A&A 560, A35 (2013)
DOI: 10.1051/0004-6361/201322281

Astronomy & Astrophysics

Using 3D Voronoi grids in radiative transfer simulations

P. Camps, M. Baes, and W. Saftly

Sterrenkundig Observatorium, Universiteit Gent, Krijgslaan 281, 9000 Gent, Belgium
e-mail: peter.camps@ugent.be

Received 15 July 2013 / Accepted 5 October 2013

ABSTRACT

Context. Probing the structure of complex astrophysical objects requires effective three-dimensional (3D) numerical simulation of the relevant radiative transfer (RT) processes. As with any numerical simulation code, the choice of an appropriate discretization is crucial. Adaptive grids with cuboidal cells such as octrees have proven very popular; however, several recently introduced hydrodynamical and RT codes are based on a Voronoi tessellation of the spatial domain. An unstructured grid of this nature poses new challenges in laying down the rays (straight paths) needed in RT codes.
Aims. We show that it is straightforward to implement accurate and efficient RT on 3D Voronoi grids.
Methods. We present a method for computing straight paths between two arbitrary points through a 3D Voronoi grid in the context of a RT code. We implement this grid in our RT code SKIRT, using the open source library Voro++ to obtain the relevant properties of the Voronoi grid cells based solely on the generating points. We compare the results obtained through the Voronoi grid with those generated by an octree grid for two synthetic models, and we perform the well-known Pascucci RT benchmark using the Voronoi grid.
Results. The presented algorithm produces correct results for our test models. Shooting photon packages through the geometrically much more complex 3D Voronoi grid is only about three times slower than the equivalent process in an octree grid with the same number of cells, while in fact the total number of Voronoi grid cells may be lower for an equally good representation of the density field.
Conclusions. The benefits of using a Voronoi grid in RT simulation codes will often outweigh the somewhat slower performance.

Key words. hydrodynamics – radiative transfer – methods: numerical

1. Introduction

The radiation observed from most astrophysical systems has been substantially affected by the gas and/or dust residing in or in front of the system under study. Many of these systems have a complicated three-dimensional (3D) geometry, for example arm structures in spiral galaxies (Misiriotis et al. 2000; Fritz et al. 2012), filaments and clumps in star-forming regions (Goldsmith et al. 2008; Paron et al. 2013; Fallscheer et al. 2013), and bow-shocks around evolved stars (Decin et al. 2012; Wang et al. 2013). Properly probing the structure of these systems requires a 3D numerical treatment of the relevant radiative transfer (RT) processes (Disney et al. 1989; Witt et al. 1992; Baes & Dejonghe 2001), which may include dust, line, ionizing UV, Lyα, neutron, and neutrino RT.

Because of the complexities involved in multiple anisotropic scattering, absorption, and reemission, the full RT problem is highly nonlocal and nonlinear. It is not feasible to directly integrate the equations in three dimensions except for the simplest problems. Consequently virtually all modern RT codes use ray tracing or Monte Carlo techniques (Gordon et al. 2001; Ciardi et al. 2001; Kurosawa & Hillier 2001; Juvela & Padoan 2003; Wolf 2003; Stamatellos & Whitworth 2003; Bethell et al. 2004; Harries et al. 2004; Wood et al. 2004; Doty et al. 2005; Ercolano et al. 2005; Niccolini & Alcolea 2006; Jonsson 2006; Pinte et al. 2006; Verhamme et al. 2006; Tasitsiomi 2006; Steinacker et al. 2006; Bianchi 2008; Laursen et al. 2009; Baes et al. 2011; Robitaille 2011; Heymann & Siebenmorgen 2012; Abdikamalov et al. 2012; Lunttila & Juvela 2012; Steinacker et al. 2013). In addition to their intrinsic 3D nature, these techniques allow the inclusion of, for example, a clumpy medium (Witt & Gordon 1996, 2000; Bianchi et al. 2000; Doty et al. 2005; Stalevski et al. 2012; Schechtman-Rook et al. 2012), polarization (Code & Whitney 1995; Goosmann & Gaskell 2007), or kinematical information (Matthews & Wood 2001; Baes & Dejonghe 2002; Baes et al. 2003). Recent RT applications explicitly using 3D include models of young stellar objects (Wolf et al. 1998), protostellar to protoplanetary disks (Indebetouw et al. 2006; Niccolini & Alcolea 2006), reflection nebulae (Witt & Gordon 1996), molecular clouds (Pelkonen et al. 2009; Steinacker et al. 2005), spiral galaxies (Bianchi 2008; Schechtman-Rook et al. 2012; De Looze et al. 2012a), interacting and starburst galaxies (Chakrabarti et al. 2007; Hayward et al. 2011), and active galactic nuclei (Schartmann et al. 2008; Stalevski et al. 2012).

For the purpose of numerical computation, the domain under study must be discretized. Since memory requirements and computation time rapidly increase with the number of grid cells, modern 3D RT codes employ an adaptive grid, placing more and smaller cells in areas that require a higher resolution. Starting from a cuboidal root cell that spans the complete spatial domain, an adaptive mesh refinement (AMR) scheme recursively subdivides each cell into $k \times l \times m$ cuboidal subcells until the required resolution is reached. In the special case of an octree, $k = l = m = 2$ so that each cell is subdivided into eight subcells (hence the name of the data structure). Adaptive-mesh grids and especially octree grids are well established (Kurosawa & Hillier 2001; Steinacker et al. 2002; Wolf 2003; Harries et al. 2004; Niccolini & Alcolea 2006; Jonsson 2006; Bianchi 2008; Laursen et al. 2009; Robitaille 2011; Lunttila & Juvela 2012;

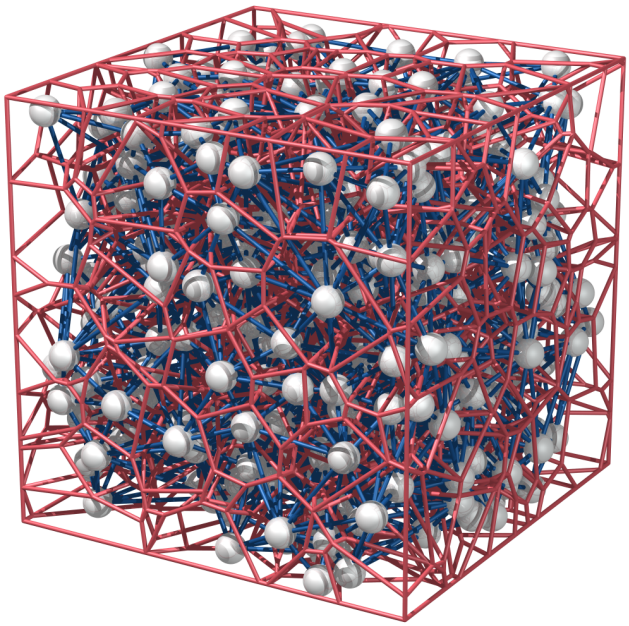

Fig. 1. A Voronoi tessellation for 400 random sites (in gray), bounded by a cube. Voronoi cell edges are shown in red, Delaunay edges in blue.

Heymann & Siebenmorgen 2012; Saftly et al. 2013) and several methods have been investigated to make them as efficient as possible (Saftly et al. 2013).

Adaptive-mesh grids seem to be an obvious choice. It is straightforward to construct an appropriate grid for any density field, whether defined by an analytical model or by a collection of smoothed particles; and it is easy to calculate a straight path through the grid, since the boundaries of the cuboidal cells are lined up with the coordinate axes and each cell has a limited number of neighbors (Saftly et al. 2013). This second point is very important in the context of RT because ray tracing and Monte Carlo RT codes determine the radiation field in each grid cell by laying down random rays (i.e., straight paths) through the domain. The simulation run time is often dominated by the portion of the code that identifies the grid cells crossed by each path and calculates the lengths of the corresponding path segments.

Adaptive-mesh grids also have drawbacks. First of all, for a given density field and required resolution, an AMR grid may not be the kind of grid with the least number of cells. To illustrate this, consider a density field defined by a set of smoothed particles. An octree grid constructed such that each cell encloses at most one particle usually has over three times more cells than there are particles; i.e., two out of three cells are empty[1]. In contrast, an unstructured grid based on a Voronoi tessellation of the spatial domain (see Sect. 2.1 and Fig. 1), using the given particles as generating sites, has exactly the same number of cells as there are particles. While not an issue in many situations, minimizing the number of cells is sometimes crucial. For example, consider a panchromatic RT simulation involving small dust grains not in local thermodynamic equilibrium (non-LTE) conditions. Because each cell stores radiation field data per wavelength bin, memory requirements are substantial. Moreover, the simulation run time is most likely dominated by the calculation of the non-LTE heating and re-emission of the dust grains in each cell. In this case, both memory usage and run time scale roughly linearly with the number of cells.

Furthermore, RT simulations frequently serve to predict the observable properties of artificial systems resulting from (magneto-)hydrodynamical (MHD) simulations (Juvela & Padoan 2003; Bethell et al. 2004; Stamatellos & Whitworth 2005; Jonsson et al. 2010; Acreman et al. 2010; Hayward et al. 2011; Robitaille 2011; Lunttila & Juvela 2012; Juvela et al. 2012). Hydrodynamical simulation codes historically employ one of two schemes: a Lagrangian formulation based on moving particles (smoothed particle hydrodynamics or SPH), for example Gadget (Springel 2005; Dolag & Stasyszyn 2009; Pakmor et al. 2012) and SEREN (Hubber et al. 2011); or a Eulerian approach based on a non-moving spatial grid, often an AMR grid, for example RAMSES (Fromang et al. 2006), Enzo (Collins et al. 2010; Bryan et al. 2013), and AMR-VAC (Keppens et al. 2012).

Recent codes including TESS (Duffell & MacFadyen 2011) and AREPO (Springel 2010, 2011) introduce a new scheme that employs a moving mesh based on a Voronoi tessellation of the spatial domain (see Sect. 2.1 and Fig. 1). This new scheme is claimed to combine the best features of SPH and the traditional Eulerian approach, and it is becoming increasingly popular. It has already been applied to various problems including the formation of stars, galaxies, and cosmological structures (Greif et al. 2011; Bauer & Springel 2012; Sijacki et al. 2012; Kereš et al. 2012; Vogelsberger et al. 2012; Torrey et al. 2012; Nelson et al. 2013; Marinacci et al. 2013). While the output from a moving mesh code can be re-gridded to an AMR grid to perform RT, the resampling process unavoidably introduces inaccuracies and represents additional overhead; it seems preferable to perform both aspects of the simulation (MHD and RT) on the same grid.

These considerations lead to the question of whether it is possible to perform accurate and efficient RT on unstructured Voronoi grids.

One approach is to approximate a straight path through the grid by a sequence of non-collinear segments connecting neighboring sites. For example in the SimpleX code (Paardekooper et al. 2010) and in the LIME code (Brinch & Hogerheijde 2010) radiation travels along the edges of the Delaunay triangulation corresponding to the Voronoi grid (see Sect. 2.1; the Delaunay edges are shown in blue in Fig. 1). While it facilitates calculating the paths, this approximation requires additional mechanisms to compensate for errors in path length (see Fig. 5 in Paardekooper et al. 2010) and direction (see Fig. 4 in Brinch & Hogerheijde 2010). The distance covered by the path inside a particular grid cell becomes a fuzzy concept, while this is an important quantity in many RT codes, e.g., for tracking the amount of energy absorbed in the cell. And finally the spread on direction makes it hard to produce high-resolution images of the simulated object.

In Sect. 2 we present instead an efficient method of calculating a straight path between two arbitrary points through a 3D Voronoi grid, applicable in any RT code based on ray tracing or Monte Carlo techniques. The path segments inside each grid cell are calculated to high precision using a straightforward algorithm that relies on the mathematical properties of Voronoi tessellations. In Sect. 3 we introduce an implementation of the method in our dust RT code SKIRT (Baes et al. 2011). We demonstrate the method's reliability, accuracy, and efficiency by comparing results obtained through the Voronoi grid with those generated by existing well-tested grids. In Sect. 4 we summarize our conclusions.

[1] To verify this claim, we ran a few tests with particles distributed uniformly over the spatial domain, and with particles representing a galaxy generated by a hydrodynamical simulation.

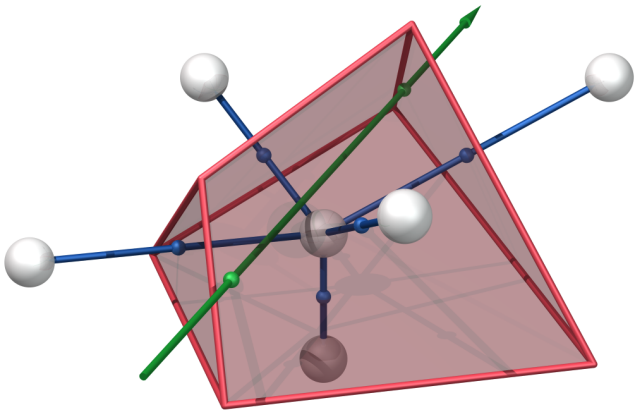

Fig. 2. A single Voronoi cell (in red) with its neighboring sites (in gray) and corresponding Delaunay edges (in blue). A straight path through the cell is shown (in green) with its intersection points with the cell boundary at entry and exit.

2. Method

2.1. Voronoi tesselations of 3D space

Given a set of points $\{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots \boldsymbol{p}_n\}$ in 3D space, called *sites*, the corresponding Voronoi tessellation (Dirichlet 1850; Voronoi 1908) is a set of cells $\{C_i\}$ where each cell C_i consists of all the points $\boldsymbol{p}$ at least as close to $\boldsymbol{p}_i$ as to any other site. The corresponding Delaunay triangulation (Delaunay 1934) is a graph created by placing a straight edge between any two sites that share a cell boundary in the Voronoi tessellation. Thus every site is connected to its nearest neighbors.

An example Voronoi tessellation is shown in Fig. 1, and a single Voronoi cell is illustrated in Fig. 2. A Voronoi cell is delimited by a convex polyhedron. A Delaunay edge, i.e., a line segment that connects two sites sharing a polygonal face, is perpendicularly bisected by the plane containing the face, although the bisection point may lie outside the face. For a set of sites chosen randomly from a uniform distribution, the number of nearest neighbors (or equivalently the number of cells sharing a face with any given cell) has an expectation value of 15.54 (van de Weygaert 1994).

To obtain an optimal grid in the context of a RT simulation, the Voronoi sites should be more densely packed (generating smaller cells) in regions where a higher resolution is desired. For example one could select the positions randomly from a probability distribution that follows the density of the RT medium, and perhaps place extra sites near sharp edges or large gradients. If the density field is defined by a set of smoothed particles, the particle locations form natural Voronoi sites. And of course if the density field is already defined on a Voronoi mesh, the original site locations can be used.

2.2. A straight path through a Voronoi grid

We consider a cuboidal spatial domain $\mathcal{D}$ defined by its corner points $(\boldsymbol{D}_{\min}, \boldsymbol{D}_{\max})$, and a set of sites $\{\, \boldsymbol{p}_m \in \mathcal{D},\, m = 1 \ldots M \,\}$. All sites are inside the domain, and the corresponding Voronoi cells are clipped by the domain walls, as illustrated in Fig. 1. Given a ray describing the path of a photon package, defined by a starting point $\boldsymbol{r_0} \in \mathcal{D}$ and a direction $\boldsymbol{k}$, our aim is to calculate the ray's consecutive intersection points with the Voronoi cell walls – or equivalently, the distance travelled in each cell – until the ray leaves the domain. This is illustrated in Fig. 2 for a single cell. The presented method can easily be adjusted for other domain geometries, or for rays that originate outside the domain.

During a setup phase, before any straight paths are calculated, the following data are prepared:

1. The domain boundaries $(\boldsymbol{D}_{\min}, \boldsymbol{D}_{\max})$.
2. The positions of the sites $\{\, \boldsymbol{p}_m,\, m = 1 \ldots M \,\}$.
3. For each site $\boldsymbol{p}_n$, the indices $\{\, m_{n,i},\, i = 1 \ldots I_n \,\}$ of all sites neighboring that site or, equivalently, of all cells neighboring the cell corresponding to that site. Domain walls are represented by special (negative) index values.

Data items (1) and (2) are externally specified as part of the problem definition. The neighbor lists (3) can easily be derived from a Voronoi tessellation or Delaunay triangulation for the specified set of sites, since nearest neighbor information is the defining characteristic of these concepts. *No information is needed on the vertices, edges or faces of the polyhedra delimiting the Voronoi cells.*

To begin calculating a straight path, the current point $\boldsymbol{r}$ is set to the starting point, and the current cell index m_r is set to the index of the cell containing the starting point. By definition of a Voronoi tessellation, finding the cell containing a given point $\boldsymbol{r}$ is equivalent to locating the site $\boldsymbol{p}_i$ nearest to $\boldsymbol{r}$. This is a straightforward operation that can easily be optimized as described later in Sect. 2.3. For the time being we assume that there is a function $C(\boldsymbol{r})$ that returns the index m of the cell containing a given point.

Once initialized, the method loops over the algorithm that computes the exit point from the current cell, i.e., the intersection of the ray formed by the current point $\boldsymbol{r}$ and the direction $\boldsymbol{k}$ with the current cell's boundary. The algorithm also produces the index of the neighboring cell without extra cost. If an exit point is found, the loop adds a path segment to the output, updates the current point and the current cell index, and continues to the next iteration. If the exit is towards a domain wall, the loop is terminated. Because of computational inaccuracies it may occur that no exit point is found. In that case, no path segment is added to the output, the current point is advanced in the direction $\boldsymbol{k}$ over an infinitesimal distance $\epsilon \ll \|\boldsymbol{D}_{\max} - \boldsymbol{D}_{\min}\|$, and the new current cell index is determined by calling the function $C(\boldsymbol{r})$.

The algorithm computing the exit point from the current cell requires the following input data: the current point $\boldsymbol{r}$; the ray's direction $\boldsymbol{k}$ as a unit vector; the index m_r of the current cell; the indices $\{\, m_i,\, i = 1 \ldots I \,\}$ of all cells neighboring the current cell, with domain walls represented by special (negative) values; the positions of the sites $\{\, \boldsymbol{p}_m,\, m = 1 \ldots M \,\}$; and the domain boundaries $(\boldsymbol{D}_{\min}, \boldsymbol{D}_{\max})$.

The line containing the ray under consideration can be written as $\mathcal{L}(s) = \boldsymbol{r} + s\,\boldsymbol{k}$ with $s \in \mathbb{R}$. The exit point can similarly be written as $\boldsymbol{q} = \boldsymbol{r} + s_q\,\boldsymbol{k}$ with $s_q > 0$, and the distance covered within the cell is given by s_q. The index of the cell next to the exit point is denoted m_q and is easily determined as follows.

1. Calculate the set of values $\{s_i\}$ for the intersection points between the line $\mathcal{L}(s)$ and the planes defined by the neighbors m_i (see below for details on this calculation).
2. Select the smallest nonnegative value $s_q = \min\{s_i | s_i > 0\}$ in the set to determine the exit point and the corresponding neighbor m_q.
3. If there is no nonnegative value in the set, no exit point has been found.

To calculate s_i in step (1) for a regular neighbor $m_i > 0$, intersect the line $\mathcal{L}(s) = \boldsymbol{r} + s\,\boldsymbol{k}$ with the plane bisecting the sites $\boldsymbol{p}(m_i)$

and $\boldsymbol{p}(m_r)$. An unnormalized vector perpendicular to this plane is given by

$$\boldsymbol{n} = \boldsymbol{p}(m_i) - \boldsymbol{p}(m_r) \tag{1}$$

and a point on the plane is given by

$$\boldsymbol{p} = \frac{\boldsymbol{p}(m_i) + \boldsymbol{p}(m_r)}{2}. \tag{2}$$

The equation of the plane can then be written as

$$\boldsymbol{n} \cdot (\boldsymbol{x} - \boldsymbol{p}) = 0. \tag{3}$$

Substituting $\boldsymbol{x} = \boldsymbol{r} + s_i\,\boldsymbol{k}$ and solving for s_i provides

$$s_i = \frac{\boldsymbol{n} \cdot (\boldsymbol{p} - \boldsymbol{r})}{\boldsymbol{n} \cdot \boldsymbol{k}}. \tag{4}$$

If $\boldsymbol{n} \cdot \boldsymbol{k} = 0$ the line and the plane are parallel so that there is no intersection, and the above equation produces $s_i = \pm\infty$. When using standard IEEE 754 floating point arithmetic there is no reason to test for this special case, since the infinite value will never be selected as the exit point in step (2).

To calculate s_i in step (1) for a domain wall $m_i < 0$, substitute the appropriate normal and position vectors for the wall plane in Eq. (4). For example, for the left wall one has $\boldsymbol{n} = (-1, 0, 0)$ and $\boldsymbol{p} = (D_{\mathrm{min,x}}, 0, 0)$ so that

$$s_i = \frac{D_{\mathrm{min,x}} - r_{\mathrm{x}}}{k_{\mathrm{x}}}. \tag{5}$$

In an actual implementation of this algorithm there is no need to accumulate the complete set of $\{s_i\}$ values; one can simply keep track of the smallest nonnegative value. As a further optimization, part of the intersection calculation can be avoided for about half of the planes by noting that the sign of $\boldsymbol{n} \cdot \boldsymbol{k}$ determines the sign of s_i in Eq. (4). Indeed, the site $\boldsymbol{p}(m_r)$ and the current point $\boldsymbol{r}$ are on the same side of the plane defined by Eq. (3) (unless $\boldsymbol{r}$ lies *in* the plane), so that the numerator of Eq. (4) is always positive (or zero if $\boldsymbol{r}$ lies in the plane).

2.3. Finding the cell containing a given point

We now return to the implementation of the function $C(\boldsymbol{r})$ that identifies the Voronoi cell containing a given query point. By definition of a Voronoi tessellation, this operation is equivalent to finding the site closest to the query point. There are many sophisticated ways to accelerate this nearest neighbor search, for example by building a k-d tree (Friedman et al. 1977) or an R-tree (Guttman 1984) data structure. We chose to use a simple mechanism, since this function is usually invoked only once per path and thus its performance is not overly critical.

We assume the domain $\mathcal{D}$ is partitioned in a set of cuboidal *blocks* $\{\mathcal{B}_k,\ k = 1 \ldots K\}$ according to a regular linear grid. During the setup phase described in the beginning of Sect. 2.2, an additional data structure is constructed containing, for each block $\mathcal{B}_k$ in the partition of the domain $\mathcal{D}$, the indices $\{m_{k,j},\ j = 1 \ldots J_k\}$ of all Voronoi cells that possibly overlap that block. Determining these lists in principle requires an intersection test between each block and each cell. In practice it suffices to consider the cell's bounding box, which can be easily intersected with the blocks.

One might be tempted to derive a Voronoi cell's bounding box from the positions of the neighboring sites; however, the convex hull of a cell's neighboring sites does not necessarily fully enclose the cell. Because a Voronoi cell is convex, its bounding box can be easily calculated from the list of its vertices. This requires fully constructing the cell; however, this is needed anyway to calculate the cell volume for use in other areas of the RT simulation (e.g., determining the specific energy absorbed per unit mass by the medium in the cell). Regardless, the cell geometry is needed solely during setup and does not have to be retained thereafter.

The function $C(\boldsymbol{r})$ receives the following input data: the query point $\boldsymbol{r}$; for each block $\mathcal{B}_k$, the indices $\{m_{k,j},\ j = 1 \ldots J_k\}$ of all Voronoi cells that possibly overlap that block; the positions of the sites $\{\boldsymbol{p}_m,\ m = 1 \ldots M\}$; and the domain boundaries $(\boldsymbol{D}_{\mathrm{min}}, \boldsymbol{D}_{\mathrm{max}})$.

For each query, the function $C(\boldsymbol{r})$ performs these steps:

1. Verify that the query point is inside the domain; if not return a special (negative) index value.
2. Locate the block containing the query point; this is trivial since blocks are on a regular linear grid.
3. Retrieve the list of cells possibly overlapping that block, and thus possibly containing the query point.
4. Calculate the squared distance from the query point to the sites for each of these cells. By definition of a Voronoi tessellation, the closest site determines the Voronoi cell containing the query point.

3. Tests, results, and discussion

3.1. Implementation

The SKIRT code (Baes et al. 2011) performs 3D continuum RT using the Monte Carlo technique. It is used for studying dusty astrophysical objects including spiral galaxies (De Looze et al. 2012a,b; De Geyter et al. 2013) and active galactic nuclei (Stalevski et al. 2012, 2013). Input models can be defined through a range of built-in geometries, or imported from the results of a MHD simulation. The SKIRT code also offers various dust grid options, including regular grids and adaptive grids (Saftly et al. 2013).

We implemented a Voronoi dust grid in SKIRT according to the method presented in Sect. 2. This allowed us to use the built-in geometries for creating synthetic test models, and to compare the results with those produced by the existing and well-tested grids.

We employed the open source library Voro++ (Rycroft 2009) to set up the input data described in Sects. 2.2 and 2.3. The library and its data structures are used only during setup. All relevant information is extracted and stored in our own data structures for reference after setup.

3.2. Test models

We tested the Voronoi dust grid with two synthetic models of our own making, called *torus* and *spiral*, and we ran the RT benchmark described by Pascucci et al. (2004). We first present the results for our models, and in Sect. 3.5 we discuss the results for the Pascucci benchmark.

The *torus* model consists of a central light source surrounded by an axisymmetric dusty torus, as might be present in the center of active galactic nuclei. The dust geometry is described by a radial power-law density from a given inner to outer radius, with an opening angle of 50 degrees. A cut through the dust distribution is shown in the top row of Fig. 4. The sites for the Voronoi dust grid are selected randomly from a uniform distribution over the cuboidal domain enclosing the torus. Since the model is axisymmetric, we can compare the results of the Voronoi grid with those

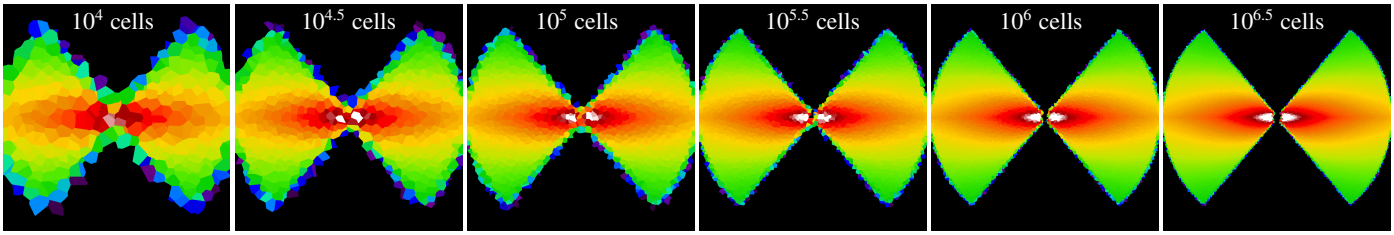

Fig. 3. A cut through the dust density distribution of the torus model, discretized on Voronoi grids with a resolution varying from 10^4 cells (*left*) to $10^{6.5}$ cells (*right*). All grids were constructed from a set of uniformly distributed sites.

produced by a regular two-dimensional (2D) cylindrical grid, in addition to those produced by an adaptive (3D) octree grid. In Fig. 3 we illustrate the effect of the number of Voronoi grid cells for the torus model.

The *spiral* model represents an idealized spiral galaxy with three arms, similar to the spiral model presented in Saftly et al. (2013). The stellar distribution includes a flattened Sérsic bulge and a double-exponential disk with a spiral arm perturbation. The dust is distributed in a thinner, similarly perturbed double-exponential disk. Cuts through the dust distribution are shown in the top row of Figs. 5 and 6. In this case, the sites for the Voronoi dust grid are selected randomly from the dust distribution, as opposed to a uniform distribution. Areas with a higher dust density are thus, on average, covered with smaller cells.

3.3. Test grids

For the torus model we ran simulations with three different dust grids: a regular 2D cylindrical grid with $250^2 = 62\,500$ cells; an adaptive octree grid with ≈950 000 cells; and a Voronoi grid with about the same number of uniformly distributed cells. The top row of Fig. 4 shows a cut through the gridded dust density distribution for each of these grids. The cylindrical grid captures the sharp edges of the model perfectly, because the cylindrical coordinate axes are lined up with the edges. The octree grid does a fine job as well because of its adaptive nature: smaller cells are automatically created along the sharp edges. The Voronoi grid does not do particularly well at the edges because of the random placement of its cells. This would not be an issue when importing a grid from a moving mesh code, because the cell sizes would already be properly adjusted to the underlying gradients.

For the spiral model we ran simulations with two different dust grids: an adaptive octree grid with ≈1 350 000 cells; and a Voronoi grid with about the same number of cells, placed using a weighed distribution according to the dust density (smaller cells in higher density areas). The top rows of Figs. 5 and 6 show a cut through the gridded dust density distribution for each of these grids. The differences between the grids are most easily seen in the lower density areas.

Although this study does not focus on grid quality, we still need to ensure that our Voronoi grid implementation properly represents the theoretical dust densities defined by the synthetic models. To obtain an objective quality measure, we sample the theoretical dust density ρ_t and the gridded dust density ρ_g at a large number of random points uniformly distributed over the domain. We use the standard deviation of the difference $\rho_t - \rho_g$ as a measure for how well the grid reflects the theoretical density distribution. Table 1 lists the resulting numbers for the various grids and models. For each model the value for the octree grid is normalized to unity.

Taking into account our naive cell placement, the Voronoi grid compares well with the highly tuned adaptive octree grid, thus verifying this aspect of our implementation.

3.4. Results

Shooting photon packages through the grid is the most important test in the context of this study.

The middle row of Fig. 4 shows the dust temperature calculated by a panchromatic simulation for the torus model, using the three grids describe above. All quantities, including the radiation field and the amount of dust absorption, are discretized on the same grid as the dust density. In each simulation, the central light source emits 10^5 photon packages for each of 100 wavelength bins on a logarithmic grid. Scattering events cause additional photon packages to be created, which is particularly relevant for this model because of the high optical depth of the torus. In the end, each simulation traces about 700 million photon packages through the dust grid.

The bottom rows of Figs. 4–6 show the flux density calculated by a monochromatic simulation for each model and grid combination. The Poisson noise is caused by the statistical nature of the Monte Carlo technique. In each simulation, the light sources emit 10 million photon packages at a fixed wavelength, and scattering events again cause additional photon packages to be created.

Other than the effects of grid resolution and the unavoidable noise, the calculated temperature and flux density maps are the same for the various grids. In particular, as noted in Sect. 3.3, the Voronoi grid does not resolve the central area of the dust distribution as well as the other grids, causing some deviation in the central area of the calculated flux density field. This effect is ultimately due to the naive placement of the Voronoi cells in our tests, and would not be present for a properly adjusted grid.

These results validate the accuracy of our straight path calculation method for Voronoi grids.

Table 2 provides an indication of the processing time spent per cell crossing for each simulation. To obtain these numbers, the elapsed time for the photon shooting phase of a simulation is divided by the number of grid cells crossed during that phase. The result thus includes some overhead for generating the random paths and for storing results, in addition to the grid traversal calculation itself. The tests were performed on a typical desktop computer using a single core. The last column lists the ratio between the cell crossing times for the Voronoi and octree grids. The Voronoi grid performs roughly three times slower than our highly optimized octree implementation (which maintains, for example, a neighbor list for each cell to accelerate the process of finding the next cell on a path). This seems surprisingly fast in view of the high geometric complexity of a Voronoi grid (illustrated in Figs. 1 and 2) compared to the cuboidal cells in an octree. Moreover, as noted in the introduction, an octree grid

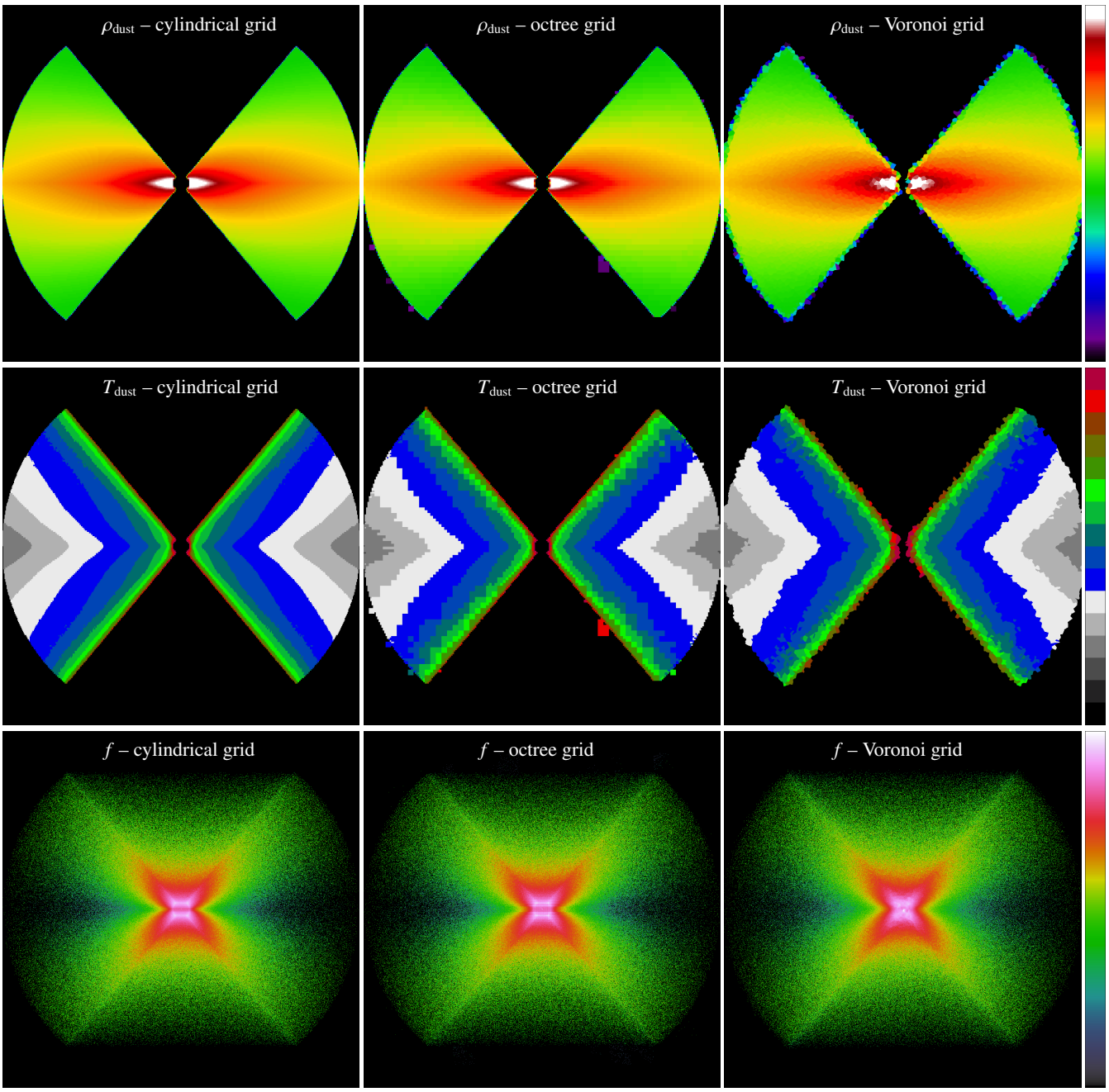

Fig. 4. Illustration of the results for the *torus* model with three different dust grids. *Rows – top*: the dust density distribution (cut through the central edge-on plane); *middle*: the calculated dust temperature (cut through the central edge-on plane); *bottom*: the calculated flux density escaping from the model (edge-on view). *Columns – left*: regular 2D cylindrical grid with $250^2 = 62\,500$ cells; *middle*: adaptive octree grid with ≈950 000 cells; *right*: Voronoi grid with ≈950 000 uniformly distributed cells.

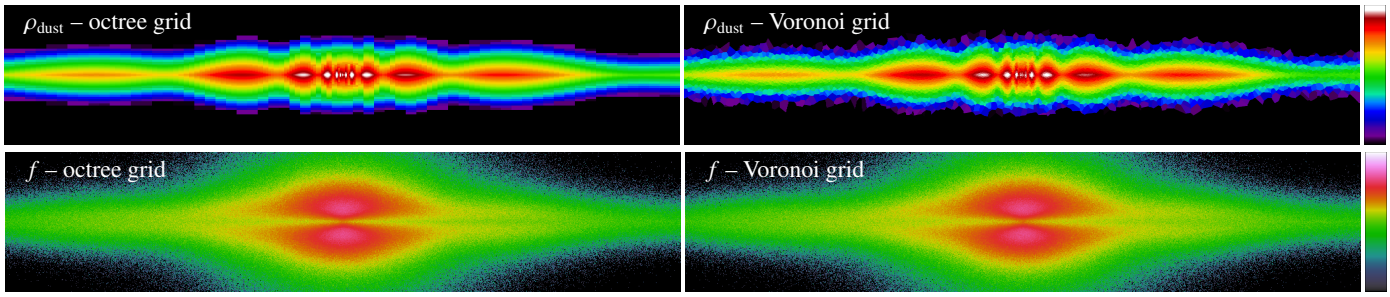

Fig. 5. Illustration of the results for the *spiral* model, edge-on view. *Rows – top*: the dust density distribution (cut through the central edge-on plane); *bottom*: the calculated flux density escaping from the model (edge-on view). *Columns – left*: adaptive octree grid with ≈1 350 000 cells; *right*: Voronoi grid with ≈1 350 000 cells with a non-uniform, weighed distribution.

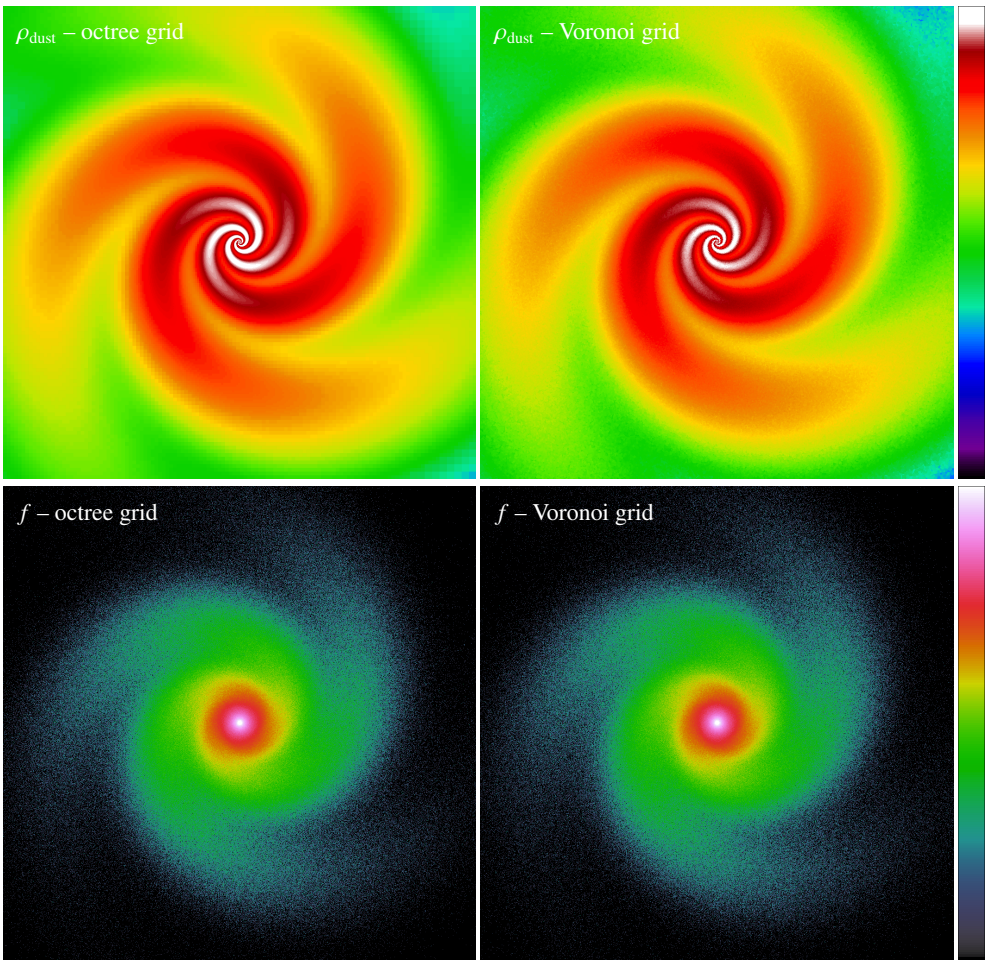

Fig. 6. Illustration of the results for the *spiral* model, face-on view. *Rows* – *top*: the dust density distribution (cut through the central face-on plane); *bottom*: the calculated flux density escaping from the model (face-on view). *Columns* – *left*: adaptive octree grid with ≈1 350 000 cells; *right*: Voronoi grid with ≈1 350 000 cells with a non-uniform, weighed distribution.

Table 1. Grid quality.

Model	Cylindrical	Octree	Voronoi
Torus	0.82	1	1.75
Spiral	–	1	1.68

Notes. The difference between the theoretical and gridded dust density is sampled at a large number of random points, uniformly distributed over the domain. The standard deviation on this difference is used as a quality measure for the grid. In the table, the value for the octree grid is normalized to unity for each model. Smaller numbers indicate better quality.

Table 2. Run time.

		Time per cell crossing (ns)		
Model	Simulation type	Octree	Voronoi	Vor./Oct.
Torus	monochromatic	219	693	3.2
Torus	panchromatic	400	1006	2.5
Spiral	monochromatic	309	903	2.9
Spiral	panchromatic	442	1095	2.5

Notes. The elapsed time for the photon shooting phase of a simulation is divided by the number of grid cells crossed during that phase. The result is an indication of the time spent per cell crossing, including grid traversal calculations and some overhead for generating the random paths and for storing results. The tests were performed on a typical desktop computer using a single core. The last column lists the ratio between the run times for the Voronoi and octree grids. Larger numbers indicate slower performance.

may need many more cells than the Voronoi grid to represent a particular density field, further balancing performance in favor of the Voronoi grid.

As discussed in Sect. 2.2, the cell crossing algorithm may occasionally fail to find an exit point because of computational inaccuracies. In our tests this occurred at most once per 50 million cell crossings, so this issue does not affect the algorithm's performance.

3.5. The Pascucci benchmark

The *Pascucci* benchmark (Pascucci et al. 2004) models a star embedded in a circumstellar disk with an inner cavity free of dust, prescribing an analytical 2D distribution and a set of optical depths and viewing angles. A cut through the central edge-on plane of the dust density distribution is shown in the left panel of Fig. 7.

We ran panchromatic simulations for this model with optical depths $\tau = 0.1, 1$, and 10 using a 3D Voronoi grid consisting of one million cells randomly placed according to a $1/r$ distribution. This distribution serves to properly resolve the intense radiation field in the center of the model. In each simulation, the central light source emits 10^5 photon packages for each of 150 wavelength bins on a logarithmic grid.

The two rightmost panels of Fig. 7 compare the spectral energy distribution (SED) produced by our SKIRT simulations

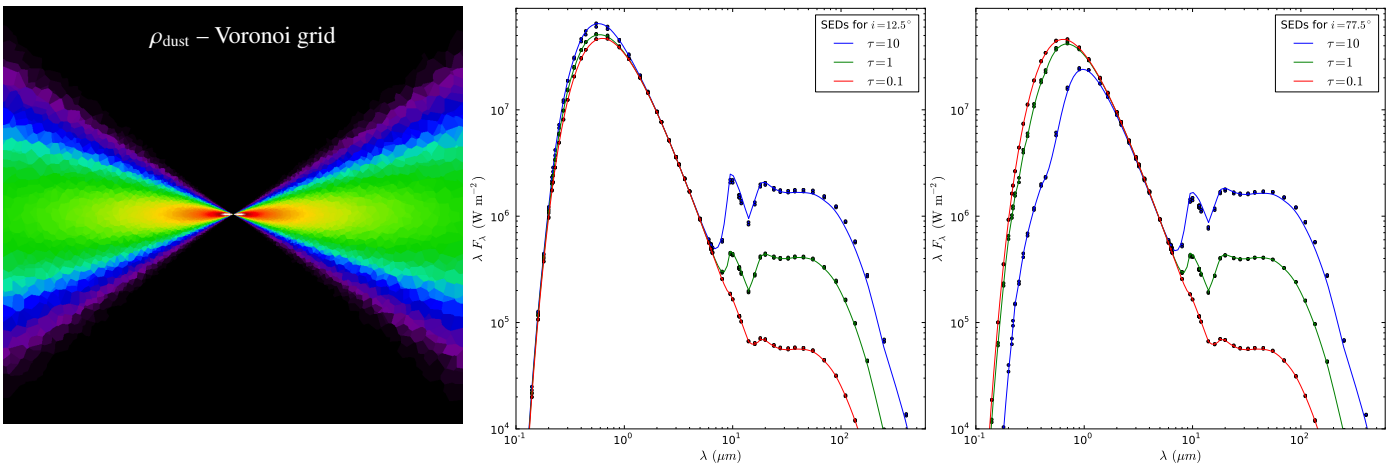

Fig. 7. Illustration of the results for the *Pascucci* benchmark (Pascucci et al. 2004). The *left panel* shows a cut through the central edge-on plane of the dust density distribution discretized on a 3D Voronoi grid with one million cells randomly placed according to a $1/r$ distribution. The other panels show the simulated spectral energy distribution (SED) for disk inclinations equal to 12.5° (*center*) and 77.5° (*right*), for optical depths $\tau = 0.1, 1$, and 10. Dots indicate benchmark reference points; solid lines represent our simulation results using the 3D Voronoi grid shown in the *left panel*.

with the corresponding benchmark results published in Pascucci et al. (2004). The center panel shows the SEDs for the various optical depths at a nearly face-on disk inclination of 12.5°, the right panel at a nearly edge-on inclination of 77.5°. Dots indicate benchmark reference points; solid lines represent our simulation results.

For higher optical depths our simulation results deviate slightly because the SKIRT code is not optimized for operation in this regime; running the benchmark with a 2D axially symmetric logarithmic grid results in the same deviation (not shown). These results further validate our method for calculating straight paths through a Voronoi grid.

3.6. Applicability

From Sect. 2 it follows that the presented method requires as input data solely the coordinates of the Voronoi sites, plus any relevant physical properties (such as mass densities) for the cell surrounding each site. In other words, the interface between the input model and the RT code is very thin, opening up a wide range of possibilities. An input model can be defined by SPH particles, serving as Voronoi sites; or by a Voronoi mesh produced by an MHD code; or by appropriately distributed random points generated from (semi)-analytical density or opacity fields, similar to the approach in Paardekooper et al. (2010), for example.

We also note in Sects. 2.2 and 2.3 that the path calculation algorithm itself requires no information on a Voronoi cell other than its bounding box and the locations of its own site and all neighboring sites. The required data structures can be easily built from the input data using a publicly available Voronoi library. The library code is invoked only during the initialization phase, minimizing its impact on performance and robustness, and allowing it to be easily replaced by another code if the need arises. For example, while we are happy with the Voro++ library's ease of use and with its performance during the tests, we may in the future consider using a parallelized method (Lo 2012; Springel 2011).

As a consequence, the presented method allows the RT code to support a Voronoi grid while remaining uncoupled from the code producing the input model. This decreases the complexity of the interface, and allows cooperation even when source code is not publicly available. In contrast to this approach, for example, the Sunrise RT code (Jonsson 2006) directly invokes parts of the non-public Arepo moving mesh code (Springel 2011) to implement the interface[2].

4. Conclusions

The choice of an appropriate discretization is crucial in any numerical simulation code. Because of the large dynamic range of the physical quantities, in most problems the resolution of the grid must scale with the field densities or gradients. Adaptive grids with cuboidal cells, such as octrees or more generally AMR grids, have proven very popular in part because of their relative ease of implementation. However, several recent codes have adopted unstructured grids based on Voronoi tessellations, or equivalently, Delaunay triangulations. These grids tend to more closely reflect dynamic ranges in the model with fewer cells, presenting cell boundaries that are more adjusted to the underlying gradients. Since a Voronoi grid is defined solely by its generating points, the cell size and distribution can be easily fine-tuned by placing these sites in the appropriate locations.

In a RT simulation the Voronoi grid can be a very flexible tool. Appropriate sites can be generated randomly, distributed according to the input model's density or opacity fields; if needed extra sites can be added in high-gradient areas. In the case of a particle-based input model, the particle locations themselves can serve as sites; and for an input model already based on a Voronoi mesh no re-gridding is required at all.

In this work we have shown that it is straightforward to implement accurate and efficient RT on Voronoi grids. In spite of the geometric complexity of the cell boundaries, calculating straight paths between two arbitrary points through a 3D Voronoi grid is only about three times slower than a highly optimized octree implementation with the same number of cells, while in practice the total number of Voronoi grid cells may be lower for an equally good representation of the density field. The presented method automatically yields the precise distance covered

[2] http://code.google.com/p/sunrise/wiki/RunningWithArepo

by the path inside each grid cell, and eliminates the need for approximate corrections or work-arounds required by alternate approaches where the radiation travels only along the Delaunay edges. The method requires only a thin interface with the input model and with the actual construction of the grid, allowing codes to remain largely uncoupled and enabling the use of a publicly available Voronoi library.

While we implemented and tested the method in our continuum RT code SKIRT, focusing on the effects of dust, it is widely applicable to all RT codes using ray tracing or Monte Carlo techniques.

We conclude that the benefits of using a Voronoi grid in RT simulation codes will often outweigh the somewhat slower performance.

Acknowledgements. This work fits in the CHARM framework (Contemporary physical challenges in Heliospheric and AstRophysical Models), a phase VII Interuniversity Attraction Pole (IAP) programme organised by BELSPO, the BELgian federal Science Policy Office. W.S. acknowledges the support of Al-Baath University and The Ministry of High Education in Syria in the form of a research grant.

References

Abdikamalov, E., Burrows, A., Ott, C. D., et al. 2012, ApJ, 755, 111
Acreman, D. M., Harries, T. J., & Rundle, D. A. 2010, MNRAS, 403, 1143
Baes, M., & Dejonghe, H. 2001, MNRAS, 326, 733
Baes, M., & Dejonghe, H. 2002, MNRAS, 335, 441
Baes, M., Davies, J. I., Dejonghe, H., et al. 2003, MNRAS, 343, 1081
Baes, M., Verstappen, J., De Looze, I., et al. 2011, ApJS, 196, 22
Bauer, A., & Springel, V. 2012, MNRAS, 423, 2558
Bethell, T. J., Zweibel, E. G., Heitsch, F., & Mathis, J. S. 2004, ApJ, 610, 801
Bianchi, S. 2008, A&A, 490, 461
Bianchi, S., Ferrara, A., Davies, J. I., & Alton, P. B. 2000, MNRAS, 311, 601
Brinch, C., & Hogerheijde, M. R. 2010, A&A, 523, A25
Bryan, G. L., Norman, M. L., et al. (The Enzo Collaboration) 2013, ApJS, submitted [arXiv:1307.2265]
Chakrabarti, S., Cox, T. J., Hernquist, L., et al. 2007, ApJ, 658, 840
Ciardi, B., Ferrara, A., Marri, S., & Raimondo, G. 2001, MNRAS, 324, 381
Code, A. D., & Whitney, B. A. 1995, ApJ, 441, 400
Collins, D. C., Xu, H., Norman, M. L., Li, H., & Li, S. 2010, ApJS, 186, 308
De Geyter, G., Baes, M., Fritz, J., & Camps, P. 2013, A&A, 550, A74
De Looze, I., Baes, M., Bendo, G. J., et al. 2012a, MNRAS, 427, 2797
De Looze, I., Baes, M., Fritz, J., & Verstappen, J. 2012b, MNRAS, 419, 895
Decin, L., Cox, N. L. J., Royer, P., et al. 2012, A&A, 548, A113
Delaunay, B. 1934, Classe des Sciences Mathématiques et Naturelles, 7, 793
Dirichlet, L. 1850, Journal für die reine und angewandte Mathematik, 40, 209
Disney, M., Davies, J., & Phillipps, S. 1989, MNRAS, 239, 939
Dolag, K., & Stasyszyn, F. 2009, MNRAS, 398, 1678
Doty, S. D., Metzler, R. A., & Palotti, M. L. 2005, MNRAS, 362, 737
Duffell, P. C., & MacFadyen, A. I. 2011, ApJS, 197, 15
Ercolano, B., Barlow, M. J., & Storey, P. J. 2005, MNRAS, 362, 1038
Fallscheer, C., Reid, M. A., Di Francesco, J., et al. 2013, ApJ, 773, 102
Friedman, J. H., Bentley, J. L., & Finkel, R. A. 1977, ACM Trans. Math. Softw., 3, 209
Fritz, J., Gentile, G., Smith, M. W. L., et al. 2012, A&A, 546, A34
Fromang, S., Hennebelle, P., & Teyssier, R. 2006, A&A, 457, 371
Goldsmith, P. F., Heyer, M., Narayanan, G., et al. 2008, ApJ, 680, 428
Goosmann, R. W., & Gaskell, C. M. 2007, A&A, 465, 129
Gordon, K. D., Misselt, K. A., Witt, A. N., & Clayton, G. C. 2001, ApJ, 551, 269
Greif, T. H., Springel, V., White, S. D. M., et al. 2011, ApJ, 737, 75
Guttman, A. 1984, SIGMOD Rec., 14, 47
Harries, T. J., Monnier, J. D., Symington, N. H., & Kurosawa, R. 2004, MNRAS, 350, 565
Hayward, C. C., Kereš, D., Jonsson, P., et al. 2011, ApJ, 743, 159
Heymann, F., & Siebenmorgen, R. 2012, ApJ, 751, 27
Hubber, D. A., Batty, C. P., McLeod, A., & Whitworth, A. P. 2011, A&A, 529, A27
Indebetouw, R., Whitney, B. A., Johnson, K. E., & Wood, K. 2006, ApJ, 636, 362
Jonsson, P. 2006, MNRAS, 372, 2
Jonsson, P., Groves, B. A., & Cox, T. J. 2010, MNRAS, 403, 17
Juvela, M., & Padoan, P. 2003, A&A, 397, 201
Juvela, M., Malinen, J., & Lunttila, T. 2012, A&A, 544, A141
Keppens, R., Meliani, Z., van Marle, A., et al. 2012, J. Comput. Phys., 231, 718
Kereš, D., Vogelsberger, M., Sijacki, D., Springel, V., & Hernquist, L. 2012, MNRAS, 425, 2027
Kurosawa, R., & Hillier, D. J. 2001, A&A, 379, 336
Laursen, P., Razoumov, A. O., & Sommer-Larsen, J. 2009, ApJ, 696, 853
Lo, S. 2012, Comput. Meth. Appl. Mech. Eng., 237 88
Lunttila, T., & Juvela, M. 2012, A&A, 544, A52
Marinacci, F., Pakmor, R., & Springel, V. 2013, MNRAS, accepted [arXiv:1305.5360]
Matthews, L. D., & Wood, K. 2001, ApJ, 548, 150
Misiriotis, A., Kylafis, N. D., Papamastorakis, J., & Xilouris, E. M. 2000, A&A, 353, 117
Nelson, D., Vogelsberger, M., Genel, S., et al. 2013, MNRAS, 429, 3353
Niccolini, G., & Alcolea, J. 2006, A&A, 456, 1
Paardekooper, J.-P., Kruip, C. J. H., & Icke, V. 2010, A&A, 515, A79
Pakmor, R., Edelmann, P., Röpke, F. K., & Hillebrandt, W. 2012, MNRAS, 424, 2222
Paron, S., Weidmann, W., Ortega, M. E., Albacete Colombo, J. F., & Pichel, A. 2013, MNRAS, 433, 1619
Pascucci, I., Wolf, S., Steinacker, J., et al. 2004, A&A, 417, 793
Pelkonen, V.-M., Juvela, M., & Padoan, P. 2009, A&A, 502, 833
Pinte, C., Ménard, F., Duchêne, G., & Bastien, P. 2006, A&A, 459, 797
Robitaille, T. P. 2011, A&A, 536, A79
Rycroft, C. H. 2009, Chaos, 19, 041111
Saftly, W., Camps, P., Baes, M., et al. 2013, A&A, 554, A10
Schartmann, M., Meisenheimer, K., Camenzind, M., et al. 2008, A&A, 482, 67
Schechtman-Rook, A., Bershady, M. A., & Wood, K. 2012, ApJ, 746, 70
Sijacki, D., Vogelsberger, M., Kereš, D., Springel, V., & Hernquist, L. 2012, MNRAS, 424, 2999
Springel, V. 2005, MNRAS, 364, 1105
Springel, V. 2010, MNRAS, 401, 791
Springel, V. 2011 [arXiv:1109.2218]
Stalevski, M., Fritz, J., Baes, M., Nakos, T., & Popović, L. Č. 2012, MNRAS, 420, 2756
Stalevski, M., Fritz, J., Baes, M., & Popovic, L. C. 2013 [arXiv:1301.4244]
Stamatellos, D., & Whitworth, A. P. 2003, A&A, 407, 941
Stamatellos, D., & Whitworth, A. P. 2005, A&A, 439, 153
Steinacker, J., Bacmann, A., & Henning, T. 2002, J. Quant. Spectr. Rad. Transf., 75, 765
Steinacker, J., Bacmann, A., Henning, T., Klessen, R., & Stickel, M. 2005, A&A, 434, 167
Steinacker, J., Bacmann, A., & Henning, T. 2006, ApJ, 645, 920
Steinacker, J., Baes, M., & Gordon, K. D. 2013, ARA&A, 51, 63
Tasitsiomi, A. 2006, ApJ, 645, 792
Torrey, P., Vogelsberger, M., Sijacki, D., Springel, V., & Hernquist, L. 2012, MNRAS, 427, 2224
van de Weygaert, R. 1994, A&A, 283, 361
Verhamme, A., Schaerer, D., & Maselli, A. 2006, A&A, 460, 397
Vogelsberger, M., Sijacki, D., Kereš, D., Springel, V., & Hernquist, L. 2012, MNRAS, 425, 3024
Voronoi, G. 1908, Journal für die reine und angewandte Mathematik, 134, 198
Wang, Z., Kaplan, D. L., Slane, P., Morrell, N., & Kaspi, V. M. 2013, ApJ, 769, 122
Witt, A. N., & Gordon, K. D. 1996, ApJ, 463, 681
Witt, A. N., & Gordon, K. D. 2000, ApJ, 528, 799
Witt, A. N., Thronson, Jr., H. A., & Capuano, Jr., J. M. 1992, ApJ, 393, 611
Wolf, S. 2003, Comput. Phys. Commun., 150, 99
Wolf, S., Fischer, O., & Pfau, W. 1998, A&A, 340, 103
Wood, K., Mathis, J. S., & Ercolano, B. 2004, MNRAS, 348, 1337