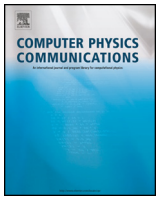
Computer Physics Communications 286 (2023) 108665

Contents lists available at ScienceDirect

Computer Physics Communications

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



Geometric multigrid method for solving Poisson's equation on octree grids with irregular boundaries ☆,☆☆

Jannis Teunissen [a,*], Francesca Schiavello [b]

[a] Centrum Wiskunde & Informatica, Amsterdam, the Netherlands
[b] UKRI, STFC, Hartree Centre, United Kingdom

ARTICLE INFO

Article history:
Received 25 August 2022
Received in revised form 6 January 2023
Accepted 10 January 2023
Available online 20 January 2023

Keywords:
Multigrid
Irregular boundary
Poisson equation
Adaptive mesh refinement
Level set function

ABSTRACT

A method is presented to include irregular domain boundaries in a geometric multigrid solver. Dirichlet boundary conditions can be imposed on an irregular boundary defined by a level set function. Our implementation employs quadtree/octree grids with adaptive refinement, a cell-centered discretization and pointwise smoothing. Boundary locations are determined at a subgrid resolution by performing line searches. For grid blocks near the interface, custom operator stencils are stored that take the interface into account. For grid block away from boundaries, a standard second-order accurate discretization is used. The convergence properties, robustness and computational cost of the method are illustrated with several test cases.

New version program summary

Program Title: Afivo
CPC Library link to program files: https://doi.org/10.17632/5y43rjdmxd.2
Developer's repository link: https://github.com/MD-CWI/afivo
Licensing provisions: GPLv3
Programming language: Fortran
Journal reference of previous version: Comput. Phys. Commun. 233 (2018) 156–166. https://doi.org/10.1016/j.cpc.2018.06.018
Does the new version supersede the previous version?: Yes.
Reasons for the new version: Add support for internal boundaries in the geometric multigrid solver.
Summary of revisions: The geometric multigrid solver was generalized in several ways: a coarse grid solver from the Hypre library is used, operator stencils are now stored per grid block, and methods for including boundaries via a level set function were added.
Nature of problem: The goal is to solve Poisson's equation in the presence of irregular boundaries that are not aligned with the computational grid. It is assumed these irregular boundaries are defined by a level set function, and that a Dirichlet type boundary condition is applied. The main applications are 2D and 3D simulations with octree-based adaptive mesh refinement, in which the mesh frequently changes but the irregular boundaries do not.
Solution method: A geometric multigrid method compatible with octree grids is developed, using a cell-centered discretization and point-wise smoothing. Near irregular boundaries, custom operator stencils are stored. Line searches are performed to locate interfaces with sub-grid resolution. To increase the methods robustness, this line search is modified on coarse grids if boundaries are otherwise not resolved. The multigrid solver uses OpenMP parallelization.

1. Introduction

A common elliptic partial differential equation (PDE) is Poisson's equation

$$\nabla \cdot (a(\boldsymbol{x}) \nabla \phi) = g, \tag{1}$$

☆ The review of this paper was arranged by Prof. N.S. Scott.
☆☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).
* Corresponding author.
E-mail address: jannis.teunissen@cwi.nl (J. Teunissen).
https://doi.org/10.1016/j.cpc.2023.108665

where the right-hand side g and coefficient $a(\boldsymbol{x})$ are given and ϕ has to be obtained given certain boundary conditions. Equation (1) can numerically be solved with a variety of techniques, for example using fast Fourier transforms (FFTs), cyclic reduction, direct sparse solvers, (preconditioned) Krylov methods, multipole methods and multigrid methods, see e.g. [1,2]. The most suitable method depends on the type of computational grid, the boundary conditions, the spatial variation in $a(\boldsymbol{x})$, and the available computational hardware. Our goal is to develop an efficient geometric multigrid scheme for the following case:

- There are irregular Dirichlet boundary conditions. These boundaries are located inside the computational domain, but they are not aligned with the numerical grid.
- Equation (1) has to be solved several times for different right-hand sides, but with the same irregular boundaries.
- The coefficient $a(\boldsymbol{x})$ is constant.
- The computational grid is a quadtree/octree mesh that is frequently adapted, so that it is desirable to have a (mostly) matrix-free method.

Multigrid methods [3–5] can be used to solve equations like (1) with great efficiency. The main idea is to iteratively damp the error on a hierarchy of grids with a smoother. On coarse grids, the long-wavelength components of the error are damped, and on fine grids the short-wavelength components. Information from different grid levels is combined via prolongation (i.e., interpolation) to finer grids, and via restriction to coarser grids. Multigrid methods can have a computational cost linear in the number of unknowns, which is ideal. We focus on geometric multigrid (GMG) methods, which solve problems on a given hierarchy of numerical grids. In contrast, algebraic multigrid (AMG) methods can be used to solve more general linear systems. This flexibility is attractive for problems with irregular boundaries, but the cost of AMG methods is generally higher [6].

Considerable work has been done on solving equations like (1) with geometric multigrid in the presence of irregular boundaries. We briefly mention some relevant work below. In [7], a matrix-free geometric multigrid was developed that could handle irregular boundaries, which were tracked by a level-set function. Node-centered grids were considered from 1D to 3D, and interpolation was performed by locally solving the elliptic PDE for a grid point. Besides Dirichlet boundaries, the authors also consider discontinuities in the PDE coefficient $a(\boldsymbol{x})$. In [8], a geometric multigrid scheme was presented to apply irregular Dirichlet boundary conditions on AMR grids, with a focus on self-gravitating astrophysical flows. On the fine grid boundaries were described by a mask, leading to a staircase pattern. The authors discuss a trade-off between a first and second order accurate scheme for representing boundaries, with the second order scheme suffering from a lack of convergence on coarse grids when boundaries are not well resolved. In [9], a multigrid solver was presented for elliptic and parabolic problems on quadtree and octree grids. A node-centered discretization was used and irregular boundaries were described by a level-set function. So-called ghost values were obtained by third-order extrapolation near refinement boundaries. In [10], a geometric multigrid solver compatible with irregular Neumann boundaries was presented. The boundaries were represented by a staircase pattern on the fine grid. The authors highlight the importance of a conservative discretization, which is also referred to as a compatibility condition, see e.g. [11,12]. Interpolation was avoided near boundaries, leading to a first order accurate method. In [13], a cut-cell geometric multigrid solver was presented supporting both Dirichlet and Neumann boundary conditions, with a focus on the efficient simulation and visualization of incompressible flow. A cell-centered discretization was used, and a constant (zeroth-order) prolongation scheme. The method was shown to be first order accurate for Dirichlet boundaries and second order accurate for Neumann boundaries.

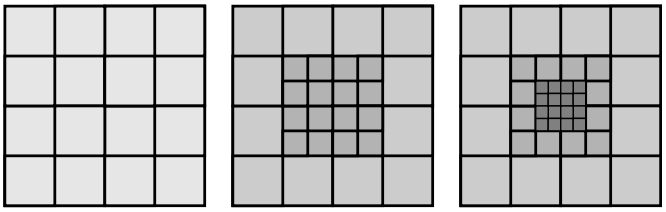


Fig. 1. Illustration of a quadtree grid. The squares indicate grid blocks, which each contain $N \times N$ cells (not indicated). From left to right, refinement is added around the center.

The main novelty of the method presented here is that it combines the following aspects:

- The flexible handling of different geometries via a level-set function.
- An (approximately) second-order accurate cell-centered discretization that is compatible with adaptive mesh refinement (AMR) on quadtree/octree grids.
- The use of a line search method to accurately locate interfaces.
- A correction for unresolved boundaries on coarse grids.
- An efficient open-source implementation, with custom stencils only stored for grid blocks that contain a boundary.

2. Multigrid method without irregular boundaries

Below, the basis of multigrid method used in this paper is briefly introduced. The extension to irregular boundaries is discussed in section 3.

2.1. Mesh

We consider so-called octree meshes, see Fig. 1. In our implementation, which is based on the `afivo` framework [12], such a mesh consists of blocks of N^D cells, where D denotes the problem dimension. These blocks can be refined by halving the grid spacing, so that 2^D refined child blocks cover a parent block. Nearby blocks are refined, if necessary, to ensure that adjacent blocks differ by at most one refinement level. A tree fulfilling such a condition is called 2:1 balanced.

Octree meshes balance adaptivity and computational efficiency. Because each block has the same shape, computations, communication and mesh refinement can be implemented rather efficiently. We use a cell-centered approach, in which the solution ϕ and right-hand side of equation (1) are defined at cell centers, and the components of $\nabla\phi$ are defined at cell faces.

2.2. Geometric multigrid method

The algorithms presented here for irregular boundaries extend the geometric multigrid solver of the Afivo framework [12]. This solver implements the Full Approximation Scheme (FAS), in which the solution ϕ is approximated on all grid levels. A brief overview is given below; further details can be found in [12] and in [14], which describes an MPI-parallel version.

Operators A standard finite difference discretizations of the Laplacian is used, with 3, 5 and 7-point numerical stencils in 1D, 2D and 3D, respectively. If the grid spacing Δx is constant and there are no boundaries, a second order accurate discretization of equation (1) in 1D is given by

$$\nabla^2 \phi_i = \frac{1}{\Delta x}\left(\frac{\phi_{i+1}-\phi_i}{\Delta x} - \frac{\phi_i - \phi_{i-1}}{\Delta x}\right) = g_i, \tag{2}$$

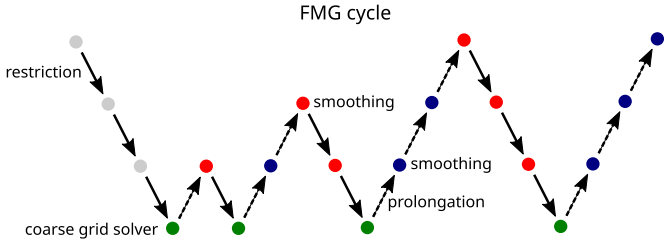


Fig. 2. Illustration of the FMG cycle for a grid with four levels.

where g is the right-hand side. The residual for an approximate solution ϕ^* is defined as

$$r = g - \nabla^2 \phi^*. \tag{3}$$

Smoother Gauss-Seidel red-black (GSRB) smoothers are used. The unknowns are first divided into red and black groups, in a checkerboard fashion. Equations like (2) can then be solved in parallel for one group, assuming the other group's values stay fixed. For equation (2), this results in

$$\phi_i = \frac{1}{2}\left(\phi_{i+1} + \phi_{i-1} - \Delta x^2 g\right).$$

The error is damped by alternatingly solving for the red and black groups.

Prolongation and restriction Prolongation is the transfer of coarse-grid corrections to a finer grid, which is a key part of a geometric multigrid method. Standard (bi/tri)linear interpolation is here used, also when irregular boundaries are present. Restriction is the transfer of information to a coarser grid. This is implemented by taking the average of the 2^D fine-grid cells covering a coarse grid cell.

Multigrid cycle A standard V-cycle and full multigrid (FMG) cycle are implemented, see Fig. 2. The V-cycle goes from fine to coarse, and then back to fine. The FMG cycle iteratively performs V-cycles from the coarsest grid up to the finest grid. Although FMG cycles are more expensive than V-cycles, they can guarantee a reduction of the residual that is independent of the problem size [4]. This results in the ideal $\mathcal{O}(N)$ computational cost of FMG, where N is the number of unknowns.

Every time a grid level is visited in a cycle, smoothing is performed. In the upward part of a cycle, N_{up} smoothing steps are performed before prolongation, and in the downward part of a cycle, N_{down} smoothing steps performed before restriction. We here use $N_{\text{up}} = N_{\text{down}} = 2$. The handling of the coarse grid is discussed in section 3.5.

Ghost cells When performing multigrid on an adaptive mesh, it is convenient to extend grid blocks with a layer of ghost cells. It is important that the ghost cells near refinement boundaries are filled in such a way that the fine-grid discretization is consistent with the underlying coarse grid. We here follow the same approach as in [12]. The basic idea is that ghost cells are filled in such a way that the coarse and averaged fine 'flux' across the refinement boundary (e.g., $\partial_x \phi$) agree.

3. Implementation of boundaries

3.1. Level set function

Internal boundaries are here defined by the zero contour of a level set function (LSF) [15,16]:

$$f(\boldsymbol{x}) = 0.$$

For example, a spherical boundary of radius R centered at $\boldsymbol{x}_c$ can be described by

$$f(\boldsymbol{x}) = ||\boldsymbol{x} - \boldsymbol{x}_c|| - R. \tag{4}$$

In this case, the LSF is a signed distance function, with a negative sign inside the sphere. Examples of other LSFs are given in section 4.

3.2. Distance computation

For geometric multigrid, it is important that the locations of boundaries (i.e., roots of the LSF) agree well between grid levels. We therefore use a line search method to locate boundaries at a sub-grid resolution.

Let $\boldsymbol{a}$ denote a start point, e.g., the center of a grid cell, and $\boldsymbol{b}$ a neighboring point. We want to know if there is a root in the LSF on the line segment from $\boldsymbol{a}$ to $\boldsymbol{b}$, and if so, how far this root is from $\boldsymbol{a}$. This information is here stored in a single value d, which denotes the relative distance to the boundary. If there is no boundary between $\boldsymbol{a}$ and $\boldsymbol{b}$, $d = 1$. Otherwise, if there is root at $\boldsymbol{x}_0$, d is given by

$$d = ||\boldsymbol{x}_0 - \boldsymbol{a}||/||\boldsymbol{b} - \boldsymbol{a}||. \tag{5}$$

The procedure for locating roots is illustrated in Fig. 3. If $f(\boldsymbol{a}) \times f(\boldsymbol{b}) \leq 0$, bisection is used to locate the root $\boldsymbol{x}_0$ between $\boldsymbol{a}$ and $\boldsymbol{b}$, with a relative tolerance of ϵ_{tol}. Otherwise, a bracket for the potential root first has to be determined. We use golden section search to minimize $f(\boldsymbol{x}) \times f(\boldsymbol{a})$ on the line from $\boldsymbol{a}$ to $\boldsymbol{b}$. As soon as $f(\boldsymbol{x}) \times f(\boldsymbol{a}) \leq 0$, bisection is again applied on the interval between $\boldsymbol{a}$ and $\boldsymbol{x}$. If this condition is not met within a given number of iterations, corresponding to the same relative tolerance ϵ_{tol}, it is assumed there is no boundary.

Note that if there are two roots on the interval between $\boldsymbol{a}$ and $\boldsymbol{b}$, the bracket search will eliminate the one farthest from $\boldsymbol{a}$. If there are three or more roots, it is not guaranteed that the above procedure finds root closest to $\boldsymbol{a}$. By default, we use a small relative tolerance of $\epsilon_{\text{tol}} = 10^{-8}$.

3.3. Discretization of Laplacian with boundaries

When there are irregular boundaries, the distances between an unknown ϕ_i and neighboring values are no longer fixed. The numerical Laplacian of equation (2) can then be generalized to

$$\nabla^2 \phi_i = \frac{2}{(d_{i+1} + d_{i-1})\Delta x}\left(\frac{\phi_{i+1} - \phi_i}{d_{i+1}\Delta x} - \frac{\phi_i - \phi_{i-1}}{d_{i-1}\Delta x}\right) = g_i, \tag{6}$$

where $0 < d_j \leq 1$ denotes the relative distances from ϕ_i to neighboring values ϕ_j, see section 3.2. Note that in the above notation the ϕ_j (for $j \neq i$) do not always correspond to unknowns on the grid. For example, if there is a boundary between cell i and $i+1$, then ϕ_{i+1} will correspond to a boundary value ϕ_b. If the corresponding term is moved to the right-hand side, equation (6) becomes

$$\frac{2}{(d_{i+1} + d_{i-1})\Delta x}\left(\frac{-\phi_i}{d_{i+1}\Delta x} - \frac{\phi_i - \phi_{i-1}}{d_{i-1}\Delta x}\right) = g_i - \frac{2\phi_b}{(d_{i+1} + d_{i-1})d_{i+1}\Delta x^2}.$$

Equation (6) is a non-symmetric discretization that was used before in e.g. [17–19]. Near boundary points this discretization is first order ($\mathcal{O}(\Delta x)$) accurate, because the second derivative is not evaluated at the center of the two gradient terms. However, if the number of boundary cells is small the global error can still be approximately second order accurate [18,19].

The extension of equation (6) to multiple dimensions is straightforward, with the same type of terms appearing for each dimension. For example, in 2D, the Laplacian can be written as

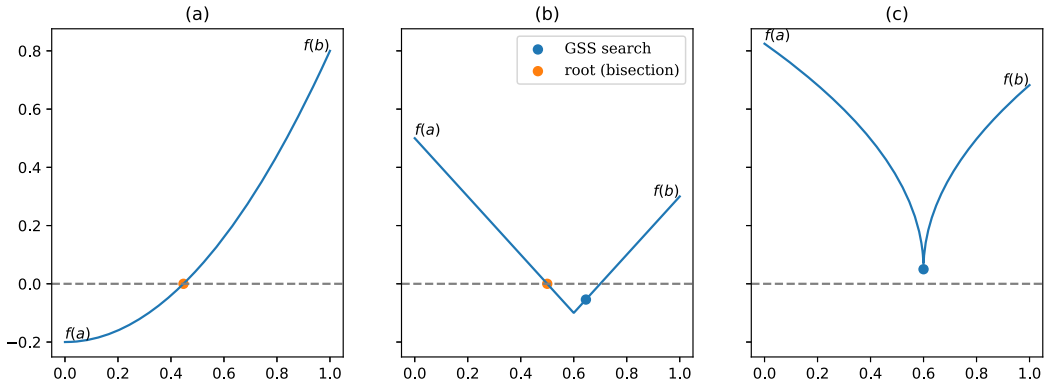


Fig. 3. Illustration of the procedure for locating roots in the level set function between two points $\boldsymbol{a}$ and $\boldsymbol{b}$. Three cases are illustrated: (a) The root is already bracketed, so bisection is directly applied; (b) A bracket is determined using Golden section search, then bisection is applied; (c) No bracket is found, and thus also no root.

$$\nabla^2\phi_{i,j} = \frac{2}{(d_{i+1,j}+d_{i-1,j})\Delta x}\left(\frac{\phi_{i+1,j}-\phi_{i,j}}{d_{i+1,j}\Delta x} - \frac{\phi_{i,j}-\phi_{i-1,j}}{d_{i-1,j}\Delta x}\right) + \frac{2}{(d_{i,j+1}+d_{i,j-1})\Delta x}\left(\frac{\phi_{i,j+1}-\phi_{i,j}}{d_{i,j+1}\Delta x} - \frac{\phi_{i,j}-\phi_{i,j-1}}{d_{i,j-1}\Delta x}\right) = g_{i,j}. \quad (7)$$

3.4. Prolongation

Standard (bi/tri)linear prolongation is used, also when boundaries are present. We did experiment with a custom prolongation scheme, in which a linear function was constructed between the nearest $D+1$ neighbors and/or boundaries, but this scheme did typically not lead to faster convergence.

3.5. Coarse grid solver

At the coarsest grid there are essentially two options. The first is to apply the same smoother as on other grid levels. However, depending on the size and geometry of the coarse grid, it could take a large number of smoothing steps to achieve a desired reduction of the residual. Therefore, we here solve the coarse grid equations using a different multigrid solver, provided by the Hypre library [20].

The coarse grid is frequently visited in an FMG cycle, see Fig. 2. It is therefore important to keep the computational cost of the coarse grid solver as low as possible. For this reason, we by default use Hypre's PFMG solver, which is a parallel semicoarsening multigrid solver that uses pointwise smoothing [21,22]. In 1D, the PFMG solver is not available and we use Hypre's PCG solver instead. Hypre's default tolerance of 10^{-6} is used for these solvers.

3.6. Implementation aspects

Below, we provide information on implementation aspects relevant for the computational efficiency of the method.

3.6.1. Boundary detection

The line search for boundaries described in section 3.2 can be expensive. For computational efficiency, we only perform such a search for cells that are sufficiently close to a boundary. The distance to a boundary can be approximated by the ratio $|f(\boldsymbol{x})|/||\nabla f(\boldsymbol{x})||$, which for a linear function $f(\boldsymbol{x})$ would be exact. This inspires the following condition for a potential boundary, evaluated at every cell center:

$$|f(\boldsymbol{x})| < L \times ||\nabla f(\boldsymbol{x})||, \quad (8)$$

where L should be proportional to the grid spacing Δx. For the tests presented in section 4 we use $L = 1.5 \times \sqrt{D}\Delta x$, with D the problem dimension. The components of $\nabla f(\boldsymbol{x})$ are computed numerically using central differencing.

Note that if $||\nabla f(\boldsymbol{x})||$ varies rapidly near boundaries, a larger safety factor than the $1.5 \times \sqrt{D}$ used above might be necessary. One way to avoid this is to use a LSF that is (approximately) a signed distance function, such as equation (4).

3.6.2. Storing stencils and boundary information

In our implementation, boundary information and numerical stencils are stored per grid block. For grid blocks without boundaries, stencils are constant, so they can be stored compactly. If there is a boundary passing through the grid block, the following information is stored per grid cell:

- The relative distances d_i to boundaries from the cell center to neighboring cell centers, according to equation (5). These distances are only stored for cells with at least one adjacent boundary.
- Operator stencil coefficients. For example, in 2D, five values have to be stored per cell for the operator in equation (7). Furthermore, the sum of the stencil coefficients that were moved to the right-hand side is stored, so that the value imposed at the boundary can be changed.

3.6.3. Unresolved LSF roots on coarse grids

If an irregular boundary has a small spatial extent in two of its dimensions, it might not be detected on a coarse grid by the line search described in section 3.6.1. An example is shown in Fig. 4. A boundary that is not detected on the coarse grid can lead to convergence issues. We therefore perform an additional boundary search for grid cells that satisfy the following conditions:

- Equation (8) holds, indicating there is a nearby boundary.
- No boundaries are detected between the cell's center and the neighboring cell centers.
- The grid spacing Δx is larger than a user-defined threshold $w_{\min}$.

For these cells, gradient descent is performed in the direction in which f goes to zero, starting from the cell's center ($\boldsymbol{a}$). At most $\Delta x/w_{\min}$ steps are performed with a step size $w_{\min}$. If after one of these steps, a location $\boldsymbol{x}$ is found such that $f(\boldsymbol{a})f(\boldsymbol{x}) \leq 0$, a line search is performed between $\boldsymbol{a}$ and $\boldsymbol{x}$ to determine the relative distance d to the boundary (normalized to the grid spacing). This relative distance is then used in the direction of the neighboring cell closest to $\boldsymbol{x}$, as illustrated in Fig. 4. The resulting discretization does not accurately represent the unresolved object, but this discretization is only used on coarse grids, so it does not affect the converged fine-grid solution.

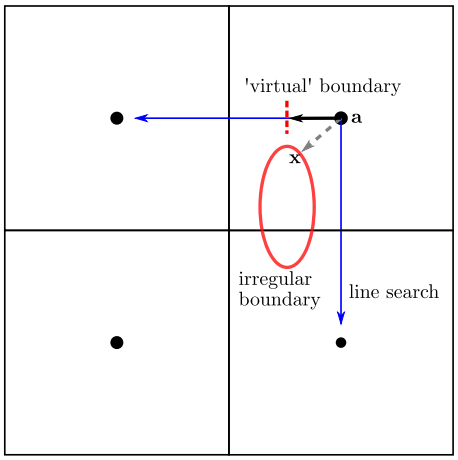


Fig. 4. Illustration of procedure for unresolved boundaries on coarse grids, for the grid cell marked **a**. The object (red ellipse) has a small extent in both dimensions, and is therefore not detected by a line search between cell centers (blue arrows). First, the distance to the closest point **x** on the boundary is determined (gray arrow), and then a virtual boundary is placed between **a** and the neighboring cell center closest to **x**. (For interpretation of the colors in the figures, the reader is referred to the web version of this article.)

4. Numerical experiments

4.1. Convergence tests on sphere

To test the numerical convergence of the method, we solve the Laplace equation

$$\nabla^2\phi = 0$$

for a spherical LSF of the form

$$f_{\text{sphere}}(\boldsymbol{x}) = ||\boldsymbol{x}|| - R, \tag{9}$$

using a computational domain of unit size (e.g., the unit cube in 3D) centered at the origin. On the spherical boundary, a Dirichlet condition $\phi = \phi_b$ is imposed. On the boundaries of the computational domain, the following analytic solutions are imposed

$$\phi_{2\text{d}}(r) = \phi_b + a\log(||\boldsymbol{x}||/R), \tag{10}$$

$$\phi_{3\text{d}}(r) = \phi_b + a(1 - R/||\boldsymbol{x}||), \tag{11}$$

using $\phi_b = 0$ and $a = 1$.

As a first test, we consider the case $R = 1/4$ on uniformly refined grids. The coarsest grid contains 8^D cells, and the finest grid $(8 \times 2^{l_{\text{max}}-1})^D$ cells, where l_{max} is the maximal refinement level. Fig. 5a shows convergence results in 2D and 3D. The residual reduction factor per FMG iteration is about 40–80 in 2D and about 30–40 in 3D. Due to numerical round-off errors, the residual eventually stops decreasing. The resulting 'converged' residual is larger on finer grids because of the division by Δx^2 in equation (3).

After two FMG iterations, the solution error (as compared with the analytic solutions) hardly changes anymore. For example, for the 3D case with $l_{\text{max}} = 6$, the maximal error after the first three iterations is 0.32×10^{-3}, 0.11×10^{-3} and 0.11×10^{-3}. This means that after two iterations, the discretization error dominates the convergence error. Fig. 5b shows that the discretization error reduces proportional to Δx^2, both in the L_∞ and in the L_2 norm, indicating second order convergence.

As a second test, we consider the case $R = 5 \times 10^{-3}$ in 3D, in combination with grid refinement. The following refinement criterion is used: refine if $\Delta x > \Delta x_{\text{min}} \times \max(1, r/R)$, where $r = \sqrt{x^2 + y^2 + z^2}$ and Δx_{min} is the grid spacing at level l_{max}. Due to its small radius, the spherical boundary will not be resolved on the coarsest grids, but the approach described in section 3.6.3 ensures that the method still converges. Fig. 6 shows that the residual re-

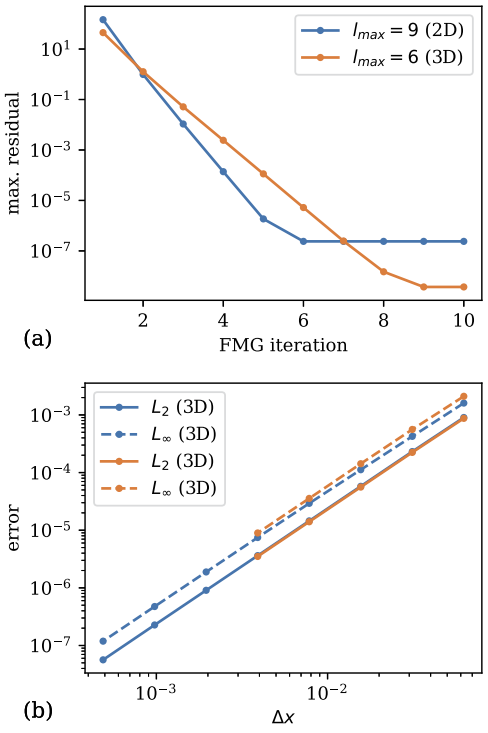


Fig. 5. Convergence results the spherical LSF given by equation (9), with $R = 1/4$ and a uniform grid. a) Maximal residual versus FMG iteration, with the maximum refinement level indicated by l_{max}. b) Error in converged solution for different grid spacings Δx, determined by comparing with equation (10). Both the maximal error (indicated by L_∞) and the RMSE (indicated by L_2) are shown.

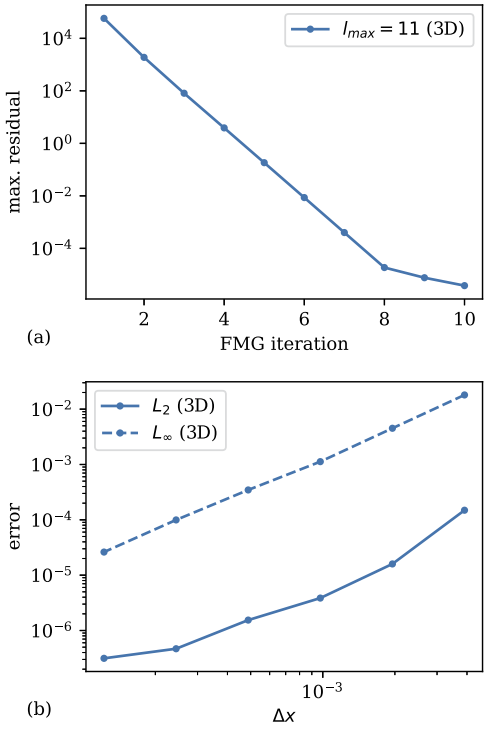


Fig. 6. Convergence results for 3D spherical problem with grid refinement, in which a boundary condition is imposed at a small radius ($R = 5 \times 10^{-3}$). a) Residual reduction. b) Error in the converged solution for different finest-grid spacings Δx_{min}.

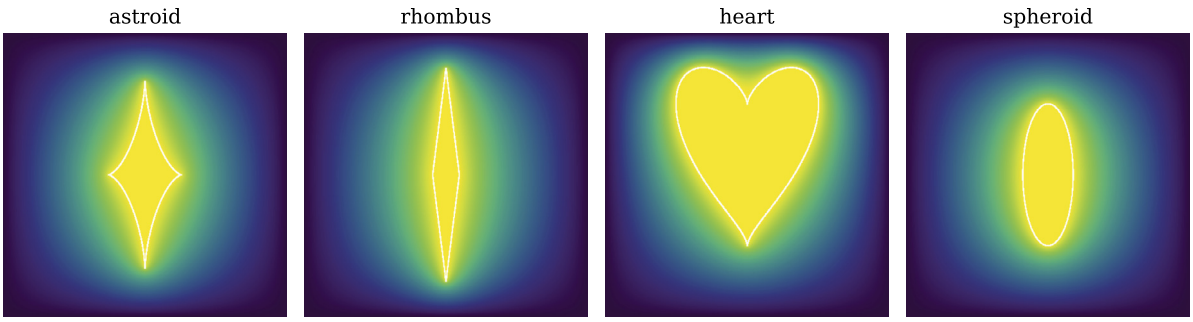


Fig. 7. Illustration of solutions with the level set functions of equations (12)–(15), shown on the unit square. Contours of the boundaries are indicated in white. Note that the astroid shape has very sharp features.

duction factor per FMG iteration is again about 30–40. The error in the solution is still approximately proportional to Δx^2, where Δx is the finest grid spacing. Note that this convergence behavior also depends on how well the mesh refinement is adapted to the problem.

4.2. Sharp boundaries

Irregular boundaries with sharp features are a more challenging test for a multigrid-based solver. On coarser grids, such sharp features cannot be accurately described, potentially reducing the effectiveness of the coarse-grid correction. Furthermore, near sharp features the solution will have steep gradients, which increases interpolation errors. To test the robustness of our solver, we consider the following level set functions

$$f_{\text{spheroid}}(p,q) = \sqrt{8p^2+q^2} - 1, \tag{12}$$

$$f_{\text{rhombus}}(p,q) = 8|p| + |q| - 1.5, \tag{13}$$

$$f_{\text{heart}}(p,q) = p^2 + (q - |p|^{2/3})^2 - 1, \tag{14}$$

$$f_{\text{astroid}}(p,q) = |p|^{2/3}/0.8 + |q|^{2/3}/1.5 - 0.8. \tag{15}$$

These LSFs are evaluated on the unit square using transformed coordinates $p = (x - 0.5)/4$ and $q = (y - 0.5)/4$, see Fig. 7.

As a first test, we consider uniformly refined grids in 2D of size 1024^2 and 2048^2, using a coarse grid size of 8×8. The corresponding maximum refinement levels are thus $l_{\text{max}} = 8$ and $l_{\text{max}} = 9$. At the irregular boundary, a boundary condition $\phi = 1$ is applied, and $\phi = 0$ on the boundaries of the computational domain. The smallest width to resolve on coarse grids (see section 3.6.3) was set to $w_{\text{min}} = 4 \times 10^{-3}$.

Fig. 8a shows the reduction in the residual per FMG iteration for each test case. The residual reduction factor is similar for the spheroid, rhombus and heart shapes. For the astroid shape the reduction factor is lower, and it changes with the refinement level. We have also noticed that the reduction factor for this test case can depend on the position of the astroid. This is probably due to an inconsistent description of the sharp endpoints on different refinement levels.

We generalize the above LSFs to cylindrical geometries in 3D by using transformed coordinates $p = (\sqrt{x^2+y^2} - 0.5)/4$ and $q = (z - 0.5)/4$. Note that in 3D there is curvature along an extra coordinate, so that solution gradients become even steeper near sharp features. We consider the shapes given above in a 3D unit cube, using the same boundary conditions as in 2D, a coarse grid size of 8^3 and $w_{\text{min}} = 8 \times 10^{-3}$. Fig. 8b shows the residual reduction per FMG iteration on a uniformly refined grid of 256^3 ($l_{\text{max}} = 6$). Note that the residual reduction factor is again lowest for the astroid shape, which has the sharpest features.

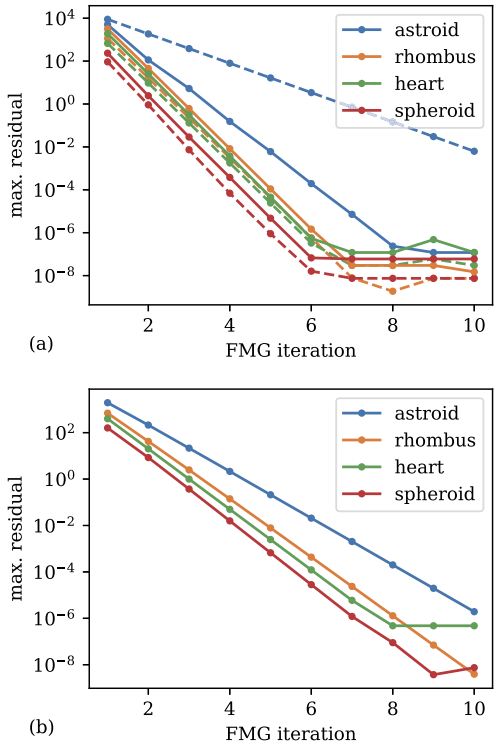


Fig. 8. a) Maximal residual versus FMG iteration for the 2D shapes shown in Fig. 7. The dashed lines correspond to a 1024^2 grid, and the solid lines to a 2048^2 grid on the unit square. b) Results for the axisymmetric 3D generalizations of the shapes on a 256^3 grid.

4.3. Application example

We briefly present an example relevant for the simulation of pulsed electric discharges such as streamers [23]. In discharge simulations, electrostatic fields have to be computed at every time step. This is done by solving equation (1) for a given electrode configuration, after which the electric field is obtained as $\boldsymbol{E} = -\nabla\phi$. Solving Poisson's equation is typically one of the most expensive components of these simulations. Various methods have been used to incorporate electrodes, ranging from a simple charge simulation technique (see e.g. [24]) to the ghost fluid method [25]. Electrodes have also been included on structured grids with different AMR framework [26,27], in combination with multigrid-based solvers. For complex geometries, the use of finite element methods can also be advantageous [28].

We consider a 3D geometry in which two electrodes are present, illustrated in Fig. 9. The computational domain is of size unity, and the following boundary conditions are used on its sides: $\phi = 1$ at the top, $\phi = 0$ at the bottom, and Neumann zero bound-

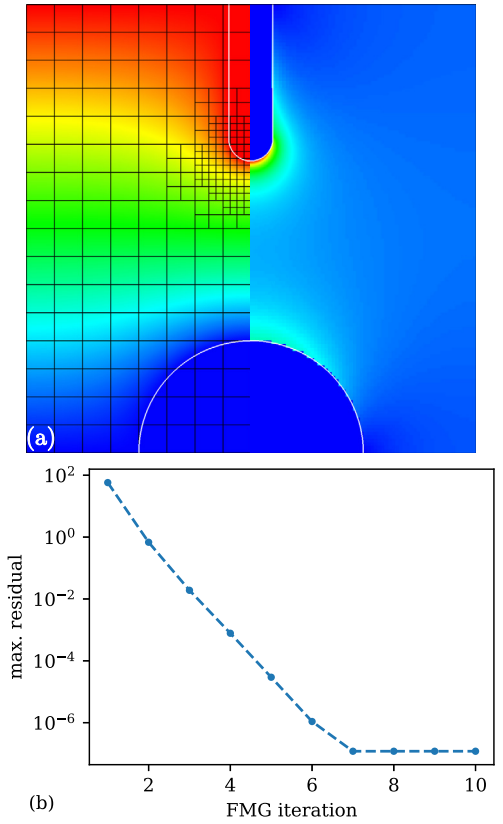


Fig. 9. a) The electric potential ϕ (left half) and the magnitude of its gradient $||\nabla\phi||$ (right half) for a two-electrode problem. The figure is a cross section of the 3D domain of size unity. Electrode contours are white, and the numerical mesh is illustrated, with each black square corresponding to a grid block of 8^3 cells. b) Maximal residual versus FMG iteration.

ary conditions on the other sides. A rod electrode is placed at the top, with radius $r_{\text{rod}} = 0.05$, at which $\phi = 1$. The corresponding level set function (in the top half) is obtained by computing the distance from a line segment, and then subtracting r_{rod}. On the bottom of the domain, a semi-sphere is placed with radius $r_{\text{sphere}} = 0.25$, at which $\phi = 0$.

The numerical mesh has a spacing of $\Delta x = 1/512$ near the tip of the top electrode, and a resolution $\Delta x = 1/128$ elsewhere, as illustrated in Fig. 9, which also shows the solution ϕ, $||\nabla\phi||$ and the maximal residual versus FMG iteration. The residual reduction factor is about 30-40 per FMG iteration. The components of $\nabla\phi$ were computed on a staggered grid (on cell faces), taking the stored distances to boundaries into account. For example, if there is a boundary between $\phi_{i,j,k}$ and $\phi_{i+1,j,k}$, then $\partial_x\phi$ at $i+1/2$ is approximated by $(\phi_b - \phi_{i,j,k})/(d\Delta x)$, where ϕ_b is the boundary value and d is the relative distance to the boundary. For cells whose center lies near the boundary but inside the electrodes, $||\nabla\phi||$ was set to zero.

4.4. Computational cost

When new refinement is added to the mesh, the boundary detection method described in section 3.6.1 is performed. This requires the evaluation of the numerical gradient of the LSF at every newly added grid cell.[1] Afterwards, the distance computation described in section 3.2 is performed for grid cells that are close to the boundary. This requires a few tens of evaluations of the level set function per grid cell. To keep these costs low, the LSF should be cheap to compute. When removing refinement, no extra work is required.

[1] Note that in the majority of cases, the absence of an irregular boundary can be deduced from the parent grid, but some sharp features might only be detected on the fine grid.

Table 1
Computational time (in seconds) per FMG cycle for the 3D test case with a spherical boundary described in section 4.1, on a uniform grid with $512^3 \approx 134 \times 10^6$ cells. Results with block sizes of 8^3, 16^3 and 32^3 are included, and parallel efficiencies compared to the case with 4 cores are indicated. The tests were performed using up to 32 cores of an AMD Epyc 7H12 processor. The computational times are an average over 40 consecutive FMG cycles.

	$512^3/8^3$	$512^3/16^3$	$512^3/32^3$
4 cores	5.55	2.97	2.36
8 cores	2.77 (100%)	1.53 (97%)	1.31 (90%)
16 cores	1.62 (86%)	1.01 (74%)	0.87 (68%)
32 cores	1.46 (48%)	0.95 (39%)	0.84 (35%)

In many applications, solutions have to be computed multiple times on the same numerical mesh, but with different right-hand sides. The cost per multigrid iteration is then most important. To illustrate these costs, we solve the test case with the spherical boundary described in section 4.1 in 3D on a 512^3 uniformly refined grid. We consider block sizes of 8^3, 16^3 and 32^3. A smaller block size increases the adaptivity of the mesh, and it will reduce the total volume of grid blocks that intersect the boundary. On the other hand, a smaller block size means that more blocks are required, leading to extra communication costs.

Table 1 gives the time per FMG cycle in seconds for the various cases. Note that the parallel scaling is not ideal. The reason for this is that computations in a geometric multigrid method are relatively cheap, so that the speed with which data can be accessed from and written to memory is often the limiting factor.

5. Conclusions

We have presented a method to include irregular domain boundaries in a geometric multigrid solver. The method was developed for quadtree/octree grids with adaptive refinement, using a cell-centered discretization, and it supports Dirichlet-type boundary conditions. The location of boundary intersections is automatically determined from a level set function, which is to be provided as input. For grid blocks near the interface, custom operator stencils are stored. However, the computational cost of handling these custom blocks is comparable to that of regular blocks away from boundaries, and in both cases, point-wise multigrid smoothers are employed. We have illustrated the numerical convergence of the method by considering spherical boundaries in the unit square and unit cube. Furthermore, the robustness and computational efficiency of the method were examined with several test cases with sharp boundaries.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] C.C. Douglas, G. Haase, U. Langer, A Tutorial on Elliptic PDE Solvers and Their Parallelization, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2003.

[2] A. Gholami, D. Malhotra, H. Sundar, G. Biros, SIAM J. Sci. Comput. 38 (3) (2016) C280–C306, https://doi.org/10.1137/15m1010798.

[3] W. Hackbusch, Multi-grid Methods and Applications, Springer Series in Computational Mathematics, 1985.

[4] U. Trottenberg, C. Oosterlee, A. Schuller, Multigrid, Elsevier Science, 2000.

[5] A. Brandt, O.E. Livne, Multigrid Techniques, Society for Industrial & Applied Mathematics (SIAM), 2011.

[6] K. Stuben, J. Comput. Appl. Math. (2001) 29.

[7] J.W.L. Wan, X.-D. Liu, SIAM J. Sci. Comput. 25 (6) (2004) 1982–2003, https://doi.org/10.1137/S1064827503428540.

[8] T. Guillet, R. Teyssier, J. Comput. Phys. 230 (12) (2011) 4756–4771, https://doi.org/10.1016/j.jcp.2011.02.044.

[9] M. Theillard, C.H. Rycroft, F. Gibou, J. Sci. Comput. 55 (1) (2013) 1–15, https://doi.org/10.1007/s10915-012-9619-2.

[10] L. Botto, Comput. Phys. Commun. 184 (3) (2013) 1033–1044, https://doi.org/10.1016/j.cpc.2012.11.008.

[11] Y. Lee, H. Thompson, P. Gaskell, Comput. Fluids 36 (5) (2007) 838–855, https://doi.org/10.1016/j.compfluid.2006.08.006.

[12] J. Teunissen, U. Ebert, Comput. Phys. Commun. 233 (2018) 156–166, https://doi.org/10.1016/j.cpc.2018.06.018.

[13] D. Weber, J. Mueller-Roemer, A. Stork, D. Fellner, Comput. Graph. Forum 34 (2) (2015) 481–491, https://doi.org/10.1111/cgf.12577.

[14] J. Teunissen, R. Keppens, Comput. Phys. Commun. 245 (2019) 106866, https://doi.org/10.1016/j.cpc.2019.106866.

[15] S. Osher, J.A. Sethian, J. Comput. Phys. 79 (1) (1988) 12–49, https://doi.org/10.1016/0021-9991(88)90002-2.

[16] J.A. Sethian, Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science, vol. 3, Cambridge University Press, 1999.

[17] S. Chen, B. Merriman, S. Osher, P. Smereka, J. Comput. Phys. 135 (1) (1997) 8–29, https://doi.org/10.1006/jcph.1997.5721.

[18] F. Gibou, R.P. Fedkiw, L.-T. Cheng, M. Kang, J. Comput. Phys. 176 (1) (2002) 205–227, https://doi.org/10.1006/jcph.2001.6977.

[19] H. Udaykumar, R. Mittal, W. Shyy, J. Comput. Phys. 153 (2) (1999) 535–574, https://doi.org/10.1006/jcph.1999.6294.

[20] R.D. Falgout, U.M. Yang, in: Proceedings of the International Conference on Computational Science-Part III, ICCS '02, Springer-Verlag, London, UK, 2002, pp. 632–641.

[21] S.F. Ashby, R.D. Falgout, Nucl. Sci. Eng. 124 (1) (1996) 145–159, https://doi.org/10.13182/NSE96-A24230.

[22] R.D. Falgout, J.E. Jones, in: M. Griebel, D.E. Keyes, R.M. Nieminen, D. Roose, T. Schlick, E. Dick, K. Riemslagh, J. Vierendeels (Eds.), Multigrid Methods VI, vol. 14, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 101–107.

[23] S. Nijdam, J. Teunissen, U. Ebert, Plasma Sources Sci. Technol. 29 (10) (2020) 103001, https://doi.org/10.1088/1361-6595/abaa05.

[24] A. Luque, V. Ratushnaya, U. Ebert, J. Phys. D, Appl. Phys. 41 (23) (2008) 234005, https://doi.org/10.1088/0022-3727/41/23/234005.

[25] S. Celestin, Z. Bonaventura, B. Zeghondy, A. Bourdon, P. Ségur, J. Phys. D, Appl. Phys. 42 (6) (2009) 065203, https://doi.org/10.1088/0022-3727/42/6/065203.

[26] V. Kolobov, R. Arslanbekov, J. Comput. Phys. 231 (3) (2012) 839–869, https://doi.org/10.1016/j.jcp.2011.05.036.

[27] R. Marskar, J. Comput. Phys. 388 (2019) 624–654, https://doi.org/10.1016/j.jcp.2019.03.036.

[28] A.P. Jovanovic, M.N. Stankov, D. Loffhagen, M.M. Becker, IEEE Trans. Plasma Sci. 49 (11) (2021) 3710–3718, https://doi.org/10.1109/TPS.2021.3120507.