0.5$ *(*middle*), and* $t = 0.75$ *(*bottom*).*

and therefore behave very chaotically. We can however compare the evolution of the large scale shock wave between both methods.

The results at three different times are shown in Fig. 4.15. The large scale shock wave is clearly visible and behaves in the same way for both methods. The small central instabilities are more pronounced in the SHADOWFAX result, indicating that a moving mesh method is more sensitive to these instabilities. This is probably caused by (a) a higher sensitivity of a moving mesh to noise in the velocity component of the fluid, and (b) the Lagrangian character of the method, which will keep instabilities intact, while numerical diffusion washes them out in an Eulerian method. It is interesting to see that these instabilities are not symmetric in the MPI-AMRVAC result, indicating that they are indeed completely governed by numerical noise.

5

Sub-grid physics

The methods in Chapter 3 allow us to simulate a hydrodynamical fluid inside a gravitational potential, under the influence of cooling and heating. This is a large step towards modelling the real interstellar medium of galaxies, but it is not enough. We still treat the gas as being a monatomic ideal gas, and have not specified how the gas cools. The same applies to the heating terms, which still need to be physically motivated. In this chapter, we will discuss these aspects in more detail.

We start with the gas, explain how it cools and behaves, and what the effect of an external UV background is on the cooling and heating of the gas. We then discuss the different types of stellar feedback, and detail how these affect the gas.

5.1 Gas physics

The interstellar medium (ISM) is the general name for all baryonic matter inside a galaxy that is not in the form of stars or stellar remnants like black holes, neutron stars, ... It not only consists of neutral or ionised gas, but also of molecules and even small dust grains. Although its density can locally reach high values, it is overall quite diffuse. For this reason, the neutral ISM is one of the best realisations of an ideal gas, so that the ideal gas approximation is certainly acceptable.

Element-wise, the ISM mainly consists of ^{1}H ($\approx 90\%$) and ^{4}He ($\approx 9\%$), in number abundances that did not change much since the Big Bang nucleosynthesis, which took place in the first 20 minutes after the Big Bang. The remaining 1% of the ISM consists of a mixture of other elements, which astronomers just denote as *metals*.

When discussing stellar feedback, we will see that this mixture can be characterised by two independent *metallicities*, corresponding to a fast contribution by the supernova explosions of massive stars, and a slow contribution by the supernova explosions of less massive stars (De Rijcke *et al.*, 2013): a [Fe/H] value and a [Mg/Fe] value, denoting respectively the logarithm of the iron abundance and

the magnesium abundance in units of the corresponding solar abundance. Primordial gas that has not been enriched by stellar feedback has element abundances set by the Big Bang nucleosynthesis, and zero metallicity.

Apart from being composed of different elements, the ISM is also a mixture of a neutral gas and an ionised plasma, depending on the local temperature and density. Not all elements are ionised at the same temperature and even for a single element, the transition from neutral to ionised takes place in some broad temperature range, so that we end up with a complex mixture of atoms and ions in different stages of ionisation. It is this complex fluid that we want to model.

5.1.1 Radiative cooling

When a ^{1}H ion recombines to a neutral atom by capturing a free electron, a small amount of *binding energy* is released in the form of one or multiple photons (with an energy of 13.6 eV). This photon can be captured by another neutral ^{1}H atom and ionise it, but if the local ISM is diffuse enough, it will escape the ISM and will cause an effective energy loss in the system.

Similarly, transitions from excited states of the ^{1}H atom to the ground state, or between excited states, will also release photons that carry away energy from the system. The same applies to other elements in the ISM. All these energy losses together constitute a cooling term in the energy equation.

In regions with a very high temperature, the ISM will be completely ionised, and the losses due to recombination processes become negligible. However, in these regions there will still be losses due to other radiative processes, like bremsstrahlung emitted by electrons that are accelerated by magnetic fields in the plasma.

It may be clear from the above that calculating a physical cooling term requires a full modelling of all radiative processes in the multiphase ISM. This means we need to determine the ionisation state and the abundance of all elements in the ISM. In the absence of external heating, these are set by three parameters: the temperature of the local ISM and its metal content, characterised by the two metallicity parameters.

Given these parameters, we use the CHIANTI spectral database (Dere *et al.*, 2009) to calculate the ionisation equilibrium, which then gives us the net cooling rate of the gas. Doing this for every set of parameters at runtime during the simulation is however way too expensive. To remedy this, we precalculate the cooling rates for a number of parameter values and store them in *cooling tables*. At runtime, we then linearly interpolate on these tables to obtain an approximate cooling rate.

5.1.2 Ionisation and recombination

In Chapter 3, we introduced the temperature T of the gas through a linear relation with the thermal energy of the gas. For an ideal gas with adiabatic index $\gamma = 5/3$, the relation is

$$u = \frac{3}{2}\frac{kT}{m},$$

with m the mass of a single microscopic constituent of the gas. This relation expresses the *equipartition theorem*, which assigns a thermal energy of $\frac{1}{2}kT$ to every degree of freedom of a microscopic constituent of the gas, with a single constituent having only three spatial degrees of freedom.

We can rewrite the equation above by introducing a constituent *number density*, n:

$$\rho u = \frac{3}{2}nkT.$$

This relation is valid for a monatomic ideal gas consisting of only one type of constituents. However, if the gas consists of multiple elements, that are allowed to ionise, then we have to adapt the equation to (Vandenbroucke *et al.*, 2013)

$$\rho u = \frac{3}{2}\left(\sum_X n_X + n_e(T)\right)kT + \sum_X \left[\sum_i n_{X,i}(T)\left(\sum_{j=1}^{i}\varepsilon_{X,j}\right)\right], \tag{5.1}$$

where n_X now represents the number densities of the different elements in the gas. $n_e(T)$ is the number density of electrons, which depends on the ionisation state of the different elements and hence the temperature. $n_{X,i}(T)$ is the number density of the ith ion of element X, while $\varepsilon_{X,j}$ is the energy needed to ionise the $(j-1)$th ion of element X to the jth ion of X. Charge conservation dictates that the number density of electrons is coupled to that of the ions:

$$n_e(T) = \sum_i i n_{X,i}(T).$$

In a monatomic ideal gas with a single type of constituents, the pressure is given by

$$P = nkT.$$

In a multiphase, multicomponent gas, this becomes

$$P = \left(\sum_X n_X + n_e(T)\right)kT = \frac{\rho kT}{\mu(T)},$$

where we introduced the mean constituent mass $\mu(T)$. It is immediately clear from the above that in the case of the real ISM, the relation $P = (\gamma - 1)\rho u$ is no longer valid.

This simple fact has some far-reaching consequences for the hydrodynamical integration, as it forces us to adapt the energy equation. In practice, it forces us to introduce the temperature T as a new primitive variable that is integrated along with the other variables, and that is used to calculate the pressure when it is needed. To this end, we need to know the mean constituent mass $\mu(T)$, which will depend on the ionisation state of the fluid. Just as for the cooling, we will precalculate the mean constituent mass for a number of temperature and metallicity values, and then calculate it at runtime by linear interpolation on these three dimensional tables.

In Vandenbroucke *et al.* (2013), we compared the change in the thermal energy by the ionisation of ^{1}H and ^{4}He to the effect caused by the ionisation of other metals. Due to their low abundances, the latter is negligible. Similarly, the change in thermal energy caused by transitions between excited states of a single ion is negligible compared to the change due to ionisation. For this reason, we will only take into account the contributions of ^{1}H and ^{4}He for the mean constituent mass, and the contribution of the ionisation of ^{1}H for the ionisation potential energy reservoir.

The ionisation fractions of ^{1}H and ^{4}He do however depend on the presence of other metals, so that we do effectively end up with three dimensional, composition-dependent tables. Note also that metals do need to be taken into account for the radiative cooling, especially in low temperature regions, where metal line cooling is the only efficient cooling mechanism.

To adapt the energy equation, various methods have been tested. In Vandenbroucke *et al.* (2013), we tabulated the thermal energy-temperature relation (5.1) and used the standard thermal energy equation to integrate the thermal energy. Whenever a temperature was needed, we calculated it by linear interpolation on the three dimensional thermal energy-temperature tables. This approach however turned out to be problematic when using it for the more extended five dimensional tables needed when an external UV background is present, since the thermal energy-temperature relation becomes degenerate in some parts of its domain.

The most recent version of our code uses an alternative approach. We split the thermal energy in two parts, a kinetic part due to the movement of the microscopic constituents of the fluid, and a potential energy part due to the ionisation of the ^{1}H:

$$u = u_{\mathrm{kin}} + \frac{\chi_{\mathrm{H}}}{m_{\mathrm{H}}} x,$$

with $\chi_{\mathrm{H}} = 13.6\,\mathrm{eV}$ the ionisation energy of ^{1}H, $m_{\mathrm{H}} = 1.6737236 \times 10^{-27}\,\mathrm{kg}$ the mass of the ^{1}H atom, and x the ionisation fraction of the gas, $x = 0$ being a completely neutral, and $x = 1$ a completely ionised gas.

In the absence of ionisation, the kinetic thermal energy equals the total

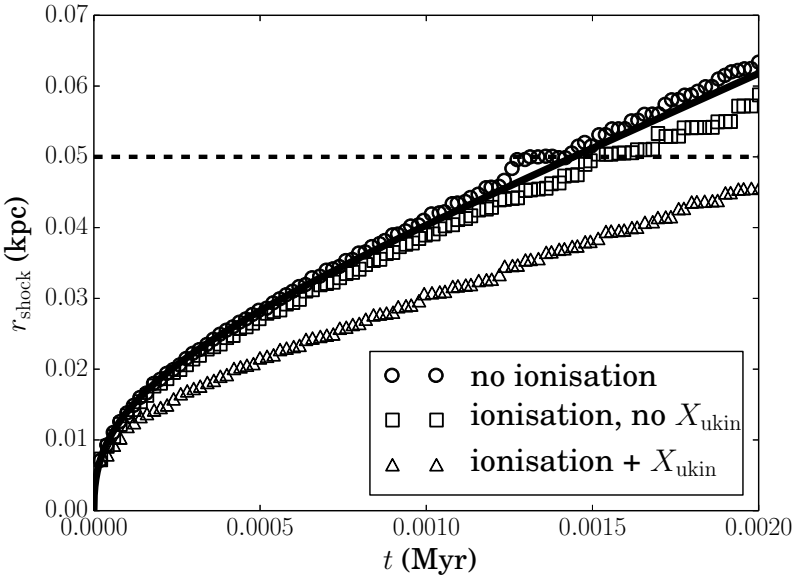


Figure 5.1: *The position of the shock front as a function of time for the Sedov-Taylor blast wave test in an ambient medium with a density of* $81\,amu\,cm^{-3}$ *and a temperature of* $15,000\,K$*, using three different treatments for the ionisation. The circles represent a simulation that does not take into account the effect of ionisation. The squares represent a simulation that takes into account the change in mean particle mass, but does not take into account the potential energy associated with the ionisation. The triangles represent a simulation that takes the full effect of ionisation into account. The full black line represents the analytic solution due to Sedov (1977). The dashed line represents the boundary of the periodic simulation box.*

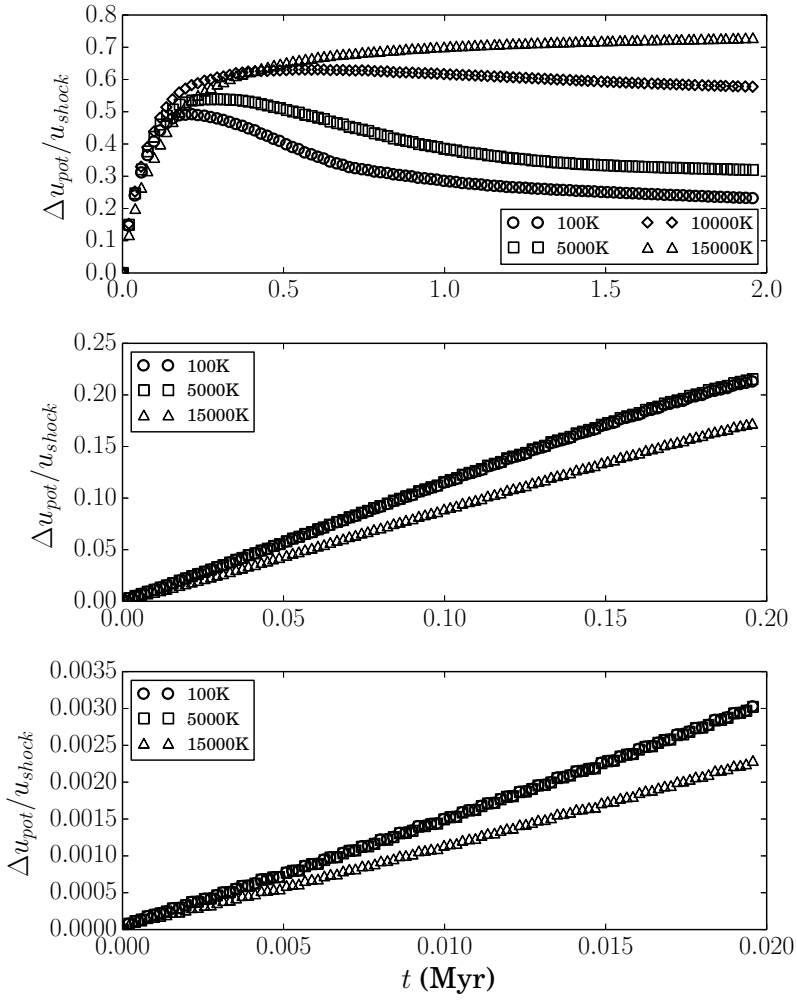


Figure 5.2: *The ratio of the total potential energy change and the injected energy as a function of time for simulations with different ambient temperatures and densities.* Top: $\rho = 81\,amu\,cm^{-3}$, middle: $\rho = 0.81\,amu\,cm^{-3}$, bottom: $\rho = 0.0081\,amu\,cm^{-3}$.

thermal energy, so that the kinetic thermal energy is the quantity that is evolved by the thermal energy equation. Instead of replacing the kinetic thermal energy by the total thermal energy when taking into account ionisation, we can also still evolve the kinetic thermal energy, using an adapted thermal energy equation:

$$\frac{\mathrm{d}u_{\mathrm{kin}}}{\mathrm{d}t} = \frac{\mathrm{d}u}{\mathrm{d}t} - \frac{\chi_{\mathrm{H}}}{m_{\mathrm{H}}}\frac{\mathrm{d}x}{\mathrm{d}t},$$

where we took into account the time dependence of the ionisation fraction.

If we neglect temporal changes in the composition of the gas (which we do implicitly by considering gas in ionisation equilibrium), then the ionisation fraction to first order only depends on the temperature, or hence the kinetic thermal energy of the gas, so that

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \frac{\mathrm{d}x}{\mathrm{d}u_{\mathrm{kin}}}\frac{\mathrm{d}u_{\mathrm{kin}}}{\mathrm{d}t}.$$

If we insert this in the equation for the kinetic thermal energy, we end up with the following evolution equation for the kinetic thermal energy:

$$\frac{\mathrm{d}u_{\mathrm{kin}}}{\mathrm{d}t} = \frac{1}{1 + X_{\mathrm{ukin}}}\frac{\mathrm{d}u}{\mathrm{d}t},$$

where $X_{\mathrm{ukin}} = \frac{\chi_{\mathrm{H}}}{m_{\mathrm{H}}}\frac{\mathrm{d}x}{\mathrm{d}u_{\mathrm{kin}}}$ represents a dimensionless quantity that describes the absorption of thermal energy by the ionisation potential energy reservoir. Instead of evolving the total thermal energy, we can hence evolve the kinetic thermal energy, and adapt the thermal energy equation by applying the correction term given by X_{ukin}. X_{ukin} is tabulated like the mean constituent mass and the cooling above.

X_{ukin} does not take into account the effect of the change in mean constituent mass due to the ionisation on the kinetic thermal energy, so that in practice we still need to tabulate the kinetic thermal energy as well. However, the relation between the kinetic thermal energy and temperature is no longer degenerate, so that this approach effectively solves the integration problems caused by evolving the total thermal energy.

The effect of the treatment of ionisation on the dynamics of the gas is illustrated in Fig. 5.1, by means of a Sedov-Taylor blast wave test. For this test, an energy of $10^{52}\,\mathrm{erg}$ is inserted in the center of a cubic simulation box with side $0.1\,\mathrm{kpc}$, containing gas with a density of $\rho = 81\,\mathrm{amu\,cm^{-3}}$ and a temperature $T = 15,000\,\mathrm{K}$, which is initially at rest. The central energy injection causes a strong shock wave to spread out from the center of the box. We keep track of the position of the shock front as a function of time, and compare the results for three different simulations with the analytic solution of Sedov (1977). As can be seen in the figure, the effect of the changing mean constituent mass is rather

small, while the potential energy reservoir associated with the ionisation is able to absorb a significant fraction of the shock energy.

The fraction of the initial shock energy that is absorbed as a function of time is shown in Fig. 5.2 for a number of simulations with different initial gas densities and temperatures. For a gas with a density of $81\,\mathrm{amu\,cm^{-3}}$, similar to the density of a star forming region in our galaxy simulations, this fraction can be more than 50%. For lower densities, the effect is a lot smaller (on the same length scale, because the whole problem is in fact dimensionless).

5.1.3 UVB

In the discussions above, we always assumed the gas to be in collisional ionisation equilibrium, i.e. the ionisation of the gas is set by the dynamic equilibrium of collisional excitation and ionisation, and the radiative recombination and deexcitation of the atoms in the gas. The rate at which atoms collide only depends on the temperature of the gas, so that the equilibrium is completely determined by three parameters.

In the presence of an external UV photon background (UVB), atoms are no longer only ionised by collisions, but can also be ionised by absorbing a UV photon with an energy of at least 13.6 eV from the background. The rate at which this will happen depends on the strength of the UVB, and on the ability of the local gas to *shield* itself from the UVB. If the ionisation rate in some region of the gas is high enough to absorb all ionising photons from the external UVB, then gas internal to this region does no longer "see" the UVB, and is effectively shielded from it. The strength of this shielding will depend on the number density of neutral ^{1}H, and hence on the density of the gas. We adopt an exponential self-shielding with a density threshold of $n_\mathrm{H} = 0.007\,\mathrm{amu\,cm^{-3}}$ (De Rijcke *et al.*, 2013).

The strength of the UVB gradually increases over time as more galaxies host young UV bright stars, starting from a redshift of ~ 10.5 and reaching its peak strength at a redshift of ~ 2 (Faucher-Giguère *et al.*, 2009). This introduces a time dependence for all quantities that depend on the ionisation equilibrium of the gas.

We hence need to add two extra parameters to the three parameters we already had: the gas density and the current simulation redshift (or time), yielding five parameters in total. Furthermore, the self-shielding depends on the density of neutral ^{1}H rather than the actual gas density, which itself depends on the temperature and the composition of the gas. We hence need to combine the five dimensional interpolation with a three dimensional interpolation to obtain the correct neutral ^{1}H density.

Five dimensional tables are precalculated for the gas cooling, mean constituent

mass, thermal energy and $X_{\rm ukin}$. Moreover, the UVB also acts as a heating term in the thermal energy equation, which also needs to be tabulated and added to the integration.

The UVB actually introduces an explicit time and density dependence for the ionisation potential energy part of the thermal energy as well, so that we should replace the total time derivative in the definition of $X_{\rm ukin}$ with a partial derivative, calculated for a fixed density and redshift. We should also include correction terms for the density dependence to the kinetic thermal energy equation. However, the effect of these will likely be smaller than that of $X_{\rm ukin}$, so that this is left for future work.

5.2 Stellar feedback

We have seen in Chapter 2 that the star particles in our simulations do not correspond to single stars, but rather to small stellar populations that are born at roughly the same time and have very similar metal contents. These Single Stellar Populations (SSPs) consist of a statistical mixture of stars with different masses, distributed according to a stellar Initial Mass Function (IMF), which gives the number of stars within a given stellar mass bin. Different prescriptions for the IMF exist (Salpeter, 1955; Kroupa, 2002; Chabrier, 2003), and it is not yet clear if the IMF depends on the metallicity of the gas from which the stars are born (Geha *et al.*, 2013).

We will use a Chabrier IMF (Chabrier, 2003) throughout this work, both to calculate the statistical properties of SSPs, and to calculate the contribution of the UV light of young stars to the UV background, which affects the ionisation equilibrium of the ISM and hence its cooling and heating.

During their lifetime, stars emit energy in the form of radiation. Since the ISM is transparent to most wavelengths of light, most of this energy is lost from the galaxy. An exception is the UV light emitted by young, massive stars, which is locally absorbed by the ISM, and acts as an extra heating term in the thermal energy equation. When stars reach the end of their lifetime, their fate largely depends on their mass. The least massive stars silently go out, without affecting their surroundings. However, the most massive stars explode in violent supernova explosions, which put large amounts of energy and debris in the surrounding ISM and will affect the local hydrodynamics.

In this section, we will discuss these different forms of stellar feedback in more detail, including the feedback from the first stars, that has properties that are very different from those of stars that were formed later. But first, we need to explain how stars are formed.

5.2.1 Star formation

Stars form when a dense region of the ISM becomes *Jeans unstable*, i.e. the hydrodynamical pressure inside the ISM can no longer support the region against the gravitational pull of its own mass, and the whole region starts to fragment into smaller clumps at the core of which stars are formed. The typical length scale at which a cloud is in hydrostatic equilibrium is called the *Jeans length*, and is given by (Jeans, 1902)

$$l_J = \sqrt{\frac{15kT}{4\pi G\mu\rho}}.$$

In a monatomic ideal gas that evolves only under its own gravity, clouds smaller than the Jeans length will expand, since the pressure is a strictly rising function of the density and temperature, and both temperature and density will rise when the gas gravitationally collapses. However, when the gas is allowed to cool radiatively, then the temperature will go down when the density rises, breaking this stability. Since for high densities the cooling scales quadratically with the density, while the pressure only scales linearly, the cloud will effectively become unstable for some critical density. Including the external gravitational potential generated by the dark matter halo of a galaxy only speeds up this process.

Due to the limited resolution of our simulations, we cannot resolve the collapse and fragmentation of individual clouds. We therefore will model star formation by replacing gas by star particles, representing SSPs, when the hydrodynamics we can resolve indicates that the gas has become Jeans unstable. In practice, there are three star formation criteria which need to be satisfied before gas is allowed to form a star particle: the gas should be cold enough (below $15,000\,\mathrm{K}$), should be in a region of collapsing flow (measured by a negative divergence of the fluid velocity), and should be dense enough (we adopt a density criterion of $100\,\mathrm{amu\,cm^{-3}}$) (Valcke *et al.*, 2008; Schroyen *et al.*, 2013).

To make sure we effectively resolve the Jeans length, we need to make sure that the gravitational resolution (set by the softening length) is at least equal to the Jeans length at the critical density and critical temperature.

When the three star formation criteria are satisfied, we remove a fraction of the local mass (corresponding to a single gas particle in SPH simulations) from the simulation and replace it by a star particle with the same mass. The star particle adopts the metallicity of the gas from which it was formed, and the current simulation time is taken as birth time of the star particle. For the remainder of the simulation, the star particle is treated as an N-body particle, and only affects the gas through stellar feedback.

5.2.2 SW and SNII

The most massive stars (with masses in the range $8-70\,M_\odot$) are short-lived, with lifetimes in the range $3.8-31\,\mathrm{Myr}$. During their lifetime, these stars already emit a significant amount of energy in the form of UV radiation. These Stellar Winds (SW) create hot, ionised bubbles around the young stars, which prevent further star formation in the vicinity of the star particle. To model this process, we heat the gas resolution elements (particles or cells) in the vicinity of the young stellar particle during the time interval in which young stars live, i.e. $0-31\,\mathrm{Myr}$. In total, we insert a fiducial $1.0\times10^{50}\,\mathrm{erg}$ per massive star into the surrounding gas during this time interval (Thornton *et al.*, 1998), uniformly spread out in time. These values differ from the values used in Valcke *et al.* (2008) and subsequent works due to the use of a different IMF.

When the heavy stars run out of fuel for the fusion reactions in their core that prevent them from gravitationally collapsing, they undergo a massive gravitational implosion, which causes a supernova explosion, whereby the outer shells of the star are violently expelled. Due to their spectral characteristics, these supernovae are called supernova type II explosions (SNII). An estimated $1.0\times10^{51}\,\mathrm{erg}$ is inserted in the ISM during this event (Thornton *et al.*, 1998), together with metal rich material from the outer shells of the star. In total, a fraction of 0.191 of the stellar mass is returned, of which most is ^{1}H and ^{4}He. In the two metallicity model we use, a fraction of 9.33×10^{-4} of the total stellar mass is returned to the ISM in the form of Fe, while a fraction 1.51×10^{-3} of the mass is returned as Mg. The energy and metal injection due to SNII is also spread out uniformly in time.

These early forms of stellar feedback have been proven to be crucial in galaxy evolution simulations, as they disperse the dense ISM and suppress further star formation (Valcke *et al.*, 2008; Cloet-Osselaer *et al.*, 2012; Schroyen *et al.*, 2013). The dispersed ISM can then cool and fall into the central galactic potential again to fuel later star formation.

There is however a problem with directly injecting the stellar feedback energy into the surrounding hydrodynamical resolution elements, if the resolution is low. The hot bubbles around young stars created by SW, or the shock bubbles around SNII are relatively small. Inside these bubbles, radiative cooling is very inefficient due to the high temperature of the ISM. Outside these bubbles, temperatures are significantly lower. If the resolution of the hydrodynamical model is too low to capture the bubbles, then the temperature inside the hydrodynamical cell or particle will be the average of the hot bubble and the surrounding cool ISM, resulting in a temperature that is too low for the hot bubbles, and too high for the surrounding gas. This average temperature will likely lie in a temperature range where radiative cooling is very efficient, leading to a serious over-cooling in the cell or particle. Therefore, most of the energy that we inject in the resolution

element will be immediately lost, and feedback becomes very inefficient.

To remedy this problem, various methods have been proposed. Some authors inject part of the supernova energy as *kinetic energy* rather than thermal energy, by directly injecting momentum in the neighbouring gas and temporarily decoupling the resolution element from its surroundings (Dalla Vecchia & Schaye, 2008). Although this mimics the supernova explosion, it leaves little control over the precise amount of energy that is injected and can drive unphysical outflows. Other authors inject thermal energy over a larger time interval, by explicitly keeping track of the shock front and inserting energy in all cells that lie inside the shock radius (R. Cen, personal communication, October 2013), although this method seems only feasible when a fixed grid is used. A popular method for Lagrangian methods is switching off radiative cooling for particles that receive stellar feedback (Thacker & Couchman, 2000), thus allowing the local ISM to go through a phase of adiabatic expansion before allowing it to cool again. We will use this last method for SW and SNII feedback, but not for the SNIa feedback discussed next.

5.2.3 SNIa

Stars with masses lower than $8\,\mathrm{M}_\odot$ do not go supernova, but expire relatively silently without a significant impact on the ISM. When the fusion reactions in their core no longer support them against gravitational collapse, they collapse to form *white dwarfs*, which are very compact objects that are supported by the pressure generated by the repulsive forces between electrons due to the Pauli exclusion principle. These white dwarfs have no significant impact on the ISM.

However, there is an upper limit on the mass of white dwarfs, set by the strange properties of the electron degenerate gas that supports it against collapse. For non-rotating stars with a composition similar to that of the Sun, this mass limit (called the Chandrasekhar limit) is $\sim 1.4\,\mathrm{M}_\odot$. When the mass of the white dwarf increases beyond this limit, the electron degenerate force can no longer support the star, and the entire star explodes in a very violent supernova type Ia (SNIa) explosion.

When a white dwarf forms, its mass will be below the Chandrasekhar limit, so that it needs to gain mass to cause a SNIa. For isolated white dwarfs, the chances of gaining enough mass are very small, so that we do not expect isolated white dwarfs to ever go supernova. However, a significant fraction of stars does not form in isolation, but rather forms as part of a binary system. For the Milky Way, an estimated one third of the stars is part of a binary system (Lada, 2006). In some of these systems, mass transfer between the two companions is possible if the *Roche lobes* overlap, i.e. the regions where the gravitational pull of one star is larger than that of the other. If one of the companions is a white dwarf, and

the other is a red giant that is expanding, mass of the red giant can be accreted onto the white dwarf. This increases the mass of the white dwarf, leading to an expansion of its Roche lobe and a further mass accretion, ultimately pushing the mass of the white dwarf over the Chandrasekhar limit (Mazzali *et al.*, 2007).

Due to the high fraction of binary systems and the fact that most stars evolve to white dwarfs, SNIa hence do occur. In our simulations, we assume a ratio of 0.15 SNIa explosions for every SNII explosion, each also releasing a fiducial energy amount of 1.0×10^{51} erg to the ISM (Valcke *et al.*, 2008). Just like SNII, SNIa also return a fraction of 6.55×10^{-3} of the mass of the star particle to the ISM; a fraction of 1.65×10^{-3} of its mass as Fe, and a fraction of $2.58 \times {}^{-4}$ of its mass as Mg. Note that SNIa return more Fe and less Mg than SNII, which is also the reason why we use a two metallicity model for the cooling and heating, to distinguish between these two contributions (De Rijcke *et al.*, 2013).

Contrary to SW and SNII feedback, SNIa feedback is spread out over a large time interval. Unlike Valcke *et al.* (2008) and subsequent work, we use the Gaussian model of (Strolger *et al.*, 2010) and return the total energy using a normal distribution that is centred on a delay time of 4 Gyr, and with a standard deviation of 0.8 Gyr (Bonaparte *et al.*, 2013). This decreases the large impact the old SNIa feedback model had on the gas of the simulation when including the UVB in our simulations (see Chapter 6).

Since SNIa go off when the ISM is already dispersed and the star particle has moved away from the region where it was born, the ISM surrounding the SNIa will have a low density and high temperature. We do not expect significant over-cooling in this case, so that we do not switch off radiative cooling in the case of SNIa feedback.

5.2.4 Population III stars

Radiative cooling at low temperatures is only efficient in metal rich gas, since it is dominated by the contribution of metal line cooling. Primordial gas has very low metallicities (Big Bang nucleosynthesis only produces stable elements up to ^{7}Li), so that the gravitational collapse of this primordial gas to form the first stars will be very different than for later star formation.

As a result, the first stars are thought to have been significantly more massive than stars that were formed later (Nomoto *et al.*, 2013; Susa *et al.*, 2014), and are expected to have an IMF that is very different from the Chabrier IMF. Due to their very low metallicities, these stars should have specific spectral properties that distinguish them from other stars. We call them population III (Pop III) stars, Pop I stars being the stars that we are most familiar with, while Pop II stars are low metallicity stars that still form like Pop I stars. Up to date, there are no confirmed observations of Pop III stars in the Universe (which, for dwarf

galaxies, is in line with what models predict, see Verbeke *et al.* (2015)). Recent observations of high redshift Lyα emitters however provide strong evidence for their existence (Sobral *et al.*, 2015).

Since they are more massive, Pop III stars will have even shorter lifespans than the most massive Pop I stars. When they go supernova, they insert more energy in the ISM than normal SNII (Nomoto *et al.*, 2013), while also emitting a lot of UV radiation during their short lifetimes (Heger & Woosley, 2010). They also return metals to the ISM, but in lower fractions than for SNII and SNIa.

For this work, we experimented with different models for Pop III feedback, ranging from very basic ad hoc models, to a full Pop III SW and SN model including return of metals to the ISM.

Model 1

As a first method to include the effect of Pop III feedback, we simply scale up the number of SNII explosions in metal poor star particles to match the energy output of a Pop III SSP with the same mass. We neglect Pop III SW for this model, and keep the same metal return as ordinary SNII (but then scaled up just like the energy output). The only degrees of freedom that are left then are (a) the mass range of Pop III stars, which sets the time interval for the feedback, and (b) the precise form of the Pop III IMF in this mass range, which sets the total energy output of the Pop III SSP. We use two different mass ranges, which we call Model 1a and Model 1b, and for each model assume different forms for the IMF.

Model 1a assumes Pop III stellar masses in the range $60 - 300\,\mathrm{M}_\odot$, giving a feedback time interval from $0.006 - 0.36\,\mathrm{Myr}$ after the star particle was formed. If we assume the same Chabrier IMF as for Pop I and Pop II stars, and extrapolate it out to this mass range, then we end up with a total energy feedback of $0.06358 \times 10^{51}\,\mathrm{erg}\,{\mathrm{M}_\odot}^{-1}$ (low feedback). If on the other hand we assume a flat IMF in this mass range (more in agreement with the IMF found by Susa *et al.* (2014)), this increases to $4.9025 \times 10^{51}\,\mathrm{erg}\,{\mathrm{M}_\odot}^{-1}$ (high feedback). To also allow for more complex IMF shapes, we also included a model with a feedback energy of $0.1467 \times 10^{51}\,\mathrm{erg}\,{\mathrm{M}_\odot}^{-1}$ (intermediate feedback).

Model 1b assumes a smaller Pop III mass interval of $140 - 300\,\mathrm{M}_\odot$, leading to a much shorter feedback time interval of $0.006 - 0.043\,\mathrm{Myr}$ over which the feedback is spread out. We again assume both a Chabrier and a flat IMF, leading to respective feedback energies of $0.1814 \times 10^{51}\,\mathrm{erg}\,{\mathrm{M}_\odot}^{-1}$ (low feedback) and $0.32 \times 10^{51}\,\mathrm{erg}\,{\mathrm{M}_\odot}^{-1}$ (high feedback).

Model 2

As a second, more advanced method for including Pop III feedback, we also include the effect of Pop III SW (Heger & Woosley, 2010), by scaling up the corresponding energy of a Pop I SSP. This feedback starts from the moment the Pop III star particle is born, and lasts until the last Pop III SN. We use the same time interval as for Model 1b. We compare two different versions of this model: one with high SW energy (10^{52} erg) and low SN energy ($0.007361 \times 10^{51}\,\mathrm{erg\,M_\odot}^{-1}$), and one with low SW energy (10^{51} erg) and high SN energy ($0.051765 \times 10^{51}\,\mathrm{erg\,M_\odot}^{-1}$).

Model 3

Our most advanced Pop III feedback model is based on the version of Model 2 with low SW feedback and high SN feedback, and also includes a more realistic metal return to the ISM. Contrary to Model 2, the feedback is now given in a time interval of $1.25 - 16.7$ Myr, and takes into account the contributions of lower mass stars that explode as ordinary SNII. A Pop III star returns a fraction of 0.45 of its mass to the ISM. A fraction of 9.327×10^{-5} of its mass is returned in the form of Fe, while a fraction of 1.514×10^{-4} of its mass is returned as Mg. These values are 10% of the corresponding value for a normal SNII, and were based on the metal yields from Nomoto *et al.* (2013). By keeping the ratio [Mg/Fe] fixed, we make sure we can still use our two metallicity model for cooling and heating.

This model was used for Verbeke *et al.* (2015).

5.2.5 Feedback efficiency

The numbers given above correspond to the expected energy output of the various stellar feedback processes. However, this energy is released in many different channels, and it is not a priori clear how much of it will effectively be absorbed by the ISM. Another problem is that it is also not clear if all of the energy we deposit into the ISM will effectively do what we expect it to do, owing to all sorts of resolution issues, consider e.g. the over-cooling problem.

As a result, we will not treat the feedback processes as being fixed, but make them a parameter in the model, which needs to be calibrated. This parameter is called the feedback efficiency parameter, and is present in almost all numerical simulations of galaxy formation or evolution (Governato *et al.*, 2010; Cloet-Osselaer *et al.*, 2012; Vogelsberger *et al.*, 2014b; Shen *et al.*, 2014; Schaye *et al.*, 2015). When calculating the amount of feedback energy to put into the local ISM, we will multiply this value with the feedback efficiency parameter. The parameter is the same for all forms of feedback, ranging from SW, SNII and SNIa feedback to Pop III feedback, since the uncertainties and numerical issues

with feedback equally apply to all these forms of feedback. Since different forms of feedback have different time dependencies and different relative importance, it still makes sense to distinguish between them.

As Cloet-Osselaer *et al.* (2012) showed, the feedback efficiency parameter closely relates to another parameter of the model: the star formation efficiency parameter. This parameter sets the probability that gas that is eligible for star formation (i.e. satisfies the star formation criteria) effectively forms a star particle, and is chosen to reproduce an observational Schmidt law (Schmidt, 1959). It was found that varying the star formation efficiency does not significantly change the total stellar mass produced by the simulation, since less efficient star formation leads to more initial star formation, leading to more feedback that shuts down further star formation. Changing the feedback efficiency parameter has similar effects, since more initial feedback leads to less initial star formation, which leads to an overall lower initial feedback which increases later star formation.

Changing the star formation efficiency parameter and the feedback efficiency parameter together does hence not make much sense, and we choose to keep the star formation efficiency fixed at a value of 0.1 throughout this work. Changing the feedback efficiency parameter has only effect when the changes are significant enough, so that we distinguish between low feedback efficiencies (~ 0.1), high feedback efficiencies (~ 0.7), and unphysically high feedback efficiencies (1 or more). The latter can be justified due to the uncertainties on the hydrodynamical modelling, and we will only use them to illustrate the effect of increasing the stellar feedback. The low feedback efficiencies were shown to produce too many stars when combined with the high density threshold for star formation (Cloet-Osselaer *et al.*, 2012; Schroyen *et al.*, 2013), so that we will use the high feedback efficiency parameter 0.7 for most of our work.

6

Constraining the sub-grid physics in simulations of isolated dwarf galaxies

In this chapter, we use the ingredients introduced in the previous chapters to study the formation and evolution of dwarf galaxies. This work is a natural extension of the work of Valcke *et al.* (2008), Schroyen *et al.* (2011), Cloet-Osselaer *et al.* (2012) and Schroyen *et al.* (2013), and introduces the UV background as a new ingredient in our model. As we will show, this leads to major problems, that can be solved by introducing Pop III feedback (Vandenbroucke *et al.*, 2016), combined with the use of merger trees (Cloet-Osselaer *et al.*, 2014). This then led to the worked presented in Verbeke *et al.* (2015).

6.1 Influence of the UVB

Fig. 6.1 shows the star formation rate (SFR) for a typical dwarf galaxy in isolation, as discussed in Schroyen *et al.* (2013). The star formation shows a clear peak at the start of the simulation, when the gas first collapses to form stars. Feedback from these first stars then disperses the dense gas at the center and causes the subsequent SFR to be lower. However, there is ongoing star formation until the end of the simulation.

Fig. 6.2 shows the same simulations with one extra ingredient in the models: the cosmological UV background (UVB). The UVB has a huge effect on the SFR: after the first star formation peak, the gas is dispersed as before, but now it is unable to form any more stars, causing an effective halt of the star formation. The cause of this dramatic effect is UVB heating of the dispersed, low density gas, which cannot shield against the UVB. All gas that at some time during the simulation receives enough feedback to drop below the density threshold for self-shielding ($n_{\mathrm{H}} = 0.007\,\mathrm{amu\,cm^{-3}}$) will be heated and can never become dense

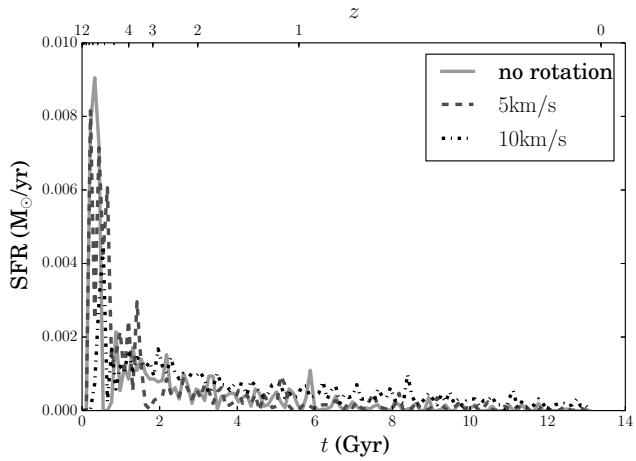


Figure 6.1: *The star formation rate for a model without UVB. After an initial star formation peak, star formation continues on a lower level until the end of the simulation. The models shown here correspond to the models C4P2M1R00L, C4P2M1R05L and C4P2M1R10L introduced below.*

enough again to form stars. In simulations without a UVB, this same gas is recycled and fuels further star formation.

Observed SFRs for isolated dwarf galaxies show a decrease in SFR around $z = 2$ (Weisz *et al.*, 2014), corresponding to the peak strength of the UVB. The effect of the UVB can hence be observed. However, these observations also show signs of ongoing star formation after $z = 2$, indicating that the effect in our simulations is somehow too strong. Furthermore, stellar masses for observed dwarf galaxies are typically lower than for simulated dwarf galaxies that live in a dark matter halo with the same estimated mass (Sawala *et al.*, 2015), and have a significantly higher neutral gas content (McGaugh, 2012). This means that real dwarf galaxies are able to retain a significant fraction of their neutral gas mass and keep it dense enough to self-shield against the UVB, yet dispersed enough to prevent it from forming stars.

Low mass halos with masses in the range $10^7 - 10^9\,\mathrm{M}_\odot$ are massive enough to keep the gas in their center in the density range $0.007 - 100\,\mathrm{amu\,cm^{-3}}$, which corresponds to gas that does not form stars, but does self-shield against the UVB. Since in the ΛCDM model of galaxy formation all halos form by the merging of less massive halos, these halos could well provide the neutral gas reservoir that

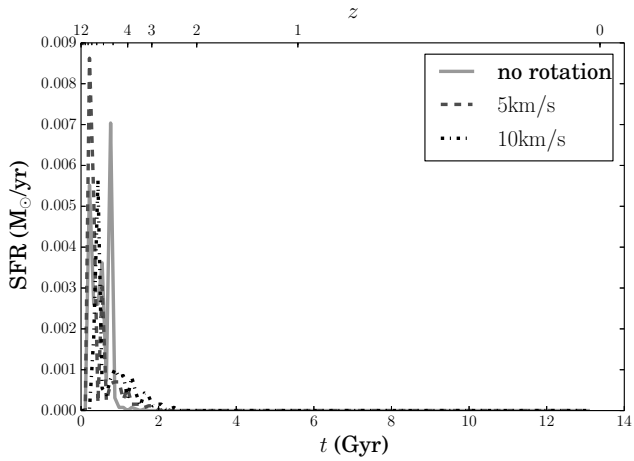


Figure 6.2: *The star formation rate for the same model as Fig. 6.1, but with a UVB. After the initial star formation peak, star formation is completely shut down. The models shown here correspond to the models C3P2M1R00L, C3P2M1R05L and C3P2M1R10L introduced below.*

fuels star formation after the initial star formation peak. However, simulations that take into account the effect of mergers (Verbeke *et al.*, 2015), still show a significant initial star formation peak and very low level late star formation.

The only way to reconcile theory and observations seems to be the suppression of the initial star formation peak (Trujillo-Gomez *et al.*, 2015). In the remainder of this chapter, we focus on a large parameter study of the models discussed in Chapter 5 and the next section, with the aim to achieve this. We pay special attention to a fair comparison of simulation results and observations, by using the same techniques observers use to analyse the simulation snapshots.

6.2 Models

Since we want to explore a large parameter range, we need to use simulations that are computationally cheap enough to run a large number of them. For this reason, we focus on simulations of isolated galaxies. We note that this approach is not ideal for reproducing dwarf galaxies that are to be compared directly to observations, since these simulations miss the important effect of cosmological gas accretion. Isolated simulations do however allow to qualify the effect of changing

sub-grid parameters on some important global properties of the dwarf galaxy, like the SFR, so that they are fit for our purpose, which is finding a sub-grid model that reduces the initial star formation peak.

6.2.1 Initial conditions

Halo

The initial conditions for our simulations consist of a virialized halo, containing only cold dark matter (DM) and gas. The former is set up as an NFW-halo (Navarro *et al.*, 1997):

$$\rho_{\rm DM}(r) = \frac{\rho_{{\rm DM},c}}{\frac{r}{r_c}\left(1+\frac{r}{r_c}\right)^2},$$

with a concentration parameter (Cloet-Osselaer *et al.*, 2014)

$$c \approx 33\left(\frac{M_{\rm h}}{10^8\,{\rm M}_\odot}\right)^{-0.06},$$

where $M_{\rm h}$ is the total mass of the DM halo.

The gas halo is set up as a pseudo isothermal sphere, with a density profile of the form (Schroyen *et al.*, 2013)

$$\rho_{\rm gas}(r) = \frac{\rho_{{\rm gas},c}}{1+\left(\frac{r}{r_c}\right)^2},$$

with r_c the scale length of the NFW-halo, and $\rho_{{\rm gas},c}$ the central gas density, which is related to the scale density $\rho_{{\rm DM},c}$ of the NFW-halo:

$$\rho_{{\rm gas},c} = \frac{\Omega_{\rm b}}{\Omega_{\rm DM}}\rho_{{\rm DM},c},$$

with $\Omega_{\rm b}/\Omega_{\rm DM} = 0.2115$ (Spergel *et al.*, 2007).

The velocities of the dark matter particles that sample the halo are drawn from the isotropic distribution function that corresponds to the NFW density profile, with so called 'quiet' initial conditions (Cloet-Osselaer *et al.*, 2012), that make sure the initial density cusp in the DM profile is not washed out by Poisson noise.

The gas particles that sample the gas halo are initially at rest, or are given a constant solid body rotation, with $v_{\rm rot}$ being a model parameter.

Resolution

We sampled our halos using 50,000 particles for both the DM and the gas halo. To assess whether this is enough to obtain a converged SFR, we also used a high resolution version of some of the initial conditions using 200,000 particles to run convergence tests. An alternative sampling of the initial conditions using 50,000 particles but a different random seed was used to assess the effect of stochastic differences between simulations using the same model.

As the total halo mass is a parameter in our study, not all simulations have the same mass resolution. The softening length of the different components (DM and gas, and later in the simulation also stars) is set to a value which roughly corresponds to the smoothing length of a gas particle that has a density equal to the star formation density ($100\,\mathrm{amu\,cm^{-3}}$), and hence also depends on the gas particle mass and the total halo mass.

6.2.2 Code

The simulations for this work were run using an adapted version of GADGET2, and include the 5D cooling, heating and gas physics discussed in Chapter 5. Stellar feedback by SW, SNII and SNIa is used in all models, while Pop III feedback is only included in a subset of our models. The UVB is included in all but one model, and to qualify the effect of different UVB models, we also ran simulations with a UVB that only starts at a redshift of 7 (instead of 10.5), and with a UVB with a strength that is only 10% of the normal strength. To quantify the effect of the adiabatic cooling period for gas particles receiving stellar feedback, we ran one model with cooling enabled for gas particles that received SW and SNII feedback.

The simulations start at a redshift of 12, which corresponds to a lookback-time of 13.37 Gyr, and were run until redshift 0 on our local computing infrastructure, consisting of 5 computing nodes with multicore Intel CPUs.

To limit the use of computational resources, a 3 month time limit was imposed on all simulations. 7 simulations exceeded this limit because the stellar feedback contribution was computationally too expensive. All these simulations formed too many stars from the very beginning of the simulation, so that we would not learn anything new from them anyway.

3 simulations crashed during the run, because of problems in the stellar feedback algorithm. These problems are also related to these simulations forming way too many stars.

Table 6.1: *Naming convention for code and parameter values.*

Code	Symbol	UVB model	$f_{\rm feedback}$	Pop III model	#
C1P1	▲	low and late	0.7	no Pop III stars	30
C1P1bis	▲	low and late	0.7	no Pop III stars	15
C2P1	◇	full and late	0.7	no Pop III stars	16
C3P1	■	full and early	0.7	no Pop III stars	16
C3P2	■	full and early	1.0	no Pop III stars	30
C3P3	□	full and early	2.0	no Pop III stars	16
C4P2	○	no UVB	1.0	no Pop III stars	16
C7P4	◀	full and early	0.7	Pop III model 1A: low feedback	16
C7P6	◀	full and early	0.7	Pop III model 1A: high feedback	16
C7P7	◁	full and early	0.7	Pop III model 1A: intermediate feedback	16
C9P8	▶	full and early	0.7	Pop III model 1B: low feedback	16
C9P9	▶	full and early	0.7	Pop III model 1B: high feedback	6
CaPa	▼	full and early	0.7	Pop III model 2: high stellar winds, low feedback	16
CbPc	♠	full and early	0.7	Pop III model 2: low stellar winds, high feedback	16
CcPd	♣	full and early	0.7	Pop III model 3	6
CeP1	♡	full and early	0.7	no Pop III stars, no adiabatic cooling period	16

6.2.3 Model names

To keep track of the different models, we adopt a simple naming convention for our simulations. The name of a simulation is composed of two parts: a part that indicates which code and parameter model was used (denoted by 4 characters), and a part indicating which initial condition file was used (usually consisting of 6 characters). The former are listed in Table 6.1 together with the number of simulations run with this model and a symbol that will be used to depict simulations of these model in general overview figures. The latter are listed in Table 6.2, together with the corresponding resolution parameters.

In general, models are run with the 15 low resolution initial conditions, and

Table 6.2: *IC naming convention.*

Mass and resolution code	DM particle mass ($10^3\,M_\odot$)	gas particle mass ($10^3\,M_\odot$)	softening length (pc)
M1L	20.0	4.23	9.76
M1H	5.0	1.06	6.15
M3L	60.0	12.7	14.1
M3H	15.0	3.17	8.87
M5L	100.0	21.2	16.7
M5H	25.0	5.29	10.5
M7L	140.0	29.6	18.7
M7H	35.0	7.40	11.8
M9L	180.0	38.1	20.3
M9H	45.0	9.52	12.8

Rotation code	Physical velocity ($\mathrm{km\,s^{-1}}$)
R00	0.0
R05	5.0
R10	10.0

one high resolution initial condition (M9R10H) is used for the convergence test. For some models, only the 5 R10 low resolution initial conditions were used, together with the high resolution M9R10H convergence simulation. For two models, all 15 high resolution initial conditions were used for a more extensive convergence test. Model C1P1bis is identical to C1P1, but uses initial conditions sampled using a different random seed as a test for stochastic effects. Only the 15 low resolution initial conditions were used in this case.

In total, 263 simulations were run, of which 250 will be discussed in the remainder of this chapter. Of the 13 simulations not discussed, 7 exceeded the 3 month time limit imposed. 3 crashed, while 3 were discarded because they form an excessive amount of stars, which makes further analysis impossible. Furthermore, these last 3 show a strong increase in circular velocity and have almost no neutral gas at the end of the simulations, contrary to other simulations using the same model. The models that exceeded the time limit showed similar behaviour, which leads us to conclude that the more massive models with no rotation are unphysical. The simulations that were discarded are listed in Table 6.3.

6.3 Analysis

We compare our simulations with the observed baryonic Tully-Fischer relation (BTFR) of McGaugh (2012). This relation is an interesting test, since it relates

Table 6.3: *Simulations that were discarded from our set.*

Name	Reason
C1P1M7R00H	exceeded time limit
C1P1M9R00L	excessive star formation
C1P1M9R00H	exceeded time limit
C1P1M9R05H	exceeded time limit
C1P1bisM9R00L	excessive star formation
C3P2M9R00L	excessive star formation
C3P2M9R00H	exceeded time limit
CeP1M1R00L	crashed
CeP1M3R00L	crashed
CeP1M5R00L	exceeded time limit
CeP1M7R00L	exceeded time limit
CeP1M9R00L	crashed
CeP1M9R10H	exceeded time limit

two model independent observable quantities, unlike other methods that depend on model fits to data to estimate quantities like the halo mass or the half-light radius (Wolf *et al.*, 2010; Garrison-Kimmel *et al.*, 2014). The circular velocity itself cannot be directly measured either, but a clearly defined proxy is used that can be: the rotation of the neutral gas. We show below that this rotation velocity is not a particularly good proxy for the real circular velocity of a halo, but it is a quantity that we can calculate from the simulations to at least make a fair comparison between simulations and observations.

Similarly, we use observational techniques to determine the stellar mass of a simulated galaxy, rather than just use the sum of the masses of the star particles. This will appropriately weigh the contributions of old and young stars, and will limit the mass estimate to the central parts of the galaxy that can be observed.

In this section, we detail the techniques used to analyse the results. We start by showing a number of general simulation properties, and show how they correlate with the mock observational values we obtain. This can help us assess how well observational proxies actually predict the real value for a quantity, as determined from the simulation, and can be of use for observational astronomers as well.

6.3.1 General properties

Fig. 6.3 shows some general properties of the simulated halos: the total halo mass, stellar mass and neutral gas mass, and the virial radius. The masses are taken to be the masses within the virial radius, with the virial radius calculated

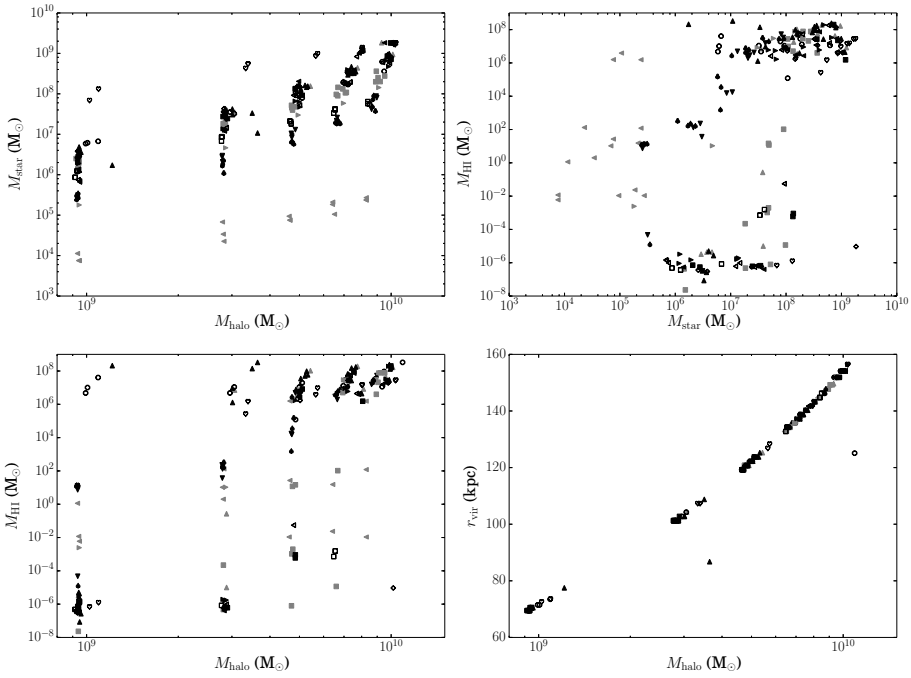


Figure 6.3: *General properties of the simulations.* Top left: *stellar mass as a function of halo mass,* top right: *neutral gas mass as a function of stellar mass,* bottom left: *neutral gas mass as a function of halo mass,* bottom right: *virial radius as a function of halo mass. The different symbols correspond to the different models in Table 6.1.*

as the radius at which the mean density inside the spherical halo equals 200 times the mean density of the Universe, $4.5 \times 10^{-31}\,\mathrm{g\,cm^{-3}}$ (Spergel *et al.*, 2007).

Since the initial conditions are parametrized by the mass of the DM halo, the simulations form groups with similar halo masses. Furthermore, there is a strong correlation between halo mass and virial radius. There is no clear link between halo mass and stellar mass or neutral gas mass, with different models leading to very different observed galaxies. Halos which form more stars are more likely to have more neutral gas, although there are a lot of exceptions.

We can also compare some general mock observational properties of our simulations with observed galaxies on the so called *scaling relations*. These scaling relations are correlations between observational quantities that were found to

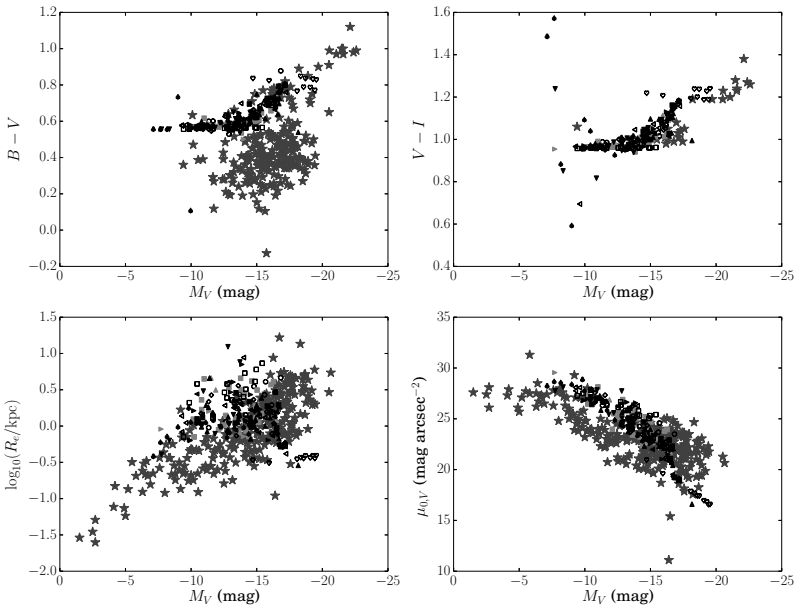


Figure 6.4: *Our simulations on four observational scaling relations.* Top left: $B-V$ *colour,* top right: $V-I$ *colour,* bottom left: *half-light radius,* bottom right: *central surface brightness. The stars correspond to the observational data, while the other symbols correspond to the different models in Table 6.1.*

hold for observed dwarf galaxies, and that set the typical size and luminosity of a dwarf galaxy. As Cloet-Osselaer *et al.* (2012) showed, producing simulated dwarf galaxies that are in line with these scaling relations is almost trivial, and hence we cannot use them to constrain our models. They are however a good first check on the results, since simulations that cannot produce galaxies consistent with these relations should certainly be discarded.

The luminosities and half-light radii of the simulated galaxies were estimated by fitting a Sérsic profile to the surface brightness profile of the galaxies, cut off at a surface brightness of $30\,\mathrm{mag\,arcsec^{-2}}$. The surface brightnesses in the B, V and I band were estimated from the age and metallicity of the star particles by interpolating on the tables of Vazdekis *et al.* (2012). Not all simulations contain enough stars to fit a general Sérsic profile, so that we resort to a simple

exponential curve if the general profile visibly yields a bad fit. The C7P6 models form so little stars that even an exponential fit does not work, so that we do not show them on the scaling relations. In total, 234 simulations were fitted, of which 216 with a general Sérsic profile, and 18 with a simpler exponential profile.

Fig. 6.4 shows four scaling relations, together with the observational data for early and late type galaxies in the Local Volume (van Zee, 2000; Grebel *et al.*, 2003; Hunter & Elmegreen, 2006; McConnachie, 2012), including recent additions like Leo P (McQuinn *et al.*, 2013; Rhode *et al.*, 2013), and Pisces A and B (Tollerud *et al.*, 2015), galaxies within the Coma (Graham & Guzmán, 2003), Virgo (van Zee *et al.*, 2004) and M81 cluster (Lianou *et al.*, 2010), and isolated dwarf galaxies (van Zee, 2000; Magorrian & Ballantyne, 2001; Geha *et al.*, 2003; Grebel *et al.*, 2003; Hunter & Elmegreen, 2006; Dunn, 2010). The simulations are clearly in agreement with the observed relations, although the $B-V$ colours and the half-light radii are rather high, two effects that can be linked to a large initial peak of star formation in all our simulations. There is a large scatter on the $V-I$ colour for fainter galaxies, due to the relatively bad quality of the fits to the low stellar mass surface brightness profiles at this end.

6.3.2 Stellar mass

Observationally, stellar masses are estimated from the total luminosity of the galaxy, taking into account information about the age and metallicity of the stars through a colour. We will do the same to obtain mock observational masses for our simulations, using the relation of Bell & de Jong (2001) and the I band luminosity and $V-I$ colour obtained above.

For the C7P6 models where we were unable to fit a surface brightness profile, we will just sum up the contributions of the star particles, which corresponds to the values in Fig. 6.3.

In Fig. 6.5, we show the mock observational mass as a function of the actual stellar mass, obtained by summing the masses of the individual star particles. There is a clear correlation between both quantities, but the mock observational mass is in general lower than the true value, with the ratio given by 0.63 ± 0.38. We have discarded the C7P6 and CeP1 models to fit the ratio, as well as models with a fitted Sérsic index larger than 1.5.

The systematic offset between the mock observational mass and the true value is both due to the differences in assumed IMF between the mass to light ratio model of Bell & de Jong (2001) and the IMF assumed to derive luminosities for the star particles (Vazdekis *et al.*, 2012), and due to the low luminosity star particles at the outskirts of the halo which are too faint to be observed.

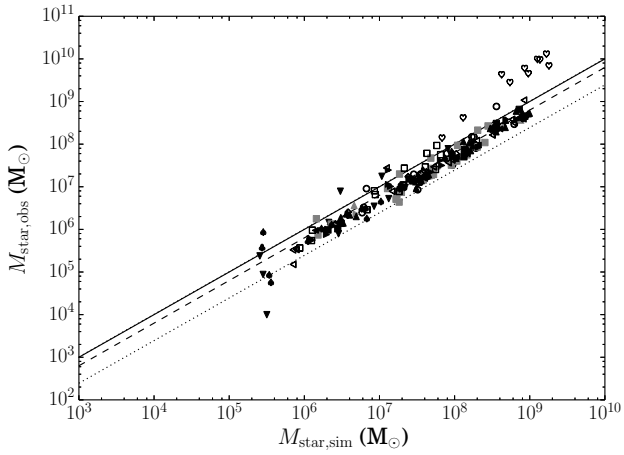


Figure 6.5: *The mock observational stellar mass as a function of the true stellar mass for all our models. The full line is a 1:1 relation, the dashed line corresponds to the fitted mean ratio, while the dotted lines correspond to a 1σ interval around this ratio. The different symbols correspond to the different models in Table 6.1.*

6.3.3 Neutral gas mass

Neutral gas can be observed as HI clouds through the 21 cm radio emission line. Neutral gas is not necessarily confined to the center of the galaxy, although none of our simulations has neutral gas far away from the center. We will therefore estimate the mock observational neutral gas mass by simply summing the contributions of the different gas particles in the simulation, without imposing an artificial cutoff.

To this end, we estimate the neutral fraction of a gas particle by a 5D interpolation on the precalculated tables of De Rijcke *et al.* (2013), that are also used for the cooling, heating and gas physics, as described in Chapter 5. These tables take into account background ionising radiation from local stars and the cosmic UVB, and hence also depend on the details of our models.

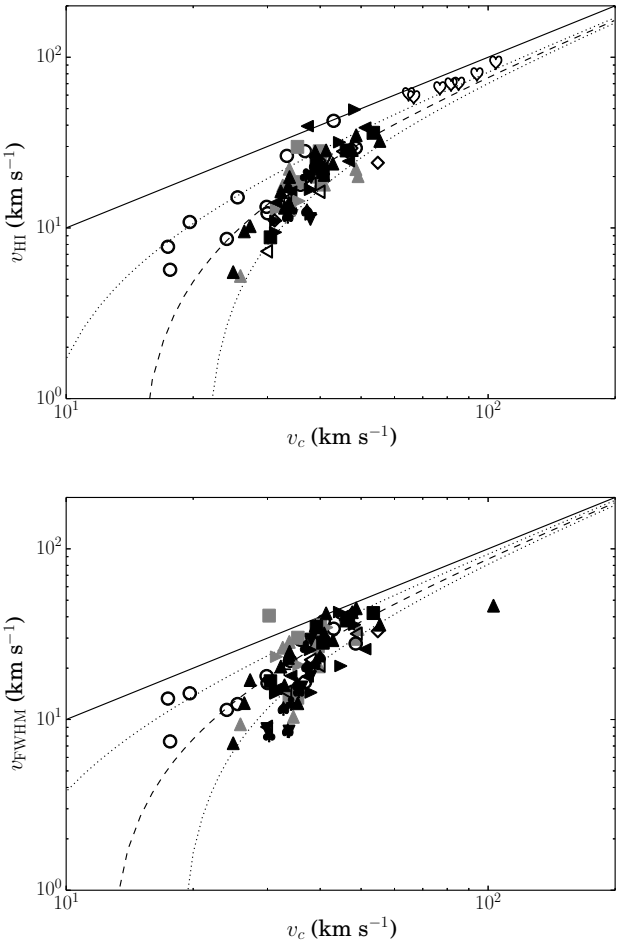


Figure 6.6: *The circular velocity derived from the rotation curve of the neutral gas (*top*), and derived from mock spectral line widths (*bottom*) as a function of the theoretical circular velocity. The full line is a 1:1 relation, while the dashed and dotted lines correspond to a least squares fit to the data and the corresponding 1σ confidence interval. The different symbols correspond to the different models in Table 6.1.*

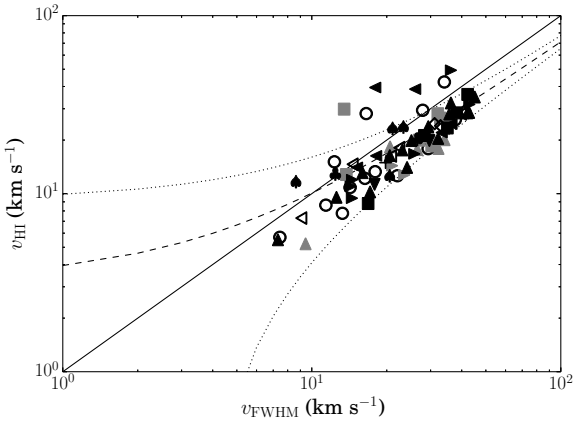


Figure 6.7: *The circular velocity derived from the rotation curve of the neutral gas as a function of that derived from mock spectral line widths. The solid line corresponds to a 1:1 relation, while the dashed and dotted lines are a least squares fit and the corresponding 1σ confidence interval. The different symbols represent the different models from Table 6.1.*

6.3.4 Circular velocity

The circular velocity of a spherically symmetric halo is a theoretical quantity, that is defined as

$$v_c(r) = \sqrt{\frac{GM(< r)}{r}},$$

with $M(< r)$ the total mass within radius r. It is a measure for the strength of the total gravitational potential of the halo, including the mass contributions that cannot be observed, and sets a limit to the movement of stars and gas within this potential. The circular velocity is usually characterised by quoting the value at a given radius, or by taking the maximal value. Both methods converge at large enough radii, where the circular velocity profile tends to become flat.

Observationally, the circular velocity can only be traced by observing the motions of gas and stars within the potential. Since the maximum of the circular velocity profile usually lies outside the main stellar body of the galaxy, these observations will likely underestimate the true circular velocity. It is therefore very important to use a good mock observational circular velocity to compare simulations and observations, as using the theoretical value for the simulations would lead to a systematic offset with respect to the observations.

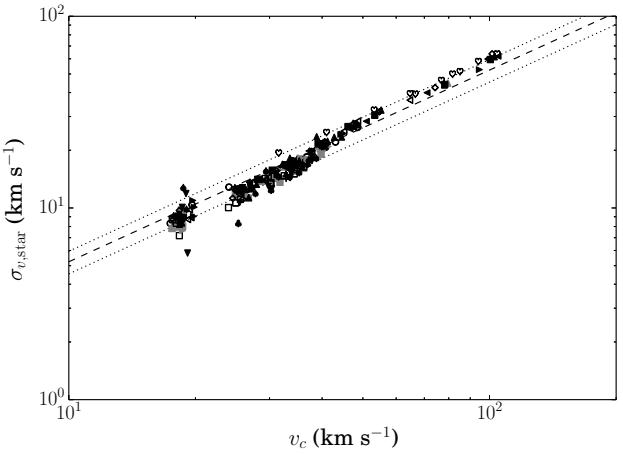


Figure 6.8: *The stellar velocity dispersion in the x, y and z direction within a sphere with radius two times the half-light radius, as a function of the theoretical circular velocity of the halo. The dashed and dotted lines show a least squares fit and the corresponding 1σ confidence interval. The different symbols represent the different models from Table 6.1.*

We use two different tracers to determine mock circular velocities: the rotation of the neutral gas and the movement of the stars. For the former, we produce mock rotation curves for the neutral gas and determine the maximal value of the velocity. We visually checked the rotation curves and only accepted the values that were obtained from curves with a clear rotation and enough neutral gas. The resulting circular velocities are compared with the theoretical circular velocity in the top panel of Fig. 6.6. Alternatively, we also produced mock spectral lines for the neutral gas, and fitted a normal distribution to the broadened spectral lines. The circular velocity can then be estimated as half W_{20}, the width of the Gaussian bell curve at 20% of its maximal value (McGaugh, 2012). The values obtained in this way are compared to the theoretical circular velocity in the bottom panel of Fig. 6.6. We use these estimates if enough neutral gas mass is available, but the rotation curve yields no clear rotation.

In total, 113 simulations contained enough neutral gas to estimate circular velocities. For 90, we were able to estimate a circular velocity using both techniques, these simulations are shown in Fig. 6.7. 11 more had a clear HI rotation profile, so that we can use rotation curve based values for 101 simulations. For

the 12 remaining simulations, we use the values derived from the spectral line widths instead. None of the CeP1 simulations contain enough neutral gas to be observable, but the neutral gas shows a very strong rotation, so that we have calculated circular velocities for them nonetheless. This allows us to show them on the BTFR below.

The circular velocity estimates are clearly lower than the theoretical value, and there is a significant amount of scatter on the values. The circular velocities derived from spectral line widths are a bit higher than those derived from rotation curves.

For the 150 simulations that do not contain enough neutral gas to estimate circular velocities, we use the stars to trace the gravitational potential. For an isothermal sphere, the stellar velocity dispersion along a line of sight can be shown to correlate linearly with the circular velocity (Binney & Tremaine, 2008), and we expect a similar relation to hold for our galaxies. Fig. 6.8 shows the velocity dispersion in the x, y and z direction of the stars within a sphere with radius 2 times the half-light radius, as a function of the theoretical circular velocity. There is a clear correlation, given by

$$\sigma_v = (0.52 \pm 0.07) v_c.$$

6.3.5 BTFR

The top panel of Fig. 6.9 shows our simulations on the BTFR of McGaugh (2012). Since not all our simulations have a circular velocity derived from their neutral gas, we have extended this figure with an alternative representation that uses the circular velocity derived from the stellar velocity dispersion in the bottom panel, and compare it with the observations from McGaugh & Wolf (2010). We also indicated the fit of McGaugh (2012) on both panels.

It is clear that the simulations are systematically above the observed relation in the top panel, indicating that they form too many stars, or that their circular velocity is gravely underestimated. The simulations seem to be in better agreement with the observations in the bottom panel. The bottom panel also shows a clear drop in baryonic mass at a circular velocity of $\sim 30\,\mathrm{km\,s^{-1}}$, signalling the transition from gas-rich to gas-poor galaxies. A similar drop off is seen in large cosmological simulations (Sales *et al.*, 2016), and in observations, although there it is less pronounced and happens at lower circular velocity. These gas-poor simulations are of course absent from the top panel.

6.3.6 Metallicities

We have seen above that our simulations lie well within the range of the observed scaling relations. They lie above the BTFR of McGaugh (2012), but are in relative

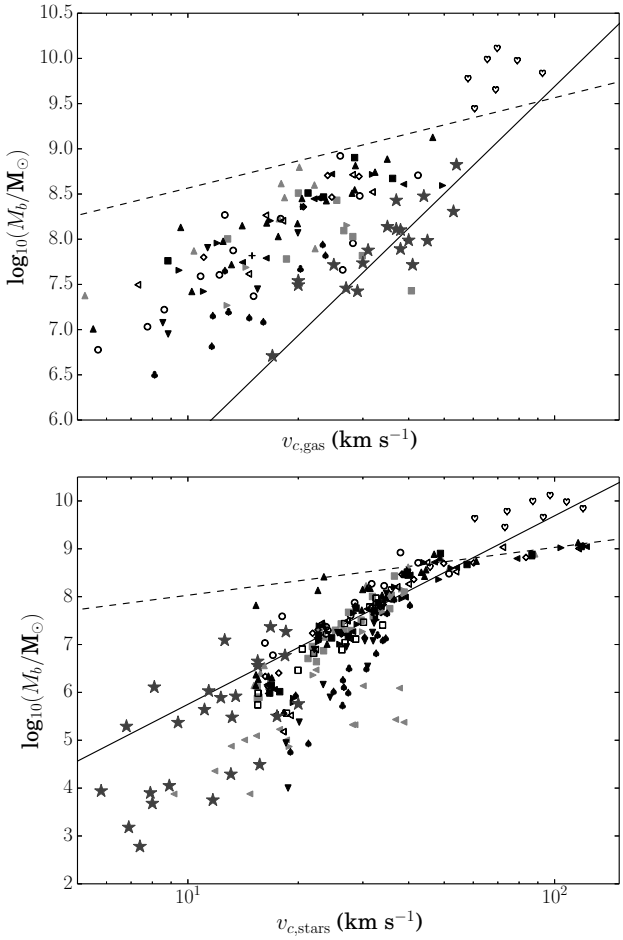


Figure 6.9: *The BTFR for all simulations in the set.* Top: *baryonic mass as a function of the circular velocity derived from the neutral gas,* bottom: *baryonic mass as a function of the circular velocity derived from the stellar velocity dispersion. The stars indicate the observations, while the different symbols correspond to the different models from Table 6.1. The full line is the fit of McGaugh (2012), and the dashed line is a least squares fit to the simulation values.*

agreement with the alternative BTFR of McGaugh & Wolf (2010). Apart from the systematically high $B-V$ colour and relatively large half-light radii, there is hence no clear indication that the large initial star formation peak in the simulations leads to unrealistic dwarf galaxies. This indication is provided when we look at the metallicities of our galaxies.

As already discussed in Chapter 5, the metallicity is a measure of the amount of elements heavier than ^{1}H and ^{4}He present in the ISM. A popular metallicity tracer is [Fe/H], as Fe is not formed during Big Bang nucleosynthesis, and hence has to be produced during stellar nucleosynthesis.

Metal lines can be observed in the absorption line spectra of stars; we compare our simulations with the metallicities of Kirby *et al.* (2013), which are based on the optical colours of red giant branch (RGB) stars. Since the absorption spectrum of a star originates from the outer shells of the star, it traces the metal contents of the gas from which the star was formed. Stars that formed early on from primordial gas will have low metallicities, and the global metallicity is expected to be an increasing function of time.

To estimate stellar metallicities, we use the stellar evolution tracks of Bertelli *et al.* (2008, 2009) for Pop I and Pop II stars, and those of Marigo *et al.* (2001) for Pop III stars to calculate the fraction of the star particle that resides on the RGB at a given time. We then weigh the metallicity of the star particle with that fraction to calculate an average [Fe/H] value. This method is biased towards older star particles, as they have a larger mass fraction of RGB stars.

A peculiarity from Kirby *et al.* (2013) is their definition of the average metallicity of a galaxy, as they define this is as the averaged [Fe/H] value for the observed RGB stars in that galaxy, $\langle[\mathrm{Fe/H}]_{RGB}\rangle$. As the [Fe/H] value is the logarithm of the metal mass, it would be more meaningful to average the metal masses and calculate a [Fe/H] value from this average, $[\langle\mathrm{Fe/H}\rangle]_{RGB}$. Both approaches are illustrated in Fig. 6.10, and compared with the average luminosity weighted metallicity of the star particles, $[\langle\mathrm{Fe/H}\rangle]_{lum}$. The method of Kirby *et al.* (2013) clearly underestimates the metallicity, so that it is important to use a similar definition when comparing our simulations with their data.

From Fig. 6.11 it is then clear that the simulated galaxies have significantly lower metallicities than observed. This is a clear indication of the large initial star formation peak, as the stars that formed during this peak will have low metallicities. The only way to increase the metallicity would be an increase of the star formation at later times. However, since the simulations already form too many stars, this is only possible if we also drastically reduce the initial star formation peak.

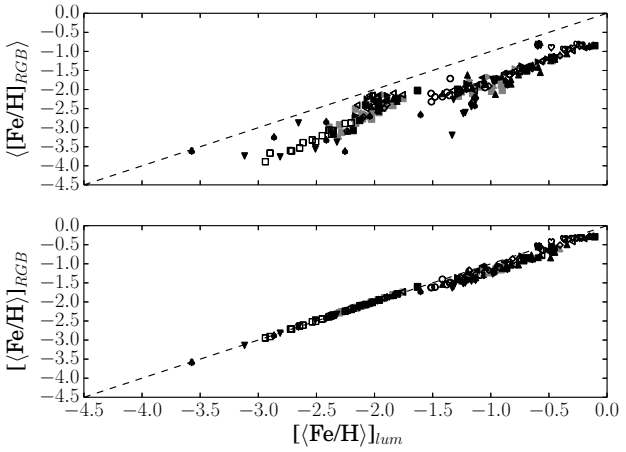


Figure 6.10: *The RGB weighted metallicity as a function of the luminosity weighted metallicity for our simulations.* Top: *averaged* $[Fe/H]$ *value, as used by Kirby et al. (2013),* bottom: $[Fe/H]$ *value of the averaged metallicity. The dashed line represents a 1:1 relation, the different symbols represent the different models from Table 6.1.*

6.4 Results

In the previous section, we showed general properties of all our simulations. It is clear that all simulations behave well for some properties, like the global scaling relations. However, we also showed that all simulations have metallicities that are too low compared with observations, and lie above the observed BTFR. In this section, we focus on the individual models, and discuss the effect of the variation of a single model parameter on the properties of the simulations. We begin by quantifying the influence of stochastic effects and resolution.

6.4.1 Stochastic effects

As mentioned in Chapter 2, sampling a density distribution with a finite number of particles will always lead to Poisson noise. This noise can be reduced by using special techniques, but it can never be completely eliminated. There will hence always be small density fluctuations in the initial setup of the simulations, both in the DM and in the gas component. As these components are evolved under the force of gravity, the small random overdensities might grow into larger

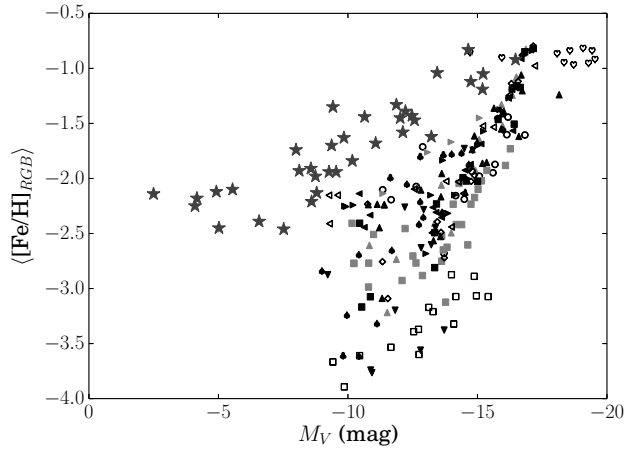


Figure 6.11: *The averaged $[Fe/H]$ value as a function of the V band luminosity. The stars are the observations from Kirby et al. (2013), the different symbols correspond to the different models from Table 6.1.*

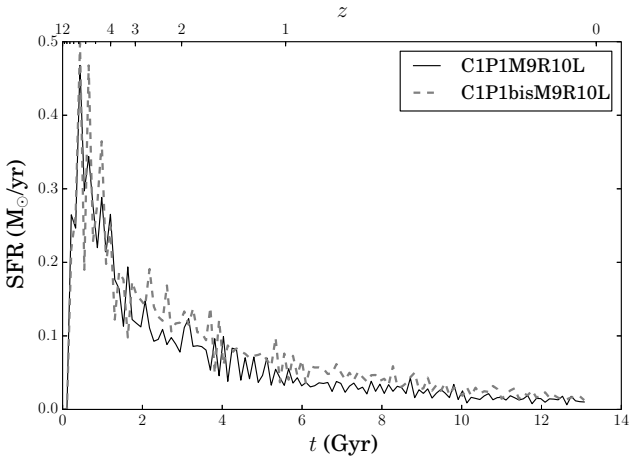


Figure 6.12: *SFR for two simulations using exactly the same model, but with differently sampled initial conditions.*

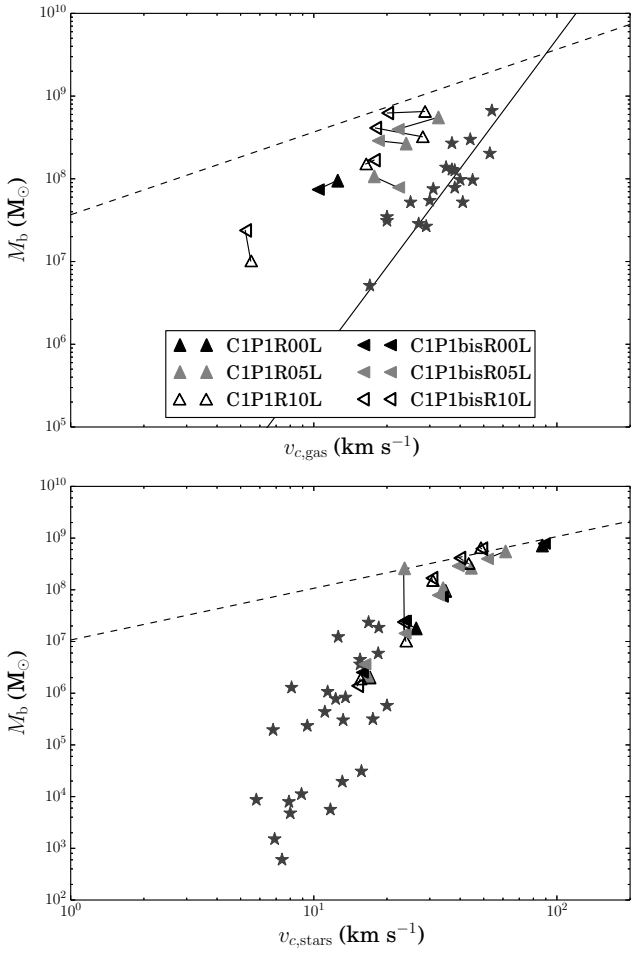


Figure 6.13: *BTFR for the models C1P1 and C1P1bis. The stars and full line are the observations from McGaugh & Wolf (2010) and McGaugh (2012) and the fit from McGaugh (2012). The dashed lines are the least squares fits to all simulations.*

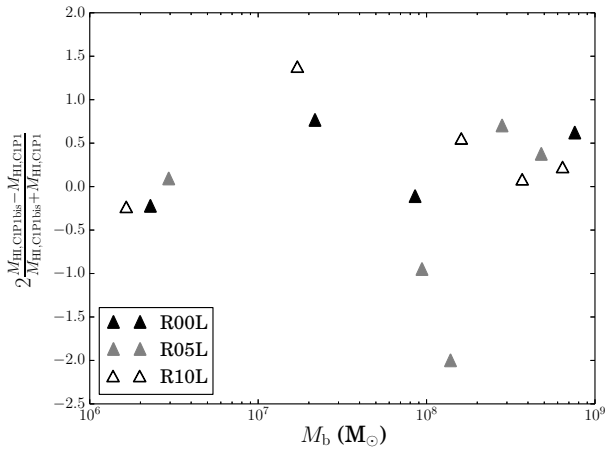


Figure 6.14: *Relative difference in neutral gas mass for the models using differently sampled initial conditions.*

overdensities that will grow out to be the first star forming regions. Since the sampling noise is different when a different set of particles is used to represent the same density distribution, the initial star forming regions for the same model can hence be different if a differently sampled initial condition is used.

It is important to know how large this effect of stochastic differences is, because it means that some of the differences that we would otherwise attribute to the difference in model parameters could well be due to stochastic effects. We already eliminated these effects where possible by using the exact same initial conditions when comparing different models, but we cannot eliminate these effects when comparing two simulations with different resolutions, as for the convergence test below.

Fig. 6.12 shows the SFR for simulations C1P1M9R10L and C1P1bisM9R10L, which use exactly the same model, but differently sampled initial conditions. The stochastic differences lead to small differences in SFR, but the overall form of the curve, with a large initial peak and low level subsequent star formation, is the same.

The simulations of model C1P1 and C1P1bis trace out very similar BTFRs, as can be seen from Fig. 6.13. There is some difference in circular velocities in the top panel for the matching simulations, which is a good indication of the noise on the circular velocity estimates. Both models are in excellent agreement on the bottom panel, except for simulations C1P1M3R05L and C1P1bisM3R05L,

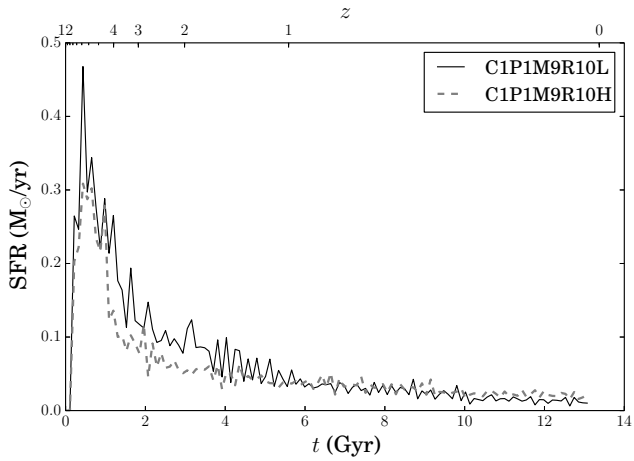


Figure 6.15: *SFR for two simulations using the same model, but different resolutions.*

which have very different baryonic masses. It turns out that for these simulations, model C1P1 has a significant higher neutral gas content than C1P1bis. Both have a stellar circular velocity of $\sim 30\,\mathrm{km\,s^{-1}}$, which is close to the transition from gas-rich to gas-poor. As is illustrated in Fig. 6.14, there is generally a larger difference in neutral gas mass between the two models in this baryonic mass range. This makes sense, as the galaxy will have lost most of its gas in this case, and the remaining gas mass is largely determined by the details of the stellar feedback, which is sensitive to stochastic differences.

6.4.2 Convergence

As was already illustrated in Chapter 2 and Chapter 3, numerical simulations are more accurate when more resolution elements are used. However, we cannot limitlessly increase the resolution of the simulations for two reasons. The first reason is of course the limitations of our computational resources. An increase in resolution leads to a larger memory imprint, and more CPU time needed to carry out the simulation. Since GADGET2 does not have particularly good strong scaling properties (Gonnet, 2014), this inevitably leads to longer simulation times.

A second reason that we cannot increase the resolution below a particle mass of $\sim 10^3\,\mathrm{M_\odot}$ is the limited validity of the sub-grid physics model. As discussed in Chapter 5, we calculate the feedback values for the star particles by assuming every star particle to represent an SSP. However, this approach is only valid if

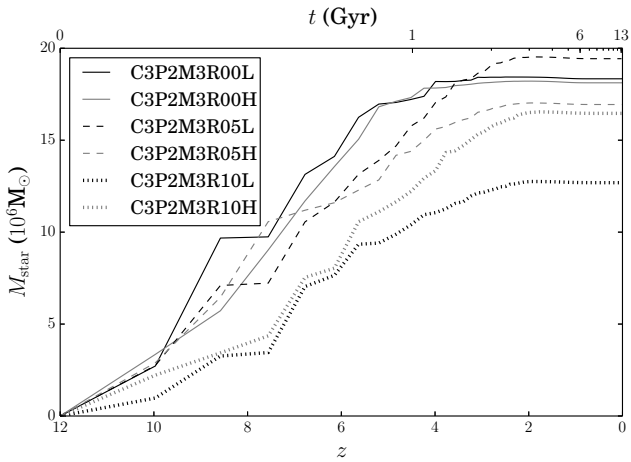


Figure 6.16: *Cumulative SFR for the M3 simulations of model C3P2.*

the SSP is massive enough to represent a statistically significant population with statistical properties. When less massive SSPs are used, the number of massive stars in the SSP becomes too small, so that a real population with that mass will either contain only a few massive stars, or will not contain massive stars at all. It is clear that this will significantly affect the feedback of the SSP. For our least massive halo model, this effectively limits the maximal resolution to 200,000 gas particles.

Even simulations with 200,000 gas particles (and the same number of DM particles) are already computationally expensive, so that we prefer to run lower resolution simulations containing only 50,000 particles of every type. We investigate whether these low resolution simulations have enough resolution to reproduce the same behaviour that is found in the corresponding high resolution simulation by running two models, C1P1 and C3P2, with both low resolution and high resolution.

Fig. 6.17 shows the BTFR for model C1P1 (the BTFR for C3P2 is very similar and is not shown, see Vandenbroucke *et al.* (2016). The BTFR is very similar to that in Fig. 6.13: the top panel shows significant scatter, while the bottom panel is in much better agreement. The low circular velocity simulations have less converged neutral gas masses, as can also be seen from Fig. 6.18. Since this effect is similar to the stochastic effects discussed above, this is likely due to the inevitable stochastic differences between the low and high resolution initial conditions.

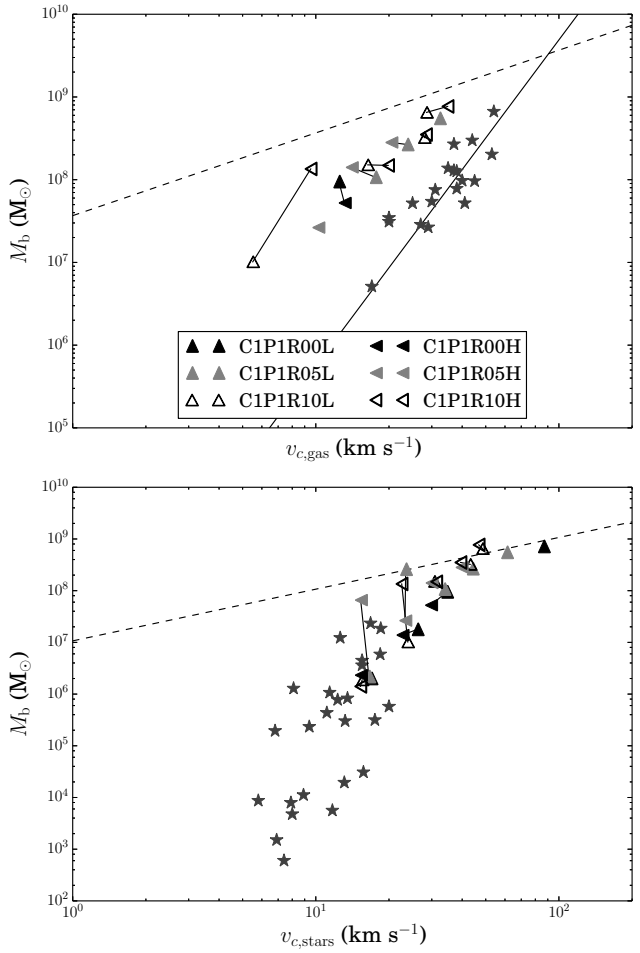


Figure 6.17: *BTFR for the simulations of model C1P1. The stars and the full line represent the observations from McGaugh & Wolf (2010) and McGaugh (2012) and the fit from McGaugh (2012). The dashed lines represent the least squares fits to all simulations.*

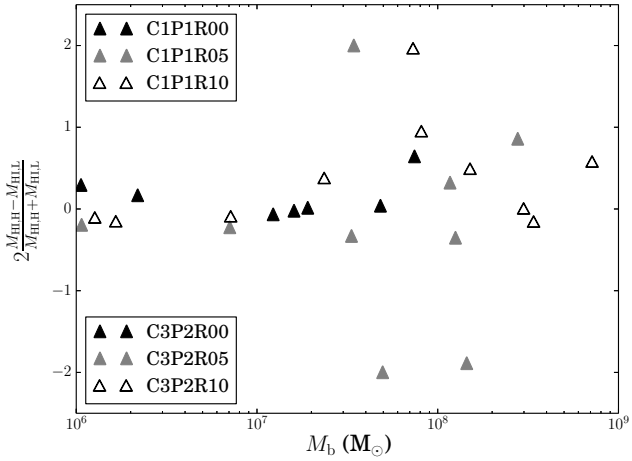


Figure 6.18: *Relative difference in neutral gas mass between the high resolution and low resolution simulations of models C1P1 and C3P2.*

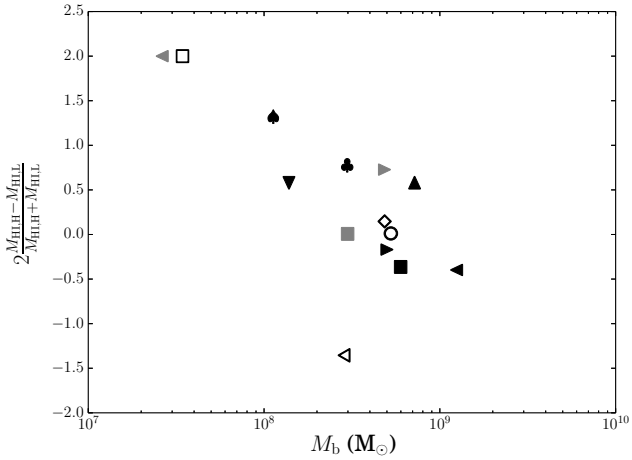


Figure 6.19: *Relative difference in neutral gas mass between the M9R10 simulations with different resolutions, for all models.*

Fig. 6.15 shows the SFR for simulations C1P1M9R10L and C1P1M9R10H. Both curves have the same overall form, although the initial star formation peak is a bit lower for the high resolution simulation. Fig. 6.16 shows the cumulative SFR for the M3 simulations of model C3P2. The build up of stellar mass is similar for the high and low resolution simulations, although there is a difference in final stellar mass for the simulations with initial rotation, which have more spread out initial star formation which is more sensitive to stochastic differences.

Fig. 6.19 shows the relative difference in neutral gas mass between simulations M9R10L and M9R10H for all models (except models C1P1bis and CeP1). It is clear that not all models are equally well converged, with convergence being best for models with a large neutral gas mass. Since we are interested in producing galaxies with large neutral gas masses, we hence conclude that the simulations are enough converged for this purpose.

6.4.3 UV background

We already illustrated the effect of including a UVB on the SFR of the simulated galaxies above. Here, we investigate the effect of the strength and timing of the UVB on the simulations, by comparing models C1P1, C2P1 and C3P1. The resulting BTFR is shown in Fig. 6.20.

The BTFR is clearly resilient against changes in both the timing and the strength of the UVB. The former is easy to explain, as a change in the onset from redshift 10.5 to redshift 7 corresponds to only a small shift in time. The initial star formation peak in the simulations typically occurs around a redshift of 7, so that the gas will only be dispersed and susceptible to UVB heating after the UVB started, both for the early and late UVB.

The apparent independence of the UVB strength is less expected, and is due to the non-linearity of the UVB heating. If the strength of the UVB decreases, this will tilt the ionisation balance in the gas towards more neutral gas. A larger fraction of neutral gas will absorb a larger fraction of the UVB energy, so that the total heating energy absorbed from the UVB stays approximately the same. To obtain a significant change in UVB heating, we have to decrease the strength of the UVB by many orders of magnitude, far below what could be physically explained, as we know that the Universe was completely reionized by redshift 6 (Becker *et al.*, 2001).

6.4.4 Over-cooling

Before we can discuss the effect of different feedback strengths, we have to make sure that the feedback we put into the ISM is effectively converted into kinetic and thermal energy, and not just radiated away, an effect known as over-cooling. We therefore ran one model, CeP1, without switching off cooling for gas particles

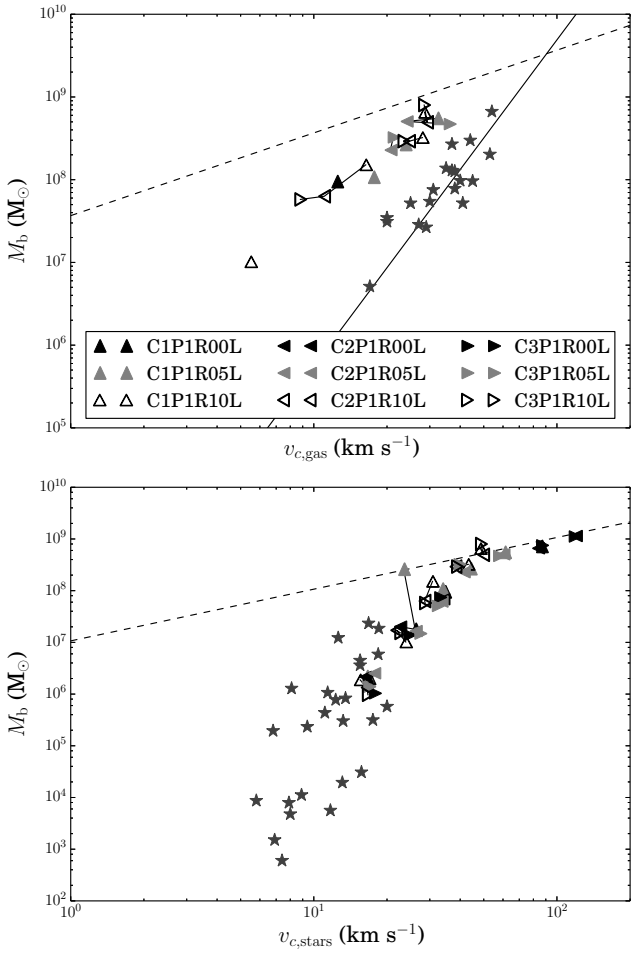


Figure 6.20: *The BTFR for models C1P1, C2P1 and C3P1. The stars and full line represent the observations from McGaugh & Wolf (2010) and McGaugh (2012) and fit from McGaugh (2012). The dashed lines represent the least squares fits to all simulations.*

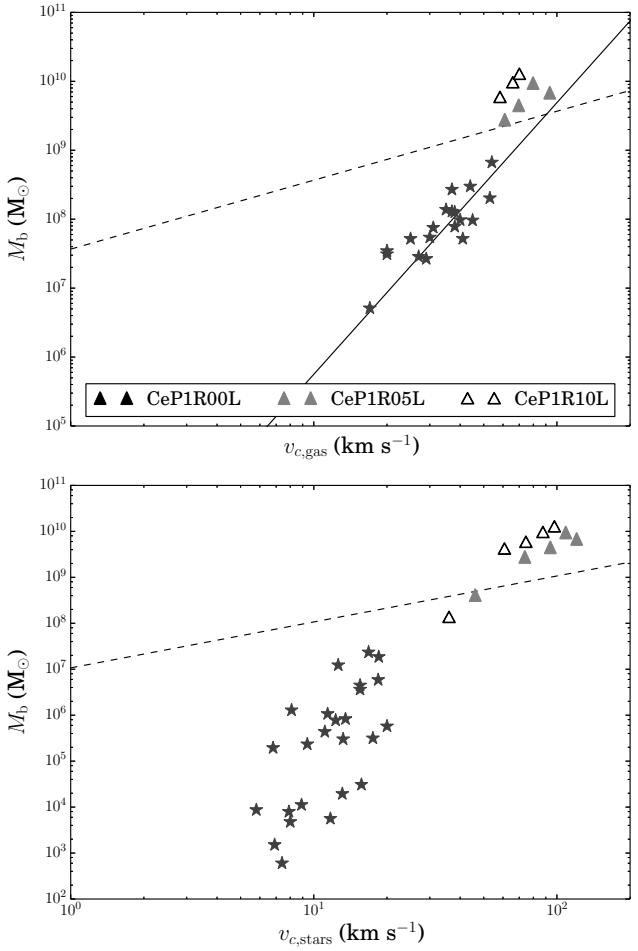


Figure 6.21: *BTFR for the CeP1 simulations. The stars and full line represent the data of McGaugh & Wolf (2010) and McGaugh (2012) and fit from McGaugh (2012). The dashed lines are the least squares fits to all simulations.*

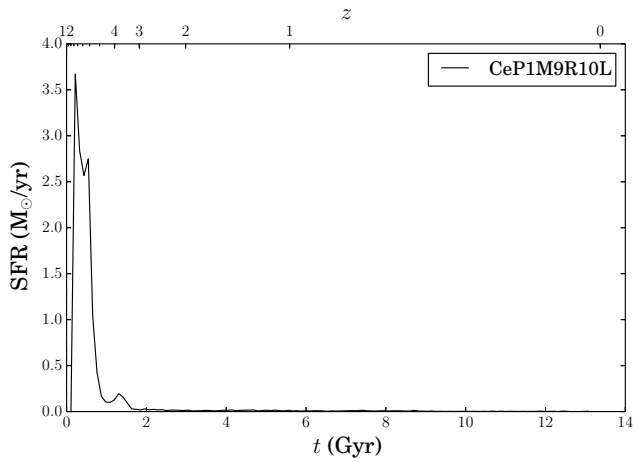


Figure 6.22: *SFR for a simulation without adiabatic cooling period.*

that received SW and SNII feedback. A typical SFR for one of these models is shown in Fig. 6.22. It is immediately clear that the initial star formation peak in this case is significantly stronger than for the other simulations. This indicates that the feedback from the initial star formation is indeed very ineffective.

The resulting BTFR, shown in Fig. 6.21 strongly deviates from the BTFR for the other simulations, forming way too many stars and having a stronger gravitational potential. We hence conclude that over-cooling also causes a strong initial star formation peak, but that it is significantly stronger than what is found for the other simulations. By switching off radiative cooling for gas particles that receive feedback, we make sure the feedback energy effectively ends up in the gas, so that we can play with the feedback strength parameter to quantify the influence of the feedback energy.

6.4.5 Stellar feedback efficiency

As already discusses in Chapter 5, increasing the stellar feedback efficiency also mimics the effect of lowering the star formation efficiency, as the stellar feedback efficiency and star formation efficiency are linked. At the current resolution, we cannot a priori predict values for both of these parameters, so that we have to treat at least one of them as a real model parameter.

Fig. 6.24 shows the BTFR for the models C3P1, C3P2 and C3P3, which have different feedback strengths. Model C3P3 is completely absent from the

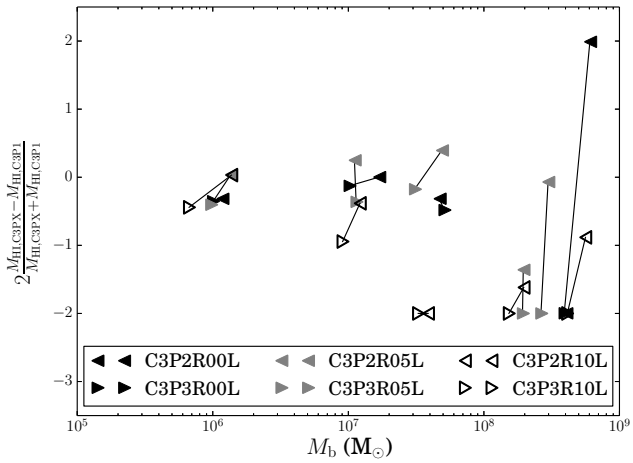


Figure 6.23: *Relative difference in neutral gas mass between the models with feedback strength 1.0 and 2.0, and the reference model with feedback strength 0.7. Corresponding simulations have been linked by a full line.*

top panel, since none of these simulations had enough neutral gas to estimate a circular velocity. From the bottom panel, we see that an increasing feedback strength leads to a lower stellar mass and circular velocity, but in line with the observed relation.

From Fig. 6.23, we see that increasing the feedback strength not necessarily leads to a lower final neutral gas mass, but affects the cutoff halo mass at which a galaxy loses its neutral gas completely, to the extent that the model with the highest feedback strength loses all its gas for all simulations. A higher feedback energy clearly helps to reduce the number of stars formed, but also affects the neutral gas. The entire SFR is suppressed, but the relative difference between the initial star formation peak and the subsequent star formation is not lowered.

A lower feedback strength would lead to more neutral gas, but would also lead to more stellar mass, while the stellar masses are already too high.

6.4.6 Pop III feedback

Until now, we only played with the parameters of the old dwarf galaxy model used by Valcke *et al.* (2008), but then with a UVB. We showed that the parameters setting the timing and strength of the UVB do not influence the simulations significantly, while tuning the stellar feedback strength does not resort in the

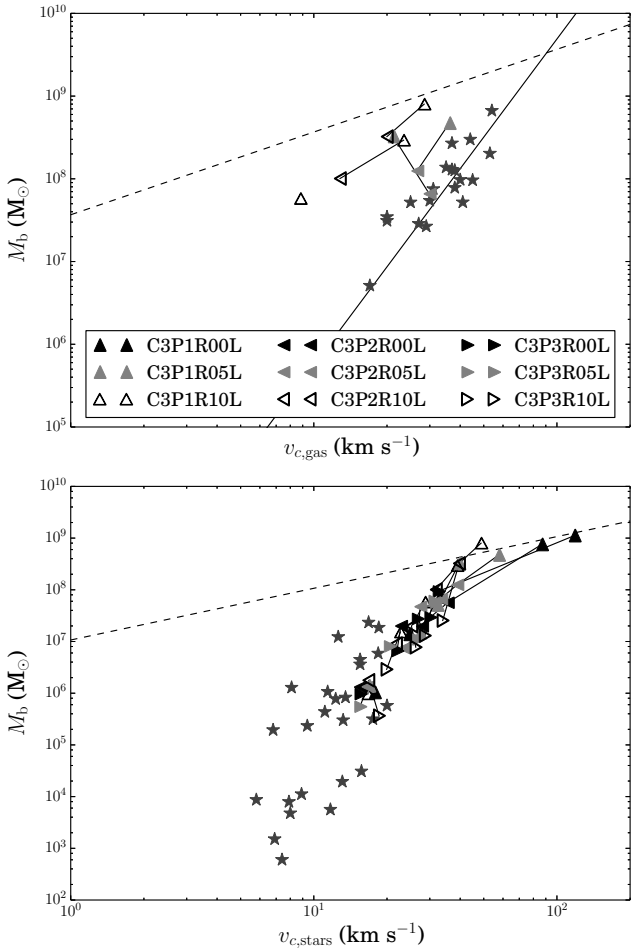


Figure 6.24: *BTFR for the C3P1, C3P2 and C3P3 models. The stars and full line represent the data from McGaugh & Wolf (2010) and McGaugh (2012) and fit from McGaugh (2012). The dashed lines are the least squares fits to all simulations.*

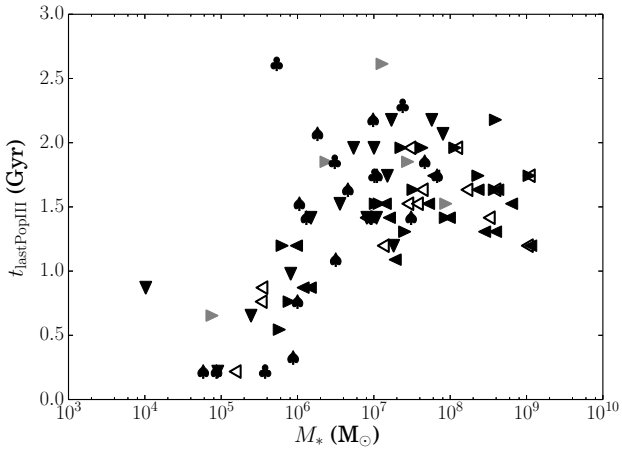


Figure 6.25: *The latest formation time of a Pop III star as a function of the final stellar mass of the simulation. The different symbols correspond to the different models in Table 6.1.*

desired suppression of the initial star formation peak.

What we would actually like to do, is play with the timing of the stellar feedback, since a time dependence of the SW and SNII feedback would potentially allow us to reduce the initial star formation peak, without affecting subsequent star formation. This could be achieved by having a higher feedback strength early in the simulation, which then decreases to the normal level later on.

In Chapter 5, we introduced Pop III stars and showed that they have properties that differ significantly from that of Pop I and Pop II stars. Since they are formed out of primordial, unenriched gas, we expect them to be only formed early on in the simulation, so that Pop III feedback is potentially limited to the beginning of the simulation. This is confirmed by Fig. 6.25, which shows the latest formation time of a Pop III star in our simulations as a function of the stellar mass of the galaxy. Most galaxies form the last Pop III star before 2 Gyr in the simulation, with the bulk of Pop III stars being formed during an even earlier star formation peak. The trend that can be observed in this figure indicates that galaxies which form little stellar mass likely only formed Pop III stars.

Below, we discuss the different Pop III feedback models that were introduced in Chapter 5.

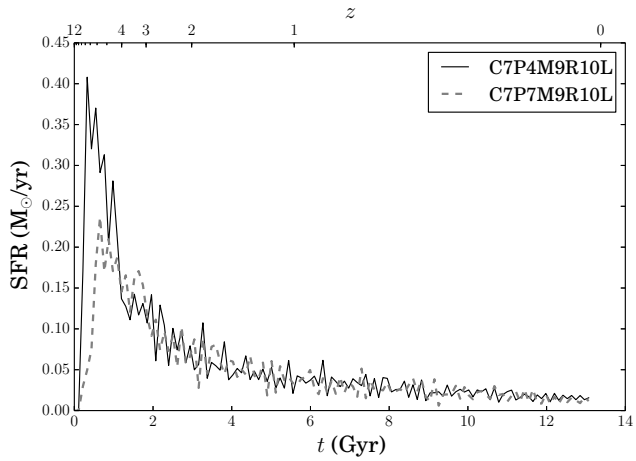


Figure 6.26: *SFR for models C7P4 and C7P7.*

Model 1

The first Pop III feedback model assumes only SN feedback and simply scales up the feedback parameter for SNII feedback for stars with very low metallicities. The different variants of the model depend on different assumed lower mass limits for the Pop III stars, and different forms of the IMF.

Model 1a is characterised by a relatively long Pop III feedback interval. The BTFR for models C7P4, C7P6 and C7P7 is shown in Fig. 6.27, together with that of model C3P1, which has the same parameters but no Pop III feedback. It is clear that model C7P6, which has the highest Pop III feedback energy, leads to gas-poor galaxies that form very little stars. The other two models yield BTFRs that are similar to that of the model without Pop III feedback.

At first sight then, Pop III feedback has very little effect on the simulations if the feedback energy is reasonable. If we look at the SFR for the simulations in Fig. 6.26 however, we see that an increase in Pop III feedback energy effectively reduces the initial star formation peak, without affecting the subsequent star formation, just as we wanted.

Model 1b has a much shorter Pop III feedback interval and comparable feedback energies. In this case, the reduction in initial star formation peak is even more pronounced, as can be seen from Fig. 6.29. The BTFR in Fig. 6.28 now

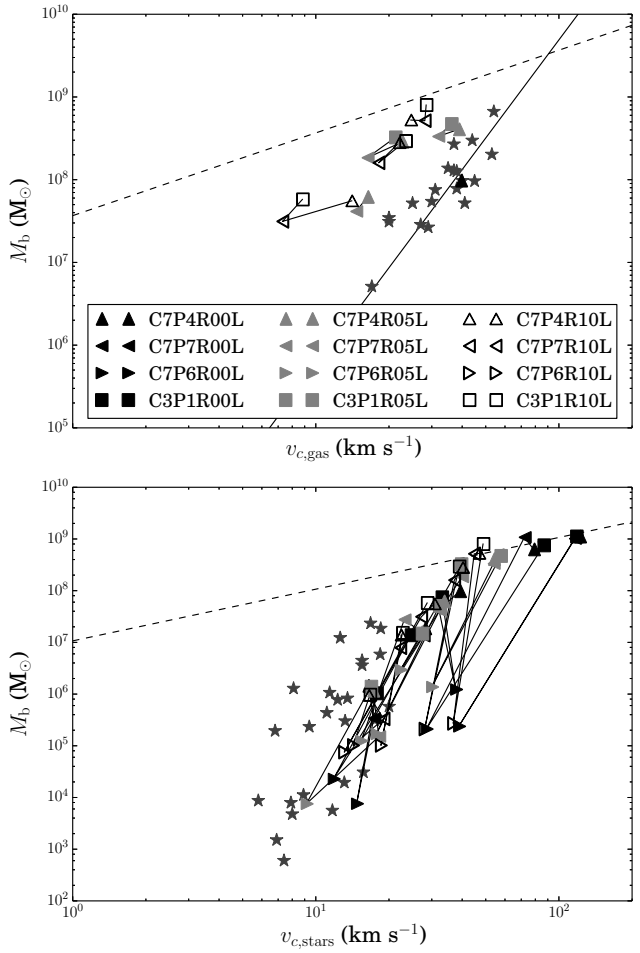


Figure 6.27: *BTFR for models C7P4, C7P6 and C7P7, and reference model C3P1. The stars and full line represent the data from McGaugh & Wolf (2010) and McGaugh (2012) and fit from McGaugh (2012). The dashed lines are the least squares fits to all simulations.*

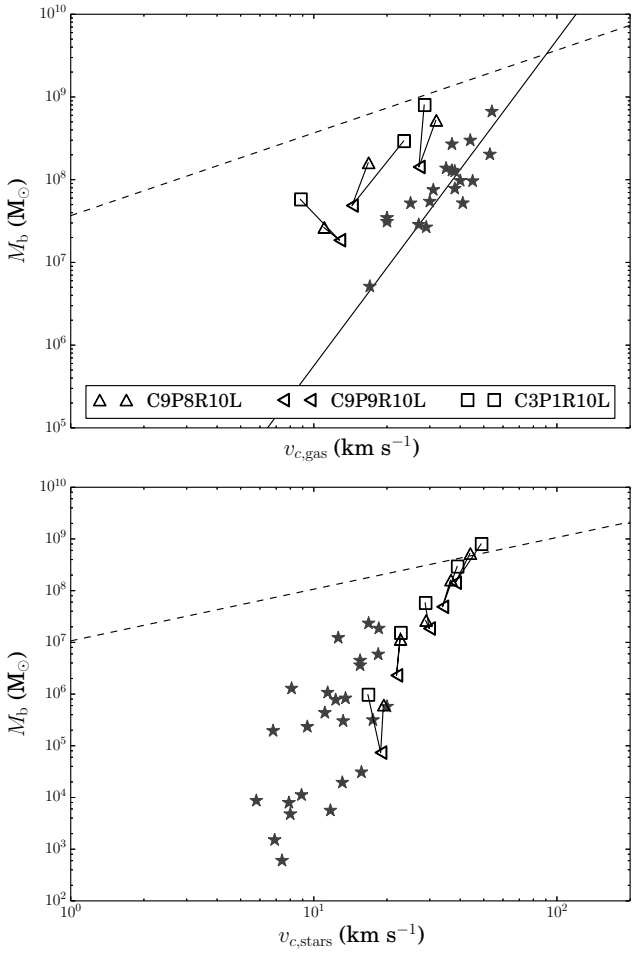


Figure 6.28: *BTFR for models C9P8 and C9P9, and reference model C3P1. The stars and full line represent the data from McGaugh & Wolf (2010) and McGaugh (2012) and fit from McGaugh (2012). The dashed lines are the least squares fits to all simulations.*

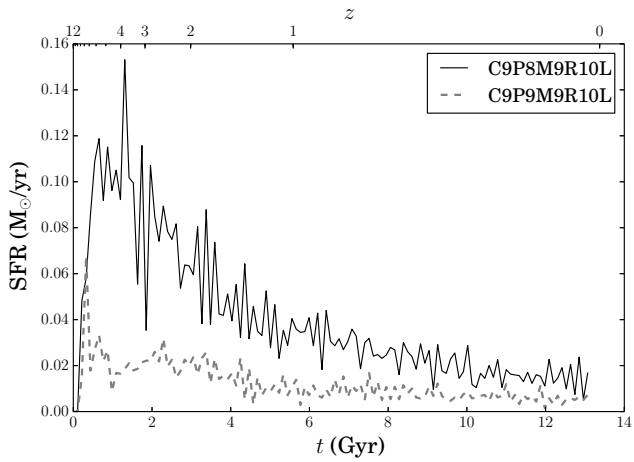


Figure 6.29: *The SFR for models C9P8 and C9P9.*

indicates that the models with Pop III feedback form less stars than the models without.

Model 2

A short Pop III feedback interval with a high feedback energy completely suppresses the initial star formation peak. However, the heavy lower limit on the Pop III masses assumed for model 1b is not very realistic. We can obtain a very similar effect by using a more realistic Pop III mass interval, and include the effect of Pop III SW. As illustrated in Fig. 6.30, this does no longer completely suppress the initial star formation peak, but still leads to a similar SFR. The resulting BTFR (Fig. 6.31) is similar to that of model 1b.

Model 3

The previous Pop III models did not include a realistic chemical enrichment model for Pop III feedback. As mentioned in Chapter 5, our 5D cooling, heating and gas physics model crucially depends on the metal content of the ISM being parametrized by two tracers, [Fe/H] and [Mg/Fe], which only makes sense if there are two feedback mechanisms with different [Fe/H] and [Mg/Fe] outputs. If we would give Pop III stars a different [Fe/H] and [Mg/Fe] output, this simple model would break down. We therefore opted to scale down the SNII chemical

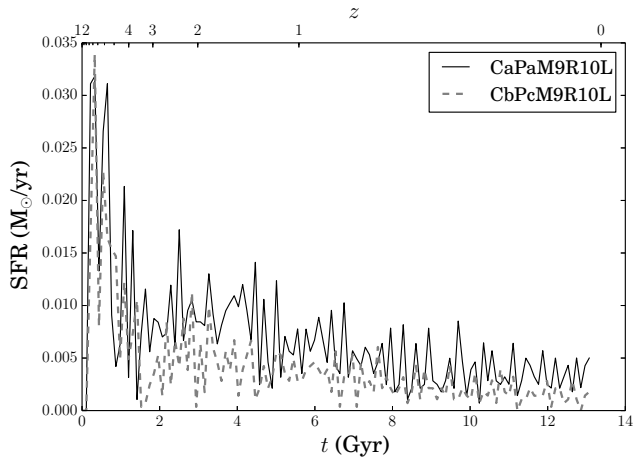


Figure 6.30: *SFR for models CaPa and CbPc.*

enrichment values, so that Pop III return values are more realistic, but are still proportional to the old model.

The resulting BTFR for this most advanced Pop III feedback model is shown in Fig. 6.32. It is very similar to that of the other Pop III models. The top panel even lies very close to the observed BTFR, although this could be a coincidence due to the overall large error flags on the circular velocities derived from the gas.

6.5 Discussion

In this chapter, we ran a large parameter study to try to suppress the initial star formation peak that kills our dwarf galaxy models in the presence of an ionising UV background. We showed that this star formation peak is very resilient against changes in the UVB itself, and that playing around with the feedback strength of regular SW, SNII and SNIa feedback as in the models of Valcke *et al.* (2008) does not help to suppress the star formation peak without at the same time suppressing subsequent star formation.

We conclude that a viable way to suppress the initial star formation peak is the inclusion of a new form of feedback, Pop III feedback, that introduces a time dependence for the feedback strength. Although not yet observed, Pop III stars are expected to have significantly different properties from regular Pop I and Pop II stars, so that our Pop III feedback model has a well defined physical meaning. We showed that the Pop III star particles are indeed only formed early

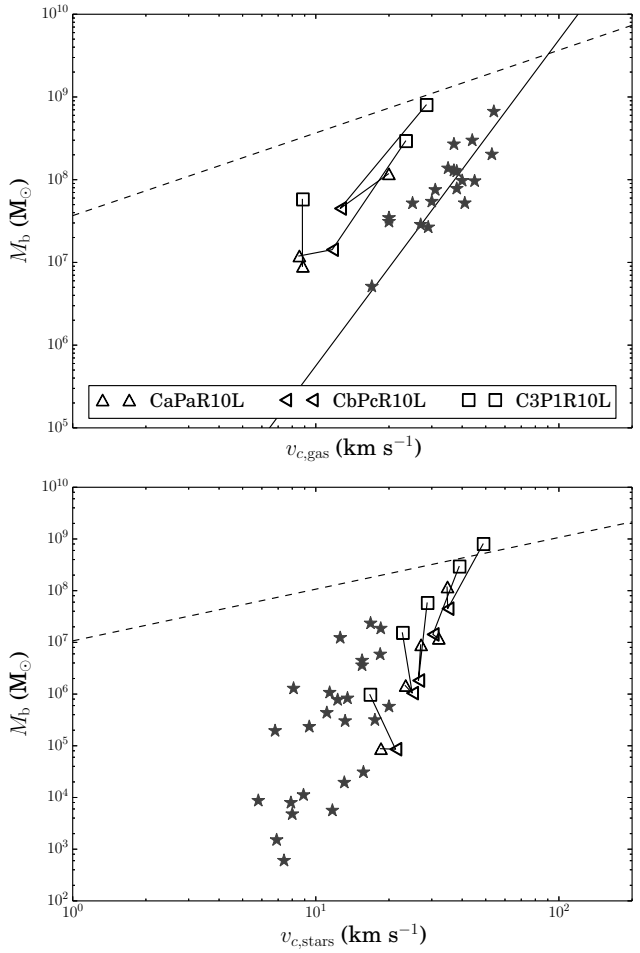


Figure 6.31: *BTFR for models CaPa and CbPc, and reference model C3P1. The stars and full line are the data from McGaugh & Wolf (2010) and McGaugh (2012) and the fit from McGaugh (2012). The dashed lines are the least squares fits to all simulations.*

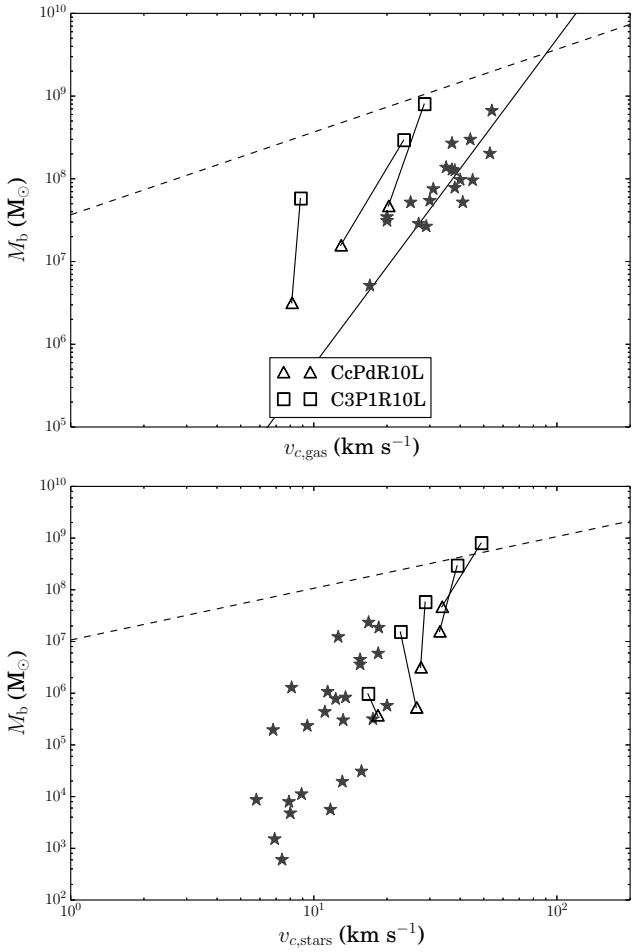


Figure 6.32: *BTFR for model CcPd and reference model C3P1. The stars and full line represent the data from McGaugh & Wolf (2010) and McGaugh (2012) and the fit from McGaugh (2012). The dashed lines are the least squares fits to all simulations.*

in the simulation, so that they effectively give rise to a different feedback strength during the initial star formation peak, while the feedback during subsequent star formation is unaffected. We showed that this model helps reducing the initial star formation peak, as desired.

We compared our simulations with the observed BTFR of McGaugh & Wolf (2010) and McGaugh (2012) by constructing mock observations of the simulated galaxies, that resemble the real observations as closely as possible. We compared the mock observational quantities with the true values from the simulations, and conclude that some quantities, like the stellar mass, are recovered reasonably well by using observational tracers. Other quantities, like the circular velocity, are harder to estimate observationally, especially if neutral gas is used as a tracer. As a result, we found a better agreement between the simulations and the BTFR of McGaugh & Wolf (2010), which uses the stellar velocity dispersion as a circular velocity tracer.

Even with Pop III feedback, our models form too many stars and have too little neutral gas, which is probably caused by neglecting the effect of cosmic gas accretion in the simulations of isolated galaxies. By using a merger tree to simulate the effect of a hierarchical growth of the halo, these problems are overcome (Verbeke *et al.*, 2015).

7

Future research

With the model described in Chapter 5 and tested in Chapter 6, we can simulate dwarf galaxies that have the same properties as observed late-type dwarf irregular galaxies (Verbeke *et al.*, 2015). Although our models still carry a lot of assumptions, and have a number of free parameters that need to be fixed, we are clearly starting to get a grasp of the processes that regulate the formation and evolution of these galaxies. The model is hence mature enough to tackle more complex questions, like the transformation of late-type into early-type dwarf galaxies by environmental effects, mentioned in Chapter 1.

Of course, to tackle this question, we also need a reliable hydrodynamical solver. In Chapter 3, we introduced two Lagrangian methods that are promising candidates to replace the SPH scheme on which our current model is based: the moving mesh scheme and the mesh-free method. During the work that led to this thesis, significant progress has been made in developing codes that implement these methods, both by the development of our own moving mesh code Shadowfax (see Chapter 4), and by collaboration on the development of the SPH/mesh-free code swift. Developing a code for astrophysical simulations however has become more work than fits in a single PhD; some effort is still required to port the sub-grid model of Chapter 5 to these codes.

Apart from these obvious subjects, the work in this thesis also opens up more exotic possibilities. With the development of arepo and gizmo, a growing interest has risen in the astrophysical community to include magnetic fields in a more broad range of simulations (Pakmor *et al.*, 2011; Hopkins & Raives, 2016), like cosmological simulations (Marinacci *et al.*, 2015; Marinacci & Vogelsberger, 2016) and simulations of galaxy formation and evolution (Pakmor *et al.*, 2014). These magnetic fields are believed to become important on smaller scales, regulating small-scale turbulence in fragmenting clouds and hence affecting star formation (Seifried & Walch, 2015). On the other hand, these fields are less constrained by observations, so that it is currently unclear whether they play an important role. If we would expand Shadowfax with an MHD implementation, this would open up the exciting possibility to simulate dwarf galaxies with

magnetic fields.

Another recent development in the astrophysical community is the coupling of real time radiative transfer and hydrodynamics, whereby the transport of ionising photons in the fluid is resolved by integrating the local radiation field (Skinner & Ostriker, 2013; Kolb *et al.*, 2013; Rosdahl *et al.*, 2013; Ramsey & Dullemond, 2015; Roth & Kasen, 2015). This problem is incredibly complex and requires an enormous amount of computing power, but solving it is crucial for a thorough understanding of reionization and the effect of the UV background on galaxy formation. With the growing use of hybrid codes that use both CPU and GPU power in parallel, radiation hydrodynamics is starting to become a reality. Grid-based codes like SHADOWFAX are more easy to couple to a radiative transfer scheme, so there lies another possible future challenge.

On a science level, a number of interesting questions about dwarf galaxies are waiting to be answered, like the too big to fail and cusp to core problems mentioned in Chapter 1. For a thorough solution of these problems, we need to use full-fledged cosmological zoom simulations, to accurately resolve the effect of cosmic gas accretion on the galaxies. The too big to fail problem furthermore requires us to resolve dwarf galaxies that are satellites of Local Group analogues, which can only be done in zoom simulations.

In this work, we have shown that it is not so straightforward to compare simulations and observations, and that some tensions between both can be alleviated by probing the simulations using mock observables. This raises the question if problems like the cusp to core problem and the too big to fail problem could be caused by similar issues. Many of these problems originate from the comparison of dark matter only simulations with observations. These observations necessarily trace the dark matter using the movement of stars and gas, which requires a model fit to estimate dark matter halo properties that can then be compared with the simulations (Garrison-Kimmel *et al.*, 2014). This approach likely suffers from serious systematic uncertainties that affect the results. We would advocate the use of simulations that include baryons, and that use the baryons from the simulation to construct mock observations that are directly compared with the real observations. This way, all model uncertainties are at least limited to the model. However, the current generation of large scale simulations does not have the resolution necessary for such an approach.

Using our model in a cosmological simulation would suffice to do this, but requires simulations that are too computationally demanding for the current generation of high performance systems. Progress in these simulations will have to be accompanied by progress in the algorithms that are used, whereby novel algorithms should make more efficient use of highly parallel infrastructures. Codes like SWIFT provide a step in the right direction.

Exciting times lie ahead!

Bibliography

Agertz, O. *et al.*, 2007. *Mon. Not. R. Astron. Soc.*, 380, 963.

Alvarez, M. A., Busha, M., Abel, T. & Wechsler, R. H., 2009. *Astrophys. J. Lett.*, 703, L167.

Barnes, J. & Hut, P., 1986. *Nature*, 324, 446.

Becker, R. H. *et al.*, 2001. *Astron. J.*, 122, 2850.

Bédorf, J., Gaburov, E., Fujii, M. S., Nitadori, K., Ishiyama, T. & Portegies Zwart, S., In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14 (IEEE Press, 2014), 54–65.

Bell, E. F. & de Jong, R. S., 2001. *Astrophys. J.*, 550, 212.

Benítez-Llambay, A. *et al.*, 2015. *Mon. Not. R. Astron. Soc.*, 450, 4207.

Berger, M. J. & Colella, P., 1989. *J. Comput. Phys.*, 82, 64.

Bertelli, G., Girardi, L., Marigo, P. & Nasi, E., 2008. *Astron. Astrophys.*, 484, 815.

Bertelli, G., Nasi, E., Girardi, L. & Marigo, P., 2009. *Astron. Astrophys.*, 508, 355.

Binney, J. & Tremaine, S., *Galactic Dynamics: Second Edition* (Princeton University Press, 2008).

Blanes, S., Casas, F., Farrés, A., Laskar, J., Makazaga, J. & Murua, A., 2013. *Appl. Numer. Math.*, 68, 58.

Boekholt, T. & Portegies Zwart, S., 2015. *Comp. Astrophys. Cosmol.*, 2, 2.

Bonaparte, I., Matteucci, F., Recchi, S., Spitoni, E., Pipino, A. & Grieco, V., 2013. *Mon. Not. R. Astron. Soc.*, 435, 2460.

Bond, J. R. & Efstathiou, G., 1984. *Astrophys. J. Lett.*, 285, L45.

Boylan-Kolchin, M., Springel, V., White, S. D. M., Jenkins, A. & Lemson, G., 2009. *Mon. Not. R. Astron. Soc.*, 398, 1150.

Brent, R., *Algorithms for Minimization Without Derivatives* (Dover Publications, 1973).

Cha, S.-H., Inutsuka, S.-I. & Nayakshin, S., 2010. *Mon. Not. R. Astron. Soc.*, 403, 1165.

Chabrier, G., 2003. *Publ. Astron. Soc. Pac.*, 115, 763.

Chan, T. K. *et al.*, 2015. *Mon. Not. R. Astron. Soc.*, 454, 2981.

Chandrasekhar, S., *Hydrodynamic and hydromagnetic stability* (Clarendon, 1961).

Cloet-Osselaer, A., De Rijcke, S., Schroyen, J. & Dury, V., 2012. *Mon. Not. R. Astron. Soc.*, 423, 735.

Cloet-Osselaer, A., De Rijcke, S., Vandenbroucke, B., Schroyen, J., Koleva, M. & Verbeke, R., 2014. *Mon. Not. R. Astron. Soc.*, 442, 2909.

Courant, R., Friedrichs, K. & Lewy, H., 1928. *Math. Ann.*, 100, 32.

Dalla Vecchia, C. & Schaye, J., 2008. *Mon. Not. R. Astron. Soc.*, 387, 1431.

Davis, M., Efstathiou, G., Frenk, C. S. & White, S. D. M., 1985. *Astrophys. J.*, 292, 371.

de Berg, M., Cheong, O., van Kreveld, M. & Overmars, M., *Computational Geometry Algorithms and Applications*. 3rd edition (Springer-Verlag, 2008).

de Blok, W. J. G., 2010. *Adv. Astron.*, 2010, 789293.

De Rijcke, S., Michielsen, D., Dejonghe, H., Zeilinger, W. W. & Hau, G. K. T., 2005. *Astron. Astrophys.*, 438, 491.

De Rijcke, S., Van Hese, E. & Buyle, P., 2010. *Astrophys. J. Lett.*, 724, L171.

De Rijcke, S. *et al.*, 2013. *Mon. Not. R. Astron. Soc.*, 433, 3005.

Deason, A., Wetzel, A. & Garrison-Kimmel, S., 2014. *Astrophys. J.*, 794, 115.

Dehnen, W. & Read, J. I., 2011. *Eur. Phys. J. Plus*, 126, 55.

Delaunay, B., 1934. *Bulletin de l'Académie des Sciences de l'URSS, Classe des sciences mathématiques et naturelles*, 6, 793.

Dere, K. P., Landi, E., Young, P. R., Del Zanna, G., Landini, M. & Mason, H. E., 2009. *Astron. Astrophys.*, 498, 915.

Diehl, S., Rockefeller, G., Fryer, C. L., Riethmiller, D. & Statler, T. S., 2015. *arXiv.* `1211.0525`.

Dirichlet, G. L., 1850. *J. Reine Angew. Math.*, 40, 209.

Draine, B. T., *Physics of the Interstellar and Intergalactic Medium* (Princeton University Press, 2011).

Dressler, A., 1980. *Astrophys. J.*, 236, 351.

Duffell, P. C. & MacFadyen, A. I., 2011. *Astrophys. J. Suppl. Ser.*, 197, 15.

Dunn, J. M., 2010. *Mon. Not. R. Astron. Soc.*, 408, 392.

Dutton, A. A. *et al.*, 2015. *arXiv.* `1512.00453`.

Dyer, C. C. & Ip, P. S. S., 1993. *Astrophys. J.*, 409, 60.

Edelsbrunner, H. & Shah, N., 1996. *Algorithmica*, 15, 223.

Emilio, M., Kuhn, J. R., Bush, R. I. & Scholl, I. F., 2012. *The Astrophysical Journal*, 750, 135.

Evrard, A. E., 1988. *Mon. Not. R. Astron. Soc.*, 235, 911.

Faucher-Giguère, C.-A., Lidz, A., Zaldarriaga, M. & Hernquist, L., 2009. *Astrophys. J.*, 703, 1416.

Flanders, H., 1973. *Am. Math. Mon.*, 80, 615.

Fortune, S., In *Proceedings of the Second Annual Symposium on Computational Geometry*, SCG '86 (ACM, 1986), 313–322.

Garrison-Kimmel, S., Boylan-Kolchin, M., Bullock, J. S. & Kirby, E. N., 2014. *Mon. Not. R. Astron. Soc.*, 444, 222.

Geha, M., Guhathakurta, P. & van der Marel, R. P., 2003. *Astron. J.*, 126, 1794.

Geha, M. *et al.*, 2013. *Astrophys. J.*, 771, 29.

Godunov, S., 1959. *Mat. Sb.*, 47, 271.

Gonnet, P., 2014. *arXiv.* `1404.2303`.

Governato, F. *et al.*, 2010. *Nature*, 463, 203.

Graham, A. W. & Guzmán, R., 2003. *Astron. J.*, 125, 2936.

Grebel, E. K., Gallagher, J. S., III & Harbeck, D., 2003. *Astron. J.*, 125, 1926.

Guibas, L., Knuth, D. & Sharir, M., 1992. *Algorithmica*, 7, 381.

Heger, A. & Woosley, S. E., 2010. *Astrophys. J.*, 724, 341.

Heggie, D. & Hut, P., *The Gravitational Million-Body Problem: A Multidisciplinary Approach to Star Cluster Dynamics* (Cambridge University Press, 2003).

Hendrix, T. & Keppens, R., 2014. *Astron. Astrophys.*, 562, A114.

Hopkins, P. F., 2013. *Mon. Not. R. Astron. Soc.*, 428, 2840.

Hopkins, P. F., 2015. *Mon. Not. R. Astron. Soc.*, 450, 53.

Hopkins, P. F. & Raives, M. J., 2016. *Mon. Not. R. Astron. Soc.*, 455, 51.

Hormann, K. & Agathos, A., 2001. *Comput. Geom.*, 20, 131.

Huang, K., *Statistical mechanics* (Wiley, 1987).

Hugoniot, H., *Journal* (Ecole polytechnique, 1887).

Hunter, D. A. & Elmegreen, B. G., 2006. *Astrophys. J. Suppl. Ser.*, 162, 49.

IEEE, 2008. *IEEE Standard for Floating-Point Arithmetic*. Technical report, IEEE Std 754-2008.

Isaacson, E. & Keller, H., *Analysis of Numerical Methods* (Dover Publications, 1994).

Ivanova, N. *et al.*, 2013. *Astron. Astrophys. Rev.*, 21, 59.

Jeans, J. H., 1902. *Philos. T. Roy. Soc*, 199, 1.

Jin, G. & Mellor-Crummey, J., 2005. *ACM Trans. Math. Softw.*, 31, 120.

Keppens, R., Meliani, Z., van Marle, A., Delmont, P., Vlasis, A. & van der Holst, B., 2012. *J. Comput. Phys.*, 231, 718.

Kirby, E. N., Cohen, J. G., Guhathakurta, P., Cheng, L., Bullock, J. S. & Gallazzi, A., 2013. *Astrophys. J.*, 779, 102.

Kolb, S. M., Stute, M., Kley, W. & Mignone, A., 2013. *Astron. Astrophys.*, 559, A80.

Kreckel, K., Joung, M. R. & Cen, R., 2011. *Astrophys. J.*, 735, 132.

Kroupa, P., 2002. *Science*, 295, 82.

Krumholz, M. R., 2014. *Phys. Rep.*, 539, 49.

Lada, C. J., 2006. *Astrophys. J. Lett.*, 640, L63.

Leaman, R., Erroz-Ferrer, S., Cisternas, M. & Knapen, J. H., 2015. *Mon. Not. R. Astron. Soc.*, 450, 2473.

Lecoanet, D. *et al.*, 2015. *arXiv*. `1509.03630`.

Lianou, S., Grebel, E. K. & Koch, A., In *Stellar Populations – Planning for the Next Decade*, volume 5 of *Proceedings of the International Astronomical Union* (2010), 115–118.

Liska, R. & Wendroff, B., 2003. *SIAM J. Sci. Comput.*, 25, 995.

Lisker, T., Weinmann, S. M., Janz, J. & Meyer, H. T., 2013. *Mon. Not. R. Astron. Soc.*, 432, 1162.

Lloyd, S., 1982. *EEE Trans. Inf. Theory*, 28, 129.

Magorrian, J. & Ballantyne, D., 2001. *Mon. Not. R. Astron. Soc.*, 322, 702.

Makino, J., 2002. *J. Comput. Appl. Math.*, 149, 131.

Marcolini, A., D'Ercole, A., Brighenti, F. & Recchi, S., 2006. *Mon. Not. R. Astron. Soc.*, 371, 643.

Marigo, P., Girardi, L., Chiosi, C. & Wood, P. R., 2001. *Astron. Astrophys.*, 371, 152.

Marinacci, F. & Vogelsberger, M., 2016. *Mon. Not. R. Astron. Soc.*, 456, L69.

Marinacci, F., Vogelsberger, M., Mocz, P. & Pakmor, R., 2015. *Mon. Not. R. Astron. Soc.*, 453, 3999.

Mavriplis, D. J., 1997. *Annu. Rev. Fluid Mech.*, 29, 473.

Mayer, L., 2010. *Adv. Astron.*, 2010, 278434.

Mazzali, P. A., Röpke, F. K., Benetti, S. & Hillebrandt, W., 2007. *Science*, 315, 825.

McConnachie, A. W., 2012. *Astron. J.*, 144, 4.

McGaugh, S. S., 2012. *Astron. J.*, 143, 40.

McGaugh, S. S. & Wolf, J., 2010. *Astrophys. J.*, 722, 248.

McMillan, P. J., 2011. *Mon. Not. R. Astron. Soc.*, 414, 2446.

McQuinn, K. B. W. *et al.*, 2013. *Astron. J.*, 146, 145.

Monaghan, J. J. & Lattanzio, J. C., 1985. *Astron. Astrophys.*, 149, 135.

Navarro, J. F., Frenk, C. S. & White, S. D. M., 1997. *Astrophys. J.*, 490, 493.

Noh, W. F., 1987. *J. Comput. Phys.*, 72, 78.

Nomoto, K., Kobayashi, C. & Tominaga, N., 2013. *Annu. Rev. Astron. Astrophys.*, 51, 457.

Oñorbe, J. *et al.*, 2015. *Mon. Not. R. Astron. Soc.*, 454, 2092.

Pakmor, R., Bauer, A. & Springel, V., 2011. *Mon. Not. R. Astron. Soc.*, 418, 1392.

Pakmor, R., Marinacci, F. & Springel, V., 2014. *Astrophys. J. Lett.*, 783, L20.

Planck Collaboration, 2014. *Astron. Astrophys.*, 571, A16.

Plummer, H. C., 1911. *Mon. Not. R. Astron. Soc.*, 71, 460.

Portegies Zwart, S. F., Belleman, R. G. & Geldof, P. M., 2007. *New Astron.*, 12, 641.

Price, D. J., 2008. *J. Comput. Phys.*, 227, 10040.

Price, D. J., 2012. *J. Comput. Phys.*, 231, 759.

Price, D. J. & Monaghan, J. J., 2007. *Mon. Not. R. Astron. Soc.*, 374, 1347.

Ramsey, J. P. & Dullemond, C. P., 2015. *Astron. Astrophys.*, 574, A81.

Rankine, W. M., 1870. *Philos. T. Roy. Soc*, 277.

Read, J. I. & Hayfield, T., 2012. *Mon. Not. R. Astron. Soc.*, 422, 3037.

Read, J. I., Hayfield, T. & Agertz, O., 2010. *Mon. Not. R. Astron. Soc.*, 405, 1513.

Recchi, S., 2014. *Adv. Astron.*, 2014, 750754.

Reem, D., In *Proceedings of the Twenty-seventh Annual Symposium on Computational Geometry*, SoCG '11 (ACM, New York, NY, USA, 2011), 254–263.

Rhode, K. L. *et al.*, 2013. *Astron. J.*, 145, 149.

Rodrigues, D. C., de Oliveira, P. L., Fabris, J. C. & Gentile, G., 2014. *Mon. Not. R. Astron. Soc.*, 445, 3823.

Rosdahl, J., Blaizot, J., Aubert, D., Stranex, T. & Teyssier, R., 2013. *Mon. Not. R. Astron. Soc.*, 436, 2188.

Roth, N. & Kasen, D., 2015. *Astrophys. J. Suppl. Ser.*, 217, 9.

Ryś, A., van de Ven, G. & Falcón-Barroso, J., 2014. *Mon. Not. R. Astron. Soc.*, 439, 284.

Saitoh, T. R. & Makino, J., 2009. *Astrophys. J. Lett.*, 697, L99.

Sales, L. V., Helmi, A. & Battaglia, G., 2010. *Adv. Astron.*, 2010, 194345.

Sales, L. V. *et al.*, 2016. *arXiv*. `1602.02155`.

Salpeter, E. E., 1955. *Astrophys. J.*, 121, 161.

Sawala, T. *et al.*, 2015. *Mon. Not. R. Astron. Soc.*, 448, 2941.

Scaramella, R. *et al.*, In *Statistical Challenges in 21st Century Cosmology*, volume 10 of *Proceedings of the International Astronomical Union* (2014), 375–378.

Schaller, M. *et al.*, 2015. *Mon. Not. R. Astron. Soc.*, 454, 2277.

Schaye, J. *et al.*, 2015. *Mon. Not. R. Astron. Soc.*, 446, 521.

Schmidt, M., 1959. *Astrophys. J.*, 129, 243.

Schroyen, J., De Rijcke, S., Koleva, M., Cloet-Osselaer, A. & Vandenbroucke, B., 2013. *Mon. Not. R. Astron. Soc.*, 434, 888.

Schroyen, J., De Rijcke, S., Valcke, S., Cloet-Osselaer, A. & Dejonghe, H., 2011. *Mon. Not. R. Astron. Soc.*, 416, 601.

Sedov, L., *Similitude et dimensions en mecanique* (Edition Mir, 1977).

Seifried, D. & Walch, S., 2015. *Mon. Not. R. Astron. Soc.*, 452, 2410.

Shen, S., Madau, P., Conroy, C., Governato, F. & Mayer, L., 2014. *Astrophys. J.*, 792, 99.

Shewchuk, J. R., 1997. *Discrete Comput. Geom.*, 18, 305.

Siebert, C. & Wolf, F., 2010. *A Scalable Parallel Sorting Algorithm Using Exact Splitting*. Technical report, German Research School for Simulation Sciences GmbH.

Silk, J., 1968. *Astrophys. J.*, 151, 459.

Skinner, M. A. & Ostriker, E. C., 2013. *Astrophys. J. Suppl. Ser.*, 206, 21.

Smith, G. D., *Numerical solution of partial differential equations: finite difference methods*. 3rd edition (Oxford University Press, 1985).

Smith, N., Hinkle, K. H. & Ryde, N., 2009. *Astron. J.*, 137, 3558.

Smith, R. *et al.*, 2015. *Mon. Not. R. Astron. Soc.*, 454, 2502.

Sobral, D. *et al.*, 2015. *Astrophys. J.*, 808, 139.

Spergel, D. N. *et al.*, 2007. *Astrophys. J. Suppl. Ser.*, 170, 377.

Springel, V., 2005. *Mon. Not. R. Astron. Soc.*, 364, 1105.

Springel, V., 2010. *Mon. Not. R. Astron. Soc.*, 401, 791.

Springel, V. & Hernquist, L., 2002. *Mon. Not. R. Astron. Soc.*, 333, 649.

Springel, V. *et al.*, 2005. *Nature*, 435, 629.

Starkenburg, T. K., Helmi, A. & Sales, L. V., 2016. *Astron. Astrophys.*, 587, A24.

Steinberg, E., Yalinewich, A., Sari, R. & Duffell, P., 2015. *Astrophys. J. Suppl. Ser.*, 216, 14.

Strolger, L.-G., Dahlen, T. & Riess, A. G., 2010. *Astrophys. J.*, 713, 32.

Strömgren, B., 1939. *Astrophys. J.*, 89, 526.

Sundar, H., Sampath, R. S. & Biros, G., 2008. *SIAM J. Sci. Comput.*, 30, 2675.

Susa, H., Hasegawa, K. & Tominaga, N., 2014. *Astrophys. J.*, 792, 32.

Sweet, S. M. *et al.*, 2016. *Mon. Not. R. Astron. Soc.*, 455, 2508.

Teyssier, R., 2002. *Astron. Astrophys.*, 385, 337.

Thacker, R. J. & Couchman, H. M. P., 2000. *Astrophys. J.*, 545, 728.

Thijssen, J., *Computational Physics* (Cambridge University Press, 1999).

Thornton, K., Gaudlitz, M., Janka, H.-T. & Steinmetz, M., 1998. *Astrophys. J.*, 500, 95.

Tollerud, E. J., Geha, M. C., Grcevich, J., Putman, M. E. & Stern, D., 2015. *Astrophys. J. Lett.*, 798, L21.

Tolstoy, E., Hill, V. & Tosi, M., 2009. *Annu. Rev. Astron. Astrophys.*, 47, 371.

Toro, E. F., *Riemann Solvers and Numerical Methods for Fluid Dynamics*. 3rd edition (Springer-Verlag, 2009).

Trujillo-Gomez, S., Klypin, A., Colín, P., Ceverino, D., Arraki, K. S. & Primack, J., 2015. *Mon. Not. R. Astron. Soc.*, 446, 1140.

Valcke, S., De Rijcke, S. & Dejonghe, H., 2008. *Mon. Not. R. Astron. Soc.*, 389, 1111.

Valcke, S., De Rijcke, S., Rödiger, E. & Dejonghe, H., 2010. *Mon. Not. R. Astron. Soc.*, 408, 71.

van Leer, B., 1979. *J. Comput. Phys.*, 32, 101.

van Zee, L., 2000. *Astron. J.*, 119, 2757.

van Zee, L., Barton, E. J. & Skillman, E. D., 2004. *Astron. J.*, 128, 2797.

Vandenbroucke, B. & De Rijcke, S., 2016. *Astron. Comput.*.

Vandenbroucke, B., De Rijcke, S., Schroyen, J. & Jachowicz, N., 2013. *Astrophys. J.*, 771, 36.

Vandenbroucke, B., Verbeke, R. & De Rijcke, S., 2016. *Mon. Not. R. Astron. Soc.*, 458, 912.

Vazdekis, A., Ricciardelli, E., Cenarro, A. J., Rivero-González, J. G., Díaz-García, L. A. & Falcón-Barroso, J., 2012. *Mon. Not. R. Astron. Soc.*, 424, 157.

Verbeke, R., De Rijcke, S., Koleva, M., Cloet-Osselaer, A., Vandenbroucke, B. & Schroyen, J., 2014. *Mon. Not. R. Astron. Soc.*, 442, 1830.

Verbeke, R., Vandenbroucke, B. & De Rijcke, S., 2015. *Astrophys. J.*, 815, 85.

Vogelsberger, M., Sijacki, D., Kereš, D., Springel, V. & Hernquist, L., 2012. *Mon. Not. R. Astron. Soc.*, 425, 3024.

Vogelsberger, M., Zavala, J., Simpson, C. & Jenkins, A., 2014a. *Mon. Not. R. Astron. Soc.*, 444, 3684.

Vogelsberger, M. *et al.*, 2014b. *Nature*, 509, 177.

Voronoi, G. F., 1908. *J. Reine Angew. Math.*, 134, 198.

Walter, F. *et al.*, 2007. *Astrophys. J.*, 661, 102.

Wang, J., Frenk, C. S., Navarro, J. F., Gao, L. & Sawala, T., 2012. *Mon. Not. R. Astron. Soc.*, 424, 2715.

Weisz, D. R. *et al.*, 2014. *Astrophys. J.*, 789, 148.

Whitham, G., *Linear and Nonlinear Waves* (Wiley, 2011).

Williams, R. J., Mathur, S., Nicastro, F. & Elvis, M., 2007. *Astrophys. J.*, 665, 247. `astro-ph/0611583`.

Wolf, J. *et al.*, 2010. *Mon. Not. R. Astron. Soc.*, 406, 1220.

Xu, Y. *et al.*, 2015. *Astrophys. J.*, 801, 105.

Yamamoto, S., Saitoh, T. R. & Makino, J., 2015. *Publ. Astron. Soc. Jpn*, 67, 37.

Zeldovich, Y. B., Kurt, V. G. & Syunyaev, R. A., 1968. *J. Exp. Theor. Phys.*, 55, 278.

Zienkiewicz, O. C., Taylor, R. L. & Zhu, J. Z., *The Finite Element Method: Its Basis and Fundamentals*. 6th edition (Butterworth-Heinemann, 2005).

A

Exact geometrical tests

As discussed in Chapter 2, we need *exact* geometrical tests during the incremental Delaunay construction algorithm. In total, we need four tests: a test to check the orientation of three points in 2D, a test to check whether a point in 2D lies inside or outside the circle through three other points, and the respective 3D equivalents of these tests.

Mathematically, these tests can be translated into finding the sign of the following determinants (Shewchuk, 1997; Springel, 2010):

$$\det{}_{\text{orient2D}}(\vec{a},\vec{b},\vec{c}) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix},$$

$$\det{}_{\text{orient3D}}(\vec{a},\vec{b},\vec{c},\vec{d}) = \begin{vmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ d_x & d_y & d_z & 1 \end{vmatrix},$$

$$\det{}_{\text{incircle}}(\vec{a},\vec{b},\vec{c},\vec{d}) = \begin{vmatrix} a_x & a_y & a_x^2+a_y^2 & 1 \\ b_x & b_y & b_x^2+b_y^2 & 1 \\ c_x & c_y & c_x^2+c_y^2 & 1 \\ d_x & d_y & d_x^2+d_y^2 & 1 \end{vmatrix},$$

$$\det{}_{\text{insphere}}(\vec{a},\vec{b},\vec{c},\vec{d},\vec{e}) = \begin{vmatrix} a_x & a_y & a_z & a_x^2+a_y^2+a_z^2 & 1 \\ b_x & b_y & b_z & b_x^2+b_y^2+b_z^2 & 1 \\ c_x & c_y & c_z & c_x^2+c_y^2+c_z^2 & 1 \\ d_x & d_y & d_z & d_x^2+d_y^2+d_z^2 & 1 \\ e_x & e_y & e_z & e_x^2+e_y^2+e_z^2 & 1 \end{vmatrix},$$

where $\vec{a}$, $\vec{b}$, $\vec{c}$, $\vec{d}$ and $\vec{e}$ are the coordinates of points in 2D or 3D (with components a_x, a_y and a_z, etc.). If the three input points for $\det{}_{\text{orient2D}}(\vec{a},\vec{b},\vec{c})$ are in counterclockwise order around the center of the triangle they form (assuming a right-handed Cartesian reference frame), the determinant is positive and we say

the triangle is positively oriented. The determinant will be zero if the three points are colinear.

Likewise, $\det_{\text{orient3D}}(\vec{a}, \vec{b}, \vec{c}, \vec{d})$ will be positive if the point $\vec{d}$ is *below* the plane through $\vec{a}$, $\vec{b}$ and $\vec{c}$, where *above* is defined as the direction from which the triangle formed by the other three points is seen as counterclockwise. In this case, we call the tetrahedron through the four points positively oriented. The determinant will be zero if the four points are coplanar.

With these definitions of positive orientation, the other two determinants will be positive if the last input point is inside the circumcircle or circumsphere of the positively oriented triangle or tetrahedron formed by the other points, negative if it is outside, and zero if the four points are cocircular or the five points are cospherical.

Calculating the sign of these determinants requires only three basic arithmetic operations: addition, subtraction and multiplication. All three operations have the pleasant property that it is possible to predict, for a given precision of input parameters, what precision is needed to exactly store the end result. It is hence possible to carry out these operations *without* any round off error, provided enough precision is used. This is not possible for e.g. a division, where the result can be unbounded and it is impossible to predict what precision is necessary to store the end result exactly or if this is even possible. In this appendix, we will discuss how we can exploit this fact to *exactly* calculate the determinants, or at least to *exactly* determine the sign they will have.

Note that this does not overcome problems associated with the round off of converting values from a decimal system to a binary system. Suppose we want to use the 2D orientation test on the specific set of points $\vec{a} = (0, 0)$, $\vec{b} = (0.4, 1.2)$ and $\vec{c} = (0.5, 1.5)$. These points are colinear, so the answer should be zero. However, 0.4 and 1.2 cannot be represented exactly in a binary floating point format. In double precision, they are represented as

$$0.4 \approx 1.1001100110011001100110011001100110011001100110011010 \times 10^{-10}$$
$$1.2 \approx 1.0011001100110011001100110011001100110011001100110011 \times 10^{0},$$

where the expressions on the right use base 2 rather than base 10. Converting these values back to base 10, we notice that the representation of 1.2 is slightly smaller than 3 times the representation of 0.4, so that the point $\vec{b}$ will lie below the line through $\vec{a}$ and $\vec{c}$ (which can be exactly represented). The triangle will hence have positive orientation, and the determinant will be positive. This result is *exact* on the computer, but it is still the wrong answer to the original problem.

However, this kind of problem is not relevant for the Delaunay construction algorithm, since there we are only interested in exact tests for values that have already been converted into a binary representation. Hence, the algorithms discussed below are sufficient for our purpose.

A.1 Error bounds

When adding, subtracting, multiplying or dividing two floating point values, the result will mathematically consist of two pieces:

$$A \square B = X + Y,$$

where A and B are two floating points and $\square$ is one of the operations listed above ($+$, $-$, $\times$ or $/$). X is the approximate floating point result of the operation (which always has finite precision and can be exactly represented), while Y is the round off error, i.e. the part of the result that cannot be expressed as a floating point value and has to be added to the approximate result to obtain the *true* result of the operation.

Following the IEEE standard, Y is bounded by a relative error factor, ϵ (Shewchuk, 1997):

$$|Y| \leq \epsilon |X|.$$

For a 64-bit floating point value, $\epsilon = 2^{-53}$.

When the result X is used as input for a next floating point operation, not taking into account Y will induce a further error on the result. Suppose for example we want to calculate $\det_{\text{orient2D}}$, given by

$$\det{}_{\text{orient2D}} = (a_x - c_x)(b_y - c_y) - (a_y - c_y)(b_x - c_x)$$

Computationally, this operation boils down to four subtractions, followed by two multiplications and a final subtraction. Each of these operations has a relative error associated with it. Let us try to see what the effect of the accumulated errors will be on the final result.

The first step in the calculation yields

$$\begin{aligned}\det{}_{\text{orient2D}} = [(a_x - c_x)' \pm \epsilon|(a_x - c_x)'|][(b_y - c_y)' \pm \epsilon|(b_y - c_y)'|] - \\ [(a_y - c_y)' \pm \epsilon|(a_y - c_y)'|][(b_x - c_x)' \pm \epsilon|(b_x - c_x)'|].\end{aligned}$$

Here, we have denoted the approximate results of the subtractions by a $'$, and we have taken into account the fact that this result might be a factor ϵ too small or too large.

The next step in the calculation are the multiplications needed to square the results in between brackets:

$$\begin{aligned}\det{}_{\text{orient2D}} = [((a_x - c_x)(b_y - c_y))' \pm 2\epsilon\,|(a_x - c_x)'(b_y - c_y)'| \\ \pm \epsilon^2\,|(a_x - c_x)'(b_y - c_y)'| \pm \epsilon\,|\,((a_x - c_x)(b_y - c_y))'\,|\,] \\ - [((a_y - c_y)(b_x - c_x))' \pm 2\epsilon\,|(a_y - c_y)'(b_x - c_x)'| \\ \pm \epsilon^2\,|(a_y - c_y)'(b_x - c_x)'| \pm \epsilon\,|\,((a_y - c_y)(b_x - c_x))'\,|\,].\end{aligned}$$

Not all error bounds in this expression are useful, since some of them still contain the individual factors. We can rewrite the entire expression in terms of the approximate results of the multiplications, by making use of

$$|(a_x - c_x)'(b_y - c_y)'| = |((a_x - c_x)(b_y - c_y))'| \pm \epsilon\,|((a_x - c_x)(b_y - c_y))'|\,.$$

This gives

$$\begin{aligned}\det{}_{\text{orient2D}} = [&((a_x - c_x)(b_y - c_y))' \\ &\pm (3\epsilon + 3\epsilon^2 + \epsilon^3)\,|((a_x - c_x)(b_y - c_y))'|\,] \\ &- [((a_y - c_y)(b_x - c_x))' \\ &\pm (3\epsilon + 3\epsilon^2 + \epsilon^3)\,|((a_y - c_y)(b_x - c_x))'|\,].\end{aligned}$$

The final step in the calculation yields

$$\begin{aligned}\det{}_{\text{orient2D}} = {} & (\det{}_{\text{orient2D}})' \pm \epsilon\,|(\det{}_{\text{orient2D}})'| \\ & \pm (3\epsilon + 3\epsilon^2 + \epsilon^3)\,(\,|((a_x - c_x)(b_y - c_y))'| + |((a_y - c_y)(b_x - c_x))'|\,)\,.\end{aligned}$$

It is clear that the approximate result $(\det{}_{\text{orient2D}})'$ can be too large or too small by a factor that could be considerably larger than ϵ. In fact, we can only be sure that the sign of $(\det{}_{\text{orient2D}})'$ is correct, if the following inequality holds:

$$\begin{aligned}(1 - \epsilon)\,|(\det{}_{\text{orient2D}})'| > {} & (3\epsilon + 3\epsilon^2 + \epsilon^3)\,(\,|((a_x - c_x)(b_y - c_y))'| \\ & + |((a_y - c_y)(b_x - c_x))'|\,)\,.\end{aligned}$$

We still need to bring the factor $1 - \epsilon$ to the other side of the inequality, making sure that we do not underestimate the error bound. If we want to be able to calculate the sum in between brackets, we also need to take into account the round off due to this calculation. Finally, we need to round the total error bound up to a number that can be expressed as a 64-bit floating point value. This finally yields the following practical error bound (Shewchuk, 1997):

$$\begin{aligned}|(\det{}_{\text{orient2D}})'| > \big((3\epsilon + 16\epsilon^2)\,(\,|((a_x - c_x)(b_y - c_y))'| \\ + |((a_y - c_y)(b_x - c_x))'|\,)\big)'\,.\end{aligned}$$

If this inequality holds, we are sure that the sign of the determinant will be correct. Since the terms between brackets are intermediate results in the computation, calculating this error bound does not cause much overhead. Furthermore, the factor $3\epsilon + 16\epsilon^2$ can be precomputed once and then reused.

It is possible to compute similar error bounds for the other determinants above. Having these, we can determine with absolute certainty if the resulting sign of the determinant will be correct. If it is, we are done. If it is not, we will need a way to do better than ordinary floating point arithmetics...

A.2 Arbitrary precision arithmetics

From the previous section, we know that the result of addition, subtraction and multiplication can be split into two parts: the approximate result obtained using floating point arithmetics, and an error term containing the part of the result that is lost due to round off error. It is actually possible to compute this second term using floating point arithmetics.

For an addition, the true result can never contain more information than fits in two floating point values, since in the worst case we add the largest and the smallest possible floating point value and we end up with the approximate result being the largest one and the error term being the smallest one. For all other additions, the extra floating point value containing the error bound can be calculated from (Shewchuk, 1997)

$$\epsilon_{a+b} = (b - ((a + b)' - a)')',$$

where every $'$ represents a floating point operation with round off. Apart from this, there are also cases where the error term ϵ_{a+b} is exactly zero (e.g. when $|a + b| \leq |a|$ and $|a + b| \leq |b|$).

For subtraction and multiplication, similar results hold and it is also possible to calculate a single floating point error term. It is hence possible to perform all of these operations exactly, by explicitly keeping track of the single floating point error term.

However, to obtain an exact result for the determinants, we need to do better than this, since we also need to be able to do further computations with these composite exact floating point results. We therefore define a notion of *expansion sum*, which is the expansion of a floating point value in non-overlapping floating point terms, so that the first term is the result rounded to double precision, and all extra terms are error terms of different order (in fact, this is what we mean by *non-overlapping*; we will not go into the mathematical definition of this term). Exact arithmetics is then just a matter of defining exact addition, subtraction and multiplication on these expansion sum representations of floating point numbers. This is far from trivial, and we will not discuss it in detail (see Shewchuk, 1997).

It might be clear from the above that exact arithmetics comes with a cost: since we need to keep track of a significant number of extra floating point terms during all operations, the calculation of the determinants will take a lot longer. Since the geometrical tests are at the core of the Delaunay construction algorithm, this will have a significant impact on the total runtime of this algorithm.

Since we do not want to waste resources on exact arithmetics when it is unnecessary, we will use *arbitrary precision arithmetics* instead: rather than using exact arithmetics everywhere, we will first use standard floating point arithmetics to obtain the error bounds that will tell us if the sign of the determinant is correct.

In the rare cases that round off error does indeed affect the result, we will use exact arithmetics for the first operations in the calculation to obtain a slightly more precise result, and again determine the error bounds. We repeat this until the result is precise enough to be sure about the sign of the determinant. In the worst case scenario, we will need to use exact arithmetics for the entire computation, but this will very rarely happen. This then justifies the computational overhead of calculating the error bounds.

A first version of SHADOWFAX used this approach by means of the predicates provided by Shewchuk (1997)[1]. It was discovered however that this did not work on all systems. The reason for this is that we need to be absolutely sure that the intermediate results are stored as 64-bit floating point values for the error term calculations to work. However, as already mentioned in Chapter 2, most modern architectures try to do better than this and store the intermediate results on the CPU as a 80-bit floating point. If result and error term are computed sequentially, then the result used in the error term calculation will have too much precision, and the error term will be wrong. This wrong error term will then propagate through the consecutive operations to yield a wrong final result. There are ways to overcome this problem, by forcing the system to write out the intermediate results to memory before using them in consecutive operations, but this inevitably leads to a slow down of the computations. Furthermore, this is not guaranteed to work on all systems.

To make the code more robust, we hence need to implement yet another method to perform exact geometrical tests. This method is discussed next.

A.3 Mapped integer arithmetics

Recall from Chapter 2 that a 64-bit floating point consists of two parts: a 53-bit mantissa, holding the sign and a fixed precision floating point value, and an 11-bit exponent. Without the exponent, the mantissas map out a much smaller range of possible values, that is formally equal to that of the 53-bit integers. It is the combination of mantissa and exponent that expands this small range to the enormous numerical dynamical range of the floating point values.

But it is also the combination of exponent and mantissa that causes much of the trouble encountered above. If floating point values would map out a similar range as the integer values, then only two types of round off error would be possible: an overflow when a value is larger than the maximal allowed value, and a real round off when the absolute value of a value is smaller than zero and becomes just zero. The latter can only occur when divisions are included in the possible set of operations, which is not the case here. All problems with round

[1] `http://www.cs.cmu.edu/quake/robust.html`

off can hence be solved very easily by increasing the number of bits of the values involved: a single bit for an addition or subtraction (since the result can never be larger than two times the maximal allowed value), and twice as many bits for multiplication (since we should be able to store the maximal allowed value squared).

Combined with an exponent, we still have both types of round off error, but now not only at the extreme values of the range of possible values, but also everywhere in between. If we add two floating point values, then we need to convert them to a common exponent first. Since overflow results in an unbounded error, this has to be the largest exponent of the two, so we will have round off in the mantissa of the smallest. We could still solve this by using more bits, but now it is not clear how many bits this should be, since this depends on the values of the exponents.

To be able to use the much simpler approach of just increasing the number of bits to get more precision, we thus need to get rid of the exponent. In practice, this means that we will extract the mantissa from the floating point values and convert it to a 53-bit (or standard 64-bit) integer (this can be done by multiplying the floating point value with an appropriate integer value and storing the result as an integer). However, to be sure that this indeed maps out a continuous range of values, we of course need to make sure that all exponents are the same.

We do not only need all exponents for the floating point inputs to one of the tests to be the same, but also that of all other possible inputs to all other possible tests. The reason for this is the same as the reason we want to use exact tests in the first place: consistency. Suppose we would allow generator coordinates to have arbitrary floating point exponents, and that we would convert these to some common exponent if the coordinates are used in a test. Then it is possible that the same coordinates will be converted differently when used as input for another test (combined with other coordinates with possible other exponents). This could mean that the extracted integer mantissa is different for one test than it is for another, which could cause inconsistent test results, which would then lead to a deadlock of the algorithm, which is what we try to avoid.

We hence need all coordinates to have the same floating point exponents. This can be realised by only using coordinates in the range $[1, 2]$, since all of these numbers will have an exponent zero. The corresponding mantissas will then also all lie in the continuous range $[1, 2]$ and map precisely to the range of 52-bit integers (since they all have the same sign) (Springel, 2010). This conversion might seem to limit the dynamical range of the tessellations that is being constructed, or worse, that of the generators that generate it. This is however not a problem if we only do it for the coordinates that are used during the mesh construction, since they need not have the same precision as the coordinates that are integrated in time. We hence will always store two coordinates for *active*

generators: their actual physical double precision coordinates, and the mesh generating coordinates in the range $[1, 2]$ that have less precision and that are obtained from the former by mapping the entire simulation box to the unit cube with sides $[1, 2]$.

We could of course also store the 53-bit mantissas of these coordinates instead, since they are what we actually use in the calculations. Just as with the arbitrary precision arithmetics, performing the exact calculations is however not always necessary and is quite computationally expensive. We will therefore use the same technique that was introduced there and first compute approximate results and error bounds. Only in the rare cases that the results are not guaranteed to have the correct sign, we will map the mantissas to integers and perform exact arithmetics. Note however that we will use the mapped coordinates in the range $[1, 2]$ for all tests, since otherwise we would again introduce inconsistencies between the treatment of the normal inexact tests, and the special more exact tests.

We have now shown that we can indeed perform exact arithmetics in a machine and architecture independent way by mapping floating point values to integers. We are left with the trivial task of determining the size integers should have to indeed be sure that the results are exact. Recall from above that for addition, subtraction and multiplication, the only possible round off error for integers is an overflow. We hence need to make sure the integers are large enough to store the largest possible result of the combination of these operations that constitutes the computation.

For $\det_{\text{orient2D}}$, given by

$$\det_{\text{orient2D}} = (a_x - c_x)(b_y - c_y) - (a_y - c_y)(b_x - c_x),$$

we find that the subtractions in between brackets can always be stored in a 54-bit integer, since the sum of two 53-bit integers can never be larger than 2^{54}. The multiplications of two 54-bit integers can never be larger than 108 bits, and the subtraction of two 108-bit integers can never be larger than 109 bits. 109-bit integers are hence enough for this computation.

Similarly, we find that $\det_{\text{orient3D}}$ requires at most 165 bits, $\det_{\text{incircle}}$ can be calculated using 220 bits, and $\det_{\text{insphere}}$ will never be larger than 277 bits.

All of these numbers are larger than the standard 64-bit integer type, so that we need to use a software library to perform the arithmetics. Springel (2010) uses the GMP library[2], which has no predefined precision. Since we know what precision we need, we will resort to the BOOST.MULTIPRECISION library[3], which allows us to define new integer types with these precisions (although timing results

[2] `https://gmplib.org/`

[3] `http://www.boost.org/doc/libs/release/libs/multiprecision/doc/html/index.html`

suggest that it is better to use somewhat larger power of two precisions in some of the cases).

B

Mesh evolution algorithm

In Chapter 2, we introduced a Voronoi mesh evolution algorithm. In this appendix, we will detail the implementation of this algorithm in the moving mesh code Shadowfax. This algorithm is not part of the current public version of Shadowfax, but will be part of a future release.

B.1 2D

The 2D adaptive Voronoi mesh is abstracted into a single class, that takes a valid already constructed Voronoi mesh (constructed using e.g. the technique discussed in Chapter 2) as input at construction. The information contained in this mesh is first translated into a more efficient structure for the evolution algorithm. At the end of this step, a valid adaptive Voronoi mesh is available and its properties can be queried.

During the remainder of the simulation, the mesh can be in two states: either it is still valid, or the generators have been moved and the mesh is not yet adapted to this situation and has become invalid. The mesh will then still contain information, but will no longer contain a valid Voronoi mesh that can be queried.

B.1.1 Structure

The information stored in the mesh consists of two aspects: the connectivity information, that tells us which cells or generators are neighbours of each other, and the actual mesh, that consists of the vertices of the cells and yields cell volumes and centroids, and interface areas and midpoints. The mesh itself strongly depends on the connectivity and the actual positions of the mesh generators, and will change whenever one of these changes. However, a change in generator positions does not necessarily induce a change in connectivity, so that we will evolve the connectivity, rather than reconstruct it.

We will hence always store a connectivity for the mesh, but this connectivity will not necessarily correspond to the specific connectivity that yields a valid

Voronoi mesh. Only when it does, it makes sense to calculate an actual Voronoi mesh. We will therefore only calculate the actual mesh when the connectivity is valid.

Connectivity

The connectivity consists of the information about which generators are neighbours. For efficient mesh construction, this information is stored on a per generator basis. With every cell generator, we associate an object of an adaptive Voronoi cell class, that stores a list of references to the neighbouring cells. In our specific implementation, these references are the indices of these cells in a general list of all cells that is stored in the adaptive Voronoi mesh class, with special values of the index flagging special neighbour types, like boundary neighbours, that are necessary to represent reflective boundary conditions.

The neighbours for a cell are furthermore ordered counterclockwise with respect to the generator of the cell. This way, we can very easily determine which neighbours of a cell are also mutual neighbours, which will be of great help during the mesh restoration step.

Mesh construction

To construct the actual Voronoi mesh from a valid connectivity, we traverse the list of neighbouring generators for every cell, and calculate the midpoints of the circumcircles through the cell generator and two consecutive neighbours from the list (with the last element of the list being combined with the first to make the cell complete). These midpoints then correspond to the vertices of the cell, and due to the counterclockwise ordering of the generators, they are also in counterclockwise order. This ordering allows for a very efficient calculation of the cell volume (the 2D face area) and centroid, as described in Chapter 2. Calculating face areas (line segment sizes) and midpoints is trivial in 2D.

B.1.2 Mesh restoration

When the positions of the generators of the mesh change, we have to signal this to the adaptive mesh by calling an appropriate method of the object, and passing on the new positions to this method. To allow for an easy identification of the new coordinates and the corresponding generator, we use the same unique identifier that is associated with the generator when it is added to the standard Voronoi mesh from which the adaptive mesh is constructed. This is also the same identifier that is used to access cell properties later on.

In parallel simulations, the change in generator positions can trigger a redecomposition of the domain, during which generators move from one process

to another. We have to account for this by moving the corresponding cells and adjusting the necessary neighbour relations.

After the positions have been updated, the adaptive mesh is no longer valid, and cell properties can no longer be queried. To make it valid again, we have to restore the mesh. This is done by adding all cells that are affected by the new positions to a queue for checking. The cells in this queue are then traversed one by one. For every cell, we subject triples of neighbours to the incircle test: if D is the cell generator, and A, C and B are its neighbours (in counterclockwise order), then we check if B is in the circumcircle through D, A and C. If it is not, the triple ACB is valid, and we move on. If it is not, we have detected a face flip, and have to carry it out. This will change neighbour relations for the four cells, so that we will need to repeat the checks for these cells. If all triples are valid, the entire cell is valid, and can be removed from the queue. It can however always be added to the queue again later on, if a face flip is detected and handled in one of its neighbours.

Face flip

Suppose a face flip is detected in the triple neighbours ACB of the cell generated by D (see Fig. 2.14). Before the face flip is carried out, the relevant parts of the neighbour lists of the different cells are

D: ... - A - C - B - ...
A: ... - C - D - ...
C: ... - B - D - A - ...
B: ... - D - C - ...

The face flip signals the invalidation of the neighbour relation between C and D, and the creation of a new neighbour relation between A and B. After the face flip, the neighbour lists for the different cells are

D: ... - A - B - ...
A: ... - C - B - D - ...
C: ... - B - A - ...
B: ... - D - A - C - ...

This means the following operations need to be carried out:

- Insertion of a new neighbour B in between neighbours C and D for cell A
- Insertion of a new neighbour A in between neighbours D and C for cell B
- Removal of neighbour C from the neighbour list of D
- Removal of neighbour D from the neighbour list of C

Boundaries

In Chapter 2, we mentioned that a Voronoi mesh is always enclosed within a box. This is necessary, since otherwise cells at the boundaries of the volume would have infinite volumes. In general, two types of boxes are used: reflective boxes, whereby the cells are cut off at the boundaries of the box, and one or more faces of the cell coincide with these boundaries, and periodic boxes, whereby points outside the box are replicated at the other side of the box, and cells at the boundaries of the box still have irregular shapes. Both are illustrated in Fig. B.1.

To construct a Voronoi mesh inside a box, we have to add *boundary ghosts*: extra generators that represent copies of generators inside the box. The faces between internal generators and these boundary ghosts will then trace out the boundary of the box. For both types of boxes, this ensures that the sum of all cell volumes equals the volume of the box.

In the classical incremental construction algorithm of Chapter 2, these boundary ghosts are treated as normal generators, that are inserted at the end of the incremental Delaunay construction algorithm (Springel, 2010). For the mesh evolution algorithm, we have to be a bit more careful, since we need to keep track of these ghosts in between two valid states of the adaptive mesh. If we adapt the position of a generator in between valid states of the mesh, we also have to update the positions of the ghosts that depend on this generator. We use different approaches for periodic and reflective boundaries.

Periodic ghost generators are nothing else than ordinary generators that have been translated. They behave exactly as normal generators; the only difference is that we need to adapt their coordinates to account for this translation whenever coordinates are required. To this end, we keep track of a second list for all cells, that associates a *wall key* to every neighbour of the cell. The wall key can be either zero, in which case we have a normal neighbour, or it can be a negative integer, in which case the neighbour is translated to the relevant periodic copy of the box (see Fig. B.2).

When a generator moves outside the periodic box through a wall, its coordinates are changed to those of the periodic copy that lies inside the box. This also means we need to change the wall key for all neighbouring generators. It is straightforward to precalculate movement tables that give the new wall key for a neighbour with a given old wall key after a movement through a wall with given key. Likewise, we can precalculate tables that give the wall key for a newly added neighbour during the face flip.

Reflective ghosts are ghost copies of the generator itself, by mirroring the generator coordinates with respect to a wall or corner of the box. Like in the case of periodic boundaries, we can assign a wall key to each reflective ghost. Reflective ghosts can however not be treated as normal generators, since then we would end up with particles that are neighbours of themselves. We hence need to

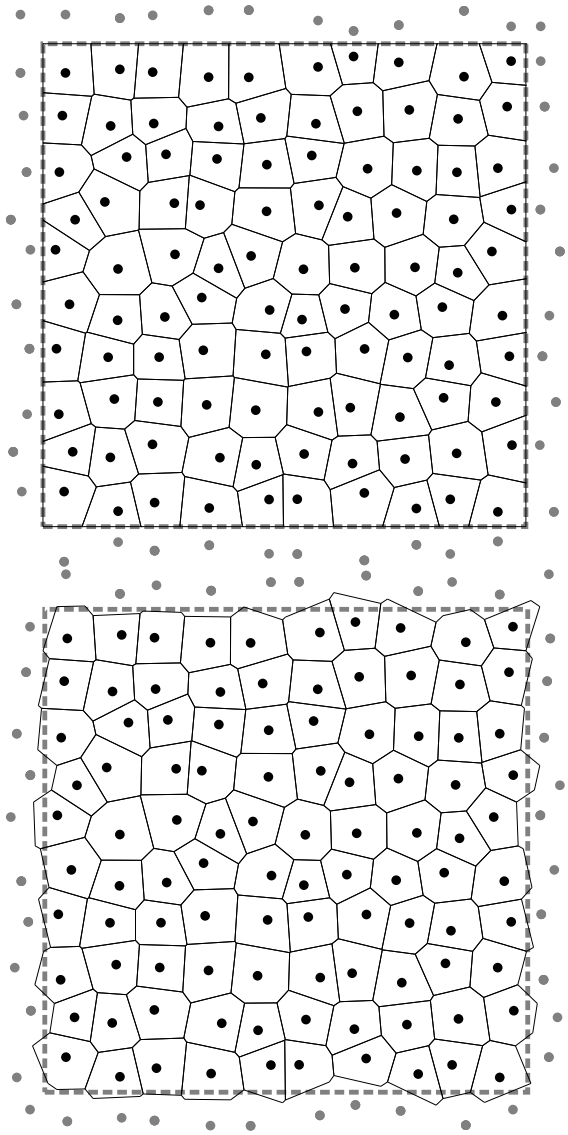


Figure B.1: *Ghost generators are used to represent the boundaries of the box.* Top: *mirror copies of generators generate reflective boundaries,* bottom: *periodic copies of generators generate periodic boundaries.*

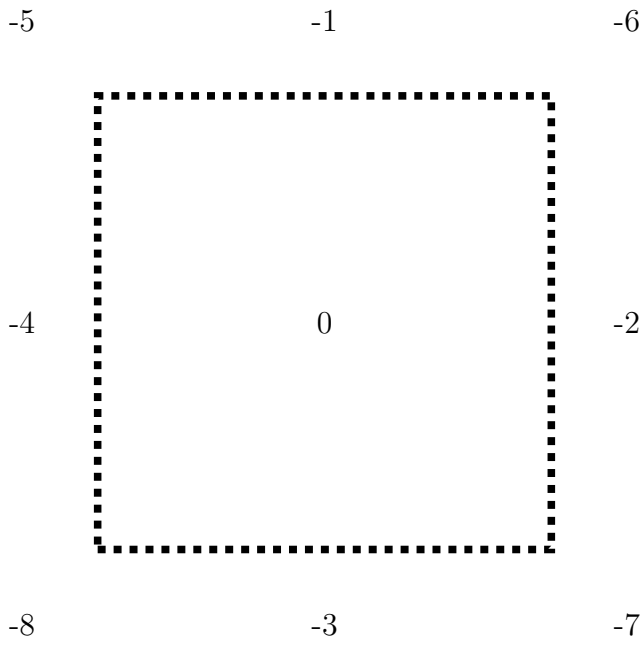


Figure B.2: *Wall keys for the 2D evolution algorithm.*

treat them on a different level throughout the algorithm. We therefore define a new type of ghost generator, that can be a neighbour of a normal generator, but that itself does not have neighbours, but only stores a reference to the generator of which it is a mirror copy, and the wall key of the appropriate mirroring boundary. After the positions of the original generators have been updated, the positions of the associated mirror copies are updated as well.

The presence of mirror copies adds extra complexity to the face flip algorithm, as cells at the boundaries will regularly contain faces of zero length caused by degenerate generator configurations. Normally, a cell will only have ghost neighbours that are mirror copies of the cell generator itself. But due to round off error, the order four vertex at the boundary of the box between the cell generator, its mirror copy, a neighbouring cell and the neighbour's mirror copy can be split into two order three vertices that are very close together (Fig. B.3). If the cell generator moves away from the boundary, we have to make sure all ghost neighbours are removed, not only the mirrors of the cell generator itself.

If a cell touches the border, we need to add new ghost generators; this also requires some extra bookkeeping.

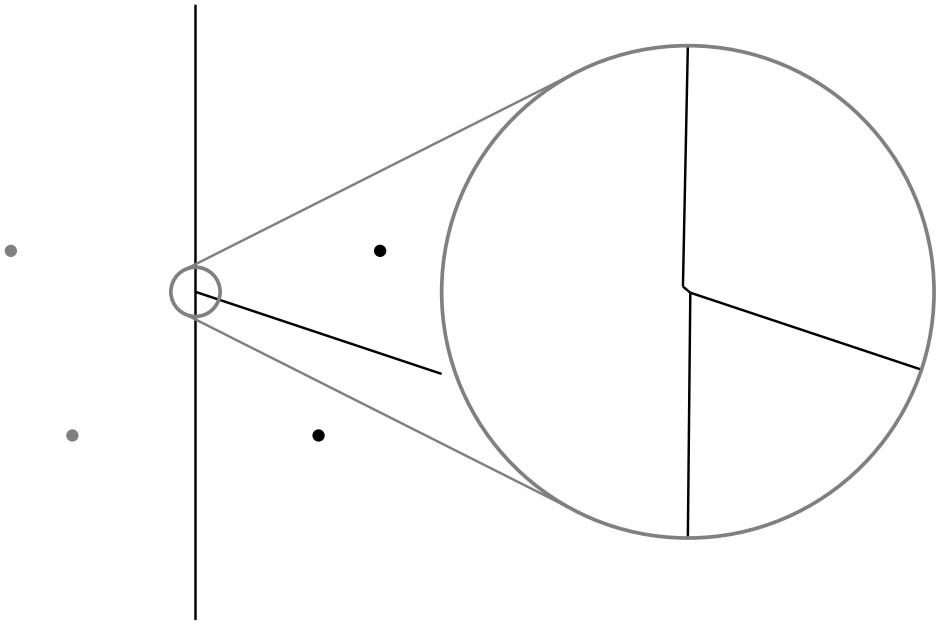


Figure B.3: *Due to round off error, cells at the reflective boundary can have ghost neighbours that are not a mirror copy of the cell generator.*

B.2 3D

The 3D adaptive Voronoi mesh is also contained in a single class, and can be either valid or invalid. Externally, the object behaves exactly like the 2D adaptive Voronoi mesh, but internally it works a bit differently, since we cannot update the generator positions all at once, like in the 3D case (see Chapter 2).

B.2.1 Structure

As in 2D, we create a cell object that holds a list of neighbours for every generator. However, this list is not enough to be able to construct the vertices of the Voronoi cell, since every vertex is now determined by four generators. For this reason, we defined a new type of neighbour, a *face neighbour*. For every normal neighbour of the cell, we also store a list of face neighbours that trace out the face between the cell and that neighbour. The list of face neighbours is ordered counterclockwise with respect to the vector pointing from the cell generator to the neighbouring generator, so that the vertices of the face are given by calculating the midpoint

of the sphere through the cell generator, the neighbour and two consecutive face neighbours in the list. Every face neighbour is also a normal neighbour of the cell, and the faces corresponding to them are the neighbouring faces of the face they trace out.

B.2.2 Mesh restoration

As mentioned in Chapter 2, there are three distinct cases of mesh deviations that can occur after a small movement of a cell generator, with each their own restoration operation. These three cases mimic the three operations that can be encountered during the validity check of newly created tetrahedra during the incremental Delaunay construction algorithm. The first two involve five distinct generators and correspond to the creation or removal of a new face. The last one is a degenerate case that involves six distinct generators and flips a face.

Face creation

Fig. 2.16 illustrates what happens when generator E enters the circumsphere through cell generator A, cell neighbour B, and face neighbour C and D. The edge of the face consisting of the midpoint of the sphere through ABCD and the midpoint of the sphere through ABED flips (and the same edge also flips in the face between A and D, and between B and D). This edge flip can be restored by inserting a new face between C and E.

Before the restoration, the relevant parts of the face neighbour lists are given by

A: B: ... - E - D - C - ...
C: ... - B - D - ...
D: ... - C - B - E - ...
E: ... - D - B - ...

B: A: ... - C - D - E - ...
C: ... - D - A - ...
D: ... - E - A - C - ...
E: ... - A - D - ...

C: A: ... - D - B - ...
B: ... - A - D - ...
D: ... - B - A - ...

D: A: ... - E - B - C - ...
B: ... - C - A - E - ...
C: ... -A - B - ...
E: ... - B - A - ...

E: A: ... - B - D - ...
B: ... - D - A - ...
D: ... - A - B - ...

After the restoration, this becomes

A: B: ... - E - C - ...
C: ... - B - E - D - ...
D: ... - C - E - ...
E: ... - D - C - B - ...

B: A: ... - C - E - ...
C: ... - D - E - A - ...
D: ... - E - C - ...
E: ... - A - C - D - ...

C: A: ... - D - E - B - ... D: A: ... - E - C - ...
B: ... - A - E - D - ... B: ... - C - E - ...
D: ... - B - E - A - ... C: ... - A - E - B - ...
E: A - D - B E: ... - B - C - A - ...
E: A: ... - B - C - D - ...
B: ... - D - C - A - ...
C: A - B - D
D: ... - A - C - B - ...

The newly created face between C and E has only three face neighbours.

To restore the mesh, we hence need the following operations:

- Insertion of the new face in between C and E, with the face neighbours given above
- Insertion of face neighbour E in the faces between A and C, B and C, and C and D
- Insertion of face neighbour C in the faces between A and E, B and E, and D and C
- Removal of face neighbour A in the face between B and D
- Removal of face neighbour B in the face between A and D
- Removal of face neighbour D in the face between A and B

The cell where the edge flip is detected automatically assumes the role of A, while the neighbour associated with the face assumes the role of B. D is the face neighbour that generates the flipped edge, and C and E are the face neighbours before and after D in the face neighbour list.

Face removal

Face removal is the reverse process of face insertion, and it occurs when the face between C and E that was created above flips: if one edge of this face flips, the other two automatically also flips, so that the entire face flips.

To restore the mesh, we execute the operations from above in reverse:

- Removal of the face between C and E
- Removal of face neighbour E in the faces between A and C, B and C, and C and D
- Removal of face neighbour C in the faces between A and E, B and E, and D and C

- Insertion of face neighbour A in the face between B and D
- Insertion of face neighbour B in the face between A and D
- Insertion of face neighbour D in the face between A and B

To find out which generator assumes which role in this case, we cannot simply use any flipped edge. The flipping face will cause edge flips in the surrounding faces as well, which could be misinterpreted as flagging a face creation. We therefore first need to locate the face with only three face neighbours that flips entirely. The generator of the cell with that face then assumes the role of C, while the neighbour associated with the face assumes the role of E. A, B and D are then the face neighbours of this face (actually ordered as ADB in the face neighbour list for C).

Face flip

If two opposing edges flip in a face with only four edges (and only four face neighbours), then this entire face is removed and replaced by a new face between the generators that correspond to the flipped edges. This effectively corresponds to the 3D equivalent of a 2D face flip.

We call the cell to which the flipping face belongs C, and its neighbour E (analogous to the case above). The face neighbours corresponding to the flipped edges (in between which the new face is created) are called A and B, while the other two face neighbours are called D and F. The face neighbour lists before the face flip are

A:	C:	... - D - E - F - ...	B:	C:	... - F - E - D - ...
	D:	... - E - C - ...		D:	... - C - E - ...
	E:	... - F - C - D - ...		E:	... - D - C - F - ...
	F:	... - C - E - ...		F:	... - E - C - ...
C:	A:	... - F - E - D - ...	D:	A:	... - C - E - ...
	B:	... - D - E - F - ...		B:	... - E - C - ...
	D:	... - A - E - B - ...		C:	... - B - E - A - ...
	E:	A - F - B - D		E:	... - A - C - B - ...
	F:	... - B - E - A - ...			
E:	A:	... - D - C - F - ...	F:	A:	... - E - C - ...
	B:	... - F - C - D - ...		B:	... - C - E - ...
	C:	D - B - F - A		C:	... -A - E - B - ...
	D:	... - B - C - A - ...		E:	... - B - C - A - ...
	F:	... - A - C - B - ...			

After the face flip, the face neighbour lists have become

A:	B:	C - D - E - F	B:	A:	F - E - D - C
	C:	... - D - B - F - ...		C:	... - F - A - D - ...
	D:	... - E - B - C - ...		D:	... - C - A - E - ...
	E:	... - F - B - D - ...		E:	... - D - A - F - ...
	F:	... - C - B - E - ...		F:	... - E - A - C - ...
C:	A:	... - F - B - D - ...	D:	A:	... - C- B - E - ...
	B:	... - D - A - F - ...		B:	... - E - A - C - ...
	D:	... - A - B - ...		C:	... - B - A - ...
	F:	... - B - A - ...		E:	... - A - B - ...
E:	A:	... - D - B - F - ...	F:	A:	... - E - B - C - ...
	B:	... - F - A - D - ...		B:	... - C - A - E - ...
	D:	... - B - A - ...		C:	... - A - B - ...
	F:	... - A - B - ...		E:	... - B - A - ...

The face flip can hence be carried out through the following operations:

- Removal of the face between C and E
- Insertion of the new face between A and B, with the four face neighbours as given above
- Replacement of face neighbour E by face neighbour B in the face between A and C
- Replacement of face neighbour C by face neighbour B in the face between A and E
- Replacement of face neighbour E by face neighbour A in the face between B and C
- Replacement of face neighbour C by face neighbour A in the face between B and E
- Insertion of face neighbour B in the faces between A and D, and A and F
- Insertion of face neighbour A in the faces between B and D, and D and F
- Removal of face neighbour E in the faces between C and D, and C and F
- Removal of face neighbour C in the faces between D and E, and E and F

Boundaries

Boundaries are treated as in 2D by the insertion of ghost generators. However, since reflective boundaries lead to similar problems with fourth order vertices as

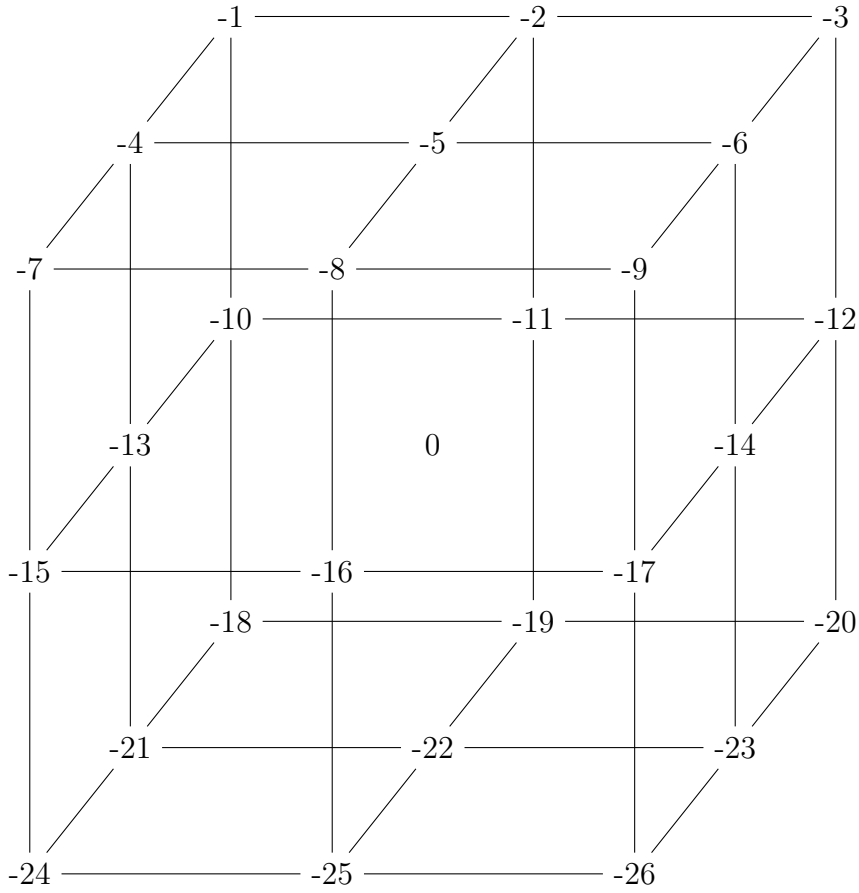


Figure B.4: *Wall keys for the 3D evolution algorithm.*

in 2D, we have not currently implemented them (we will see below that the 3D algorithm does not perform well when handling fourth order vertices).

Periodic boundaries are implemented by keeping track of two extra lists for every cell: a list of wall indices for all neighbours, and a list of face wall indices for all face neighbours of a face. The wall index convention is illustrated in Fig. B.4. The bookkeeping is similar to the 2D algorithm, although everything is a bit more complex.

B.2.3 Algorithm

In 3D, we cannot move all generators and then fix the Voronoi mesh, as is the case in 2D. Instead, we need to move the generators one by one, and refine their movement when necessary, to prevent different deviations from overlapping. To this end, we devised a counting scheme for edge flips. Once we have moved the generator of a single cell, we check all faces of this cell, and all faces of neighbouring cells that have this cell as a face neighbour, and count the number of edge flips, the number of edge flips in faces with only three edges, and the number of edge flips in faces with four edges. We then compare these numbers to the flip signature of the three different deviations described above. If no flips are encountered, the generator movement is accepted, and we move on with the next cell. If the flip signature matches one of the known cases, we also accept the movement, and restore the cell. We then check the cell and its neighbours again, since the restoration can lead to consecutive deviations. If a non-trivial signature is encountered that does not match a known case, we refine the movement: we move the generator back to its original position and then move it with a fraction of the first displacement. We repeat this procedure until the entire movement has been carried out.

A normal face creation is signalled by a single edge flip. This edge will be shared by three faces, which are each shared by two cells. In total, we hence expect six edge flips to signal a face creation. To speed up the algorithm, we check every face only once: if a face has already been checked in one of the two cells that shares it, we skip it in the other cell. We hence are left with three edge flips as the face creation signature.

A face removal corresponds to three edge flips in a single face with only three edges. These edges are all shared by two other faces (each shared by two cells), so that we end up with 18 edge flips, and two face flips as the signature for a face removal. However, we will not count the six edge flips in the flipped face, and for efficiency we only check every face once. We hence end up with one flipped face and six flipped edges as the signature for a face removal.

The degenerate face flip is signalled by two opposite edges flipping in a face with only four edges. We will again not count the edge flips in the flipping face

(but count them as a distinct double edge flip), and count every other face only once. The signature for a face flip is then a single double edge flip, and four flipping edges.

We have tried to optimise the performance of the algorithm, by immediately breaking off the counting algorithm if an invalid count is encountered: if the number of edge flips in a single face is too high, or the total number of edge flips exceeds six. We also keep track of which edges flipped, so that we already have the necessary information for the mesh restoration if a valid signature is encountered. Despite all this, the speed up obtained with the 3D evolution algorithm is less significant than for the 2D algorithm. This, and the fact that we have not yet found a safe way to use the 3D algorithm with reflective boundaries (or an efficient way to parallelize it), makes this algorithm not yet fit for scientific use.

Index