Hunger, L., Cosenza, B., Kimeswenger, S., & Fahringer, T.

# Spectral turning bands for efficient Gaussian random fields generation on GPUs and accelerators

**Journal article | Accepted manuscript (Postprint)**
This version is available at https://doi.org/10.14279/depositonce-7092

This is the peer reviewed version of the following article:

Hunger, L., Cosenza, B., Kimeswenger, S., & Fahringer, T. (2015). Spectral turning bands for efficient Gaussian random fields generation on GPUs and accelerators. Concurrency and Computation: Practice and Experience, 27(16), 4122–4136. https://doi.org/10.1002/cpe.3550

which has been published in final form at https://doi.org/10.1002/cpe.3550. This article may be used for non-commercial purposes in accordance with Wiley Terms and Conditions for Use of Self-Archived Versions.

**Terms of Use**
Copyright applies. A non-exclusive, non-transferable and limited right to use is granted. This document is intended solely for personal, non-commercial use.

WISSEN IM ZENTRUM UNIVERSITÄTSBIBLIOTHEK

Technische Universität Berlin

# Spectral turning bands for efficient Gaussian random fields generation on GPUs and accelerators

Lars Hunger[1,2,*], Biagio Cosenza[3,4], Stefan Kimeswenger[2,5], Thomas Fahringer[4]

[1] BrainLinks-Braintools, University of Freiburg, Germany
[2] Institute for Astro- and Particle Physics, University of Innsbruck, Austria
[3] Department of Computer Engineering and Microelectronics, Technische Universität Berlin, Germany
[4] Institute of Computer Science, University of Innsbruck, Austria
[5] Instituto de Astronomía, Universidad Católica del Norte Antofagasta, Chile

* Corresponding Author: Lars Hunger, BrainLinks-BrainTools, Univeristy of Freiburg, Germany

lars.hunger@blbt.uni-freiburg.de

**Abstract**

A random field (RF) is a set of correlated random variables associated with different spatial locations. RF generation algorithms are of crucial importance for many scientific areas, such as astrophysics, geostatistics, computer graphics, and many others. Current approaches commonly make use of 3D fast Fourier transform (FFT), which does not scale well for RF bigger than the available memory; they are also limited to regular rectilinear meshes.

We introduce random field generation with the turning band method (RAFT), an RF generation algorithm based on the turning band method that is optimized for massively parallel hardware such as GPUs and accelerators. Our algorithm replaces the 3D FFT with a lower-order, one-dimensional FFT followed by a projection step and is further optimized with loop unrolling and blocking. RAFT can easily generate RF on non-regular (non-uniform) meshes and efficiently produce fields with mesh sizes bigger than the available device memory by using a streaming, out-of-core approach. Our algorithm generates RF with the correct statistical behavior and is tested on a variety of modern hardware, such as NVIDIA Tesla, AMD FirePro and Intel Phi. RAFT is faster than the traditional methods on regular meshes and has been successfully applied to two real case scenarios: planetary nebulae and cosmological simulations.

**Keywords**

GPU; random field; turning band; FFT; astrophysics; non-uniform mesh; non-regular mesh; GPGPU; spectral methods

## 1. INTRODUCTION

A random field (RF) is a spatial distribution of correlated random values.

An instructive example for the generation of a two-dimensional RF is the simulation of the distri-bution of freshly fallen snow on a street. Assume that the street is a 2D grid; we want to generate the snowfall heights for each grid cell. If we decide to choose the snow height independently at random for each grid cell, we will end up with a snowfall height map that does not look like the snow distri-bution on a real street. However, to create a more realistic surface, two cells cannot be independently generated, as cells that are close by will have no correlation with each other; for example, two close cells may have a very high height difference, therefore creating an unrealistic, non-smoothed sur-face. We introduce a relation that makes close by points no longer independent; this relation is called a **correlation function** and it describes how the values of RF points behave depending on their

relative position to each other. In the snowfall example, we would choose a correlation function with a certain range so that points close to each other have similar values, but long-distance points do not influence each other. It means that the snowfall height on one end of the street does not depend on the height of snowfall on the other end (i.e. the mean snowfall height of the entire street is the same).

The value of the correlation of two points will always be $-1 \leqslant C \leqslant 1$. In this case, 1 means the two points will have a value described by a linear dependence of the form $x = d * y$ with $x$ and $y$ being the correlated points and a parameter $d$, while $-1$ indicates an anti-correlation. If we look at fluctuations around a mean of, for example, 1 and for simplicity, $d = 1$, then if one point has a value of 1.5, another point with $C = 1$ will also have a value of 1.5 while a point with $C = -1$ will have a value of 0.5. Here, correlations with a value close to 1 signify that the two points have similar values, and a correlation of 0 means that points are uncorrelated.

In our snow fall example, we can assume that the correlation has a linear dependence with a factor $d = 1$, meaning that points with a high correlation will also have similar values. The range of the chosen correlation function will also control how large clusters of similar values will be. A short correlation range would mean that a lot of small snow mounds are generated, where a long range correlation would mean that a few quite large snow mounds would be generated. An RF can also be described by the size distribution of these clusters. This way of describing a RF is called a **power spectrum**. The size of a structure is commonly identified with a corresponding frequency, where larger frequencies mean smaller structures. The higher the power at higher frequencies, the larger is the amount of small structures in the RF. According to the Wiener–Khinchin theorem [1], the power spectrum and correlation function are equivalent descriptions of a RF and can be transformed into each other via a Fourier transformation.

The snow fall example was an example of a two-dimensional RF, but an RF can have higher dimensionality. The distributions of temperature fluctuations in a swimming pool is a good example of a 3D RF.

Random field generation algorithms are of crucial importance for many scientific areas. They are widely used in computational physics; here, they are used for the generation of initial conditions for cosmological structure formation simulations such as the Millennium simulation [2], to create winds in planetary nebulae simulations (Section 7) and for the initialization of $N$-body simulations [3]. In simulations that use a turbulence-driving technique [4], an RF has to be generated in each time-step of the Magneto-hydrodynamical simulation. RFs are also often used in geo-statistical research [5] for creating topological maps or for the estimation of the yield of geological reservoirs. In other words, RFs are used when only the statistical properties of a scalar field are known and distinct realizations have to be generated.

Even though our and the traditional fast Fourier transform (FFT) approach generalize easily to higher dimensions, we focus on three-dimensional (3D) RF in this article.

Traditional approaches to compute 3D RFs make extensive use of 3D FFT. These 3D FFT-based methods are limited to regular meshes for generating random fields.

In this article, we introduce turning band random fields (RAFT)*, a new random field generation implementation based on the turning band (TB) method that has been highly optimized to run on GPUs and accelerators. The proposed algorithm replaces the 3D FFT used in a traditional approach with a two step approach: a faster, lower-dimensional FFT to generate lines, which uses a smaller set of points with respect to the traditional approach, and a multidimensional projection step, where all of the lines affect each mesh point of the random field. TB RF generators are not commonly used for generating large RFs because, on the CPU, they are much slower than a traditional 3D FFT approach. TB methods on the CPU are slower because each grid point is affected by all of the lines, while in the 3D FFT approach, the field is generated in one pass. In this work, we demonstrate that TB methods can be highly optimized for GPUs and accelerators and allow the out-of-core generation of RF on regular and non-regular meshes.

This article extends our seminal work [6] with a new section on tests of the statistical behavior, which shows the correctness of the generated RFs, and a larger set of tested hardware architectures, including two GPUs and an accelerator.

*A former version with a different name appeared in [6].

Our contributions are as follows:

- RAFT, a TB-based RF generation algorithm optimized for GPUs exploiting loop blocking and unrolling;
- Support for the fast generation of RF on irregular meshes;
- Out-of-core streaming computation of an RF, which allows the generation of a very large RF, not possible with the traditional approach on the GPU;
- Test results on two GPU architectures (NVIDIA Tesla and AMD FirePro) and one accelerator (Intel Xeon Phi);
- Tests that show that the RFs generated with RAFT have the correct statistical behavior; and
- Practical application of RAFT to two real test cases: planetary nebulae and cosmological simulations.

## 2. RELATED WORK

**Random field generation** The TB method itself was first proposed in [7]. The spectral TB method was then first proposed in [8] where a TB method like RAFT is first described in combination with a spectral line generation algorithm. A Matlab version of the TB method can be found in [9]. The line generation according to a power law power spectrum is performed with the method described in [10]. To generate lines with arbitrary power spectra, we use the algorithm proposed in [11]. For the line generation with a correlation function, we use the circulant embedding approach [12]. This method uses the special structure of a correlation matrix to quickly perform an eigenvalue decomposition. With this method, long lines with a nearly arbitrary correlation structure can be generated.

**GPU** Graphics processing units (GPUs) are used not only for 3D graphics rendering but also in general-purpose computing because of their huge computational power. GPUs' programmability has significantly improved thanks to high-level parallel programming languages such as CUDA [13] and OpenCL [14]. The GPUs' huge potential computational power comes with some drawbacks: The available device memory is limited to few gigabytes (e.g. 6 GB on NVIDIA Tesla K20); it requires slow host–device communications for big datasets. Moreover, optimizing code for GPUs means writing algorithms that are better suited for the hardware, but also exploring low-level optimizations. Traditional compiler optimizations such as loop tiling (blocking) [15] and loop unrolling [16] have been successfully tested on GPUs [17, 18]. However, the search space is quite big [19, 20], and highly optimized codes still require manual, problem-specific exploitation of the optimization space.

**FFT** Our work also focuses on one- and multidimensional FFT methods. For small-scale FFTs, if the data can be held entirely on a GPU, the computation can benefit from the high device memory bandwidth [21–24]. However, if the data does not fit the available device memory, the overhead to transfer data between host memory (i.e. the CPU main memory) and device memory are a bottleneck [25]. This problem applies whenever the dataset is bigger than the available device memory, for example, out-of-core computation or cluster computing [25].

## 3. THE TURNING BAND METHOD

**Correlation function and power spectrum** The (auto-)correlation function describes the correlation of two values of an RF depending on their spatial positions. The power spectrum describes the size distribution of clusters in the RF. For well-behaved correlation functions, these two ways of describing an RF are interchangeable. This transformation is not always possible, but RAFT is able to create an RF from both a spectral density and a correlation function. The TB method is an asymptotically correct approach of generating multidimensional Gaussian RFs, which we use for generating 3D RFs. The RAFT algorithm has multiple steps. First, discrete 1D RFs, that is, lines,

have to be generated. The correlation function or the power spectrum that the 1D lines have to follow is calculated by

$$C_{1D}(r) = \frac{d}{dr}[r \cdot C_{3D}(r)]$$
$$S_{1D}(\omega) = \frac{4\pi \left|\omega^2\right|}{6} \cdot S_{3D}(\omega)$$

where $C_{3D}$ is the correlation function, $S_{3D}$ the power spectrum of the 3D field to be generated, $r$ the distance between points, and $\omega$ the angular frequency corresponding to a structure of a certain size. To generate these lines according to a power law power spectrum, we use a simple 1D Fourier transform approach [10]. For lines with an arbitrary power spectrum, we use a pulse train method [11]. Lines according to a correlation function are generated using a circulant embedding approach [12]. The algorithm presented here is limited to generating isotropic RFs, that means RFs where the correlation function or power spectrum does not depend on the direction in space. Our algorithm can easily be extended to generate RFs with an ellipsoid correlation structure, that is anisotropic RFs where the correlation function or power spectrum is of the same form for each direction but can have a range parameter that depends on the spacial direction. This would be achieved by a coordinate transformation after an isotropic field has been generated (see [26]). For the generation of general anisotropic RFs, TB method would have to be modified more severely, but this is beyond the scope of this paper.

**Algorithm 1** Turning bands method.

$S \leftarrow computeHaltonSequence()$
$Dir \leftarrow computeLineDirection(S)$
$L \leftarrow computeLines(Y)$ // requires 1D FFT
**for all** $line \in L$ **do**
  **for all** $cell(x, y, z) \in domain$ **do**
    $lineCoord \leftarrow -(x, y, z) \cdot Dir[line]$
    $linePoint \leftarrow round(lineCoord \times resolutionFactor) + lineLength \times 0.5 + 1$
    $index = line.index * linelength + linepoint$
    $value = L[index]$
    $field[index] = field[index] + value$
  **end for**
**end for**

**Number of lines and line directions** The TB method is an approximate method. The statistical quality depends on the number of lines used to create the multidimensional field. Empirical studies have shown that for a 3D field of any size, 1000 lines are sufficient to avert banding artifacts [8, 9]. A schematic picture of the TB method is shown in Figure 1(right). The lines are laid out along unit vectors ($u_i$), starting at the origin, so that the surface of the unit sphere is covered as uniformly as possible. We create the unit vectors with the help of a pseudo-random Halton sequence, which leads to a closer to optimal coverage of the unit sphere than random vectors. After the direction vectors have been created, we rotate all vectors together by a random angle around the three major Cartesian axes. This assures that we do not produce statistical artifacts if we generate a large number of fields.

**Projection step** The last step is the projection in which the 3D RF is generated (Figure 1(right)). A point $P$ of the 3D RF is generated by projecting its location vector $X_P$ onto the line $i$ and adding the corresponding value of this line $L_i(P)$ to the value of the point $P$. For $P$, this projection is then repeated for each line. After performing the projection step for each point, we have generated the full 3D RF. Because the RF in the TB method is generated by the summation over a large number of lines that follow a Gaussian distribution, the values of the resulting RF will also follow a

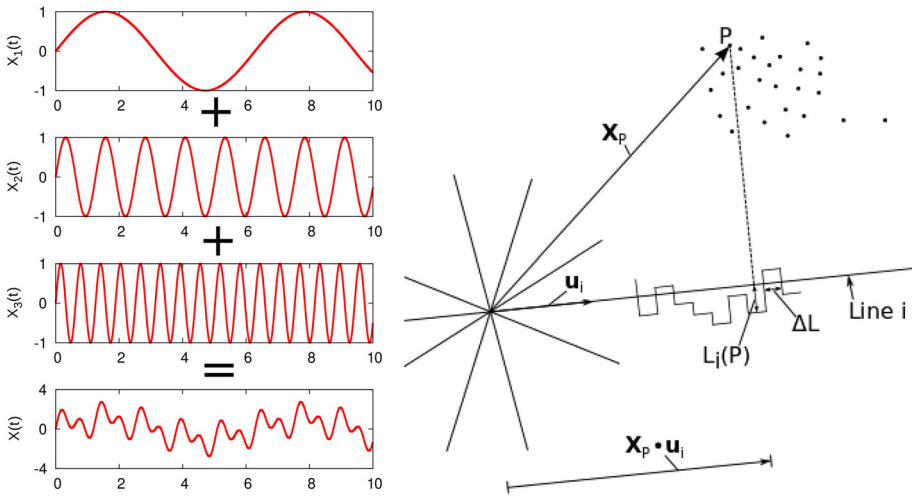



Figure 1. In the fast Fourier transform method (left), components with different frequencies (or wavevectors) are summed up according to their amplitude. This summing is performed by performing the inverse Fourier transform. In the turning band algorithm (right), the point positions $P$ are projected onto the lines $X_p \cdot u_i$, and the corresponding values $L_i(P)$ are then summed over all lines.

Gaussian distribution, which is shown in Section 6. The width of the distribution of the RF depends on the widths of the distributions of the lines which can be chosen as a parameter. RAFT generates Gaussian RF, generating RF with non Gaussian distribution functions is beyond the scope of this article.

**Line discretization** In our algorithm, we use discrete lines. In the regular field case, the resolution of these lines has to be chosen so that the lines have the same resolution as the resolution of the 3D field. In the non-regular case, we usually choose a resolution that is at least five times higher than the average point distance; this generally gives good results. This discretization should lead to some approximation error, because the projections of the 3D points are not regularly spaced. The projection of the inter-point distance on the lines can be much smaller than the average point distance (down to 0 if the projection is perpendicular to the line). But, if the projected distance is small that means that even if we would have a higher line resolution the points close to each other on the line would have similar values since they are correlated, so that the discretization error here is small. The line directions in the projection step are approximately uniformly distributed over the sphere, that means that this discretization error, with regard to two neighboring points, will only happen for a limited number of lines reducing its impact on the accuracy of the final RF. We did experiments with line resolutions up to a factor of 16 higher than the resolution of the 3D grid. These experiments showed that the target spectrum or target covariance is not significantly improved by an increase in line resolution.

**Traditional 3D FFT method** As a comparison, we also show a traditional 3D Fourier transform algorithm for creating an RF with a power law power spectrum and a power law index between $-3$ and $-5$. This algorithm is much less versatile than our TB algorithm. For the input data, we choose the amplitude $A$ for each 3D wavevector $\mathbf{k}$ according to the desired power spectrum. For each wavevector, we also choose a random phase $\Phi$ to be able to generate different realizations of the RF. We choose the random phases of our input data so that $\Phi(\mathbf{k}) = -\Phi(-\mathbf{k})$, making sure that the result of the following inverse Fourier transformation is real. After filling the 3D array with the input data ($A \cdot \Phi$), we only have to perform a 3D inverse Fourier transformation on the array to get our final field with the correct power spectrum. With the inverse Fourier transform, contributions with different wavevectors are summed up according to their amplitude to generate a real valued field (Figure 1(left)). For the power law indexes outside the range $-3$ to $-5$, this method does not work because the resulting field will show very strong generation artifacts. There are more complex

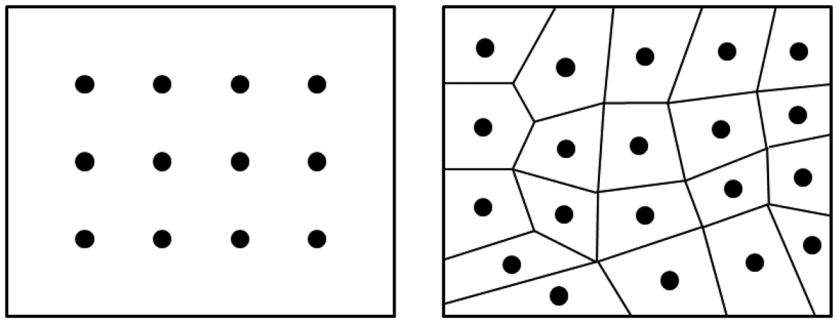


Figure 2. Grid points of a regular (left) and non-regular (right) mesh. In the irregular mesh, the shape of the corresponding Voronoi cell is shown additionally.

3D FFT methods that can generate RF according to arbitrary power spectra, but that is beyond the scope of this article.

To compare the results of both methods, we calculate the power spectrum of the resulting field and compare it with the theoretical power spectrum we aimed to generate. Both methods generate RFs with the correct power spectrum; for more details on the tests performed with RAFT, see Section 6.

**Non-regular (Non-uniform) Fields** One advantage of the TB method is its ability to generate RF on non-regular meshes. The difference between regular and non-regular meshes is shown in Figure 2. The 3D FFT methods can only generate RF on regular rectangular meshes because the FFT works only on equally spaced points. In the projection step, the TB method can generate RF with arbitrary point positions. The resolution of the 1D lines has to be chosen high enough so that the smallest distance between two grid points can be sufficiently resolved. The ability to create RFs on non-regular meshes makes RAFT a very versatile RF generator. It can be used to create RF on regular grids with different resolutions like in adaptive mesh refinement or on entirely unstructured grids. Both of these tasks are much harder to perform with traditional 3D FFT methods.

## 4. PARALLELIZATION AND OPTIMIZATIONS

The TB method, as described by Algorithm 1, comprises four main steps: the Halton sequence (line 1) and line direction generation (line 2), the one-dimensional field generation (line 3), and the final projection step (lines 4–11). Steps 1 and 2 are fast. Step 3 includes multiple 1D FFT calls with very small sizes, which are quite fast (cuFFT has an optimized $cufftPlanMany$ function for this). Therefore, the *projection code* is the main bottleneck and is where we focus our optimization efforts. In the following section, we describe how we map that algorithm, and in particular the projection phase, onto the GPU hardware.

**OpenCL** We use the OpenCL [14] model and terminology: the platform model comprises of a *host* connected to one or more *devices* (e.g. a GPU). Each device consists of one or more compute units, which are further divided into processing elements. A program running on a device is called *kernel* and represents the parallel part of an OpenCL application. A single OpenCL thread is called *work-item*. Several work-items form a *work-group*. OpenCL provides a fast *local memory*, which is shared between work-items belonging to the same work-group. Similarly, OpenCL offers fast local synchronization between work-items inside the same group. Host and device exchange data through memory buffers, which are passed as arguments to the kernel before its execution.

**Parallelization strategy** Algorithm 1 can be parallelized in two different ways. Following the original sequential formulation, it is possible to run a different OpenCL work-item for each line (*line parallelization*). Alternatively, it is possible to apply a loop interchange between the two for loops, therefore mapping a different OpenCL work-item to each cell, that is, *cell parallelization*. The *line*

*parallelization* approach has two drawbacks. First, writing cell values happens concurrently from different threads, therefore requiring an atomic addition. Unfortunately, atomic addition for double floating point precision is not included in OpenCL 1.1, but can be implemented by exploiting a 64-bit compare and exchange operation ($atom\_cmpxchg$). However, atomic operations are extremely expensive on GPUs. The second drawback is the lower parallelism: while applying our approach to a real dataset, the number of lines is too low (ranging from 1024 up to 8192) to exploit GPUs' massively parallel architecture. On the other hand, cell parallelization exposes a high level of parallelism and does not require the use atomic operations. We tested the two parallelizations on a $128^3$ mesh with 1024 lines of length 2600, where the cell parallelization was 50 times faster than the line parallelization.

```
1  __kernel void make_reg_field(int nr_lines,
2          int dim_x, int dim_y, int dim_z, int linelength,
3          __global double4* dir, __global double* L,
4          __global double* RF, double resfactor) {
5    const size_t dim_yz  = dim_y*dim_z;
6    int gid = get_global_id(0);
7    int k = gid / dim_yz;
8    int j = (gid % (fielddim_yz  )) / fielddim_y;
9    int i = gid - j * dim_y - k * dim_yz;
10   double4 id4 = {k, j, i, 0};
11   double rf_value = 0;
12   for(int l=0; l<nr_lines; l++) {
13     double linecoord = - dot(id4, dir[l]);
14     size_t linepoint = round(linecoord*resfactor)+linelength*0.5+1;
15     rf_value += L[l*linelength+linepoint];
16   }
17   RF[gid] = rf_value;
18 }
```

Listing 1. Non optimized OpenCL kernel for the cell parallelization projection kernel.

**Loop blocking and unrolling** Starting from the cell parallelization, we applied two loop optimizations to the for loop in line 12 (Listing 1). First, we tried to apply *loop blocking* (i.e. tiling) by partitioning the loop iteration space into smaller blocks (matching the work-group size), to ensure data used in a loop stays in the fast local memory available on the GPU. This technique can be applied to the line $dir$ vector (line 13), which has coalesced memory accesses. However, the $L$ array (line 15) is accessed randomly and cannot be prefetched.

We also applied *loop unrolling* (i.e. unwinding) to the same loop. The goal of loop unrolling is to reduce the number of iterations and branch penalties, as well as hiding memory access latencies while reading data from the memory [16]. The latter is particularly important in our case, as the inner loop performs many random accesses to the (slower) global memory. We applied to the projection code all the combinations of loop blocking and unrolling, with group size of 64, 128, 256, and 512, and unroll factors of 2, 4, and 8, on three different hardware architecture.

**Streaming out-of-core field generation** GPU architecture has a limited amount of memory with respect to the RF size needed in some applications (already 30 GB for an $1024^3$ grid). Especially while working with astrophysical datasets, RFs commonly exceed the memory available on a single GPU. This is a limitation for the standard approach based on 3D FFT [25]. Our approach only requires the lines to be stored on the GPU and can be further distributed to work over multiple devices (e.g. on a multi-GPU or cluster of GPUs) or to perform an out-of-core streaming computation of the field in a single machine. RAFT splits the field in fragments of $128^3$ cells to allow out-of-core RF generation.

**Non-regular fields** The TB method can also be used to generate a non-regular RF. We applied the same optimizations to a non-regular version of the projection kernel (note that other parts of the algorithm do not change) and tested different point distributions.

## 5. RESULTS

We ran different optimized versions of the RAFT code on three target devices: NVIDIA Tesla K20m, AMD FirePro S9000, and Intel Xeon Phi (details are listed in Table I). We used the OpenCL version of RAFT with all three platforms; in addition, we also used the CUDA version for the NVIDIA Tesla, for a total of four platform configurations.

We used the libWater CUDA extension [27] to support both CUDA and OpenCL kernels. All tests were performed with double precision. For the FFT implementations, we used FFTW [28] on the CPU and cuFFT [24] for the CUDA version.

**RAFT versus traditional approach** We compared the traditional approach based on 3D FFT with our approach, running on all the different platform configurations. Figure 3 shows the performance for different grid sizes and line lengths. For all the tests, we used 1024 lines and line length scaling according to the grid size (e.g. $512^3$ cells line length is 1024). The standard approach on the CPU uses 3D FFTW and supports very big grid sizes. The erratic behavior of the FFTW approach can be explained by the different algorithms employed by the FFTW library when the number of points is not equal to a power of two. The NVIDIA Tesla version of the same approach is based on cuFFT, but it is limited by the amount of memory available on the GPU (up to 262.14 million cells for our test

Table I. Target architecture.

| | NVIDIA Tesla k20m | AMD FirePro S9000 | Intel Xeon Phi 7120P |
|---|---|---|---|
| Frequency (MHz) | 705 | 900 | 1333 |
| Compute unit | 13 | 28 | 240 |
| Global memory size (MB) | 4799 | 3072 | 11634 |
| OpenCL version | 1.1 CUDA (340.29) | 1.2 AMD-APP (1268.1) | 1.2 (Build 82248) |
| CUDA compute capability | 3.5 | | |

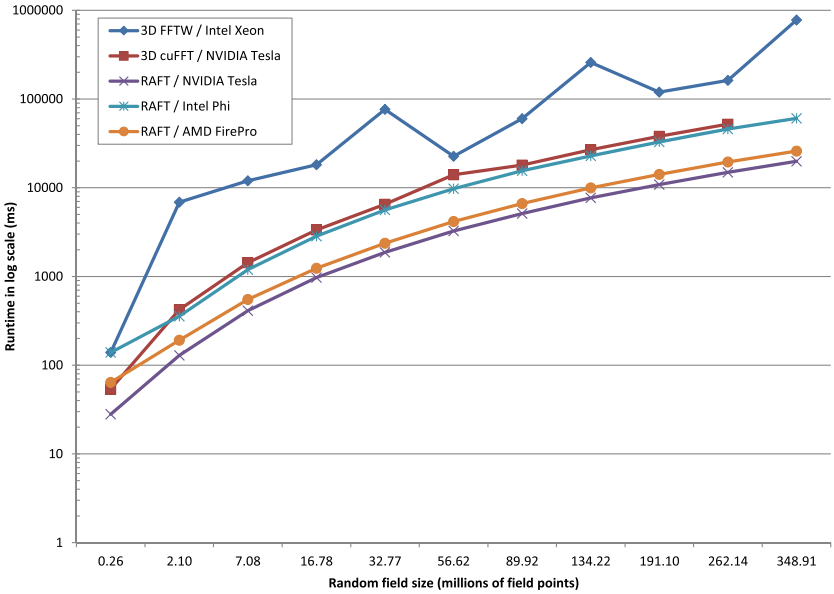
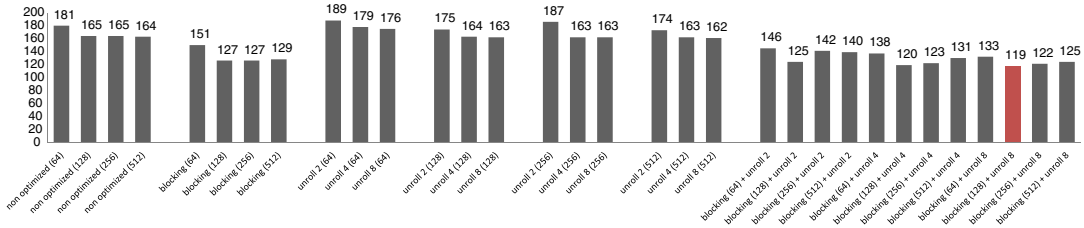
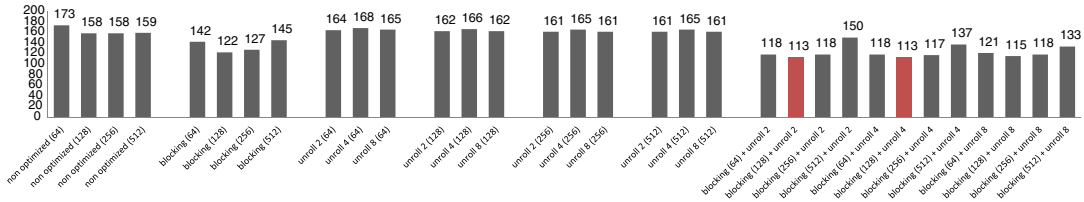


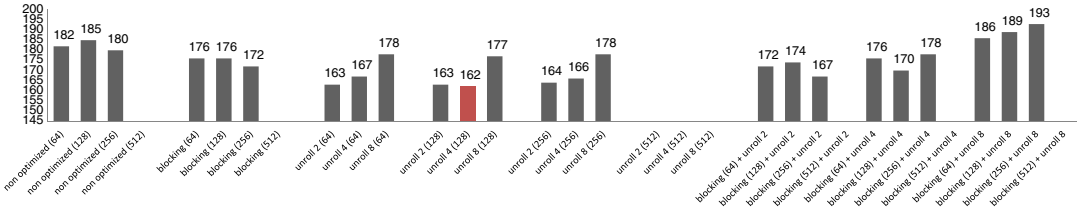

Figure 3. Performance behavior of our out-of-core random field generation on different target architectures with varying problem sizes (i.e. the number of cells), compared with that of 3D FFT method. Note that FFT on the graphics processing unit is limited by the available memory. FFT, fast Fourier transform are performed with the FFTW library (see [28]).



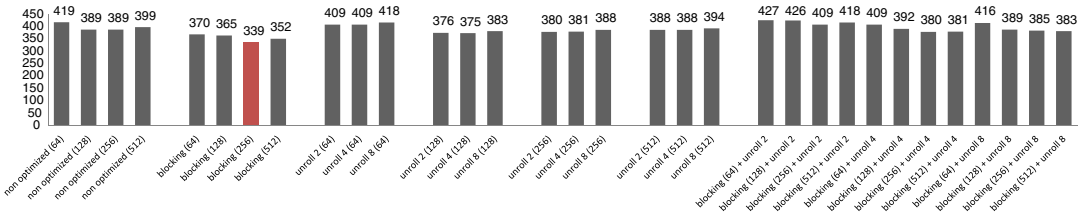

(a) NVIDIA Tesla K20m (CUDA)




(b) NVIDIA Tesla K20m (OpenCL)




(c) AMD FirePro (OpenCL)




(d) Intel Phi (OpenCL)

Figure 4. Best optimization configuration for the projection kernel on different platforms. In red, the best configurations for each target platform. Runtimes are in milliseconds.

cases). 3D FFT methods require an extra cell per dimension (i.e. to generate a field of $256^3$ elements we need a $257^3$ 3D FFT). We tested RAFT on all of our available hardware configurations. Each RAFT code was running on its optimized configuration (see next paragraph). RAFT on the AMD FirePro and the NVIDIA Tesla is faster than 3D cuFFT on the NVIDIA Tesla. Furthermore, RAFT can efficiently generate RFs bigger than the available device memory. In all tested configurations, RAFT is faster than the 3DFFTW approach on the CPU.

**Projection kernel optimizations** Figure 4 shows the runtimes for the projection kernel on a uniform mesh generation with $128^3$ cells. The optimized version is always faster than the non-optimized. However, the best optimization configurations drastically change according to the underlying platform.

On NVIDIA, Tesla blocking is more effective than unrolling; however, the best configurations apply both optimizations. The fastest OpenCL configuration is different from the CUDA one: OpenCL versions have faster execution times with lower unroll factor and higher local size (best configurations had unroll factor 2–4 and local size 128), while the faster CUDA version used much

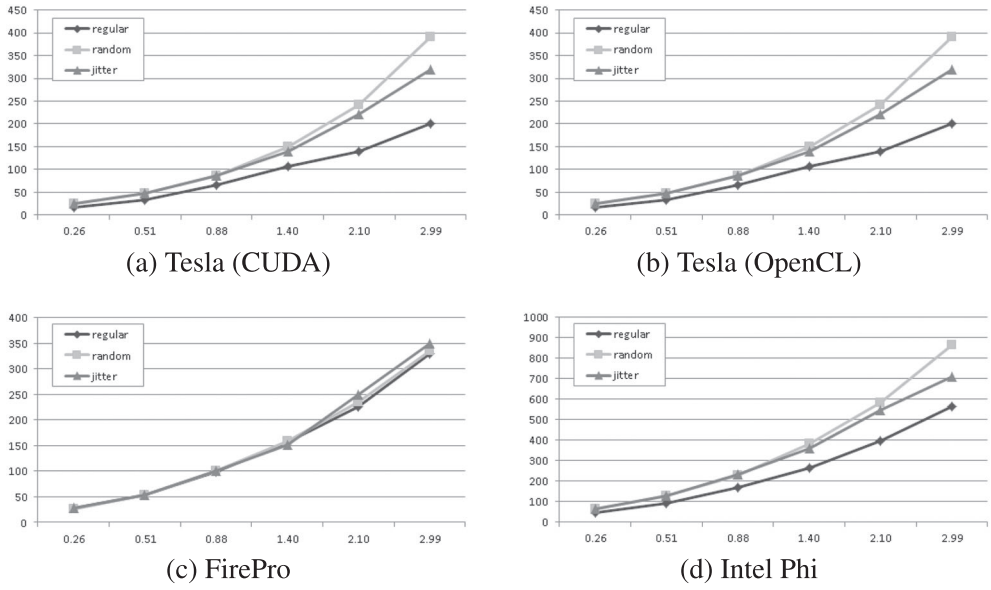



Figure 5. Non-regular field with three different point distributions.

more unrolling (i.e. factor 8). However, the difference between optimized CUDA and OpenCL is usually less than 4%.

The AMD FirePro S9000 reached faster performance by only using unrolling (i.e. unroll factor 4), with local size of 128. As shown in Figure 4c, such an architecture does not support local group size of 512, but usually, we get the best performance with a local size of 128.

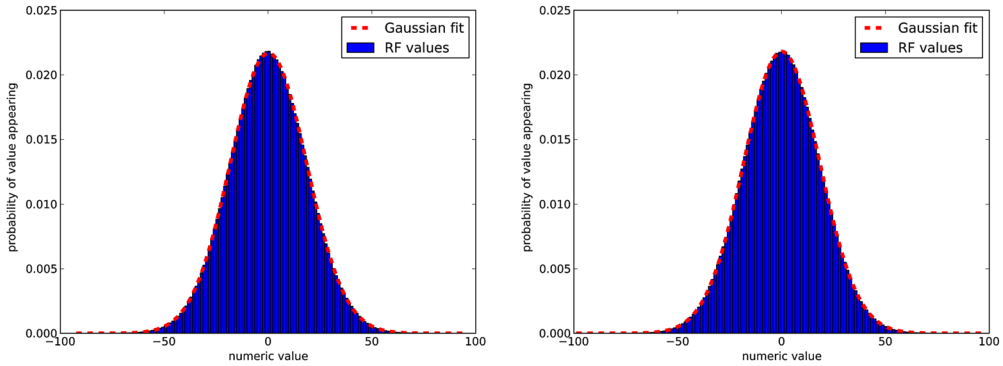
The improved locality offered by a blocking optimization is particularly beneficial also for the Intel Xeon Phi; despite that this architecture does not have a programmable local memory such as other GPUs (OpenCL local memory is allocated on the regular GDDR memory and is supported by the cache system like any other memory [29]), blocking indirectly improves cache reuse, and it is enough to provide the faster configuration (Figure 4d).

The use of blocking to improve locality has been applied to the relatively small $dir$ buffer. Unfortunately, there is no simple way to apply the same optimization to the $L$ buffer, as it shows data-dependent memory access.

**Non-regular fields** Finally, we tested the non-regular version of the RF generation algorithm against different mesh structures in order to understand how the point distribution affects the locality of the memory accesses. The first, named *regular*, has exactly the same distribution of the regular, uniform grid used before. The second uses a *jitter* sampling approach where each point has a regular position plus a random offset. The third is a completely *random* point distribution, where two points close in memory in the input array may be very distant in space. Figure 5 shows the performance for the four platform configurations. For Intel Phi (Figure 5d) and both Tesla configurations (Figure 5a and b), the random distribution is much slower than the regular one, as it exposes poor memory accesses locality. However, this difference is quite small for FirePro (Figure 5d), which confirms the potential of the rich complement of storage offered by AMD's Tahiti architecture, such as the hardware-managed, multi-level read/write cache hierarchy.

## 6. TESTS OF STATISTICAL BEHAVIOR

We performed a number of tests with RAFT of which a selection is shown in this section. The goal of these tests is to determine whether or not the generated RFs have the correct statistical behavior we aimed to generate. This is performed because the described method is an approximate method




(a) Value distribution for irregular field with double exponential correlation function.

(b) Value distribution for regular field with a power law power spectral density.

Figure 6. Normalized histogram of random field (RF) values for 10 RFs each. The histogram is shown in blue, while a Gaussian fit is shown as a red line. The Gaussian fit and the RF value distribution agree, showing that our method generates Gaussian RF.

that is only correct for a large number of lines. In this section, we show that our method performs correctly, and the approximate nature of the method is not a problem.

As mentioned in Section 3, RAFT can generate RFs with a desired power spectrum or correlation function, on regular and non-regular meshes. First, we tested if the values of the generated RFs followed a Gaussian distribution function. To achieve this, we generated two test cases consisting of 10 RFs with $128^3$ points each. The first test case was generated with a power law power spectrum (spectral index $= -2$) on a regular grid, and the second test case was generated with a double exponential correlation function on an irregular grid. We calculated the histogram of the RF values with 150 bins for both of these test cases. In Figure 6, we show the histograms of the value distributions overlayed with a fitted Gaussian distribution. For both test cases, we can see that the Gaussian fit agrees very well with the distribution of the RF values, proving that the RFs generated with our code are following a Gaussian distribution. The width of the distribution depends on the width of the distribution of the individual lines that are summed up. As a second test, we investigated if the power spectra or correlation functions of the generated RFs agreed with the target power spectrum or correlation function. To test the RF generation with a power spectrum, we choose a power law power spectrum as a test case because these RF are used in a number of applications, as shown in Section 7. This test is performed on a regular mesh, for the simple reason that our testing procedure for power spectra utilizes an FFT, which only works on regular meshes. The test case for the correlation function method is a double exponential correlation function on a non-regular mesh. In the next two paragraphs, we show that our method performs well in both test cases.

**Power law power spectra on a regular mesh** In Figure 7, we show the comparison of three theoretical power law power spectra, and the power spectra of RFs generated with RAFT. To calculate the power spectra of the generated RFs, we first generated 100 RFs for each power law index. The RFs used for this test had a size of $128^3$ regularly arranged mesh points. For each of these 100 RFs, we calculated the power spectrum by utilizing a large 3D FFT; after that, we averaged the power spectra of each spectral index to get the average power spectrum of the generated RF. We calculated an average because the power spectrum of a single RF tends to fluctuate a lot; this is not a flaw of the method but due to the statistical nature of RFs themselves. When running our code for power law power spectra, we realized that the introduction of a compression factor markedly improves the statistical behavior of the created RF. This compression factor reduces the size of a line element and thereby compresses the line before the projection step, see Section 3. The value of the compression factor depends on the power law index of the spectrum, but not on the resolution. Currently, we calculate low-resolution RFs to determine the optimal compression factor before calculating the full resolution fields, but we are working on an analytic way to determine the optimal compression factor. For Figure 7, the compression factors used are 1.32, 1.13, and 0.8 for the power law indices

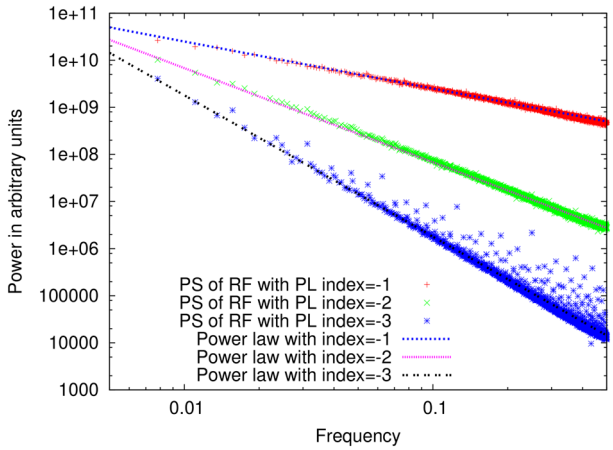



Figure 7. Power Law (PL) Power Spectrum (PS) for three different spectral indices ($-1$, $-2$, and $-3$). The theoretical power spectrum and the averaged power spectrum of 100 RFs generated with a target spectrum corresponding to the theoretical one are shown. The power is not normalized. $\omega = 0$ corresponds to a constant over the whole size of the RF, while $\omega = 0.5$ corresponds to the smallest fluctuation that can be seen on the mesh (twice the size of a mesh cell). RF, random field.

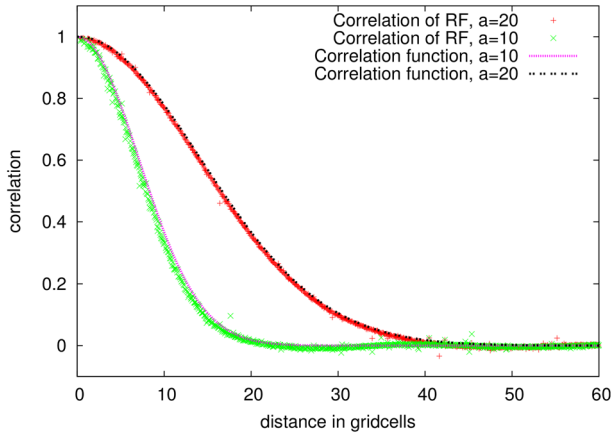



Figure 8. Double exponential correlation function for two different range parameters ($a = 10$, $a = 20$). The theoretical correlation function and the averaged correlation of 100 random fields (RFs) generated with a target correlation corresponding to the theoretical one are shown. The RFs were generated on a random non-regular mesh.

$-1$, $-2$, and $-3$, respectively. In Figure 7, it can be seen that, generally, the power spectra of the RFs we generated are in good agreement with the theoretical values. For steep power law indices ($< -3$), it seems like there are some outliers that could be overtones. We generated the RFs in this test with different resolutions of the discretized lines. That allowed us to rule out a discretization error as the source of the outliers in the steep power law case. We suspect the outliers are the effect of the regularity of the mesh.

**Double exponential correlation function on a non-regular mesh** In Figure 8, the comparison between two theoretical double exponential correlation functions

$$C(r) = e^{-(\frac{x}{a})^2}$$

and the correlation of the corresponding RF are shown for two different range parameters $a$. The RFs generated here are generated on a non-regular mesh. The coordinates of all mesh points, except for a reference point at (30,30,30) have been randomly determined in a box of size $128^3$. The amount of

mesh points calculated was $128^3$. To calculate the correlation in the RFs, we calculate the correlation with the reference point for each point. After this step, we average over all distances to the reference point with a distance bin of size 0.05, which smooths the correlation diagram. We perform this procedure for 100 RFs generated and average over the 100 realizations of the RF. In this step, all the generated RFs use the same random mesh points. As in the previous paragraph, we average over 100 RF realizations because one RF realization usually has a quite high spread of correlations due to the statistical nature of the method. We can see that the theoretical prediction and the correlation of the generated RF agree very well. This shows that our method also works for irregular meshes, which is a great advantage over traditional FFT methods.

## 7. APPLICATIONS

**Astrophysics: Planetary Nebulae** The code presented here has already been implemented to create a wind with density fluctuations in a Planetary Nebulae clump simulation. To have an inflowing wind entering on one side of the computation domain, we create a RF tube of size $256 \times 256 \times 10000$ with a power law power spectrum. The size of the tube will be larger for higher resolutions. For this problem we already use the out-of-core version of RAFT since the whole field is too large to fit into the main memory. Examples of the fields used can be found in Figure 9; for these simulations, the power law index of the power spectrum is a free parameter, so we show RFs for different power law indices. With the optimized out-of-core OpenCL kernel, NVIDIA Tesla, it takes 37 367 ms to generate a RF with $256 \times 256 \times 10000$ points using 1024 lines with a line length of 20 000.

**Astrophysics: Cosmology Simulations** In the astrophysical community, moving mesh techniques for calculating hydrodynamical simulations have become more popular. The most prominent example is AREPO, the new moving mesh *n*-body code [30]. In these codes, hydrodynamic simulations are performed on a non-regular mesh. RAFT ability to create RFs on a non-regular mesh is a clear advantage over the traditional 3D FFT methods for all simulations performed with these moving mesh codes.

Turning band random field is able to generate RFs that can be used as initial conditions for cosmological structure formation simulations with AREPO. A realization of such a RF following a Harrison-Zeldovich spectrum [31] is shown in Figure 10 (left). These new moving mesh codes can also be used to perform turbulence-driven simulations. These simulations are typically quite large so the ability of RAFT to create the fields out-of-core is another advantage. In these turbulence-driven simulations, an RF is needed in every time-step, making the RF generation a major contributor to the computational cost of the whole simulation. Until now, the runtime of TB methods prohibited them from being used in this manner. With the increased performance on the GPU, TB m

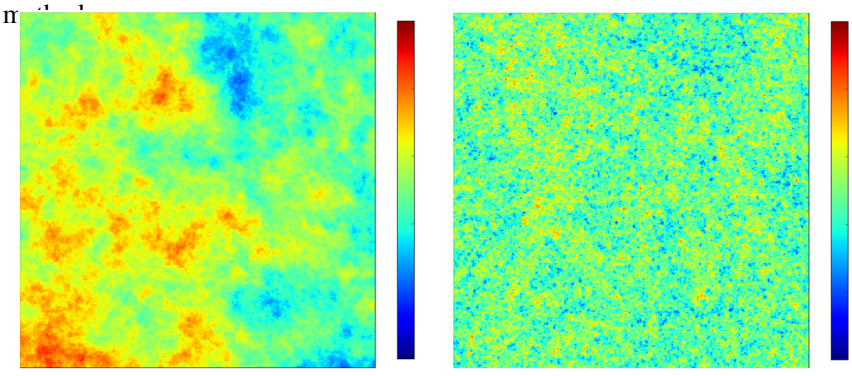


Figure 9. 2D plane slices through 3D random field (RF) used in the Planetary Nebulae simulations. Red values are positive while blue values are negative. The left image shows a field with a power spectrum $P(k) \propto k^{-3.9}$ that emphasizes larger structures while the right image shows a field with a power spectrum of $P(k) \propto k^{-2.0}$ where larger structures are less prominent.

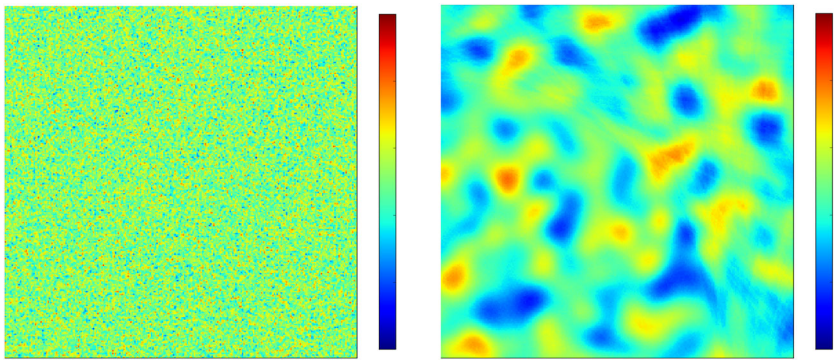


Figure 10. Red values are positive while blue values are negative. The left image shows a 2D slice through a 3D random field (RF) with a power spectrum of $P(k) \propto k^{-1.0}$ sometimes proposed as the initial fluctuations (Harrison-Zeldovich spectrum) of cosmological structure formations. The right image shows a slice through a 3D RF with a power spectrum of $P(k) \propto k^6 \cdot e^{(-k)}$ that is used for turbulence-driving simulations.

such as RAFT, are now a viable option for turbulence-driven simulations on non-regular meshes. In Figure 10 (right), we show a slice of an RF that can be used for this kind of turbulence-driven simulations. With the optimized out-of-core OpenCL kernel, on NVIDIA Tesla, it takes 61 173 ms to generate a RF with $1024^3$ points using 1024 lines with a line length of 2048.

## 8. CONCLUSIONS

This article shows a novel approach to generate Gaussian random fields by using the turning band method. We demonstrated that turning band methods can be significantly sped up by porting them onto GPUs and accelerators. We presented RAFT, our implementation of the turning band method that efficiently generates random fields on both regular and non-regular meshes. RAFT is able to create random fields that are bigger than the available device (e.g. GPU) memory efficiently, thanks to its support for out-of-core streaming computation. Traditional methods based on 3D FFT are limited to the available device memory and cannot generate random fields on non-regular meshes. These advantages make RAFT much better suited to be used in combination with, for example, moving mesh hydrodynamic codes than traditional 3D FFT RF generators. RAFT can easily be used for turbulence simulations that need to generate a large RF in each hydrodynamic time-step speeding up these simulations significantly. RAFT can also be used to generate gas flows with non-uniform density like it has been performed in the PN example. All in all, our algorithm should be considered a general tool to be used by scientists that need to quickly generate large RFs. The project source is available at https://github.com/LarsHunger/RAFT under the GNU Lesser General Public License (LGPL) license.

In future work, by means of libWater [27], we plan to further extend this work to multi-GPU and heterogeneous distributed system.


### ACKNOWLEDGEMENTS

This project was funded by the FWF Doctoral School CIM Computational Modelling under contract W 1227-N16 (DK-plus CIM) and by the Austrian Research Promotion Agency under contract 834307 (AutoCore).


### REFERENCES


1. Engelberg S. *Random Signals and Noise: A Mathematical Introduction*. CRC Press: Boca Raton, Florida, USA, 2007.
2. Springel V, White SDM, Jenkins A, Frenk CS, Yoshida N, Gao L, Navarro J, Thacker R, Croton D, Helly J, Peacock JA, Cole S, Thomas P, Couchman H, Evrard A, Colberg J, Pearce F. Simulations of the formation, evolution and clustering of galaxies and quasars. *Nature* 2005; **435**:629–636.

3. Kofler K, Steinhauser D, Cosenza B, Grasso I, Schindler S, Fahringer T. *Kd-tree based N-body simulations with volume-mass heuristic on the GPU Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC)*.
4. Stone J. Direct numerical simulations of compressible magnetohydrodynamical turbulence interstellar turbulence. *Proceedings of the 2nd Guillermo Haro Conference*, Cambridge University Press, Puebla, Mexico, 1999; 267.
5. Chiles JP, Delfiner P. *Geostatistics: Modeling Spatial Uncertainty*. John Wiley & Sons: New York, 1999.
6. Hunger L, Cosenza B, Kimeswenger S, Fahringer T. Random fields generation on the GPU with the spectral turning bands method. *European Conference on Parallel Processing (Euro-Par)* 2014; **8632**:656–667.
7. Matheron G. The intrinsic random functions and their application. *Advances in Applied Probability* 1973; **5**:439–468.
8. Mantoglou A. Digital simulation of multivariate two- and three-dimensional stochastic processes with a spectral turning bands method. *Mathematical Geology* 1987; **19**(2):129–149.
9. Emery X, Lantuéjoul C. TBSIM: a computer program for conditional simulation of three-dimensional Gaussian random fields via the turning bands method. *Computers & Geosciences* 2006; **32**:1615–1628.
10. Kasdin NJ, Walter T. Discrete simulation of power law noise. *Frequency Control Symposium, 1992. 46th., Proceedings of the 1992 IEEE*, Hershey, Pennsylvania, USA, 1992; 274–283.
11. Carrettoni M, Cremonesi O. Generation of noise time series with arbitrary power spectrum. *Computer Physics Communications* 2010; **181**(12):1982–1985.
12. Dietrich CR, Newsam GN. Fast and exact simulation of stationary Gaussian processes through circulant embedding of the covariance matrix. *SIAM Journal on Scientific Computing* 1997; **18**(4):1088–1107.
13. NVIDIA. CUDA Compute Unified Device Architecture Reference Manual.
14. Khronos OpenCL Working Group. The OpenCL Specification 1.1.
15. Wolfe M. *More iteration space tiling Proceedings of the ACMIEEE Conference on Supercomputing*, Reno, Nevada, USA, 1989; 655–664.
16. Sarkar V. Optimized unrolling of nested loops. *International Journal of Parallel Programming* 2001; **2**(5):545–581.
17. Murthy GS, Ravishankar M, Baskaran MM, Sadayappan P. Optimal loop unrolling for GPGPU programs. *IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, Altanta, Georgia, USA, 2010; 1–11.
18. Yang Y, Xiang P, Kong J, Zhou H. A GPGPU compiler for memory optimization and parallelism management. *Proceedings of the 2010 ACM SIGPLAN PLDI*, Toronto, Canada, 2010; 86–97.
19. Jordan H, Thoman P, Durillo JJ, Pellegrini S, Gschwandtner P, Fahringer T, Moritsch H. A multi-objective auto-tuning framework for parallel codes. *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, Salt Lake City, Utah, USA, 2012; 10:1–10:12.
20. Kofler K, Grasso I, B. Cosenza, Fahringer T. An automatic input-sensitive approach for heterogeneous task partitioning. *Proceedings of the 27th International ACM Conference on Supercomputing*, Eugene, Oregon, USA, 2013; 149–160.
21. Govindaraju N, Lloyd B, Dotsenko Y, Smith B, Manferdelli J. High performance discrete Fourier transforms on graphics processors. *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC)*, Austin, Texas, USA, 2008; 2:1–2:12.
22. Nukada A, Matsuoka S. Auto-tuning 3-D FFT library for Cuda GPUs. *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC)*, Portland, Oregon, USA, 2009; 30:1–30:10.
23. Volkov V, Kazian B. *Fitting FFT onto G80*. Architecture Report, University of California: Berkeley, 2008.
24. NVIDIA. *CUDA CUFFT Library, Version 2.3*, 2009.
25. Chen Y, Cui X, Mei H. Large-scale FFT on GPU clusters. *Proceedings of the 24th ACM International Conference on Supercomputing (ICS)*, Tsukuba, Japan, 2010; 315–324.
26. Tompson AFB, Ababou R, Gelhar LW. Implementation of the threedimensional turning bands random field generator. *Water Resources Research 25.10* 1989:2227–2243.
27. Grasso I, Pellegrini S, Cosenza B, Fahringer T. LibWater: heterogeneous distributed computing made easy. *Proceedings of the 27th International ACM Conference on Supercomputing*, Eugene, Oregon, USA, 2013; 161–172.
28. Frigo M, Johnson SG. The design and implementation of FFTW3. *Proceedings of the IEEE* 2005; **93**(2):216–231.
29. Intel. Intel SDK for OpenCL Applications XE Optimization Guide.
30. Springel V. E pur si muove: Galilean-invariant cosmological hydrodynamical simulations on a moving mesh. *Monthly Notices of the Royal Astronomical Society* 2010; **401**(2):791–851.
31. Harrison ER. Fluctuations at the threshold of classical cosmology. *Physical Review* 1970; **D1**(10):2726.