GenSDF: An MPI-Fortran based signed-distance-field generator for computational fluid dynamics applications.

Akshay Patil[a], Udhaya Chandiran Krishnan Paranjothi[b], Clara García-Sánchez[a]

[a] *3DGeoinformation Research Group, Delft University of Technology, a.l.patil@tudelft.nl*
[b] *Wind Energy Section, Flow Physics and Technology Department, Faculty of Aerospace Engineering, Delft University of Technology*

Abstract

This paper presents a highly efficient signed-distance field (SDF) generator designed specifically for computational fluid dynamics (CFD) workflows. Our approach combines the parallel computing power of Message Passing Interface (MPI) with the performance advantages of modern Fortran, enabling precise and scalable computations for complex geometric domains. The algorithm focuses on localized distance calculations to minimize computational overhead, ensuring efficiency across multiple processors. An adjustable input stencil width allows users to balance computational cost with the desired level of accuracy in distance approximation. Additionally, the generator supports the widely used Wavefront OBJ format, utilizing its encoded outward normal information to achieve accurate boundary definitions. Performance benchmarks demonstrate the tool's ability to handle large-scale 3D models with high fidelity and reduced computational demands. This makes it a practical and effective solution for CFD applications that require fast, reliable distance field computations while accommodating diverse geometric complexities.

Keywords: Signed-Distance-Field, Computational Fluid Dynamics, MPI

Metadata

Nr.	Code metadata	Description
C1	Current code version	v0.1
C2	Permanent link to code/repository used for this code version	`https://github.com/AkshayPatil1994/GenSDF`
C3	Permanent link to Reproducible Capsule	NA
C4	Legal Code License	AGPL-3.0 license
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	MPI + Fortran
C7	Compilation requirements, operating environments & dependencies	gfortran and MPI library
C8	If available Link to developer documentation/manual	`https://github.com/AkshayPatil1994/GenSDF`
C9	Support email for questions	a.l.patil@tudelft.nl

Table 1: Code metadata details and support information along with the link to the code repository.

1. Motivation and significance

Scale resolving turbulent flow simulations around complex objects have become increasingly accessible for engineering problems by virtue of the increasing computational power [1, 2, 3, 4]. While there exist multiple classes of methods that can be used to introduce the complex objects within the flow, such as body-conforming grids as shown in figure 1 [3, 5], the use of the immersed boundary method (IBM) has increasingly become popular due to its efficient underlying algorithm on regular grids [6, 7]. Since IBM does not require the grid to conform to the object, the underlying grid data structure can leverage the Cartesian grid's simplicity and use efficient pressure solvers for the governing equations [8]. For complex objects, it becomes essential to correctly locate the boundary of the immersed object on the computational lattice/grid, which can be done through the signed distance field (SDF). This paper presents an efficient and scalable SDF computation algorithm that can be used for geometries stored using the OBJ file format.

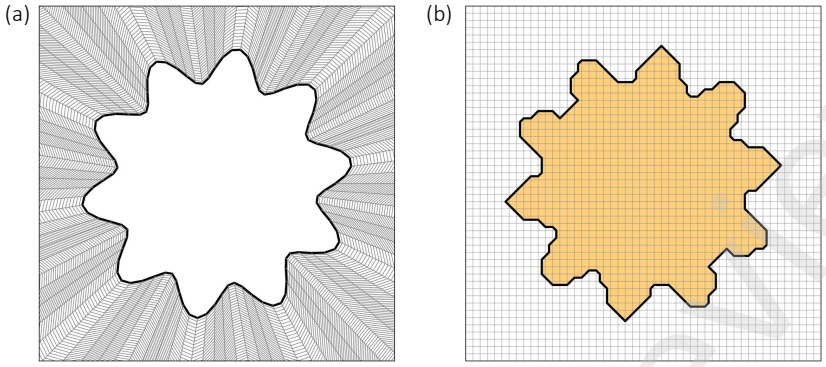

Figure 1: (a) Body conforming mesh used in typical computational fluid dynamics applications (b) Immersed Boundary Method over a regular Cartesian grid.

While SDF generators are not new [9], the closed-source nature of the existing ones has limited the accessibility for applications within the CFD community. Additionally, as the problem size for scale-resolving simulations grows, single Graphics Processing Unit (GPU) implementations can be severely limited by the total available memory per GPU. Thus, in this work, we use the distributed memory paradigm allowed by Message-Passing-Interface (MPI) to circumvent this limitation and allow for a scalable algorithm beyond billions of grid points and surface triangulation corresponding to the geometry.

2. Software description

The *GenSDF* software is a general-purpose code developed to accurately and efficiently obtain the SDF of an arbitrary triangulated geometry over a Cartesian grid. *GenSDF* was primarily motivated by the need for a scalable and distributed memory (MPI-based) tool to obtain SDFs for computational grid sizes in the order of billions.

2.1. Software architecture

The *GenSDF* software is written in modern Fortran, allowing for a simple yet computationally efficient framework to develop and maintain the code. The distributed memory parallelism is achieved through the MPI interface, where the computational grid is decomposed using linear decomposition along the x-coordinate axis into N chunks, where N corresponds to the number of MPI ranks used for the parallel version of the code. Figure 2 shows a simple flowchart detailing the central components of the software presented in this paper. As detailed in the second step of the algorithm, the bounding box coordinates of the triangulated geometry can be used to co-locate it on

the computational grid. Using this information, a large positive distance is assigned to the grid points outside the bounding box (referred to as B-Box in Figure 2) since these points are known in advance to be outside the geometry. This bounding box methodology allows for correctly excluding the computationally expensive distance query for points that are known to be outside the geometry. The bounding box also includes a user-specified stencil buffer (s_b), where s_b is the number of computational grid points around the bounding box.

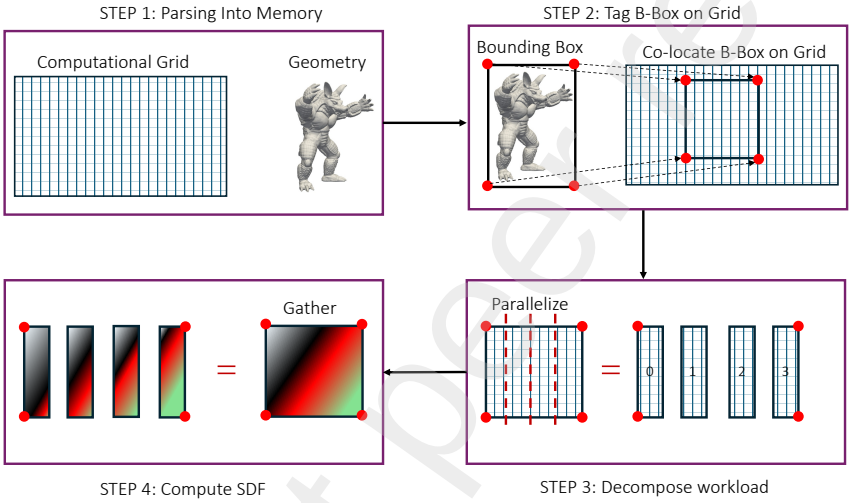

Figure 2: Flowchart of the key steps in the code. In step 3, the numbers represent the MPI rank IDs. The flowchart presents a 2-dimensional example of the bounding box; however, the code works in 3 dimensions.

After the bounding box is co-located in Step 2, the requisite B-Box is decomposed based on the user-requested MPI ranks, as detailed in Step 3. Here, the domain decomposition is carried out over the streamwise direction, usually the longest in such flow simulations. Once the domain decomposition is done, individual MPI ranks compute the local distance queries for the surface triangles within their individual coordinate extents. Finally, Step 4 gathers the decomposed domain by handling the boundary values and file write operations.

2.2. Software functionalities

The software has three core functionalities:

- Parse OBJ geometry and load it into memory

- Calculate the distance between the query point and the triangle module
- Parallelise the workload for efficient and accurate computations

Within these core functionalities, the software first parses the input file where the user provides information about the input geometry, input grid type, the default distance value for points outside the geometry, and the stencil buffer. This user input file parser is contained in the subroutine called *read_inputfile*. Once the input data is known, the computational grid is parsed into memory on each of the MPI ranks using the *read_cans_grid* subroutine. Subsequently, the surface triangulated geometry is parsed into the memory using the *read_obj* subroutine that loads the vertex, vertex normal (assumed to be pointing outwards), and face data. Once the geometry is loaded to memory, the bounding box corresponding to the geometry is calculated which is then used to co-locate the extent over which the distance is queried.

The computationally intensive workload is housed in the subroutine titled *compute_scalar_distance_face*. The algorithm to compute the signed distance is designed to scale the computational effort based on the number of vertices in the triangulated geometry. Specifically, in this implementation, the outermost loop iterates over the total number of faces within the triangulated geometry. As presented in figure 3, each triangular face on the surface geometry is composed of three vertices where an average surface normal pointing outwards is also defined (not shown in the figure). Knowing the x, y, and z coordinates of the vertices, a face-local bounding box can be obtained readily as the computational grid (shown with the blue grid lines) constitute a simple Cartesian structure. This bounding box includes all the candidate grid points where a distance calculation is considered and excludes all the other points that are relatively further away from the triangulated face under consideration. Once the face-local bounding box is known, the distance between the point and the triangular face is calculated using the well-established algorithm [10]. This approach not only localises the distance query, but also allows for an efficient parallelization strategy as there is no global communication required when such a distance query is requested. Since the outer loop iterated over the total number of faces in the triangulated surface, there are some limitations that must be observed in this algorithm. Geometries with long and slender triangles may render inaccurate and de-generate SDF, this is especially true for geometries that constitute a large collection of blocks (cubes such as buildings). Additionally, for test cases where the underlying computational grid is relatively much finer when compared to the surface triangulation, there can be instances where certain computational points can

render de-generate SDF. A simple fix for both these scenarios is to refine the surface triangulation using a meshing tool or the instructions provided within the published repository.

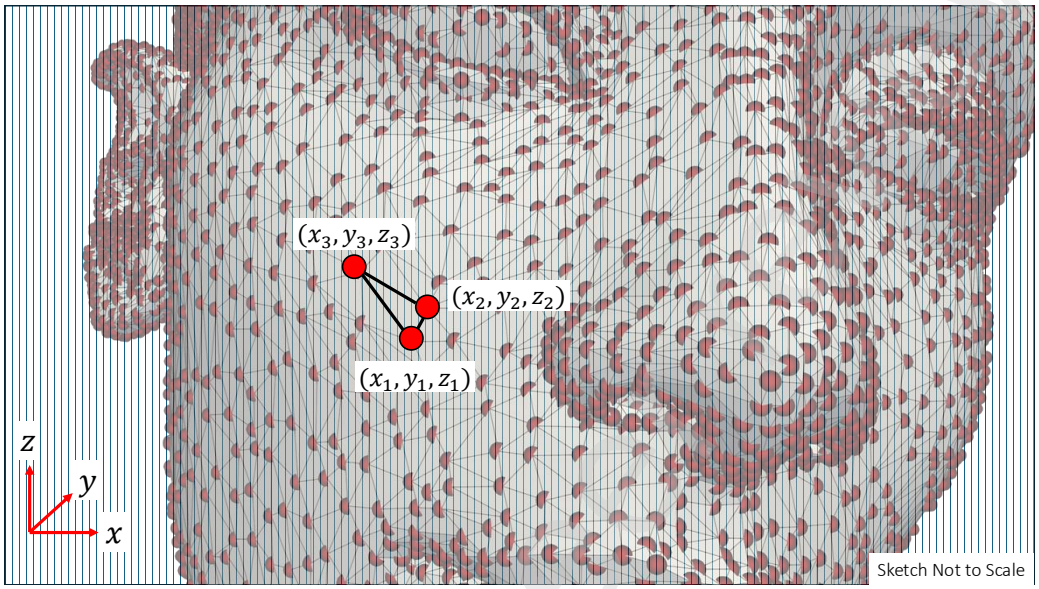

Figure 3: Example sketch of the triangulated surface geometry overlaid onto the computational grid (2D example). The three filled-red circles correspond to the vertices of the triangular face denoted by the solid black lines.

Since the user is allowed to specify a stencil buffer around the geometry where the distance is calculated, the default initialization for the SDF value does not correctly tag the values inside the geometry as negative. In this software, we make use of a simple 3D flood fill algorithm that assigns a large negative value of -10^3 for internal values bounded by the SDF. Finally, once the SDF is calculated on the individual MPI ranks, the *gather_array* subroutine collects the decomposed parts of the array and stacks it into a single contiguous array which is then written out to a binary file format for further use within the CFD solver.

3. Illustrative examples

A typical output of the program when compiled using the openmpi Fortran compiler can be seen in figure 4. By default, the code assumes that the grid is staggered such that the locations of u, v, w, and p (scalar) are different thus generating four different arrays corresponding to the location of the variables. The code also provides an indication of the expected minimum memory required to run the specific program based on the total number of

Figure 4: Example output of the MPI-enabled code.

arrays initialised by the software. This is important when the problem size is large and the user needs to parse the data into memory.

The left panel in figure 5 shows two clipped planes with the color denoting the distance field while the white region is marked by the input geometry. The right panel shows the region of the computational grid where distance is calculated (marked in yellow) and the black region marks the flood-filled default value of -10^3. The gray region in the right panel has a default value of 10^2 and the distance is not calculated here as it is known a-priori to be outside the geometry.

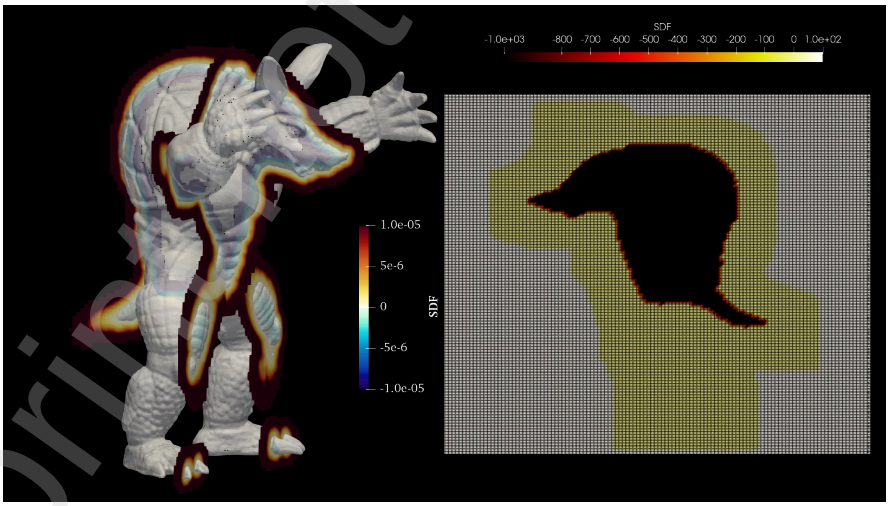

Figure 5: Signed Distance Field calculated using the software for the Stanford Armadillo (scaled) geometry.

4. Impact

The development of this software has a significant impact on enabling a pre-processing workflow for scale-resolving turbulent flow simulations around complex objects for problem sizes in the order of billions of computational grid cells. GenSDF is an open-source software that allows computational fluid dynamicists to seamlessly incorporate complex geometries into their solvers. The distributed memory implementation ensures scalability across multiple CPUs, eliminating memory limitations as a bottleneck and enabling its application to extremely large-scale simulations. While originally designed to integrate with the well-validated scale-resolving solver CaNS [8], GenSDF is highly adaptable and can be employed with other solvers requiring signed distance functions (SDFs) to handle complex objects.
The availability of GenSDF opens up avenues for exploring new research questions in computational fluid dynamics (CFD) and beyond. For example, it enables the study of turbulence and flow behavior around geometrically intricate structures at unprecedented scales. It also facilitates research into optimized solver designs that leverage SDF representations, allowing for novel investigations into mesh-free methods and hybrid grid-based techniques. Additionally, GenSDF's ability to handle arbitrary geometries with high scalability enables interdisciplinary applications, such as biomechanics and atmospheric simulations, that require accurate modeling of complex boundary interactions. The GenSDF software has been used in the following manuscript:

1. Patil, A. and García-Sánchez, C. Hydrodynamics of In-Canopy Flow in Synthetically Generated Coral Reefs Under Oscillatory Wave Motion, Journal of Geophysical Research: Oceans, (Under Review)

By streamlining the integration of complex geometries into CFD workflows, GenSDF significantly reduces the time and effort required for pre-processing, allowing researchers to focus on core simulations and analysis. This efficiency improvement accelerates studies on topics like flow-induced noise, drag reduction, and wake dynamics. Moreover, its scalability allows researchers to push the boundaries of resolution in turbulence modeling, leading to more accurate insights into multi-scale flow phenomena and better validation of theoretical models. The software's robustness and scalability have also encouraged its adoption in high-performance computing (HPC) environments, making complex simulations more accessible and routine for a broader audience.

5. Conclusions

GenSDF provides an open-source, scalable, and accurate method to generate signed-distance-field for complex objects over a Cartesian grid with variable vertical grid spacing. The development of this code was primarily motivated by the need for a scalable and distributed memory solution to generate signed-distance-field for computational fluid dynamics applications. Some of the main advantages of GenSDF are: (1) Low memory footprint (2) Scales over 100s of CPUs (3) Easily portable to other solver. The software also provides a detailed documentation on the source code and example usage. GenSDF is continually updated and tracked through the github repository where users can directly contribute.

Acknowledgements

A.P would like to thank the resources provided by the 3D Geoinformation Research Group for the testing and prototyping of this software. A.P and C.GS would also like to acknowledge that this research was carried out as a part of the EU-Project RefMAP. RefMAP has received funding from the Horizon Europe program under grant agreement No 101096698. The opinions expressed herein reflect the authors' views only. Under no circumstances shall the Horizon Europe program be responsible for any use that may be made of the information contained herein. During the preparation of this work the authors used Grammarly in order to spell and grammar check. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

References

[1] K. A. Goc, O. Lehmkuhl, G. I. Park, S. T. Bose, P. Moin, Large eddy simulation of aircraft at affordable cost: a milestone in computational fluid dynamics, Flow 1 (2021) E14.

[2] M. F. Ciarlatani, Z. Huang, D. Philips, C. Gorlé, Investigation of peak wind loading on a high-rise building in the atmospheric boundary layer using large-eddy simulations, Journal of Wind Engineering and Industrial Aerodynamics 236 (2023) 105408.

[3] C. García-Sánchez, D. Philips, C. Gorlé, Quantifying inflow uncertainties for cfd simulations of the flow in downtown oklahoma city, Building and environment 78 (2014) 118–129.

[4] J. Hochschild, C. Gorlé, Design and demonstration of a sensing network for full-scale wind pressure measurements on buildings, Journal of Wind Engineering and Industrial Aerodynamics 250 (2024) 105760.

[5] H. Jiang, L. Cheng, Large-eddy simulation of flow past a circular cylinder for reynolds numbers 400 to 3900, Physics of Fluids 33 (3) (2021).

[6] C. S. Peskin, The immersed boundary method, Acta numerica 11 (2002) 479–517.

[7] J. Yang, E. Balaras, An embedded-boundary formulation for large-eddy simulation of turbulent flows interacting with moving boundaries, Journal of computational Physics 215 (1) (2006) 12–40.

[8] P. Costa, A FFT-based finite-difference solver for massively-parallel direct numerical simulations of turbulent flows, Computers & Mathematics with Applications 76 (8) (2018) 1853–1862.

[9] A. Roosing, O. T. Strickson, N. Nikiforakis, Fast distance fields for fluid dynamics mesh generation on graphics hardware (2019). `arXiv:1903.00353`.
URL `https://arxiv.org/abs/1903.00353`

[10] D. Eberly, Distance between point and triangle in 3D, https://www.geometrictools.com/Documentation/DistancePoint3Triangle3.pdf, accessed: 2024-Nov-15 (2020).