SoftwareX 30 (2025) 102117

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

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] 3D Geoinformation Research Group, Faculty of Architecture and the Built Environment, Delft University of Technology, The Netherlands
[b] Wind Energy Section, Flow Physics and Technology Department, Faculty of Aerospace Engineering, Delft University of Technology, The Netherlands

ARTICLE INFO

Keywords:
Signed-distance-field
Computational fluid dynamics
MPI

ABSTRACT

This paper presents a highly efficient signed-distance field (SDF) generator designed specifically for computational fluid dynamics (CFD) workflows. Our approach integrates the Message Passing Interface (MPI) for parallel computing with the performance benefits of modern Fortran, enabling efficient and scalable signed distance field (SDF) computations for complex geometries. The algorithm focuses on localized distance calculations to minimize computational overhead, ensuring efficiency across multiple processors. An adjustable stencil width allows users to balance computational cost with the desired level of accuracy in the distance approximation. Additionally, GenSDF 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 ($\sim \mathcal{O}(10^7)$ triangulation faces) and computational grid points $\sim \mathcal{O}(10^9)$ 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.

Code metadata

Current code version	v0.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-24-00647
Permanent link to Reproducible Capsule	NA
Legal Code License	AGPL-3.0 license
Code versioning system used	git
Software code languages, tools, and services used	MPI + Fortran
Compilation requirements, operating environments & dependencies	gfortran and MPI library
Link to developer documentation/ manual	https://github.com/AkshayPatil1994/GenSDF
Support email for questions	a.l.patil@tudelft.nl

Software metadata

Current software version	v0.1
Permanent link to executables of this version	https://github.com/AkshayPatil1994/GenSDF
Legal Software License	AGPL-3.0 license
Computing platforms/Operating Systems	Linux, OS X, Unix-like
Installation requirements	MPI, gfortran, make
Support email for questions	a.l.patil@tudelft.nl

* Corresponding author.
E-mail address: a.l.patil@tudelft.nl (Akshay Patil).

https://doi.org/10.1016/j.softx.2025.102117
Received 30 November 2024; Received in revised form 26 February 2025; Accepted 27 February 2025
Available online 12 March 2025

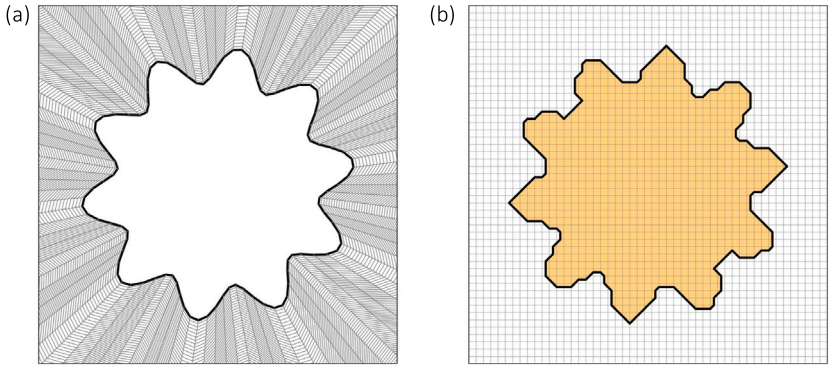

Fig. 1. (a) Body conforming mesh used in typical computational fluid dynamics applications (b) Immersed Boundary Method over a regular Cartesian grid.

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 [1–4]. While there exist multiple classes of methods that can be used to introduce complex objects within the flow, such as body-conforming grids as shown in Fig. 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.

While SDF generators are not new [9], the closed-source nature of existing solutions has limited accessibility for CFD applications. Additionally, as the problem size for scale-resolving simulation grows, the memory requirement for generating the SDF can drastically increase thus requiring a distributed-memory framework to efficiently parallelize the SDF generation. 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. In the following sections, we will first present a detailed description of the software and its performance, followed by some illustrative examples and benchmark results. We then summarize the discussion by presenting an impact statement and concluding remarks.

2. Software description

The *GenSDF* software is a general-purpose code developed to accurately and efficiently obtain the SDF over a Cartesian grid for an arbitrary triangulated geometry.

2.1. Software architecture

The *GenSDF* software is written in modern Fortran and is compatible with the Fortran 2003 standard and later [10], allowing for a simple yet computationally efficient framework to develop and maintain the code. The distributed memory parallelism is achieved through the MPI interface (compatible with MPI-4.0 standard and above), 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. Fig. 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 (henceforth referred to as B-Box in Fig. 2) since these points are known in advance to be outside the geometry. This B-Box methodology allows for correctly excluding the computationally expensive distance query for points that are known to be outside the geometry. The B-Box also includes a user-specified stencil width (s_b), where s_b is the number of computational grid points in the B-Box wall-normal direction. Additionally, s_b is also used to carry out a narrow-band distance calculation where it is defined as the number of grid points away from the geometric face, where the distance calculation is carried out.

After the B-Box is co-located in Step 2, it 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 calculations 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 a query point and a triangular face.
- Parallelize 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 width. 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 into 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 Fig. 4, 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 Fig. 4). Knowing the x, y, and z coordinates of the

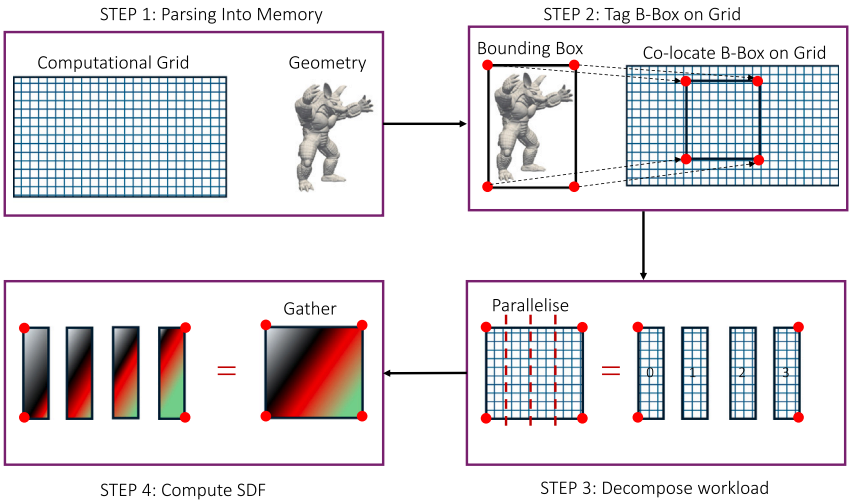

Fig. 2. Flowchart of the key steps in the code. In step 3, the numbers represent the MPI rank IDs. The flowchart presents a 2D example of the B-Box; however, the code works in 3D.

vertices, a face-local bounding box can be obtained readily as the computational grid (shown with the blue grid lines) constitutes 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 a well-established algorithm following Eberly [11]. The face-local bounding box circumvents the brute-force computational cost, which would otherwise scale as $\mathcal{O}(N_c \times N_f)$, where N_c is the number of computational grid points and N_f is the total number of triangular faces in the input geometry. The central idea is to reduce the pre-factor N_c by only considering the computational grid points in the immediate vicinity of the requisite face controlled by the s_b input parameter thus reducing the overall computational cost to $\sim \mathcal{O}(s_b \times N_f)$. For context $s_b \sim \mathcal{O}(10)$ while $N_c \sim \mathcal{O}(10^7)$, thus resulting in relatively large reduction in the computational cost associated with calculating the SDF when compared to the brute-force method. Fig. 3 presents a simple sketch considering $s_b = 1$ around the face (an edge in 2D) marked in red and the corresponding points considered for calculating the distance query marked in black- and purple-filled circles (closest point to the face). The thick-purple lines connecting the filled-black circles are chosen based on the value of s_b prescribed by the user in each lattice direction around the point of consideration in 3D. The pseudo-code of the algorithm is detailed below in Algorithmic listings 1–2.

A sign-unaware distance is calculated with the method proposed by Eberly [11], which further requires information from the underlying geometry to correctly tag the positive or negative sign associated with the point being inside or outside the geometry. Since the OBJ file format supports storing the vertex normals as shown in Fig. 3, the distance can be tagged to each query point by taking the inner product between the average vertex normals corresponding to the face and the segment connecting the query point and the face centre given by

$$L_{q_i}^{\text{tag}} = \text{sgn}(\hat{n}_f \cdot k_{q_i - o}), \tag{1}$$

where $L_{q_i}^{\text{tag}}$ is the sign tagged to the distance calculated for the respective query point q_i, sgn is the sign function that returns either a 1.0 or −1.0 (double-precision floating point), $\hat{n}_f$ is the vertex averaged normal vector associated with the face, and $k_{q_i - o}$ is the vector originating from the query point and pointing towards the face centre. Query points returning a positive $L_{q_i}^{\text{tag}}$ are tagged to be inside the geometry, while the query points that return a negative $L_{q_i}^{\text{tag}}$ are tagged to be outside as sketched in Fig. 3.

Algorithm 1 Compute the Minimum Distance Between a Point and a Triangle

Require: Point P, triangle $\triangle ABC$ with vertices A, B, C
Ensure: Minimum distance d between P and $\triangle ABC$

Compute edge vectors:
$E_0 \leftarrow B - A$
$E_1 \leftarrow C - A$
$V \leftarrow P - A$
Compute dot products:
$d_{00} \leftarrow E_0 \cdot E_0$
$d_{01} \leftarrow E_0 \cdot E_1$
$d_{11} \leftarrow E_1 \cdot E_1$
$d_{20} \leftarrow V \cdot E_0$
$d_{21} \leftarrow V \cdot E_1$
Compute barycentric coordinates:
$\text{denom} \leftarrow d_{00} d_{11} - d_{01}^2$
$v \leftarrow \frac{d_{11} d_{20} - d_{01} d_{21}}{\text{denom}}$
$w \leftarrow \frac{d_{00} d_{21} - d_{01} d_{20}}{\text{denom}}$
$u \leftarrow 1 - v - w$
if $u \geq 0 \wedge v \geq 0 \wedge w \geq 0$ **then**
    $P_c \leftarrow A + vE_0 + wE_1$
    return $d \leftarrow \|P - P_c\|$
end if
Check closest vertex cases:
if $v \leq 0 \wedge w \leq 0$ **then**
    return $d \leftarrow \|P - A\|$
else if $u \leq 0 \wedge v \leq 0$ **then**
    return $d \leftarrow \|P - C\|$
else if $u \leq 0 \wedge w \leq 0$ **then**
    return $d \leftarrow \|P - B\|$
end if
Check closest edge cases:
if $u \leq 0$ **then**
    $P_c \leftarrow \text{ClosestPointOnSegment}(P, B, C)$
else if $v \leq 0$ **then**
    $P_c \leftarrow \text{ClosestPointOnSegment}(P, C, A)$
else if $w \leq 0$ **then**
    $P_c \leftarrow \text{ClosestPointOnSegment}(P, A, B)$
end if
return $d \leftarrow \|P - P_c\|$

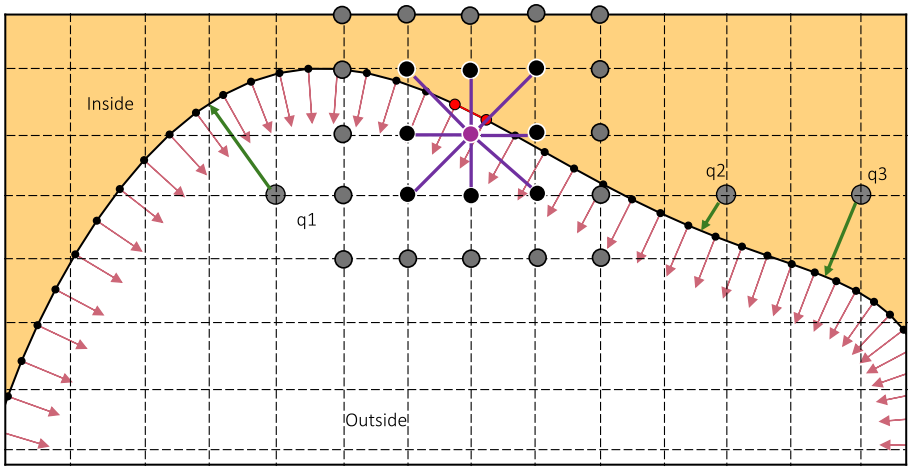

Fig. 3. 2D sketch of sign-tagging logic used in this implementation. The black solid line with black-filled circles marks the faces of the geometry where each segment corresponds to a face. The vertex normals originate at each vertex and point in the outward normal direction. The green arrows originating from the query points (q1, q2, and q3) are assumed to point towards the closest face on the geometry. The black-dashed lines mark the computational grid on which the query points are situated. The red edge connecting the red-filled circles correspond to the edge under consideration for the distance query while the black-filled circles are the computational grid points against which the distance needs to be calculated. The triangulation here i.e., the faces, are presented as isotropic in size, however, this is not a requirement for the algorithm. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

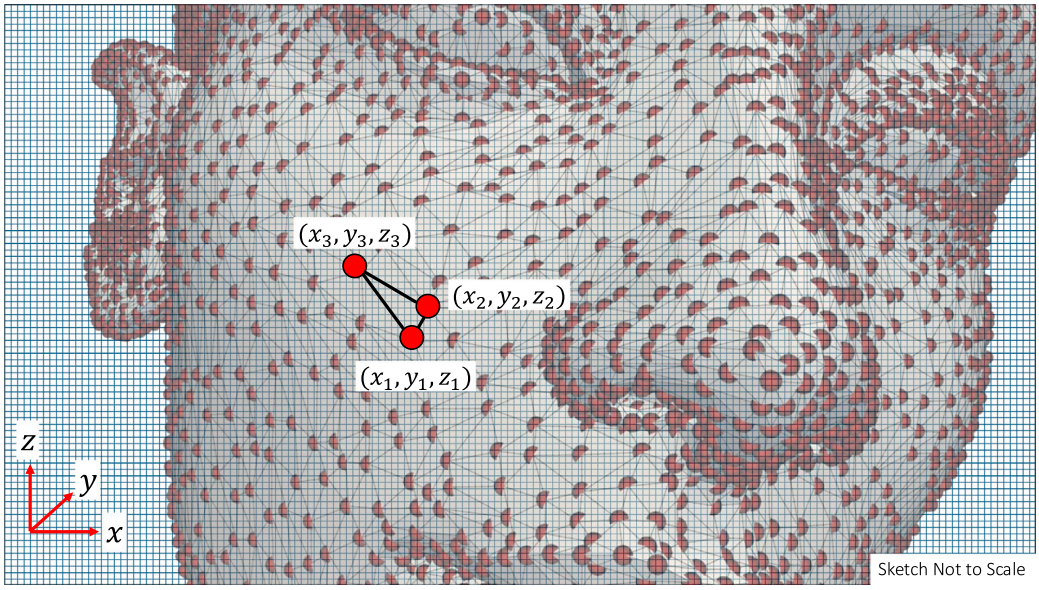

Fig. 4. Example sketch of the triangulated surface geometry overlaid onto the computational grid (2D example). The three filled-red circles correspond to the vertices of the triangular face denoted by the solid black lines.

Algorithm 2 Closest Point on a Line Segment

Require: Point P, segment endpoints X, Y
Ensure: Closest point P_c on segment XY
Compute projection parameter:
$t \leftarrow \frac{(P-X)\cdot(Y-X)}{(Y-X)\cdot(Y-X)}$
Clamp t **to** $[0, 1]$:
$t \leftarrow \max(0, \min(1, t))$
Compute closest point:
$P_c \leftarrow X + t(Y - X)$
return P_c

Since the user is allowed to specify a stencil width around the geometry where the distance is calculated, the default initialization for the SDF value does not correctly tag the values inside the geometry that should retain a negative value. During the initialization step, all points are originally tagged to be outside the geometry and assigned a large positive distance value specified by the user in the user input parameters. The effect of varying stencil width without correctly handling the internal values can be seen in Fig. 5 where the algorithm incorrectly assigns a positive default value inside the geometry. For increasing stencil width, such a situation can be potentially avoided, however, the computational cost scales with the stencil width, thus we make use of a simple 3D flood-fill algorithm that assigns a large negative value for internal values bounded by the SDF. This is a critical part of the algorithm as the SDF is used to identify the location of the solid interface (i.e., SDF = 0.0), thus without the final flood-fill process,

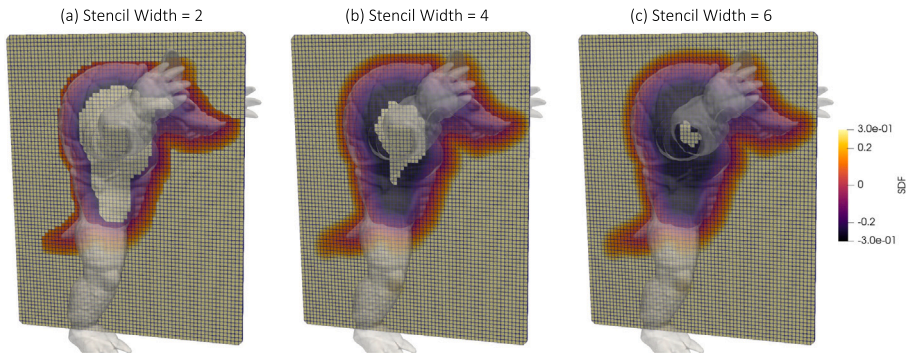

Fig. 5. Example case illustrating the effect of stencil width when flood-fill algorithm is not used post-signed-distance calculation. Size of the internal region with a default positive value decreases with increasing stencil width.

the CFD solver could potentially return multiple interface locations. 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.

This approach not only localizes 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 iterates over the total number of faces in the triangulated surface geometry, there are some limitations that must be considered for this particular algorithm. Geometries with long and slender triangles may render de-generate SDF, this is especially true for geometries that constitute a large collection of blocks (cubes such as buildings). Additionally, for cases where the underlying computational grid is relatively much finer compared to the surface triangulation, there can be instances where certain computational points render de-generate SDF. A simple fix for both scenarios is to refine the surface triangulation using a meshing tool or the instructions provided within the published repository.

2.3. Input requirements and performance

GenSDF is designed with the CFD user in mind and, like any other software, requires input parameters and data to work as detailed in this paper. Some of these requirements are:

1. Input Geometry: The geometry is required to consist of triangles and must be manifold and watertight. In cases where the computational grid spacing is larger than the triangulation of the geometry, water tightness is not a strict requirement. The geometry must be stored in the wavefront OBJ file format as it stores the vertex normal information that is central to the implementation considered in this study. Additionally, this file format is supported and used in many open-source software where the input geometry will be used for setting up scale-resolving simulations [12,13]. The algorithm enforces the coordinate axes x_i to be aligned such that $i = 1, 2, 3$ correspond to the streamwise (longest), spanwise, and vertical directions, respectively. The bounding box of the input geometry must be contained within the bounding box of the computational geometry either completely or partially. In case the input geometry is completely outside the bounds, the SDF will simply return a default value for all computational grid points.
2. Input Computational Grid: In addition to the input geometry, the computational grid over which the SDF is to be calculated needs to be provided. The code in its current form is directly compatible with one of the widely used CFD solvers CaNS developed by Costa [8]. However, since most scalable CFD solvers that would require the use of *GenSDF* have similar structure i.e., isotropic grid spacing in two directions, and non-isotropic grid in one direction; the grid can be ported to CaNS format. Specifically, the CaNS grid structure consists of two files, namely, a. `geometry.out`, and b. `grid.out`. These two files are assumed to be hosted in the `data` directory relative to where the executable is placed. The `geometry.out` file contains two lines with three columns of information where the first line contains the total number of grid points while the second line contains the length of the domain with respect to the computational grid origin (default origin 0,0,0) in the streamwise, spanwise, and vertical directions, respectively. The `grid.out` file contains five columns and n_z lines, where n_z is the number of grid points in the vertical direction. The first column is typically empty and has a dummy value of 0.0 while the second and third columns correspond to the cell centre and cell face locations in the vertical directions, respectively. The fourth and fifth columns correspond to the grid spacing for the cell centre and the cell face locations. Both the `geometry.out` and `grid.out` columns are separated by white space and do not contain a file header.
3. Input Parameters: A complete list of input parameters required in addition to the input geometry are detailed in Listing 1. Each line preceding with a '!' corresponds to a comment. The scalar value refers to the initial value set for all the points assumed to be outside the geometry. The real and int keywords correspond to floating point and integer data types, respectively. The non-uniform-grid boolean flag can either be set to 'T' or 'F' for True and False, respectively.

Listing 1: Input parameters required for the code with the default file name *parameters.in*

```
! Name and location of the inputfile
'data/armadillo_withnormals.obj'
! Scalar value (real), stencil width (int),
    progressbarsize (int)
100.0 4 10
! nx, ny, nz (Computational Grid)
512 128 128
! r0 (Origin of the computational grid)
0.0 0.0 0.0
! non_uniform_grid
F
```

To evaluate the accuracy and the performance of the software, we first consider the effect of increasing resolution for a sphere with a radius of $r_c = 0.05$ units centred at $(0.5, 0.5, 0.5)$. The background computational grid is discretized using three grid resolutions $r_c/\Delta = [12.8, 25.6, 51.2]$. Fig. 6(a) compares the results using *GenSDF* where the level-set (SDF = 0.0) is shown against the analytical geometry. With increasing grid resolution, the level-set converges towards the analytical geometry. For relatively complex geometry, the level-set accurately

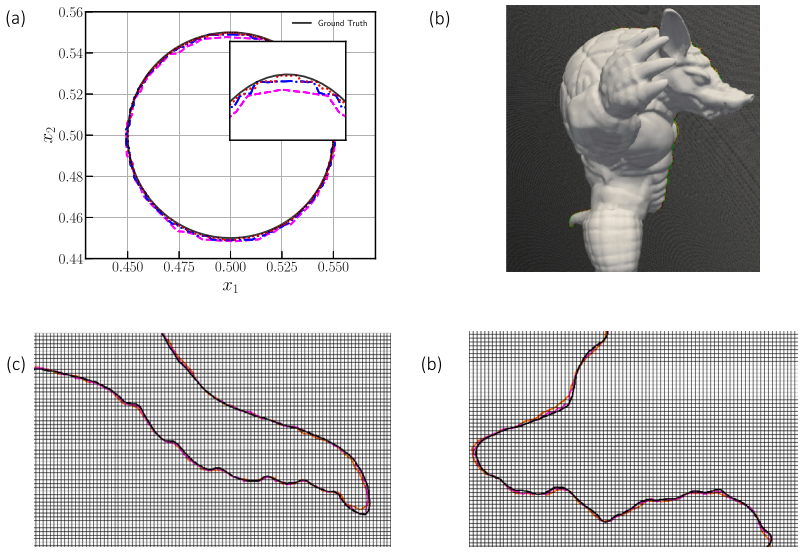

Fig. 6. (a) SDF level-set comparison against analytical geometry for varying computational grid resolution. The dashed-magenta, dash–dot-blue, and dotted-red lines correspond to $r_c/\Delta = [12.8, 25.6, 51.2]$, respectively, while the solid black line represents the ground truth. (b–d) Comparison of the level-set against a relatively complex geometry with an effective resolution of $\Delta = [1/512, 1/1024]$ marked using orange and magenta solid lines respectively, while the solid black line represents the exact geometry. The background grid corresponds to a resolution of $\Delta = 1/1024$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 1
Computational time required for the three geometries tested using *GenSDF* as a function of the number of faces (N_f) in the geometry for constant background grid resolution.

Geometry name	N_f	$\Delta = 1.0/512$	$\Delta = 1.0/1024$
Orion	51 736	1.64 s	2.46 s
Stanford Lucy	99 970	1.63 s	1.55 s
Stanford Dragon	249 882	4.90 s	5.21 s

captures the underlying geometry even for a relatively coarse resolution as shown using the orange solid line in Fig. 6(b–d). To demonstrate the performance of the algorithm, we consider three cases namely the Orion [14], the Stanford Dragon, and Lucy [15] with two grid sizes with a resolution of $\Delta = [1.0/512, 1.0/1024]$. The original geometry was re-scaled to fit a bounding box (of the geometry) with size $x_{\min} = (0.5, 0.5, 0.5)$, $x_{\max} = (1.0, 1.0, 1.0)$ within the cube (background computational grid) of side length 1.5 units. Table 1 lists the time required to generate the SDF using 8 cores on a single socket CPU. It is clear to see that despite the relatively simple computational structure, the code is fast and accurate for complex geometries.

2.4. Current limitations and future vision

GenSDF in its current form provides a foundational framework to further develop a computational tool to generate SDFs for CFD applications. Consequently, in this section we will briefly present the future vision for improving *GenSDF*.

Short-Term Milestones

1. In its current form, *GenSDF* uses a streamwise slab-type decomposition [16] which can be a limiting aspect of the implementation. A 2D pencil decomposition could further improve the scalability of *GenSDF* and will be considered in the next update cycle.
2. Finally, the underlying computational grid is assumed to be in a specific format that is not universal and does not allow non-isotropic grid spacing in the streamwise and spanwise directions. This can be relatively easily fixed by defining a universal input grid format that is agnostic to the CFD solver and will be considered in the update cycle.

Long-Term Milestone

1. With GPGPU computing hardware continually improving over the past decade, both serial- and multi-GPU support could greatly improve the usability and performance when compared to the current MPI-based CPU version of the code. This aspect will require re-writing relatively major parts of the code and thus will be considered as one of the long-term milestones.

3. Illustrative examples

Despite some of the shortcomings mentioned above, *GenSDF* has been used at extreme scale simulation setups as illustrated in the following section. By default, the code assumes that the grid is staggered such that the locations of u, v, w, (velocities in streamwise, spanwise, and vertical directions, respectively) and p (pressure and any other cell-centre parameters of interest i.e., scalars) 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 arrays initialized by the software. This is important when the problem size is large and the user requires a minimal memory estimate to schedule the SDF generation on HPCs.

Figs. 7, 8, and 9 depict the suitability of *GenSDF* for multi-scale and complex geometries. Fig. 7 depicts the SDF generated for an artificially generated coral reef containing a total of 14.85 Million vertices and 29.75 Million faces on a background non-isotropic computational grid containing 1.2 Billion grid points. The Stanford dragon exhibits relatively sharp curvature and slender cross-sections as seen in Fig. 8 and *GenSDF* is able to sufficiently capture the details without introducing a large computational overhead (see Table 1). A relatively extreme-scale case is depicted in Fig. 9 with 2 Billion grid points and 25 Million triangulation faces where the grey colour marks the level-set of 0 representing the solid boundary of the buildings and the colour marks the distance field.

4. Impact

The development of this software has a significant impact on enabling a pre-processing workflow for scale-resolving turbulent flow

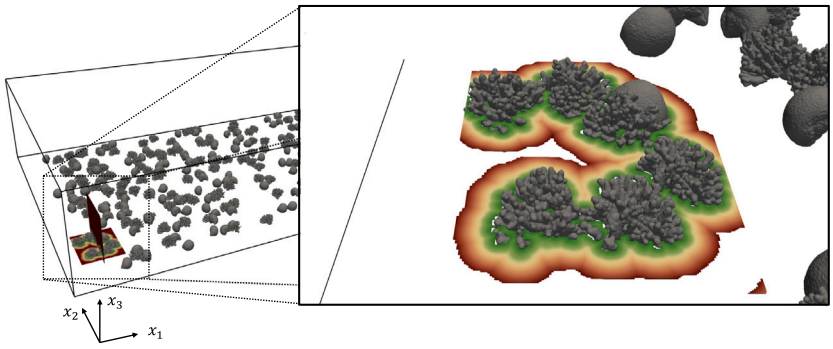

Fig. 7. SDF generated for a multi-scale artificially generated coral reef (Results from [17]).

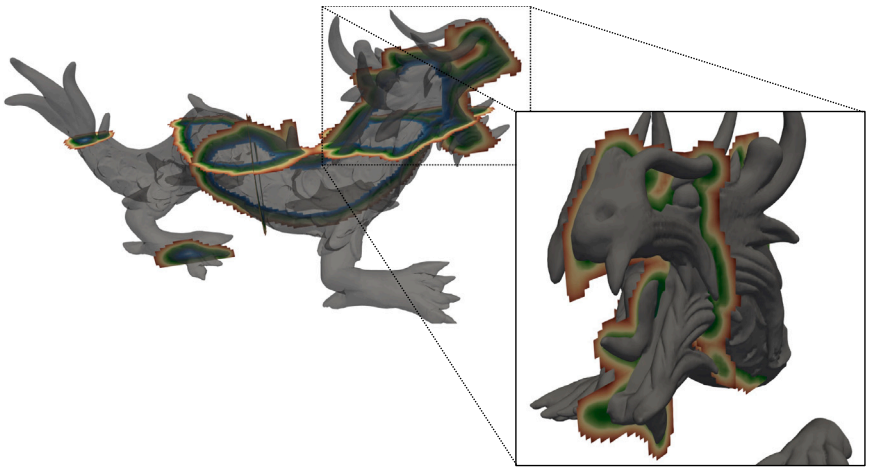

Fig. 8. SDF generated for complex geometry corresponding to the Stanford Dragon [15].

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 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 behaviour around geometrically intricate structures at high resolution. 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 modelling of complex boundary interactions. The GenSDF software has been used in the following manuscripts:

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, (submitted 2024-08-09, Under Review)
2. Patil, A. and García-Sánchez, C. A Comparative Hydrodynamic Characterization of the Flow Through Regular and Stochastically Generated Synthetic Coral Reefs Over Flat Topography, Coastal Dynamics 2025 Meeting, Aveiro, Portugal, (submitted 2025-01-15), Pre-Print: https://arxiv.org/abs/2501.15237

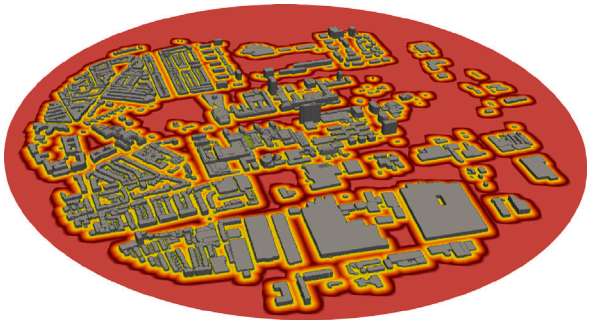

Fig. 9. SDF generated for the Delft University of Technology central campus with a background grid size of 2 Billion grid points and 25 Million triangulation faces for the underlying geometry. Only a section of the grid is shown in the figure.

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 modelling, 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 a 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 a 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 different CFD solvers. The software also provides detailed documentation of the source code and example usage. GenSDF is continually updated and tracked through the GitHub repository where users can directly contribute.

CRediT authorship contribution statement

Akshay Patil: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Udhaya Chandiran Krishnan Paranjothi:** Writing – review & editing, Software, Methodology, Formal analysis, Conceptualization. **Clara García-Sánchez:** Writing – review & editing, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Akshay Patil and Clara Garcia-Sanchez report financial support was provided by Horizon Europe. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

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. The authors express their sincere gratitude to the two anonymous reviewers for their insightful comments and constructive suggestions, which have significantly contributed to the improvement of this manuscript.

References

[1] Goc KA, Lehmkuhl O, Park GI, Bose ST, Moin P. Large eddy simulation of aircraft at affordable cost: a milestone in computational fluid dynamics. Flow 2021;1:E14. http://dx.doi.org/10.1017/flo.2021.17.

[2] Ciarlatani MF, Huang Z, Philips D, Gorlé C. Investigation of peak wind loading on a high-rise building in the atmospheric boundary layer using large-eddy simulations. J Wind Eng Ind Aerodyn 2023;236:105408. http://dx.doi.org/10.1016/j.jweia.2023.105408.

[3] García-Sánchez C, Philips D, Gorlé C. Quantifying inflow uncertainties for CFD simulations of the flow in downtown Oklahoma City. Build Environ 2014;78:118–29. http://dx.doi.org/10.1016/j.buildenv.2014.04.013.

[4] Hochschild J, Gorlé C. Design and demonstration of a sensing network for full-scale wind pressure measurements on buildings. J Wind Eng Ind Aerodyn 2024;250:105760. http://dx.doi.org/10.1016/j.jweia.2024.105760.

[5] Jiang H, Cheng L. Large-eddy simulation of flow past a circular cylinder for Reynolds numbers 400 to 3900. Phys Fluids 2021;33(3). http://dx.doi.org/10.1063/5.0041168.

[6] Peskin CS. The immersed boundary method. Acta Numer 2002;11:479–517. http://dx.doi.org/10.1017/S0962492902000077.

[7] Yang J, Balaras E. An embedded-boundary formulation for large-eddy simulation of turbulent flows interacting with moving boundaries. J Comput Phys 2006;215(1):12–40. http://dx.doi.org/10.1016/j.jcp.2005.10.035.

[8] Costa P. A FFT-based finite-difference solver for massively-parallel direct numerical simulations of turbulent flows. Comput Math Appl 2018;76(8):1853–62. http://dx.doi.org/10.1016/j.camwa.2018.07.034.

[9] Roosing A, Strickson OT, Nikiforakis N. Fast distance fields for fluid dynamics mesh generation on graphics hardware. 2019, arXiv:1903.00353.

[10] Fortran 2003 Draft International Standard. URL https://wg5-fortran.org/N1601-N1650/N1602.pdf.

[11] Eberly D. Distance between point and triangle in 3D. 2020, URL https://www.geometrictools.com/Documentation/DistancePoint3Triangle3.pdf. [Accessed 15 November 2024].

[12] Peters R, Dukai B, Vitalis S, van Liempt J, Stoter J. Automated 3D reconstruction of LoD2 and LoD1 models for all 10 million buildings of the Netherlands. Photogramm Eng Remote Sens 2022;88:165–70. http://dx.doi.org/10.14358/PERS.21-00032R2.

[13] Pađen I, García-Sánchez C, Ledoux H. Towards automatic reconstruction of 3D city models tailored for urban flow simulations. Front Built Environ 2022;8:899332. http://dx.doi.org/10.3389/fbuil.2022.899332.

[14] Orion capsule, NASA 3D resources. 2025, URL https://nasa3d.arc.nasa.gov/detail/orion-capsule. [Accessed January 2025].

[15] The stanford 3D scanning repository. 2025, URL http://graphics.stanford.edu/data/3Dscanrep/. [Accessed January 2025].

[16] Li N, Laizet S. 2Decomp & FFT-a highly scalable 2D decomposition library and FFT interface. In: Cray user group 2010 conference. 2010, p. 1–13, URL https://www.turbulencesimulation.com/uploads/5/8/7/2/58724623/2010_laizet_nag.pdf.

[17] Patil A, García-Sánchez C. A comparative hydrodynamic characterization of the flow through regular and stochastically generated synthetic coral reefs over flat topography. 2025, arXiv:2501.15237.