Available online at www.sciencedirect.com
ScienceDirect
Mathematics and Computers in Simulation 210 (2023) 49–65
MATHEMATICS AND COMPUTERS IN SIMULATION
www.elsevier.com/locate/matcom

Original articles

Adaptive partition of unity interpolation method with moving patches☆

Alfa Heryudono[a,*], Mehdi Raessi[b]

[a] Department of Mathematics, University of Massachusetts Dartmouth, 285 Old Westport Rd, Dartmouth, MA, 02747, USA
[b] Department of Mechanical Engineering, University of Massachusetts Dartmouth, 285 Old Westport Rd, Dartmouth, MA, 02747, USA

Received 16 August 2022; received in revised form 28 December 2022; accepted 8 March 2023
Available online 15 March 2023

Abstract

The adaptive partition of unity interpolation method, introduced by Aiton and Driscoll, using Chebyshev local interpolants, is explored for interpolating functions with sharp gradients representing two-medium problems. For functions that evolve under vector fields, the partition of unity patches (covers) can be shifted and resized to follow the changing dynamics of local profiles. The method is tested for selected 1D and 2D two-medium problems with linear divergence-free vector fields. In those cases, the volume fraction in each patch contributing to volume conservation throughout the domain can be kept in high accuracy down to machine precisions. Applications that could benefit from the method include volume tracking and multiphase flow modeling.

Keywords: Partition of unity method; Free boundary problems; Volume-preserving technique

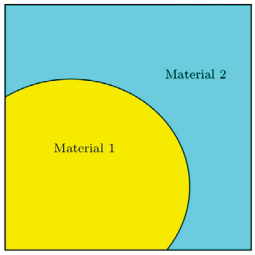
1. Introduction

The need for highly accurate interpolants for functions where the region of high activity is shifting or moving in time appears in many practical applications, such as shape deformations and multiphase flows. Depending on the methods used, such functions are usually sampled on structured/unstructured grids, meshes, or scattered points. Initially, if the methods permit, the underlying grids are typically concentrated in some regions to capture localized profiles such as peaks and sharp gradients. As the profiles move in time to some other locations in the domain, the grids usually need to be redistributed or resampled accordingly. Hence, the accuracy and quality of the interpolant can be maintained. The interpolant is typically utilized not just for data interpolation but also for finding derivatives and computing integrals.

In this work, we are interested in cases where a one- or two-dimensional time-dependent function $f(\underline{x}, t)$ defined on a rectangular domain Ω has sharp gradients representing a two-phase medium scenario; an example is shown in Fig. 1, where $f(\underline{x}, t) = 1$ (representing material/medium 1), on a particular localized region inside the domain and zero everywhere else. The locations where the sharp gradients occur are called the interface, which separates

☆ This work was funded by the US National Science Foundation DMS-2012011.
* Corresponding author.
E-mail addresses: aheryudono@umassd.edu (A. Heryudono), mraessi@umassd.edu (M. Raessi).

https://doi.org/10.1016/j.matcom.2023.03.006



$$f(\underline{x}) = \begin{cases} 1, & \underline{x} \in \text{Material 1} \\ 0, & \underline{x} \notin \text{Material 1} \end{cases}$$

Fig. 1. An illustration of two materials/mediums layout along with the interface. $f(\underline{x})$, usually called an indicator function, is commonly modeled with a discontinuous function or a function with sharp gradient.

the two mediums. With a given vector field $\underline{u}(\underline{x})$, with $\underline{x} = (x, y)$ in 2D, for example, one might be interested in studying the dynamics of the free boundary (the shape of the interface) and the conservation of several quantities associated with the mediums/materials. To be specific, with our function described above, $\int f d\Omega$ measures the volume of medium 1. In problems where volume conservation is expected, the value of the integral must stay the same throughout the simulation. Typically, one can rarely use the grid points used at time $t = 0$ for the initial condition $f(\underline{x}, 0)$ since the profile of the function might look different later. Ideally, grids should be distributed following the moving front.

Computational strategies and numerical methods for dealing with functions representing a two-phase medium evolved under a vector field consist of two challenges. (a) A highly accurate interpolant for f in space that captures the region of high activities is needed, and (b) distributing the grids/points/cells marching together with the time-evolving interface while ensuring the quality of the interpolant does not change. There has been a large body of publications and survey work (including [5,9,12,13,15,17,18,22,23,25,28,30–33,35]) in this field since the early development of numerical methods for partial differential equations. For example, it is at the heart of multiphase flow simulations using the Volume-of-Fluid (VOF) method.

This paper explores ways to mitigate challenges (a) and (b) by using the partition of unity method with non-stationary covers. To approximate functions with sharp gradients in this work, we focus on utilizing an adaptive partition of unity (APU) method based on Chebyshev polynomial interpolants introduced by Aiton, and Driscoll [2–4]. The method is able to adaptively (using bisection techniques in alternating dimensional directions) construct a highly accurate global interpolant down to machine precision with ease in 1D, 2D, and 3D. Their codes are available in MATLAB and use Chebfun [8] as a backend. Both make rapid numerical prototyping with Chebyshev technologies enjoyable with minimal effort.

During its construction, the interpolant also creates overlapping patches (covers) of the computational domain and its local interpolants. For functions with high activities around a particular region (e.g., around the interface) in the domain, more patches are concentrated there and fewer anywhere else. To follow the dynamics of the free boundary or the interface, we allow the patches to move along with the flow field as long as they meet certain conditions. For updating local function values in the patches, the resizing and shifting of the covers usually only involve scaling due to changes in covers' sizes. Hence, the accuracy of the global interpolant is not affected.

The paper is organized as follows: In Section 2, we briefly introduce the partition of unity interpolation method. Section 3 describes how patches can resize and shift due to a vector field and how function values are updated. The algorithm is briefly discussed in Section 4. Numerical cases of two-phase problems in 1D and 2D under linear divergence-free vector fields that show conservation of volume down to machine precision are provided in Section 5. A discussion section concludes our work.

2. Partition of unity method in a nutshell

The basic idea of the partition of unity approach is to break the domain into several pieces (patches), approximate the function in each subdomain (patch) separately, and then blend the local approximations together using smooth, local weights that sum up to one everywhere on the domain. Let $f(\underline{x})$ be defined as a smooth scalar multivariate

function defined on a closed domain $\Omega \subset \mathbb{R}^n$, with boundary $\partial\Omega$ where $\underline{x} = (x_1, x_2, \ldots, x_n)$. In a partition of unity method, the global approximation $\tilde{f}(\underline{x})$ to the function $f(\underline{x})$ is constructed as a weighted sum of local approximations $\tilde{f}_j(\underline{x})$ on overlapping patches Ω_j, $j = 1, \ldots, N_p$. That is,

$$\tilde{f}(\underline{x}) = \sum_{j=1}^{N_p} w_j(\underline{x}) \tilde{f}_j(\underline{x}). \tag{1}$$

where w_j, $j = 1, \ldots, N_p$ are weight functions. The patches Ω_j need to form a cover of the domain in the sense that

$$\bigcup_{j=1}^{N_p} \Omega_j \supseteq \Omega. \tag{2}$$

The partition of unity weight functions w_j are non-negative, compactly supported on Ω_j and satisfy

$$\sum_{j=1}^{N_p} w_j(\underline{x}) = 1, \quad \forall \underline{x} \in \Omega. \tag{3}$$

In the particular case where rectangular patches are used, the local approximations $\tilde{f}_j(\underline{x})$ are Chebyshev approximations on the local tensor product grid. In the best efficiency scenario, the local approximant is computed only once and replicated for each patch. However, several templates of local approximants can also be precomputed and stored in a lookup table to be used by any patch that needs them. In order to capture high degrees of localization, patches can be adapted to reflect the profile of the solutions. Hence, the method offers greater flexibility to adjust local approximants by refining/coarsening patches (h-type version) or raising/lowering the degree of local interpolants (p-type version).

Calculus operations such as differentiation and integration of the global approximant Eq. (1) can be done term by term (then summed up) in a straightforward way. This is handy when gradients, divergence, and volumes must be computed. The calculus operations should be accurate locally at patch levels and globally throughout the domain.

Constructing a PU interpolant for a function representing two-medium problems usually results in patches clustered in the interface's neighborhood. Those patches containing local interpolants with high-degree Chebyshev polynomials are needed in regions with steep gradients. Otherwise, the interpolant may not capture the function properly there, i.e., under resolution can most likely happen. Indeed, the locations of those patches can also be utilized to determine the approximate shape of the interface. This information could potentially be helpful for the purpose of tracking or reconstructing interfaces, although we have not used the Chebyshev PU interpolation method solely for that purpose here yet. Other regions away from the interface can have patches with larger sizes with lower degrees of local interpolant.

As an illustration in two dimensions, Fig. 2 shows the layout of the rectangular patches for a function $f(x, y) = \frac{1}{2}(1 + \tanh(5(1 - (x^2 + y^2))))$ in the domain $[-4, 4] \times [-4, 4]$, with polynomials degrees ranging from 3 to 129 with 10% overlap. For visualization purposes, those rectangles can be colored based on any information that might be useful for users if needed. Aiton and Driscoll use a semitransparent pastel-style coloring so that the overlap regions can also be observed.

With their codes, to construct the interpolant for the function above, one may execute the following command in MATLAB:

```
f = PUchebfun(@(x,y) 0.5*tanh(5*(1-(x.^2+y.^2)))+0.5,[-4 4;-4 4]);
```

With the backend of Chebfun, the construction usually finishes in a fraction of a second but could be longer for more complicated functions. Similar commands are available in 1D and 3D. Once the interpolant is constructed, one can use it to interpolate or find operators involving derivatives (e.g., Laplacian, gradients, divergence) and integrals. For example, one can execute `sum(f)` (i.e., $\int f d\Omega$) to compute the integral of f on the domain. In the case above, the integral of f approximates the area of a unit circle. When one increases the slope of the tanh from 5 to, say, 50, the integral gives about 14 digits accuracy in a fraction of a second. One can see the power of adaptive partition of unity representation for constructing a global interpolant with spectral accuracy.

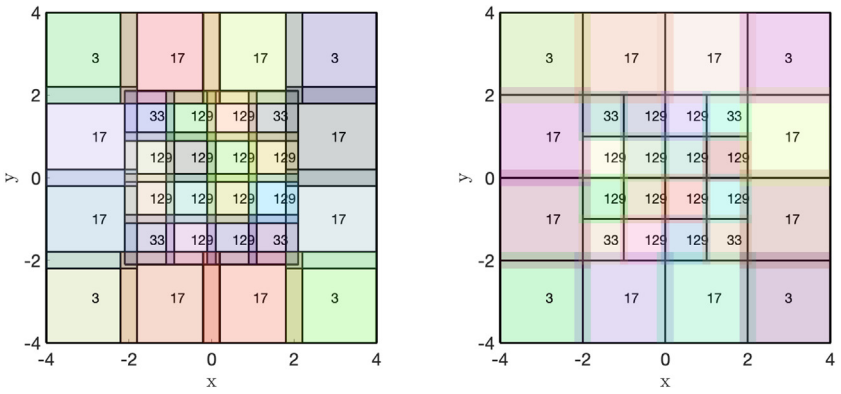


Fig. 2. The layout of patches of the function f, systematically constructed using bisections, along with their overlapped regions. The numbers show the maximum degrees of Chebyshev polynomials used in each patch. The figure on the right shows the zones: non-overlapping boxes that define the patches.

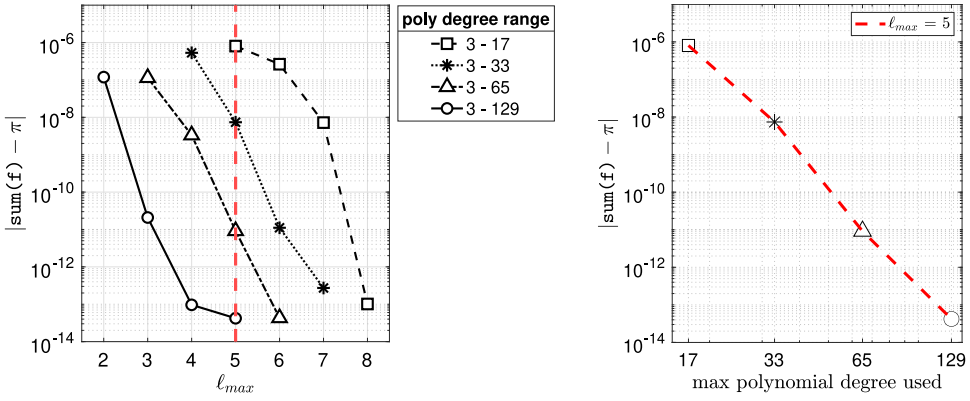


Fig. 3. The function $f(x, y) = \frac{1}{2}(1 + \tanh(100(1 - (x^2 + y^2))))$ is constructed using an APU interpolant. The `sum(f)` $\approx \int f d\Omega$ is computed and the value then is compared to π. Left: error $|\mathtt{sum(f)} - \pi|$ convergence trend (`semilogy` scale) as a function of ℓ_{max} for varying polynomial degree range used by the interpolant in each patch. In this case, the maximum subdivision of the patch/zone in either x or y direction can go down to $2^{-\ell_{max}}$. Right: the convergence (`loglog` scale) by fixing the ℓ_{max} (red dash line on the left figure) and plotting it as a function of maximum polynomial degree used in each patch. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Plotting the error convergence trend of the adaptive method, though not as straightforward as in the fixed grid method, can be done from different scenarios. This is due to the different range of polynomial degrees and the level of subdivisions used during the construction of the interpolant. We provide two example plots. Fig. 3 shows the convergence plot of $|\mathtt{sum(f)} - \pi|$ using the APU interpolant for the same hyperbolic tangent with the slope raised again to 100. The left figure of Fig. 3 shows the error convergence trend by allowing the degree of Chebyshev polynomials to vary in either x or y direction in each patch/zone but restricting the maximum level of subdivision of the patches/zones down to $2^{-\ell_{max}}$. On the other hand, the rightmost figure of Fig. 3 shows when one wants to fix the maximum level (say $\ell_{max} = 5$) and then plot the convergence along that particular vertical line. In the end, the goal of the adaptive method is to systematically use a combination of subdivisions and a range of polynomial degrees to achieve certain error tolerance with minimal/no user intervention. Moreover, the class constructor f reveals the underlying fields and structures useful for numerical experiments or prototyping. Information about local patches, weights, Chebyshev series coefficients, and others is provided as part of the data structure. The papers [2–4], along with the code, explain everything users need to know about this method.

Table 1
Types of velocity fields. All c's and ω are constants. ω is also referred to as angular velocity.

Type	$u_1(x, y)$	$u_2(x, y)$	Implementation
Translation	c_1	c_2	Cartesian
Pure Strain	$cx + c_1$	$-cy + c_2$	Cartesian
Angular Deformation	$cy + c_1 x$	$cx - c_1 y$	Polar
Pure Rotation	$-\omega y + cx$	$\omega x - cy$	Polar

3. Moving patches

When the profile of the function changes with respect to a vector field, patches may not stay stationary anymore. Ideally, patches with high-degree polynomials are always closely following the interface. Alternatively, one can always rebuild the patches by initiating the interpolant constructor at every step. However, this process should probably be done not too often or under some specific conditions only to reduce computational time. In simple cases where only translation and pure strain problems are considered, rebuilding patches are usually unnecessary. Indeed, in translation and pure-strain vector fields, a particular patch is responsible for a specific subdomain of the function at all times, regardless of whether it is stretched or shrunk. Hence "material transfer" that requires donor–acceptor [12,20] techniques between patches can be avoided.

In the current work, several constraints or conditions exist to consider when allowing rectangular patches to move following a linear divergence-free flow field. These conditions are common in computer graphics for simulating shape deformations of objects [34]. Some of these restrictions might be possibly removed in future work.

1. The movement of patches is based on the movement of the vertices of the patches.
2. The shape of the patches must stay rectangular when stretched or shrunk. Cases with patch rotations, specifically in cartesian coordinates, are left for future study. The rectangular condition is needed for this work, which uses a structured grid. However, the shape can be somewhat arbitrary for other partition of unity frameworks with unstructured grids.
3. Patches cannot detach or merge. Overlapped regions should remain intact too.
4. Their relative positions must always stay the same, i.e., neighbors for life, even when their sizes change or positions move.
5. The number of patches stays the same throughout the simulation. Some on-the-fly techniques involving refinement/coarsening strategies will be left for future study.
6. Boundaries of the domain can be fixed or freely moved with the flow. Fixed boundaries can cause a patch to stretch with a higher proportion in one direction.

Although the movement of those patches may look all over the place during the simulation, they are moving in an orchestrated way with respect to those constraints.

Given a vector field, the time-dependent position vector of the vertices of the patches can be obtained by solving systems of ODEs. However, as a proof of concept, we provide the position vector function or utilize MATLAB and Chebfun high-order built-in ODE solvers to maintain high accuracy in both space and time when needed. This may not be realistic in applications where $\underline{u}$ may come from other flow solvers. At least, with the APU interpolant, the spatial approximation is accurate, and the error of the conserved quantities may be due to the accuracy of the ODE solver used.

For the 2D problem, in each patch, the types of linear divergence-free velocity field $\underline{u}(x, y) = \langle u_1(x, y), u_2(x, y)\rangle$ we are considering in this work are classified in Table 1. The divergence-free conditions provide desired properties for shape deformation such as no path line self-intersections (local or global) in 4D space–time domain [27] and the deformation is volume-preserving [7].

In terms of a system of linear ODEs, the velocity field can be written as

$$\begin{bmatrix} u(x, y) \\ v(x, y) \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \quad A \quad \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b \end{bmatrix},$$

with $\mathrm{Tr}(A) = 0$ ensuring the divergence-free condition. Entries of A and b consist of c, c_1, c_2, ω depending on the type of vector fields.

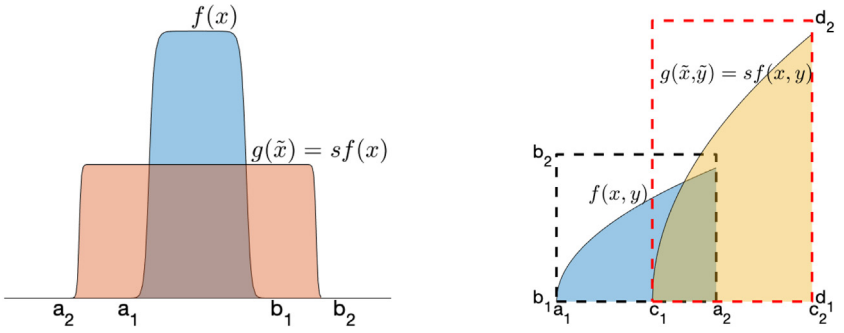


Fig. 4. After a patch is resized, the function f is scaled by a constant s to keep the volume constant. Deforming the function f this way means shape can change, but volume and mass stay constant.

We can set a condition for "deforming" the function f on Ω or f_j or $w_j f_j$ when a patch Ω_j is shifted or resized. Without loss of generality, we just use the notation f for both global and local functions. Note that the model function we are using here, for example, hyperbolic tangent in 1D, is Lipschitz continuous on $[a, b]$ or on any patch interval. It means that there is a constant C such that $|f(x) - f(y)| \leq C|x - y|$ for all $x, y \in [a, b]$. Locally, the product of the function with the partition of unity weight function in any particular patch is also Lipschitz continuous. Fig. 4 shows an illustration in 1D. When a patch of interval $[a_1, b_1]$ is resized to $[a_2, b_2]$, the goal is to scale the function $f(x)$, $x \in [a_1, b_1]$ to a new function $g(\tilde{x})$ such that $g(\tilde{x}) = sf(x)$, $\tilde{x} \in [a_2, b_2]$ with

$$\int_{a_2}^{b_2} g(\tilde{x})d\tilde{x} = \int_{a_1}^{b_1} f(x)dx$$

to keep the area/volume unchanged. In this case, $s = \frac{|b_1-a_1|}{|b_2-a_2|}$. In 2D, as illustrated in the right figure in Fig. 4, the $s = \frac{|a_2-a_1|}{|c_2-c_1|}\frac{|b_2-b_1|}{|d_2-d_1|}$. Since the change of the variable from x to $\tilde{x}$ is a linear map, s is the product of the ratio of the size of the patch before and after resizing it in each dimension. In other words, deforming the function this way means the shape can change, but volume and mass stay constant.

For approximating local interpolants with the Chebyshev series and their integrals, each patch interval can be scaled to $[-1, 1]$. The following two theorems and correct proof, see [29], can be used.

Theorem 1 (*Chebyshev Series*)**.** *If f is Lipschitz continuous on $[-1, 1]$, it has a unique representation as a Chebyshev series,*

$$f(x) = \sum_{k=1}^{\infty} a_k T_k(x),$$

which is absolutely and uniformly convergent. The coefficients are given for $k \geq 1$ by the formula

$$a_k = \frac{2}{\pi}\int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1-x^2}}dx,$$

and for $k = 1$, by the same formula with factor $2/\pi$ changed to $1/\pi$.

The adaptive partition of unity method monitors the decay of the coefficients in each dimension to decide to split the domain to construct patches and chop the series at the desired n. The highest order of polynomials it can use in each dimension is 129, though users can modify the APU code if needed.

Once the Chebyshev series coefficients are available, then they can be used to calculate, for example, an integral in each patch. The theorem below describes the formula.

Theorem 2 (*Integral of a Chebyshev Series*)**.** *The integral of a degree n polynomial expressed as a Chebyshev series*

$$\int_{-1}^{1} \sum_{k=1}^{n} c_k T_k(x)dx = \sum_{k=0,k \text{ even}}^{n} \frac{2c_k}{1-k^2}.$$

The total integral for the whole domain is the sum of the term-by-term integral of Eq. (1) contributed from each patch. In each patch, the 2D and 3D implementation of the APU constructor is the extension of the 1D version via tensor product. Note that a_k and c_k coefficients are already precomputed, and scaling the function f with s will leave them unmodified.

4. Algorithm

The central idea of the method described in this work can be classified as a vector-field-based method of transplantation [19] to advect a scalar function. Given a linear divergence-free vector field, the function is advected by transplanting it over moving patches. The algorithm of the partition of unity method with moving patches is described in Algorithm 1. We choose to use a hyperbolic tangent (also commonly used in the THINC approach [35]) with a steep slope as a template for modeling a two-medium function f.

Algorithm 1 Adaptive partition of unity method with moving patches.

Initiate a function f constructor with the adaptive partition of unity method.
Compute $v_0 =$ the total volume of f on Ω.
Set starting time $t = T_{\text{init}}$, a time step Δt, and a final time T_{fin} for the simulation.
while $t < T_{\text{fin}}$ **do**
  for each patch Ω_i **do**
    Get all the vertices coordinates $vertcoords(\Omega_i(t))$ at time t.
    Record the volume $vol(\Omega_i(t))$ at time t.
    Solve the system of ODEs at the time interval $[t, t + \Delta t]$ based on the vector field information to obtain $vertcoords(\Omega_i(t + \Delta t))$.
    Record the new volume $vol(\Omega_i(t + \Delta t))$ at time $t + \Delta t$.
    Scale the function values with the scaling factor $s = \frac{vol(\Omega_i(t))}{vol(\Omega_i(t+\Delta t))}$.
  end for
  Update all patch vertices with all new information at $t + \Delta t$.
  Compute $v =$ the total volume of f and measure the error $|v - v_0|$.
  update the time $t = t + \Delta t$.
end while

There is no adaptation of polynomial degrees in each patch during the simulation. The polynomial degrees used in each patch at the starting time T_{init} stays the same at $t > T_{\text{init}}$. Note that if the vector field $\underline{u}(\underline{x})$ is provided beforehand, the trajectory of the vertices can be precomputed. We do that here in most of our numerical experiments to show the method as a proof of concept. However, in practice, the flow field is obtained from the flow solver, and an analytical formula is unavailable. Although we have not done it here, the for-loop step in Algorithm 1 can be done in parallel.

5. Numerical experiments

All our experiments here can be classified as velocity-field-based shape deformation problems using linear divergence-free vector fields. Our numerical experiments are carried out in MATLAB 2021a on a standard workstation equipped with a quad-core Intel CPU with 16 GB RAM. Aiton provided the GitHub repository where the PUchebfun codes can be downloaded [1]. Chebfun [8] software can be downloaded from its website. Both packages should be added to the MATLAB path. We provide a simple 1D code explaining the idea in Experiment 1. As proof of concept, in all our numerical experiments, all the flow fields are provided. Hence, the timing of most computations provided here can be done in a few seconds. In the future, we will couple this with a flow solver.

5.1. Experiment 1 (expanding a 1D blob)

We begin by evolving the initial condition at time $t = 0$ given by the function

$$f(x) = \tfrac{1}{2}\tanh(\upsilon(1 - x^2)) + \tfrac{1}{2},$$

with slope $\upsilon = 100$ on the interval $-4 \leq x \leq 4$ under the flow field $u(x) = cx$, where $c = 1/\upsilon$. In terms of two-phase medium terminology, f can be thought of as an indicator function where the region $-1 \leq x \leq 1$ represents material 1 and the rest of the region represents material 2. The simulation is stopped at $t = 1.6$ to avoid the effect of boundaries.

First, we create the APU constructor for the function by executing as well as computing its volume (the area under the curve) at $t = 0$ with the following commands.

```
dom = [-4 4];
f = PUchebfun(@(x) 0.5*tanh(100*(1-x.^2))+0.5,dom)
v0 = sum(f)
```

This creates a global partition of unity interpolant f with 22 patches. The degree of Chebyshev polynomials in those patches varies between 3 and 129. As expected, patches with the highest degrees of 129 are located around $x = \pm 1$ where the interfaces exist.

In terms of time, under the flow field $u(x) = cx$, the endpoints of each patch are evolving as $x(t) = x_0 e^{ct}$. The only points not moved on purpose are the interval endpoints $x = \pm 4$. As shown in Fig. 5, the simulation results in stretching material 1 with decreasing height to maintain a constant volume. Following the scaling technique provided in Section 3, the decreasing height is due to the multiplication with a ratio of the patch size before and after resizing. The total volume of f at every step can be quickly computed because all coefficients are already precomputed.

Since the analytic trajectory is already known beforehand, we can use it immediately in the code. The rough template of the MATLAB code, with $\Delta t = c = \frac{1}{\upsilon}$ without the plot, is provided as the following:

```
Np = length(f.leafArray);
[domlen,newdomlen] = deal(zeros(Np,1));
c = 1/100; x = @(t) exp(c*t);
Tinit = 0; dt = c; Tfin = 1.6;
for t=Tinit:dt:Tfin
  for i=1:Np
   domlen(i) = diff(f.leafArray{i}.domain);
   flag = f.leafArray{i}.domain > dom(1) & ...
        f.leafArray{i}.domain < dom(end);
   f.leafArray{i}.domain(flag) = f.leafArray{i}.domain(flag)*x(t);
   f.leafArray{i}.zone(flag) = f.leafArray{i}.zone(flag)*x(t);
   newdomlen(i) = diff(f.leafArray{i}.domain);
   ratlen = domlen(i)/newdomlen(i);
   f.leafArray{i}.values = f.leafArray{i}.values*abs(ratlen);
  end
  v = sum(f);
  abs(v-v0)
end
```

The template of the code is relatively similar in 2D and 3D since one can access the data structure of the PUchebfun easily. Note that one can instead also pass a function handle obtained from ODE solvers replacing $x(t)$. The volume error, defined as the difference between the initial volume and calculated volume at any time, remains very near machine zero, as shown in the right-most figure in Fig. 5. The highest volume error was no greater than 3×10^{-15}. When one uses ODE solvers in replacing $x(t)$, one might expect the error in the volume plot to be in the order of accuracy of the time-stepping used.

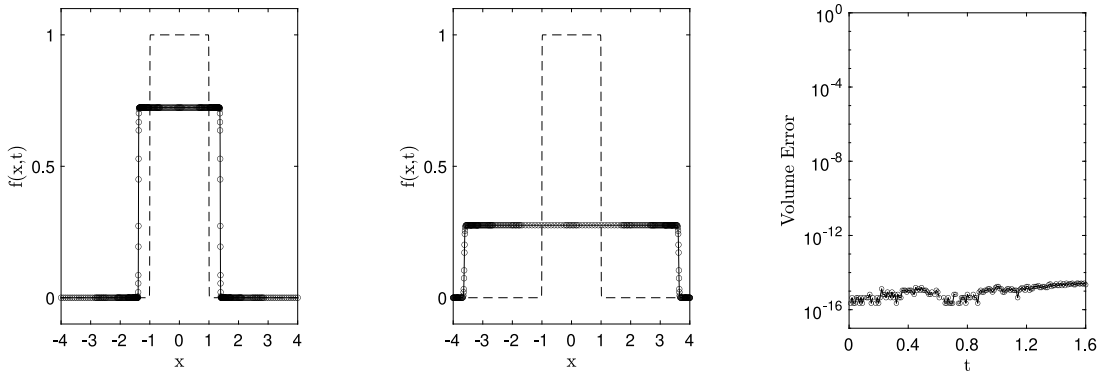


Fig. 5. The evolution of the tanh from $0 \le t \le 1.6$. The dashed line is the function at $t = 0$. The leftmost figure is the profile at time $t = 0.8$, and the middle figure is at time $t = 1.6$. The rightmost figure shows the absolute error between the volume at time t with the one computed at the initial time. The deformation does seem to preserve the sharp gradient with no smoothing or energy minimization needed.

5.2. Experiment 2 (reversible blob in 1D)

This test is similar to Experiment 1 with the flow field given by $u(x, t) = c(2\pi/T_{\text{fin}})x\cos(2\pi t/T_{\text{fin}})$, where c is a constant as defined in Experiment 1. The cosine multiplier function has the effect of decreasing the magnitude of the vector field until time $t = T_{\text{fin}}/2$ and then powering back up in a reversible way such that the evolving function f should coincide with the original shape at $t = 0$ at time T. This test can give us insights into whether the method can maintain the shape and volume or whether some smearing effect is happening. Fig. 6 shows that the volume error is near machine zero, and the original shape is maintained at the final time with no smearing. Although we can compute $x(t)$ analytically, we can instead use an ODE solver to solve it. For every endpoint/vertex of the patches, we precompute their trajectory offline. For example, using the Chebfun ODE solver for the left endpoint of the patch with index 1 (Ω_1), we can use the commands

```
N = chebop(Tinit, Tfin);
N.op = @(t,x) diff(x)-x*c*(2*pi/Tfin)*cos(2*pi*t/Tfin);
N.lbc = f.leafArray{1}.domain(1);
odesol = solvebvp(N,0) or odesol = N\0.
```

The `odesol` is a function handle that defines the path line or trajectory of the left endpoint of the patch Ω_1, e.g., `odesol(0.5)` will compute its position at time $t = 0.5$. The `solvebvp` is essentially solving the initial value problem in "space–time" where t is treated the same way as the spatial variable. As the results demonstrate, the method maintains accuracy down to machine precision in both space and time.

5.3. Experiment 3 (pure strain flow in 2D)

For this experiment, we use a 2D function with a pear-shaped form given by

$$f(x, y) = \tfrac{1}{2}\tanh\left(100\left(\left(1 + (y-1)^3\right)(1 - (y-1)) - 4x^2\right)\right) + \tfrac{1}{2}$$

on the domain $[-4, 4] \times [-0.5, 2.5]$. This pear shape, shown in Fig. 8, is a class of piriform curves commonly used to approximate droplet shape. The flow field is of type pure strain $\underline{u} = (x, -y)$. Fig. 7 shows the initial shape and its initial distribution of patches. For this simulation, we let the vertices of the domain move along with the flow. Under this flow, we expect the pear to be squeezed/deformed towards the line $y = 0$, and the patches become elongated in the horizontal direction at the final time. Fig. 8 shows the volume accuracy is still down to machine precision throughout the simulation.

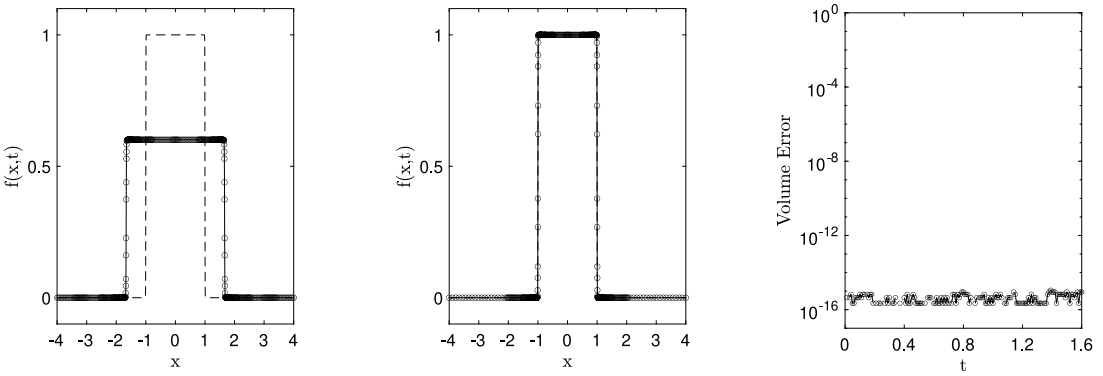


Fig. 6. The evolution of the tanh from $0 \le t \le 1.6$ with the flow field containing cosine multiplier. The dashed line is the function at $t = 0$. The left most figure is the profile at time $t = 0.8$ ($t = T_{\text{fin}}/2$) and the middle figure is at time $t = 1.6$ ($t = T_{\text{fin}}$). The rightmost figure shows the absolute error between the volume at time t with the one computed at the initial time. The 1D blob does seem to coincide back with its original shape at the final time. Although unnecessary for this purpose, the simulation trajectory is precomputed with `solvebvp` in Chebfun.

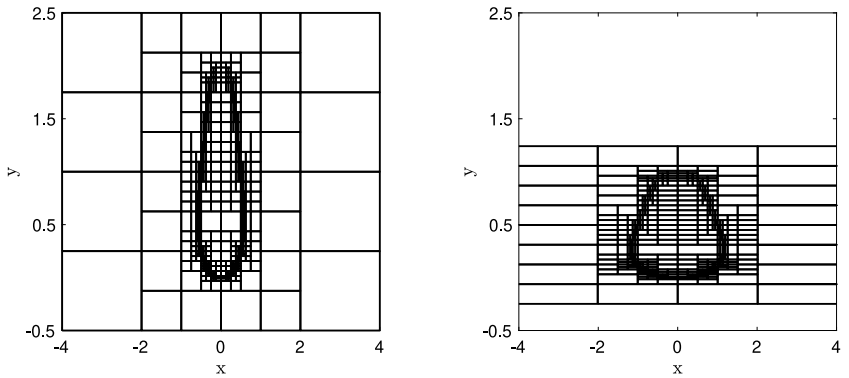


Fig. 7. Left: the pear-shaped form dynamics under the pure strain flow. The partition of unity patches distribution follows the profile of the pear. Patches with bigger sizes are away outward from the interface. Right: since we let the corners of the domain freely move, the patches at $t = 0.8$ are squeezed towards $y = 0$.

5.4. Experiment 4 (circular motion without rotation)

In this experiment, we perform a circular motion of the medium *without* rotation. The object (material/medium 1) is a unit disk initially centered at $(0, 2)$. The function to represent it is

$$f(x, y) = \tfrac{1}{2}\tanh(50(1 - (x^2 + (y - 2)^2))) + \tfrac{1}{2}.$$

With the steep gradient slope of 50, the area of the disk computed with `sum(f)` is already close to π up to 14 digits. We then let the disk move in circular motion such that the distance of its center of mass to the origin is 2 throughout the simulation. Additionally, the vertices of the domain are kept fixed. Hence, the patches are constrained to move only inside the bounding box.

The top two figures of Fig. 9 show the position of the unit disk at the initial time and the half-circular way of the motion. The patches are shown when the position of the disk is somewhere in the third quadrant. Since the vertices of the bounding box are kept fixed, some patches are stretched to maintain good domain coverage. The volume accuracy can be seen in the bottom right plot of Fig. 9. Although the location of the center of mass is available analytically during the motion, one can also check it numerically. Note that the data structure contains all information about Chebyshev grids and all necessary weights and coefficients in each patch. Hence, one can use them to compute the center of mass or centroids in each patch or other type of "moments".

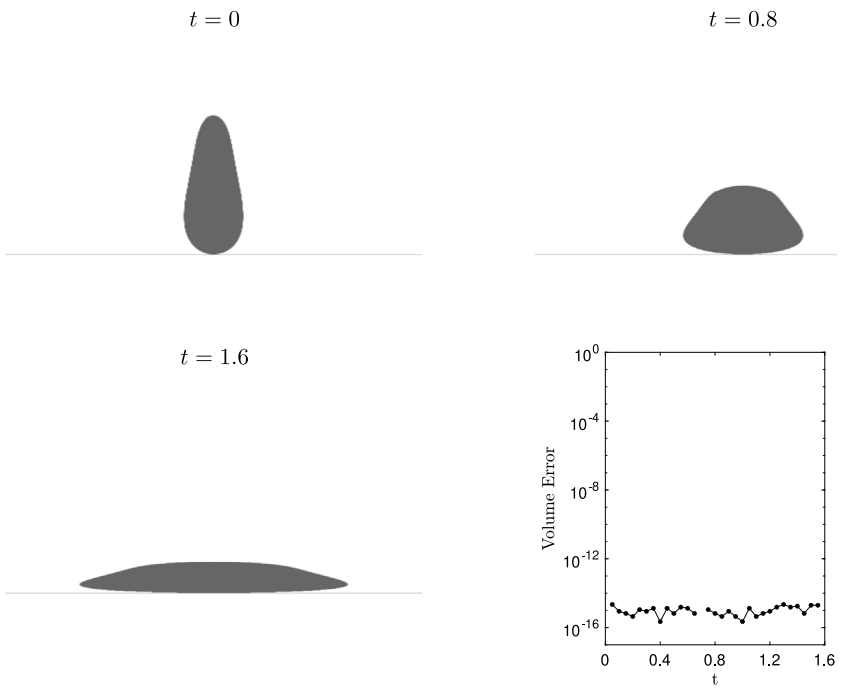


Fig. 8. Top left: the initial state of the pear shape at $t = 0$. The pear condition at $t = 0.8$ ($t = T_{\text{fin}}/2$) (top right) and at the final time $t = 1.6$ ($t = T_{\text{fin}}$) (bottom left).

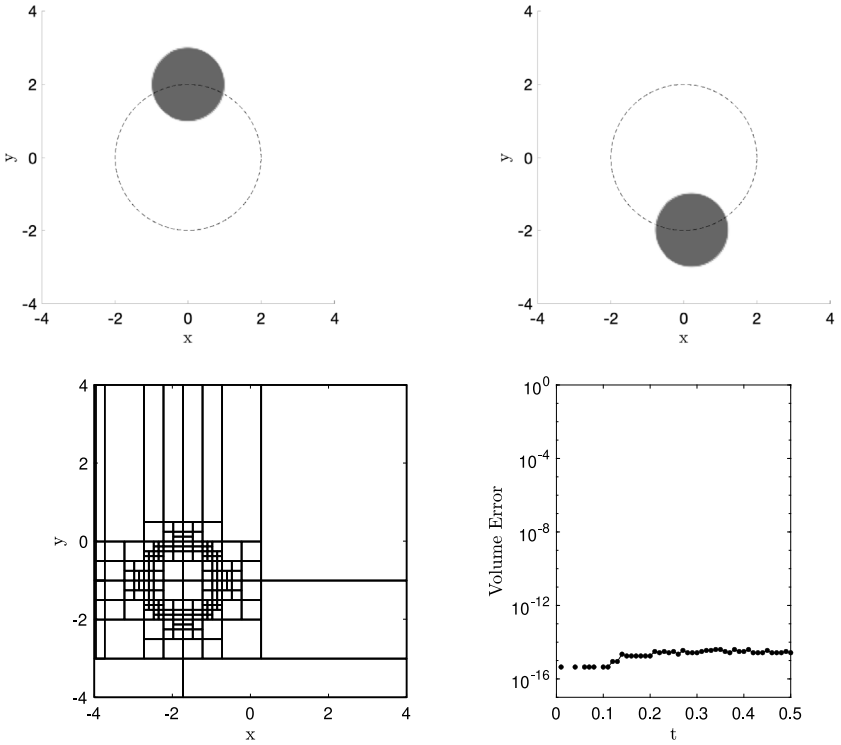


Fig. 9. The unit disk in a circular motion. The distance of its center of mass to the origin is 2 throughout the simulation. The center of mass stays on the orbit (i.e., a circle of radius 2), and the volume is conserved.

Table 2
Volume fraction, centroid, center of mass, unit gradient vector.

Volume fraction	$v_f(\Omega_i) = \frac{\int f d\Omega_i}{\max(f)_{\Omega_i} vol(\Omega_i)}$
Centroid	$\underline{x}_c = \left(\frac{\int x f d\Omega_i}{vol(\Omega_i)}, \frac{\int y f d\Omega_i}{vol(\Omega_i)}\right)$
Center of mass	$\underline{x}_m = \left(\frac{\int x f d\Omega}{vol(\Omega)}, \frac{\int y f d\Omega}{vol(\Omega)}\right)$
Unit gradient	$\widehat{\underline{g}} = \frac{\nabla f}{\lvert\nabla f\rvert}$

Even though the method has yet to be intended for interface reconstruction, we can utilize the interpolant, its derivatives, and integrals to compute the volume fractions, centroids, center of mass, and gradients needed for a very rudimentary reconstruction. The formula provided in Table 2 can be utilized. The use of volume fraction for basic interface reconstructions is in the spirit of the volume-of-fluid (VOF) method [12] and its variants. Moreover, the use of a root-finding strategy in the direction of unit gradients to find a point in the neighborhood of the interface is common in the Moment-of-Fluid methods [6,14,16] and its vast literature.

We first compute volume fractions of all patches and keep the ones between, say, 5% and 95%. The patches containing volume fractions between 0 and 1 are shared between two mediums. The top left figure of Fig. 10 shows patches/zones in bold where the volume fractions range between 5% and 95%. We can compute the centroids and the unit gradient vectors (at the centroids) for those patches. A zoom-in view of one of the patches, pointed by the arrow, is shown in the top right figure of Fig. 10. In that particular patch, the volume fraction is about 60.4%.

To find an approximate interface point in each patch, we can search for a point away from the centroid in the direction of the gradient vector using the following parametric equation.

$$\underline{x}(\tau) = \underline{x}_c + \tau \text{diam}(\Omega_c)\widehat{\underline{g}}_c,$$

where $\text{diam}(\Omega_c)$ is the diameter of the patch/zone where the centroid x_c is located. We can then use the APU interpolant f and root-finding method for finding parameter τ such that $f(\underline{x}_{\text{ifc}}) = 1/2$. Essentially, we find points where the values of the function f are $1/2$ (a halfway value of the two mediums). For VOF-based flow solvers like ours, it is common to plot an iso-contour of 0.5 of volume fractions. On a particular patch, as in the top right figure of Fig. 10, the point on the interface is colored in red. After those points are found, we can translate and scale their distances relative to the center of mass, transform them into polar coordinates, and sort their angles. One can then plot the points in a counter-clockwise way to get the rough approximate curve of the interface, as shown at the bottom of Fig. 10.

An important geometrical quantity in simulations of multiphase flows is the interface curvature, which directly determines the surface tension force along the interface at any given point. Next, the values of interface curvature at those points can be computed with (see [10])

$$\kappa = \frac{|f_y^2 f_{xx} - 2 f_x f_y f_{xy} + f_x^2 f_{yy}|}{(f_x^2 + f_y^2)^{3/2}}.$$

In this case, the exact curvature for all points at the boundary of the unit disk is $\kappa = 1$. In our computations, the curvature at those points is between 8 to 12 digits accuracy. Fig. 10 shows the log of absolute error of the curvature at each computed interface point.

5.5. Experiment 5 (rotation of multibody)

For Experiment 5, we are rotating a unit disk and a unit squircle together (both centered along the circle of radius 2) under a pure rotation unit vector field $\underline{u} = (-y/r, x/r)$, with $r = \sqrt{x^2 + y^2}$ from time $t \in [0, 2]$. The two-medium function is represented in r and θ at time $t = 0$ with a slope of 100 as

$$\begin{aligned} f(r,\theta) =& 1 + \tfrac{1}{2}\tanh(100(1 - ((r\cos(\theta + \pi/4))^2 + (r\sin(\theta + \pi/4) - 2)^2))) + \\ & \tfrac{1}{2}\tanh(100(1 - ((r\cos(\theta - \pi/4))^4 + (r\sin(\theta - \pi/4) - 2)^4))) + 1 \end{aligned}$$

If the computation is done in the cartesian coordinates, then rectangular patches are going to break up due to the rotations. However, in the polar domain, the rotation becomes a translation in the angular direction as

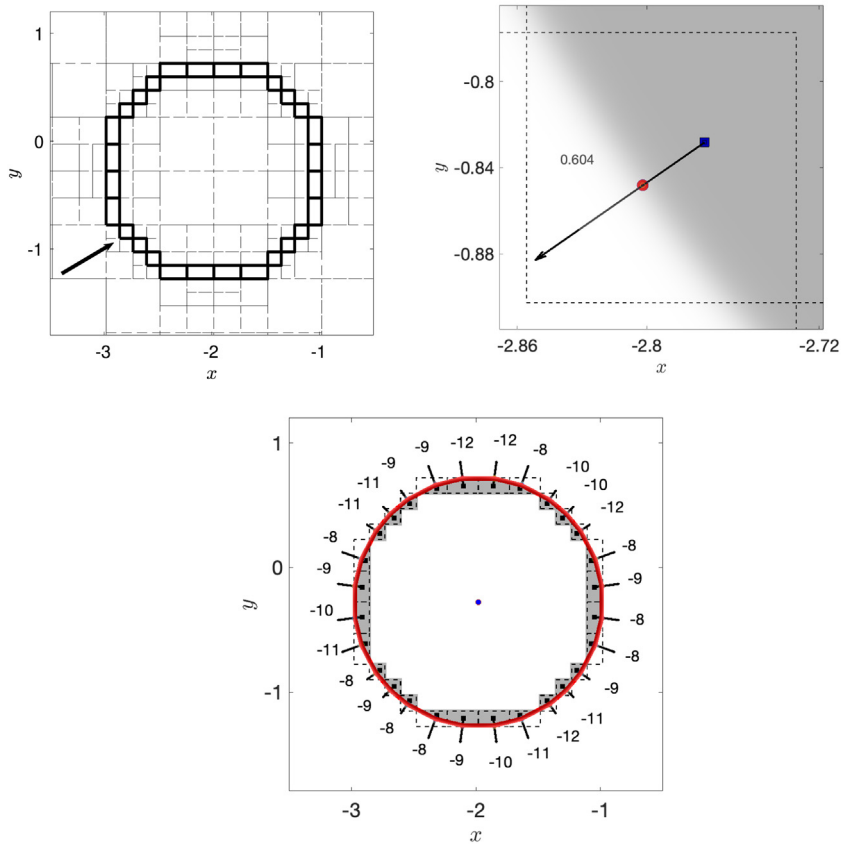


Fig. 10. Top left: a zoom-in view of the disk somewhere around the third quadrant from Fig. 9. Boxes/zones with volume fractions ranging from 5% to 95% are in bold. Top right: a zoom-in view of one of the patches pointed by the arrow on the top left figure. The patch has a volume fraction of about 60.4%. The point on the interface is obtained using a root-finding method applied to the parametric equation of a line starting from the centroid in the direction of the unit gradient outward. Bottom: points on the interface are then connected to recover an approximate curve (in red) of the interface. The error of the curvature, i.e., the $\log_{10}(|\kappa - 1|)$, with respect to the exact curvature at the boundary of a unit disk, is also shown. The center of mass location is accurate to 14 digits. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$\underline{u}(r, \theta) = (0, 1)$. Fig. 11 also shows that the circle and squircle, once transformed into the polar domain, become an egg-like and pear-like geometry.

The circle and the squircle are positioned such that the shortest distance between the two objects is no more than pi/4 in the angular direction. This tests whether the APU constructors can correctly handle and adaptively refine regions between the two interfaces. The simulation shows no degradations of the total volume and no changes in shapes. When the pear and the egg are close to $\theta = 2\pi$, one can shift the domain upward to keep the rotation since the interpolant is initially constructed for non-periodic domains.

5.6. Experiment 6 (angular deformation with cosine multiplier)

Using the unit disk provided in Experiment 5, we can perform angular deformation of it in the θ direction only. We can similarly use the cosine multiplier as in Experiment 2 so that the disk is recovered at the original position after the deformation. Fig. 12 shows its dynamic in the cartesian coordinates recorded at different times. Finally, the disk is recovered at its original position, and the volume error remains at machine zero throughout. At $t = 1.25$, we construct the approximation of the interface using the same technique done in Experiment 4. In addition to using only a centroid in a patch, to get more interface points nearby the patch, we can add an extra 1 or 2 more points along the direction of the tangent line, passing the centroids as initial guesses for the root-finding step. Having additional

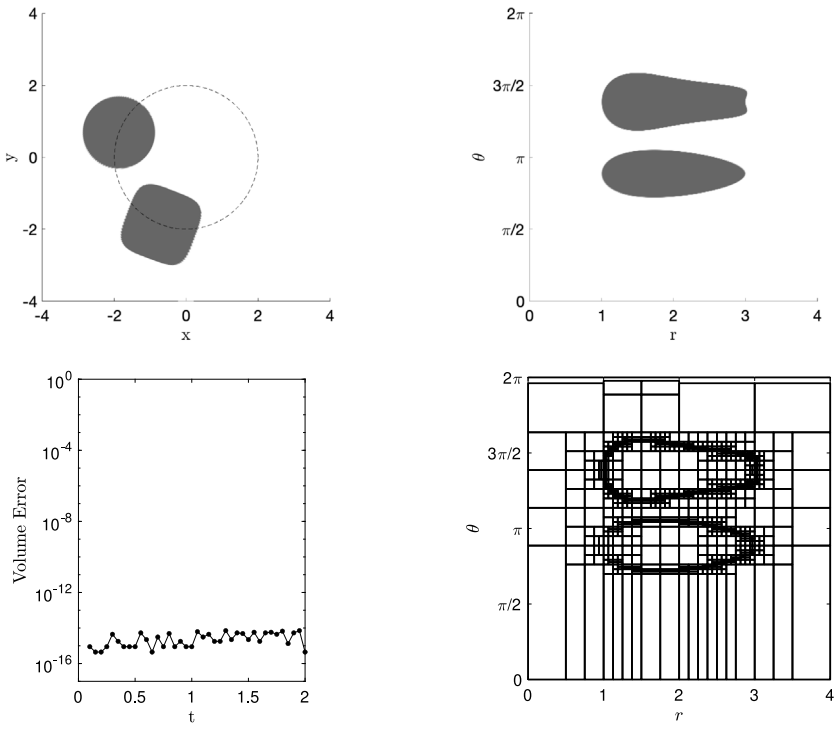


Fig. 11. Rotations of a circle and a squircle under unit pure rotational vector field. In polar coordinates, the shapes are egg-like and pear-like geometry. The top left figure shows the position of both mediums at $t = 2$. The top right figure shows their positions at $t = 2$ in polar coordinates, where the motion becomes a simple translation. The patch distributions at $t = 2$ show the level of details around the interfaces.

interface points, typically where the curvature changes quickly, will help us better parametrize the interface when needed. The bottom left figure of Fig. 12 shows the patches with volume fractions between 7.5% and 95%, and the red line is the reconstructed interface.

6. Discussion

We should point out that the adaptive partition of unity method with moving patches described here is similar in flavor to the volume-of-fluid (VOF) method [12]. We are not creating divergence-free bases (not explicitly on purpose) to approximate the two-medium function f. For the shape-deformation problems, the two main differences between the method and standard VOF are that the grids are deformed, and reconstruction steps can be avoided. If needed, the quantity of volume fraction in each deformed patch can be computed locally by integrating the scalar function. Moreover, unlike the standard VOF method, the method does not directly advect the volume fraction and reconstruct the function later. Instead, patches containing pieces of materials or volume fractions are moving along with the vector field; hence the volume conservation can be maintained with high accuracy.

The method described here looks relatively simple for linear divergence-free types of vector fields that affect material contour in a specific way. Moreover, rectangular patches can be made of different sizes in those cases, and the global interpolant can still be spectrally accurate. As provided in one of the numerical examples, one may switch to using polar domains for vector fields with pure rotations.

The patches may break one or more constraints provided in Section 3 for problems that involve more complicated nonlinear vector fields. One possible remedy is to couple it with a method that domain decomposes the flow field regions so that locally, the linearized flow behaves like the ones provided in Table 1. In other words, the linearized model (Jacobian) regions of the nonlinear ODEs. We are investigating this approach for future study, especially comparing the proposed method with some standard benchmark problems found in [9,11,21,23,24,26,31,34,36,37] for 2D and 3D cases.

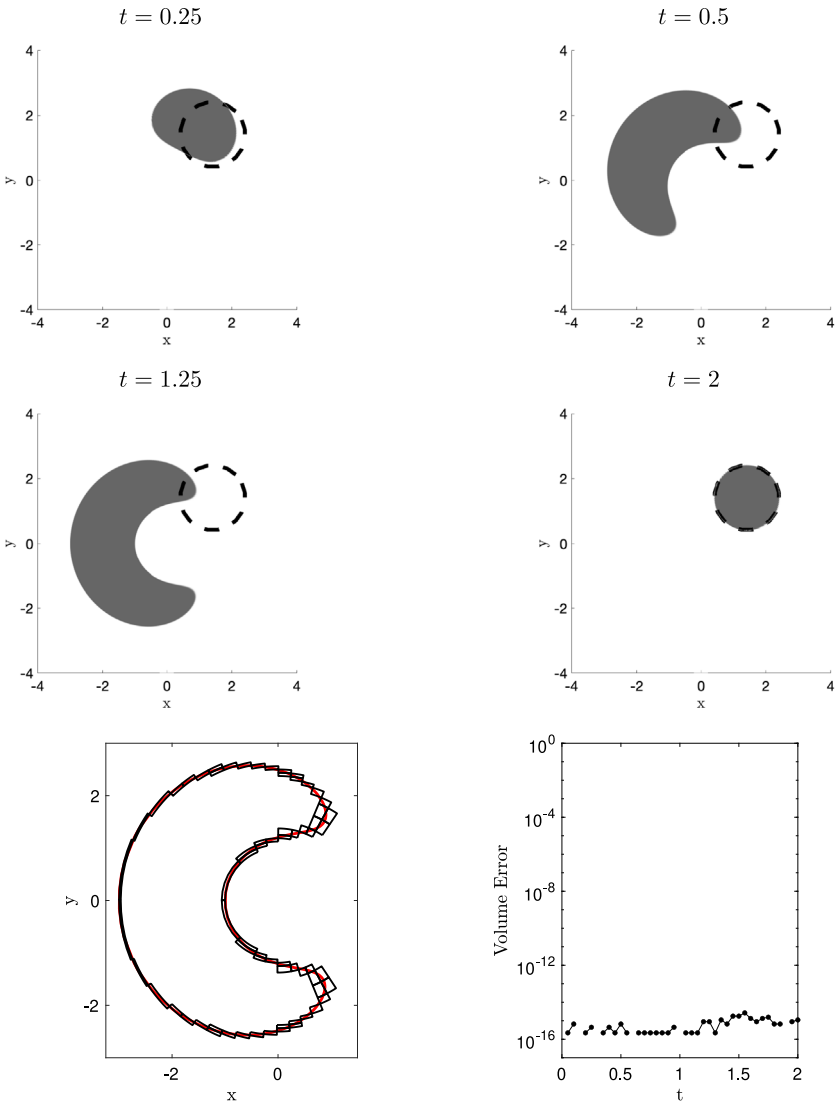


Fig. 12. Stretching a unit disk under angular deformation. The dashed line is the initial position at $t = 0$. Top left is the position at $t = 0.25$; Top right at $t = 0.5$; Middle: $t = 1.25$ (left) and finally at $t = 2$ (right) back to the original position. Bottom: a zoom-in view of the reconstructed interface (red) using patches with volume fractions between 7.5% and 95% at $t = 1.25$. The volume error stays relatively flat at the order of machine precision throughout the simulation. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Two of our numerical experiments also showed that although the interface is not explicitly tracked, the highly accurate APU interpolant can be used for reconstructing the interface. Additionally, differentiation and integration operators can be done in the same framework. This is useful when one wants to utilize them to model material structure or fluid–structure interactions in multi-phase flow simulations.

7. Conclusions

This study demonstrates that the adaptive partition of unity method with moving patches works very well with linear divergence-free vector fields while maintaining volume conservation with high accuracy. The method is simple

to implement for shape deformation problems and in the same spirit as the VOF method but with spectrally accurate global interpolants. Using the technique for other types of vector fields or other more difficult benchmark problems is currently under investigation.

Acknowledgments

We thank Kevin Aiton and Toby Driscoll for making the adaptive partition of unity codes open-source and UMass Dartmouth Center for Scientific Computing and Data Science Research (CSCDR) for providing rapid prototyping servers and computing facilities for this project. We also thank anonymous reviewers for their time and effort in reviewing our work and for feedback on improving our manuscript. This work was funded by the National Science Foundation DMS-2012011.

References

[1] K.W. Aiton, PUchebfun Github Repository, URL https://github.com/kevinwaiton/PUchebfun.
[2] K.W. Aiton, T.A. Driscoll, An adaptive partition of unity method for Chebyshev polynomial interpolation, SIAM J. Sci. Comput. 40 (1) (2018) A251–A265, http://dx.doi.org/10.1137/17M112052X.
[3] K.W. Aiton, T.A. Driscoll, An adaptive partition of unity method for multivariate Chebyshev polynomial approximations, SIAM J. Sci. Comput. 41 (5) (2019) A3230–A3245, http://dx.doi.org/10.1137/18M1184904.
[4] K.W. Aiton, T.A. Driscoll, Preconditioned nonlinear iterations for overlapping Chebyshev discretizations with independent grids, SIAM J. Sci. Comput. 42 (4) (2020) A2360–A2370, http://dx.doi.org/10.1137/19M1242483.
[5] W. Aniszewski, T. Ménard, M. Marek, Volume of Fluid (VOF) type advection methods in two-phase flow: A comparative study, Comput. & Fluids 97 (2014) 52–73, http://dx.doi.org/10.1016/j.compfluid.2014.03.027.
[6] X. Chen, X. Zhang, A predicted-Newton's method for solving the interface positioning equation in the MoF method on general polyhedrons, J. Comput. Phys. 384 (2019) 60–76, http://dx.doi.org/10.1016/j.jcp.2018.12.038.
[7] H.F. Davis, A.D. Snider, Introduction to Vector Analysis, fourth ed., Allyn and Bacon, Boston, 1979.
[8] T.A. Driscoll, N. Hale, L.N. Trefethen, Chebfun Guide, Pafnuty Publications, 2014, URL http://www.chebfun.org/docs/guide/.
[9] V. Dyadechko, M. Shashkov, Moment-of-Fluid Interface Reconstruction, Los Alamos Report la-UR-05-7571, 2005, p. 49.
[10] R. Goldman, Curvature formulas for implicit curves and surfaces, Comput. Aided Geom. Design 22 (7) (2005) 632–658, http://dx.doi.org/10.1016/j.cagd.2005.06.005.
[11] D. Harvie, J. Cooper-White, M. Davidson, Deformation of a viscoelastic droplet passing through a microfluidic contraction, J. Non-Newton. Fluid Mech. 155 (1–2) (2008) 67–79, http://dx.doi.org/10.1016/j.jnnfm.2008.05.002.
[12] C. Hirt, B. Nichols, Volume of Fluid (VOF) method for the dynamics of free boundaries, J. Comput. Phys. 39 (1) (1981) 201–225, http://dx.doi.org/10.1016/0021-9991(81)90145-5.
[13] H. Huang, M.C. Sukop, X.-Y. Lu, Multiphase Lattice Boltzmann Methods: Theory and Application, John Wiley and Sons, Inc, Chichester, West Sussex, 2015.
[14] M. Jemison, E. Loch, M. Sussman, M. Shashkov, M. Arienti, M. Ohta, Y. Wang, A Coupled Level Set-Moment of Fluid Method for Incompressible Two-Phase Flows, J. Sci. Comput. 54 (2–3) (2013) 454–491, http://dx.doi.org/10.1007/s10915-012-9614-7.
[15] M.B. Liu, G.R. Liu, Smoothed particle hydrodynamics (SPH): An overview and recent developments, Arch. Comput. Methods Eng. 17 (1) (2010) 25–76, http://dx.doi.org/10.1007/s11831-010-9040-7.
[16] T. Marić, Iterative volume-of-fluid interface positioning in general polyhedrons with Consecutive Cubic Spline interpolation, J. Comput. Phys.: X 11 (2021) 100093, http://dx.doi.org/10.1016/j.jcpx.2021.100093.
[17] S. Osher, R. Fedkiw, Level Set Methods and Dynamic Implicit Surfaces, in: Applied Mathematical Sciences, vol. 153, Springer New York, New York, NY, 2003, http://dx.doi.org/10.1007/b98879.
[18] S. Osher, J.A. Sethian, Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations, J. Comput. Phys. 79 (1) (1988) 12–49, http://dx.doi.org/10.1016/0021-9991(88)90002-2.
[19] G. Pólya, M. Schiffer, Convexity of functionals by transplantation, J. D'Anal. Math. 3 (1) (1953) 245–345, http://dx.doi.org/10.1007/BF02803593.
[20] J.D. Ramshaw, J.A. Trapp, A numerical technique for low-speed homogeneous two-phase flow with sharp interfaces, J. Comput. Phys. 21 (4) (1976) 438–453, http://dx.doi.org/10.1016/0021-9991(76)90039-5.
[21] S.J. Ruuth, B.T. Wetton, A simple scheme for volume-preserving motion by mean curvature, J. Sci. Comput. 19 (1) (2003) 373–384.
[22] J.A. Sethian, A fast marching level set method for monotonically advancing fronts, Proc. Natl. Acad. Sci. 93 (4) (1996) 1591–1595, http://dx.doi.org/10.1073/pnas.93.4.1591.
[23] J.A. Sethian, J.A. Sethian, Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science, second ed., Cambridge monographs on applied and computational mathematics, (no. 3) Cambridge University Press, Cambridge, U.K. ; New York, 1999.
[24] P.A. Stewart, N. Lay, M. Sussman, M. Ohta, An improved sharp interface method for viscoelastic and viscous two-phase flows, J. Sci. Comput. 35 (1) (2008) 43–61, http://dx.doi.org/10.1007/s10915-007-9173-5.
[25] M. Sussman, P. Smereka, S. Osher, A level set approach for computing solutions to incompressible two-phase flow, J. Comput. Phys. 114 (1) (1994) 146–159, http://dx.doi.org/10.1006/jcph.1994.1155.

[26] T. Takahashi, Y. Dobashi, I. Fujishiro, T. Nishita, Volume preserving viscoelastic fluids with large deformations using position-based velocity corrections, Vis. Comput. 32 (1) (2016) 57–66, http://dx.doi.org/10.1007/s00371-014-1055-x.
[27] H. Theisel, T. Weinkauf, H. Hege, H.-P. Seidel, Topological methods for 2D time-dependent vector fields based on stream lines and path lines, IEEE Trans. Vis. Comput. Graphics 11 (4) (2005) 383–394, http://dx.doi.org/10.1109/TVCG.2005.68.
[28] A.-K. Tornberg, B. Engquist, A finite element based level-set method for multiphase flow applications, Comput. Vis. Sci. 3 (1–2) (2000) 93–101, http://dx.doi.org/10.1007/s007910050056.
[29] L.N. Trefethen, Approximation Theory and Approximation Practice, in: Applied mathematics, Society for Industrial and Applied Mathematics, Philadelphia, 2013.
[30] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, Y.-J. Jan, A front-tracking method for the computations of multiphase flow, J. Comput. Phys. 169 (2) (2001) 708–759, http://dx.doi.org/10.1006/jcph.2001.6726.
[31] G. Tryggvason, R. Scardovelli, S. Zaleski, Direct Numerical Simulations of Gas-Liquid Multiphase Flows, Cambridge University Press, Cambridge ; New York, 2011, oCLC: ocn664324526.
[32] J. Tsitsiklis, Efficient algorithms for globally optimal trajectories, IEEE Trans. Automat. Control 40 (9) (1995) 1528–1538, http://dx.doi.org/10.1109/9.412624.
[33] S.O. Unverdi, G. Tryggvason, A front-tracking method for viscous, incompressible, multi-fluid flows, J. Comput. Phys. 100 (1) (1992) 25–37, http://dx.doi.org/10.1016/0021-9991(92)90307-K.
[34] W. von Funck, H. Theisel, H.-P. Seidel, Vector field based shape deformations, ACM Trans. Graph. 25 (3) (2006) 1118–1125, http://dx.doi.org/10.1145/1141911.1142002.
[35] F. Xiao, Y. Honma, T. Kono, A simple algebraic interface capturing scheme using hyperbolic tangent function, Internat. J. Numer. Methods Fluids 48 (9) (2005) 1023–1040, http://dx.doi.org/10.1002/fld.975.
[36] H.-K. Zhao, B. Merriman, S. Osher, L. Wang, Capturing the behavior of bubbles and drops using the variational level set approach, J. Comput. Phys. 143 (2) (1998) 495–518, http://dx.doi.org/10.1006/jcph.1997.5810.
[37] H. Zolfaghari, D. Izbassarov, M. Muradoglu, Simulations of viscoelastic two-phase flows in complex geometries, Comput. & Fluids 156 (2017) 548–561, http://dx.doi.org/10.1016/j.compfluid.2017.05.026.