An Energy Stable High-Order Cut Cell Discontinuous Galerkin Method with State Redistribution for Wave Propagation

Christina G. Taylor[a,*], Lucas C. Wilcox[b], Jesse Chan[a,*]

[a]*Department of Computational Applied Mathematics and Operations Research, Rice University,*
[b]*Department of Applied Mathematics, Naval Postgraduate School,*

Abstract

Cut meshes are a type of mesh that is formed by allowing embedded boundaries to “cut” a simple underlying mesh resulting in a hybrid mesh of cut and standard elements. While cut meshes can allow complex boundaries to be represented well regardless of the mesh resolution, their arbitrarily shaped and sized cut elements can present issues such as the *small cell problem*, where small cut elements can result in a severely restricted CFL condition. State redistribution, a technique developed by Berger and Giuliani [1], can be used to address the small cell problem. In this work, we pair state redistribution with a high-order discontinuous Galerkin scheme that is L_2 energy stable for arbitrary quadrature. We prove that state redistribution can be added to a provably L_2 energy stable discontinuous Galerkin method on a cut mesh without damaging the scheme's L_2 stability. We numerically verify the high order accuracy and stability of our scheme on two-dimensional wave propagation problems.

Keywords: Energy stable discontinuous Galerkin, State redistribution, Cut meshes, Embedded boundary methods

1. Introduction

Here we present a provably energy stable discontinuous Galerkin (DG) method on 2D cut meshes using state redistribution. Our method falls underneath the umbrella of Embedded Boundary methods, where embedded regions Ω_E are removed from the overall embedding domain, $\hat{\Omega}$, which we take to be Cartesian, to yield the final PDE domain

$$\Omega = \hat{\Omega} \setminus \Omega_E. \tag{1}$$

Here we limit ourselves only to 2D domains and thus $\hat{\Omega}, \Omega_E, \Omega \in \mathbb{R}^2$. An example of a domain with embedded boundaries is shown in Figure 1a.

Of interest to us is the simulation of hyperbolic conservation laws. Such conservation laws govern a wide variety of natural phenomena and include the wave equation, shallow water equations, and compressible Euler equations. The inclusion of embedded boundaries allows simulation of fluid flow

*Corresponding author(s)
Email addresses: `cgt3@rice.edu` (Christina G. Taylor), `jesse.chan@rice.edu` (Jesse Chan)

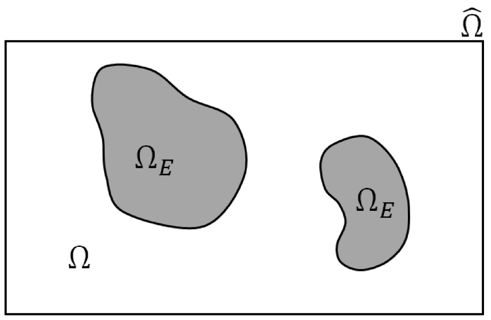
around objects and more geometrically complex domains. These laws feature wave-like behavior, making the acoustic wave equation,

$$\frac{1}{c^2}\frac{\partial p}{\partial t} + \nabla \cdot \boldsymbol{u} = 0, \tag{2}$$

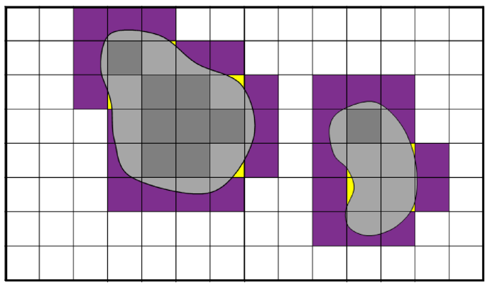
$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla p = 0, \tag{3}$$

a model problem before considering more complex hyperbolic conservation laws [2]. Here we limit our focus to the scalar wave equation, but our scheme can be applied to any linear symmetric hyperbolic system [3].

In general, linear hyperbolic conservation laws satisfy a continuous L_2 energy conservation property: the L_2 energy is constant in time. When dealing with linear conservation laws, certain numerical methods preserve this energy conservation at the semi-discrete level and are thus said to be energy conservative. If the numerical method additionally is able to dissipate energy, i.e. the L_2 energy can decrease in time, it is said to be energy stable. For high-order methods, energy stable formulations are particularly pertinent as high-order methods can suffer from instability [4]. High-order methods are of interest to us as they have greater accuracy per degrees of freedom [4] and are better suited to unsteady flows [5] due to their lower dissipation compared to low-order methods [6]. However, high-order methods also feature more degrees freedom per element and thus are more expensive under mesh refinement. To avoid the need to refine our mesh we appeal to cut meshes to capture embedded boundaries.



(a) Example domain.



(b) Example cut mesh.

Figure 1. An example of an embedding Cartesian domain, $\hat{\Omega}$, with embedded boundaries defining regions Ω_E to exclude and the resulting simulation domain Ω. Once meshed, elements remain Cartesian (white), become cut cells (active portion purple and yellow), or get excluded from the simulation domain (dark grey). Note cut meshes can result in extremely small or skewed elements, such as those shown in yellow.

Cut meshes are a form of hybrid mesh and can be linear or curvilinear in nature. In cut meshes the (typically Cartesian) embedding domain is discretized using standard meshing techniques without regard for the embedded boundaries to create a background mesh. Then, the embedded boundaries are allowed to "cut" and exclude elements from the background mesh resulting in a hybrid mesh of cut and standard elements. An example is given in Figure 1b.

Cut meshes are attractive for use with embedded boundaries as they are relatively easy to generate and can capture complex geometries well while maintaining computational efficiency [7]. Cut meshes have a long history in computational fluid dynamics field with one of their first appearances in [8] in the 1970s. Since their introduction they have been used for a number of applications such as fluid dynamics [9, 10, 11], fluid-structure interaction [12], front tracking [13], moving boundaries [14], and physiology models [15]. Here we pair our high-order DG scheme with a Cartesian cut mesh that features a coarse, computationally efficient Cartesian quadrilateral background mesh with curvilinear cut elements.

Despite their advantages, cut meshes, which features arbitrarily shaped/sized cut elements, can pose difficulties for many methods and classes of problems. A recurring challenge is the construction of volume quadrature rules on cut elements, though this challenge is relative straightforward to address using techniques from the curvilinear meshing community [3, 16] when exact quadrature is not needed. A greater challenge when simulating hyperbolic problems on cut meshes is the *small cell problem*, where the arbitrarily shaped (and sized) cut elements severely restrict the CFL (Courant-Friedichs-Lewy; i.e., the maximum stable time step) condition of the simulation.

1.1. Methods for Addressing the Small Cell Problem

Addressing the small cell problem is an active area of research for which a number of methods have been developed. One way to avoid the small cell problem altogether is to use implicit time integration as in [17]. However, the cost and complexity of implicit time integration can be prohibitive. Hybrid explicit-implicit time integration schemes such as in [18, 19] seek to marry the stability of implicit time integration with the practicality of explicit time integration by using implicit integration on small elements and explicit integration elsewhere. Such local time integration schemes can also be fully explicit by using subcycling [20] on small cells. However the use of localized time integration requires coupling of different time integration schemes and may not sufficiently stabilize all small elements [20].

More standard method-of-lines approaches using special DG and finite element (FE) schemes have also been developed that make use of special penalty terms or ghost values to stabilize small elements. One such example is presented in [21] where penalization of jumps in normal derivatives is used to build stabilization into the spatial discretization's mass and stiffness matrices. This has been done for DG and finite element [22, 23, 24, 21] formulations. The use of penalty terms can yield the attractive result of being able to assert energy stability [21] or provide a priori error estimates such as in [24, 23]. However, these error estimates may not always be robust for all geometries [25] though some provably are [23].

Another penalty-term based method is the domain-of-dependence method, which was developed solely for hyperbolic problems and uses penalty terms to redistribute mass from a cut cell to its neighbors [26]. While it was originally developed on triangular meshes for the advection equation [26] it has since been extended to general linear hyperbolic problems on general elements [27] and in 1D to nonlinear hyperbolic problems [28]. It has been shown to be L_2 energy stable [27], total variation diminishing in the L_1 norm [26], and monotone for piecewise constant solutions [29].

While penalty terms seek to provide stabilization by modifying the DG or FE scheme's formulation, flux-based stabilization is also possible for finite volume (FV) and DG methods. One such flux-based stabilization approach is h-box methods, which seek to solve a Riemann problem over a "box" of length/scale h at cell interfaces to compute the numerical fluxes for each cut cell. h-box methods were first developed for 1D FV cut cells in [30]. This method was further developed in [31, 32] for adaptation to 2D and to improve accuracy and computational efficiency [33]. Another flux-based stabilization method is flux redistribution, which was originally developed for front tracking in a hybrid finite difference, finite volume scheme in [34]. Similar to the h-box method, flux redistribution seeks to adjust small cut elements/cells' flux values but typically does so using flux values from neighboring elements. While flux redistribution is conservative, it is only first-order accurate [34, 35].

In the same spirit as flux redistribution is cell merging, also called cell agglomeration, where small elements are merged with neighbors to eliminate small elements altogether. Cell merging is a common solution to the small cell problem [11, 7] and can be used in conjunction with other stabilization methods [20]. However, determining a provably stable time step with cell merging can be difficult in 2D and 3D meshes [36]. Similar to cell merging is cell linking, which has the benefit of maintaining the original, unmerged mesh [37, 38, 39].

To address the small cell problem in our solver we use *state redistribution*, a method developed by Berger and Giuliani in [1]. State redistribution is in a sense an extension of cell merging and linking but with the issue of order dependency resolved. Importantly, it is high-order accurate and conservative in the sense that is preserves the average of the solution and polynomial degree [1].

In this paper we prove that when state redistribution is applied to an energy stable DG scheme the resulting scheme remains energy stable. The proof of energy stability under state redistribution is this project's main contribution to the Embedded Boundary method literature.

The rest of this paper is organized as follows: in Section 2 we describe our DG formulation, which is energy stable for arbitrary quadrature rules on cut meshes, before describing state redistribution in more detail in preparation for Section 2.3, where we present our proof of energy stability under state redistribution. Next, in Section 3 we numerically verify the theoretical properties of our scheme and compare our solver to the benchmark "Pacman" model for 2D acoustic wave propagation [40]. Lastly in the Appendices we highlight some of the implementational details of our scheme, such as our approach to representing the embedded boundaries and constructing volume quadrature on cut elements.

2. Methods

2.1. The DG Formulation

For our DG solver we consider the linear acoustic wave equation. On a given domain Ω with appropriate initial and boundary conditions the acoustic wave equation is given by

$$\frac{1}{c^2}\frac{\partial p}{\partial t} + \nabla \cdot \boldsymbol{u} = 0, \tag{4}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla p = 0, \tag{5}$$

where c is the speed of sound in the medium, p is the pressure, and $\boldsymbol{u} \in \mathbb{R}^2$ is the velocity. We assume a Cartesian embedding domain $\hat{\Omega}$ over which we impose a Cartesian rectangular mesh of $n_x \times n_y$ elements. As stated previously our true domain Ω is related to the Cartesian domain $\hat{\Omega}$ by

$$\Omega = \hat{\Omega} \setminus \Omega_E, \tag{6}$$

where Ω_E is the set of embedded regions to be excluded as illustrated previously in Figure 1. The domain Ω is then decomposed into a cut mesh of N_h non-overlapping Cartesian and cut elements,

D^k, satisfying

$$\Omega = \bigcup_{k=1}^{N_h} D^k. \tag{7}$$

For a given polynomial degree N, we define the solution on Cartesian elements using a reference element $\hat{D} = [-1,1]^2$. For the reference element solution space we use $\mathcal{Q}^N(\hat{D})$, the space of tensor products of degree N polynomials in the $\hat{x}$ and $\hat{y}$ direction on the reference element. For this space we use a tensor product basis of 1D Lagrange polynomials,

$$\phi_{ij}(\hat{x},\hat{y}) = \ell_i(\hat{x})\ell_j(\hat{y}), \quad i,j = 0,1,2,...,N. \tag{8}$$

We define the solution on cut elements similarly with a nodal basis, but due to the arbitrary shape of cut elements we can no longer exploit a tensor product structure or use a reference element. On cut elements we use the space of total degree N polynomials $\mathbb{P}^N(D^k) = \text{span}\{x^i y^j \mid i+j \leq N, i,j \in \mathbb{Z}_0^+\}$. For convenience, we define these polynomials nodally on the physical cut element using 2D Lagrange polynomials,

$$\varphi_i(x,y) = \ell_i(x,y), \quad \ell_i(x_j,y_j) = \delta_{ij} \text{ for } 0 \leq i,j, \quad i+j \leq N, \tag{9}$$

whose nodal points are approximate Fekete points [41]. Fekete points are a class of interpolation nodes that yield well-conditioned interpolation matrices [41] with the added benefit of being easy to generate; however, any set of unisolvent points will suffice. Our cut quadrature rules also use approximate Fekete points, but require a larger set of nodes which changes the points' distribution. In Appendix 6.3 we elaborate on how we generate approximate Fekete points for use as volume quadrature nodes via a moment-fitting scheme. The same process is used to generate the polynomial basis nodes and is a form of moment fitting [42].

We define our solution and test space as $\mathcal{Q}^N(D^k)$ on Cartesian elements and $\mathbb{P}^N(D^k)$ on cut elements. Given our solution $(p,\boldsymbol{u})$ and test functions $(q,\boldsymbol{v})$, the standard DG formulation [43] for the acoustic wave equation on a single element D^k is given by

$$\int_{D^k} \frac{1}{c^2}\frac{\partial p}{\partial t} q = \int_{D^k} \nabla\cdot\boldsymbol{u} q + \frac{1}{2}\int_{\partial D^k} [\![\boldsymbol{u}]\!]\cdot\boldsymbol{n} q, \tag{10}$$

$$\int_{D^k} \frac{\partial \boldsymbol{u}}{\partial t}\cdot\boldsymbol{v} = \int_{D^k} \nabla p\cdot\boldsymbol{v} + \frac{1}{2}\int_{\partial D^k} [\![p]\!]\,\boldsymbol{n}\cdot\boldsymbol{v}. \tag{11}$$

where $\boldsymbol{n}$ is the outward normal of the element D^k and $[\![\cdot]\!]$ is used to denote the "jump" between the solution on D^k and its neighbors

$$[\![p]\!] = p^+ - p, \qquad [\![\boldsymbol{u}]\!] = \boldsymbol{u}^+ - \boldsymbol{u}. \tag{12}$$

where the "+" superscript is used to denote the exterior value of a quantity on a neighboring element.

Under exact quadrature, this formulation is energy stable on affine meshes and standard DG approximation spaces. However, cut elements necessitate custom quadrature rules, which in our current implementation do not provide exactness guarantees. To maintain energy stability we convert the standard formulation to its skew-symmetric form which is energy stable under arbitrary quadrature [3, 16]. To derive the skew symmetric formulation, we split the volume terms into halves

$$\int_{D^k} \nabla \cdot \boldsymbol{u} q = \frac{1}{2}\int_{D^k} \nabla \cdot \boldsymbol{u} q + \frac{1}{2}\int_{D^k} \nabla \cdot \boldsymbol{u} q, \tag{13}$$

$$\int_{D^k} \nabla p \cdot \boldsymbol{v} = \frac{1}{2}\int_{D^k} \nabla p \cdot \boldsymbol{v} + \frac{1}{2}\int_{D^k} \nabla p \cdot \boldsymbol{v}, \tag{14}$$

and then apply integration by parts to *one* of the halved terms. This allows terms to be combined/canceled with the surface integral terms to yield a new expression for the RHS (right-hand side):

$$\int_{D^k} \nabla \cdot \boldsymbol{u} q + \frac{1}{2}\int_{\partial D^k} [\![\boldsymbol{u}]\!] \cdot \boldsymbol{n} q = \frac{1}{2}\int_{D^k} (\nabla \cdot \boldsymbol{u} q - \boldsymbol{u} \cdot \nabla q) + \frac{1}{2}\int_{\partial D^k} \boldsymbol{u}^+ \cdot \boldsymbol{n} q, \tag{15}$$

$$\int_{D^k} \nabla p \cdot \boldsymbol{v} + \frac{1}{2}\int_{\partial D^k} [\![p]\!]\, \boldsymbol{n}\boldsymbol{v} = \frac{1}{2}\int_{D^k} (\nabla p \cdot \boldsymbol{v} - p \nabla \cdot \boldsymbol{v}) + \frac{1}{2}\int_{\partial D^k} p^+ \boldsymbol{n} \cdot \boldsymbol{v}. \tag{16}$$

Substituting these expansions into the original formulation in Equations (10) and (11) yields the skew-symmetric formulation

$$\int_{D^k} \frac{1}{c^2}\frac{\partial p}{\partial t} q = \frac{1}{2}\int_{D^k} (\nabla \cdot \boldsymbol{u} q - \boldsymbol{u} \cdot \nabla q) + \frac{1}{2}\int_{\partial D^k} \boldsymbol{u}^+ \cdot \boldsymbol{n} q, \tag{17}$$

$$\int_{D^k} \frac{\partial \boldsymbol{u}}{\partial t} \cdot \boldsymbol{v} = \frac{1}{2}\int_{D^k} (\nabla p \cdot \boldsymbol{v} - p \nabla \cdot \boldsymbol{v}) + \frac{1}{2}\int_{\partial D^k} p^+ \boldsymbol{n} \cdot \boldsymbol{v}. \tag{18}$$

Lastly we add upwinding penalty terms to the skew-symmetric formulation. These terms dissipate energy via penalizing jumps in the solution at element boundaries. Adding these terms yields our final formulation:

$$\int_{D^k} \frac{1}{c^2}\frac{\partial p}{\partial t} q = \frac{1}{2}\int_{D^k} (\nabla \cdot \boldsymbol{u} q - \boldsymbol{u} \cdot \nabla q) + \frac{1}{2}\int_{\partial D^k} (\boldsymbol{u}^+ \cdot \boldsymbol{n} q + \tau_p [\![p]\!]\, q), \tag{19}$$

$$\int_{D^k} \frac{\partial \boldsymbol{u}}{\partial t} \cdot \boldsymbol{v} = \frac{1}{2}\int_{D^k} (\nabla p \cdot \boldsymbol{v} - p \nabla \cdot \boldsymbol{v}) + \frac{1}{2}\int_{\partial D^k} (p^+ \boldsymbol{n} \cdot \boldsymbol{v} + \tau_u [\![\boldsymbol{u}]\!] \cdot \boldsymbol{v}). \tag{20}$$

In all of our simulations we set the penalty parameters τ_p and τ_u equal to $1/2$; in general they must satisfy $\tau_p, \tau_u \geq 0$. By adding energy dissipation, the penalty terms renders our final formulation energy stable [3, 44]. While we assume here that the speed of sound in the medium c is constant for simplicity's sake, varying c can also be accommodated by incorporating its effect into the mass matrix as shown in [45].

2.2. State Redistribution

Next we briefly describe the action of state redistribution. State redistribution stabilizes the solution on small elements via a careful merging and redistribution of the solution with the solution on neighboring elements. In subsequent work since its original publication, a newer, upgraded version of state redistribution, called weighted state redistribution, uses a different weighting for the projections to allow state redistribution's effect to be smoothly activated [46]. Here we use the original, volume-weighted version of state redistribution for high-order DG methods.

In state redistribution, we first identify cut elements in need of stabilization. A common condition is cut elements with volume less than one-half the volume of the background Cartesian element, which we also use. For each small element D^k, we determine a *merging neighborhood*, over which we merge the constituent elements' solutions via a volume-weighted projection, Π_k. Mathematically we denote the merge neighborhood as

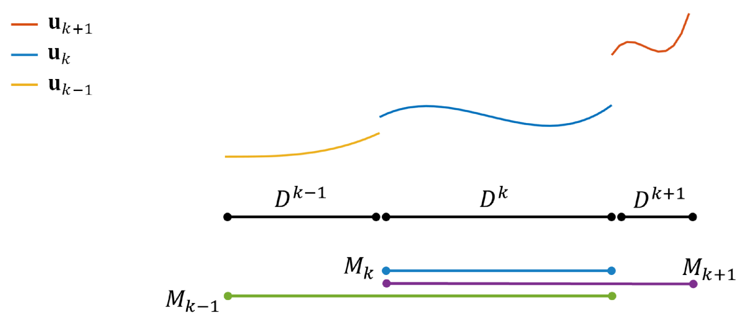
$$M_k = \{k, k_1, k_2, ...\}, \tag{21}$$

the set of all element indices in element D^k's merge neighborhood. For elements that do not require merging $M_k = \{k\}$.

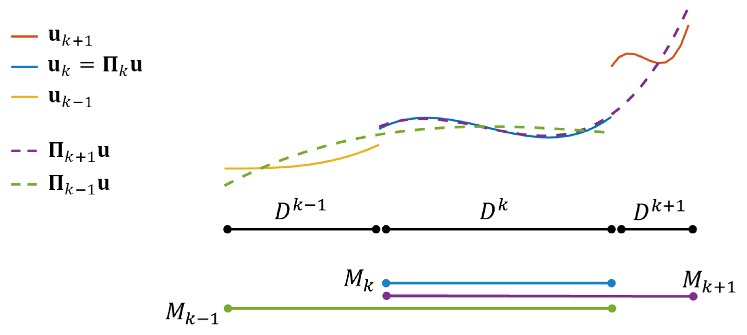
Unlike cell merging and linking, a given element may contribute to multiple merge neighborhoods. We denote all merge neighborhoods to which an element D^k contributes as the set

$$C_k = \{j : k \in M_j\}. \tag{22}$$

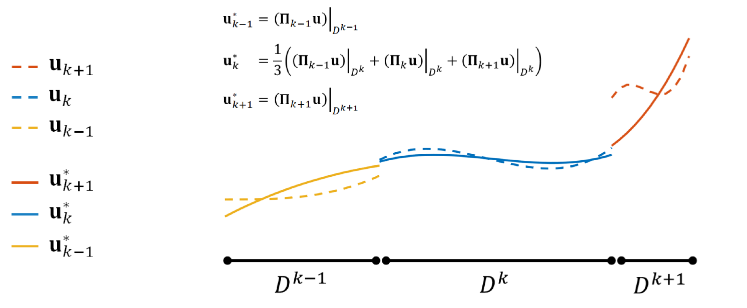
We use the notation $|C_k|$ to denote the cardinality of C_k, i.e., number of elements in C_k. The final solution on each element is the average of all projected solutions to which a given element contributed. This process is illustrated on a 1D mesh with two small elements and one full-sized element in Figure 2.



(a) Here we denote the solution on element D^k as $\boldsymbol{u}_k$. Elements $D^{(i-1)}$ and $D^{(i+1)}$ are in need of merging. For each element, including full-sized element D^k, we compute merge neighborhoods M_{k-1}, M_k, M_{k-1}.


$\mathbf{u}_{k+1}$
$\mathbf{u}_{k} = \mathbf{\Pi}_k\mathbf{u}$
$\mathbf{u}_{k-1}$
$\mathbf{\Pi}_{k+1}\mathbf{u}$
$\mathbf{\Pi}_{k-1}\mathbf{u}$
D^{k-1}
D^{k}
D^{k+1}
M_k
M_{k+1}
M_{k-1}

(b) Over each merge neighborhood we compute the projected solutions. Notice in the case of a full-sized element the projected solution on its merge neighborhood is merely the original solution on that element.


$\mathbf{u}^*_{k-1} = (\mathbf{\Pi}_{k-1}\mathbf{u})\big|_{D^{k-1}}$
$\mathbf{u}^*_{k} = \frac{1}{3}\left((\mathbf{\Pi}_{k-1}\mathbf{u})\big|_{D^{k}} + (\mathbf{\Pi}_{k}\mathbf{u})\big|_{D^{k}} + (\mathbf{\Pi}_{k+1}\mathbf{u})\big|_{D^{k}}\right)$
$\mathbf{u}^*_{k+1} = (\mathbf{\Pi}_{k+1}\mathbf{u})\big|_{D^{k+1}}$
$\mathbf{u}_{k+1}$
$\mathbf{u}_{k}$
$\mathbf{u}_{k-1}$
$\mathbf{u}^*_{k+1}$
$\mathbf{u}^*_{k}$
$\mathbf{u}^*_{k-1}$
D^{k-1}
D^{k}
D^{k+1}

(c) To compute the final updated solution $\boldsymbol{u}^*$ on each element we take the average of all projected solutions (restricted to the element of interest) to which that element contributed.

Figure 2. The three-step process of state redistribution in 1D over two-small elements and one full-sized element.

State redistribution has been shown to relax the CFL condition of cut meshes to be proportional to the CFL condition of the underlying background mesh. Importantly, state redistribution preserves polynomial order and conserves the average of the solution [1]. For systems of partial differential equations, such as the wave equation, state redistribution is applied to each component individually. State redistribution can be represented as a linear operator, S, acting on a semi-discrete solution $\boldsymbol{u}$, and is slightly dissipative in nature, making it a good partner for an energy stable scheme. The main computational complexity in using state redistribution is in setup: determining which elements are in need of stabilization, building their merge neighborhoods, and constructing the necessary operators. For a static mesh the state redistribution operators need only be computed once at the start of a simulation as a preprocessing step and then applied to the solution are each time step.

2.3. Proof of L_2 Stability of High-Order State Redistribution

To prove the energy stability of our final DG formulation with state redistribution, we appeal to work by Nordström and Winters [47] which states that an energy stable DG scheme to which a contractive filter is applied remains energy stable. Note that in general, filters can disrupt the energy stability of their underlying method [48]. We interpret the state redistribution operator as a filter, reducing our proof to showing that the state redistribution operator is contractive in the L_2 norm.

Theorem 2.1. *Let S be the state redistribution operator and $u \in L_2(\Omega)$. Then S is contractive,*

$$\|Su\|_{L_2(\Omega)} \leq \|u\|_{L_2(\Omega)} . \tag{23}$$

Before proving state redistribution is contractive we first introduce some notation and recall two basic lemmas.

Definition 2.2. *We define the weighted inner product $(\cdot,\cdot)_{M_k}$ over the elements of a merge neighborhood M_k,*

$$D^{M_k} = \{D^j : j \in M_k\} \subseteq \Omega,$$

for $u, v \in \mathbb{P}^N(D^{M_k})$ as

$$(u,v)_{M_k} = \sum_{j\in M_k} \frac{1}{|C_j|}(u,v)_{L_2(D^j)} \tag{24}$$

and denote its associated norm as

$$\|u\|_{M_k} = (u,u)_{M_k} = \sum_{j\in M_k} \frac{1}{|C_j|}\|u\|_{L_2(D^j)}. \tag{25}$$

Lemma 2.3. *(Jensen's inequality) For a set of numbers $\{x_i\}_{i=1}^n \in \mathbb{R}$ the following inequality holds [49]*

$$\left(\sum_{i=1}^n x_i\right)^2 \leq n\sum_{i=1}^n x_i^2.$$

Lemma 2.4. *Let $(\cdot,\cdot)$ denote some inner product on a domain D with inner product norm $\|\cdot\|$. For a given polynomial degree N, let $\Pi : \mathbb{P}^N(D) \to \mathbb{P}^N(D)$ be a projection operator satisfying*

$$(\Pi u, v) = (u, v), \quad \forall v \in \mathbb{P}^N(D).$$

Then the projection operator Π is contractive:

$$\|\Pi u\| \leq \|u\|.$$

The proofs of Lemmas 2.3 and 2.4 are straightforward using standard tools such as Jensen's and the Cauchy-Schwarz inequalities. We now present the proof of Theorem 2.3.

Proof. Let $(\cdot,\cdot)_{L_2(B)}$ denote the L_2 inner product on given subdomain $B \subseteq \Omega$. For a piecewise polynomial u the action of merging the element-wise solutions from all elements in a merge neighborhood M_k can be expressed as a volume-weighted L_2 projection Π_k over M_k. Since state redistribution conserves the moments of the solution, Π_k satisfies

$$(\Pi_k u, v)_{M_k} = (u, v)_{M_k}, \quad \forall v \in \mathbb{P}^N(D^{M_k}). \tag{26}$$

The state redistribution operator restricted to D^k can then be expressed as

$$Su|_{D^k} = \frac{1}{|C_k|} \sum_{j \in C_k} \Pi_j u. \tag{27}$$

Using this notation the application of state redistribution over the entire domain can be expressed as

$$\|Su\|^2_{L_2(\Omega)} = \sum_{k=1}^{N_h} \|Su|_{D^k}\|^2_{L_2(D^k)} \tag{28}$$

$$= \sum_{k=1}^{N_h} \int_{D^k} \left(\frac{1}{|C_k|} \sum_{j \in C_k} \Pi_j u \right)^2 \tag{29}$$

$$= \sum_{k=1}^{N_h} \frac{1}{|C_k|^2} \int_{D^k} \left(\sum_{j \in C_k} \Pi_j u \right)^2. \tag{30}$$

Applying Lemma 2.3 to (30), we have that

$$\sum_{k=1}^{N_h} \frac{1}{|C_k|^2} \int_{D^k} \left(\sum_{j \in C_k} \Pi_j u \right)^2 \leq \sum_{k=1}^{N_h} \frac{1}{|C_k|} \sum_{j \in C_k} \|\Pi_j u\|^2_{L_2(D^k)}, \tag{31}$$

where N_h is the number of elements in the mesh.

We can express nested sums over all k and $j \in C_k$ equivalently as a nested sum over all k and $j \in M_k$ and vice versa. The first of these corresponds to a sum over the final, merged solutions while the second corresponds to a sum of contributed terms from each element. Applying this change and the definition of the $\|\cdot\|_{M_k}$ norm yields

$$\sum_{k=1}^{N_h} \frac{1}{|C_k|} \sum_{j\in C_k} \|\Pi_j u\|^2_{L_2(D^k)} = \sum_{k=1}^{N_h} \sum_{j\in M_k} \frac{1}{|C_j|} \|\Pi_k u\|^2_{L_2(D^j)} \tag{32}$$

$$= \sum_{k=1}^{N_h} \|\Pi_k u\|^2_{M_k}. \tag{33}$$

We can then apply Lemma 2.4 to the RHS

$$\sum_{k=1}^{N_h} \|\Pi_k u\|^2_{M_k} \leq \sum_{M_k} \|u\|^2_{M_k} \tag{34}$$

before returning the sum of the weighted norm over all k and $j \in M_k$ to sums of the L_2 norm over all k and $j \in C_k$:

$$\sum_{k=1}^{N_h} \|u\|^2_{M_k} = \sum_{k=1}^{N_h} \sum_{j\in M_k} \frac{1}{|C_j|} \|u\|^2_{L_2(D^j)} = \sum_{k=1}^{N_h} \frac{1}{|C_k|} \sum_{j\in C_k} \|u\|^2_{L_2(D^k)}. \tag{35}$$

Since the summand $\|u\|_{L_2(D^k)}$ does not depend on j we have that

$$\sum_{j\in C_k} \|u\|^2_{L_2(D^k)} = |C_k| \|u\|^2_{L_2(D^k)}. \tag{36}$$

Substituting this into Equation (35) gives

$$\sum_{k=1}^{N_h} \frac{1}{|C_k|} \sum_{j\in C_k} \|u\|^2_{L_2(D^k)} = \sum_{k=1}^{N_h} \|u\|^2_{L_2(D^k)} = \|u\|^2_{L_2(\Omega)} \tag{37}$$

which via the inequalities in (31) and (34) yield the desired result:

$$\|Su\|^2_{L_2(\Omega)} \leq \|u\|^2_{L_2(\Omega)}.$$

□

3. Numerical Experiments

3.1. Manufactured Solution

To test the correctness of our solver we implement a manufactured solution for the wave equation

$$\frac{\partial p}{\partial t} + \nabla \cdot \boldsymbol{u} = f(x, y, t) \tag{38}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla p = 0. \tag{39}$$

For the manufactured solution, we prescribe the pressure to be

$$p = \cos(2\pi t)\sin(\pi x)\sin(\pi y). \tag{40}$$

Substituting the prescribed pressure into Equation (39) and taking $\boldsymbol{u}(x, y, t = 0) = 0$ yields

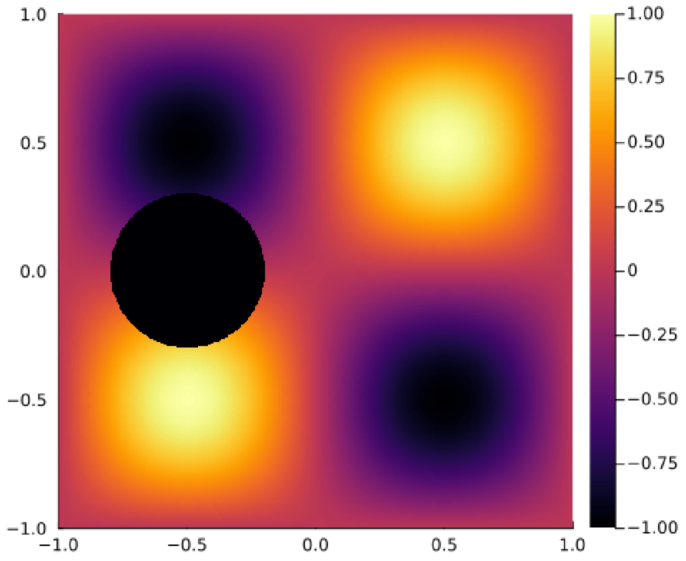
$$\boldsymbol{u} = \int_0^t \nabla p \, \mathrm{d}\tau = -\frac{1}{2}\begin{bmatrix} \sin(2\pi t)\cos(\pi x)\sin(\pi y) \\ \sin(2\pi t)\sin(\pi x)\cos(\pi y) \end{bmatrix} \tag{41}$$

and

$$f(x, y, t) = -\pi \sin(2\pi t)\sin(\pi x)\sin(\pi y). \tag{42}$$

We simulate this PDE (partial differential equation) on the Cartesian domain $[-1, 1]^2$ from which we cut out a circle C of radius $R = 0.3$ centered at $(x_0, y_0) = (-0.5, 0)$. We use the manufactured solution to prescribe initial and boundary conditions. To assess our solver we performed a h-convergence study using degree $N = 1, 2, 3,$ and 4 polynomials and simulated the manufactured solution using $h = \Delta x = \Delta y = 1/2,\ 1/4,\ 1/8,\ 1/16$.

Figure 3 shows the pressure solution at $t = 0$. Note the velocity field is zero at $t = 0$. For a degree N solution, we expect h^{N+1} convergence. Figure 4 shows the L_2 error convergence results with h^{N+1} rates for comparison.



.

Figure 3. Pressure field for the manufactured solution at $t = 0$ showing the embedded object.

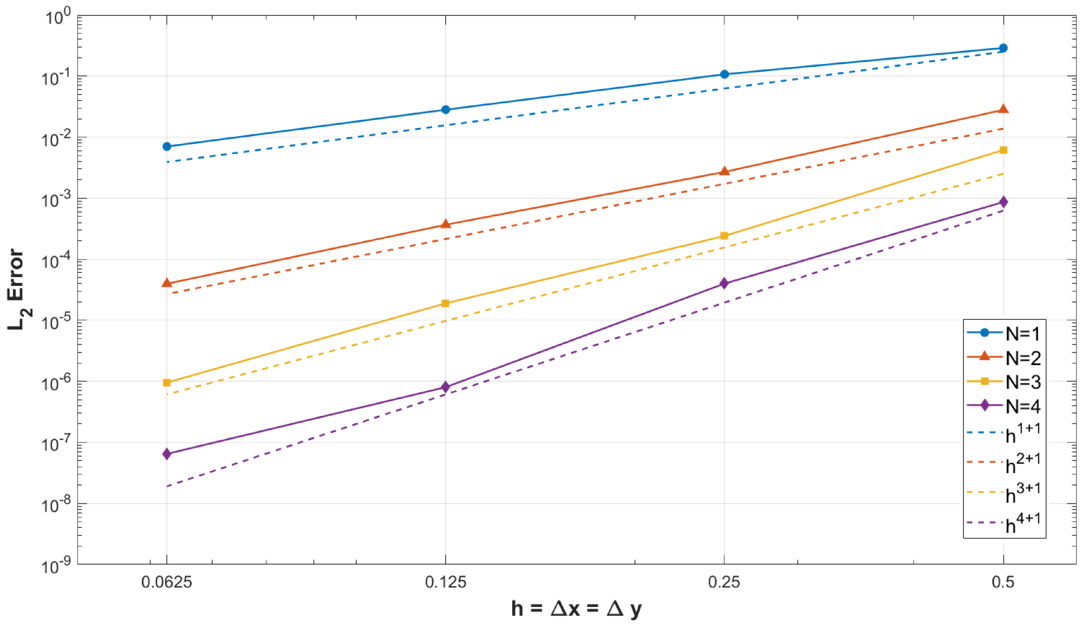


Figure 4. L_2 error for the manufactured solution at various grid sizes, h, and polynomial degree, N, at time $t = 1.3$.

3.2. Eigenvalues of RHS Operator

In this experiment we seek to highlight the influence of state redistribution on CFL condition in the presence of a small cell by examining the eigenvalues of the RHS operator with the eigenvalues of the combined RHS-state redistribution operator.

After the applying the DG discretization to the wave equation, we are left with the global system of ordinary differential equations

$$\frac{\partial \boldsymbol{U}}{\partial t} = \boldsymbol{M}^{-1}\boldsymbol{Q}\boldsymbol{U} = \mathbf{A}\boldsymbol{U} \tag{43}$$

where $\boldsymbol{U}$ is the global vector of unknowns, $\boldsymbol{M}$ is the global mass matrix, $\boldsymbol{Q}$ is the discrete action of the spatial derivatives, and $\mathbf{A} = \boldsymbol{M}^{-1}\boldsymbol{Q}$ is the overall action of the DG discretization prior to applying any time integration.

State redistribution can be seen as a linear operator, which we will denote as S, that acts on the current solution vector. Adding state redistribution to our scheme modifies Equation (43) by applying state redistribution to the solution vector prior to applying the discretization operator:

$$\frac{\partial \boldsymbol{U}}{\partial t} = AS\boldsymbol{U}. \tag{44}$$

We wish to compare the eigenvalues of A with the eigenvalues of AS. To perform this comparison we consider a circle of radius $R = 0.699$ embedded at the origin of a 8×8 Cartesian mesh on the domain $[-1, 1]^2$. This setup yields a volume ratio of 940 between a full Cartesian cell and the smallest cut cell and a length ratio of 21.7. Note that we expect state redistribution to improve the CFL condition/maximum eigenvalue magnitude by a factor similar to the length ratio between a full cell and the smallest cut cell. For our solution space we use degree 4 polynomials. The mesh for the discretization is shown in Figure 5.

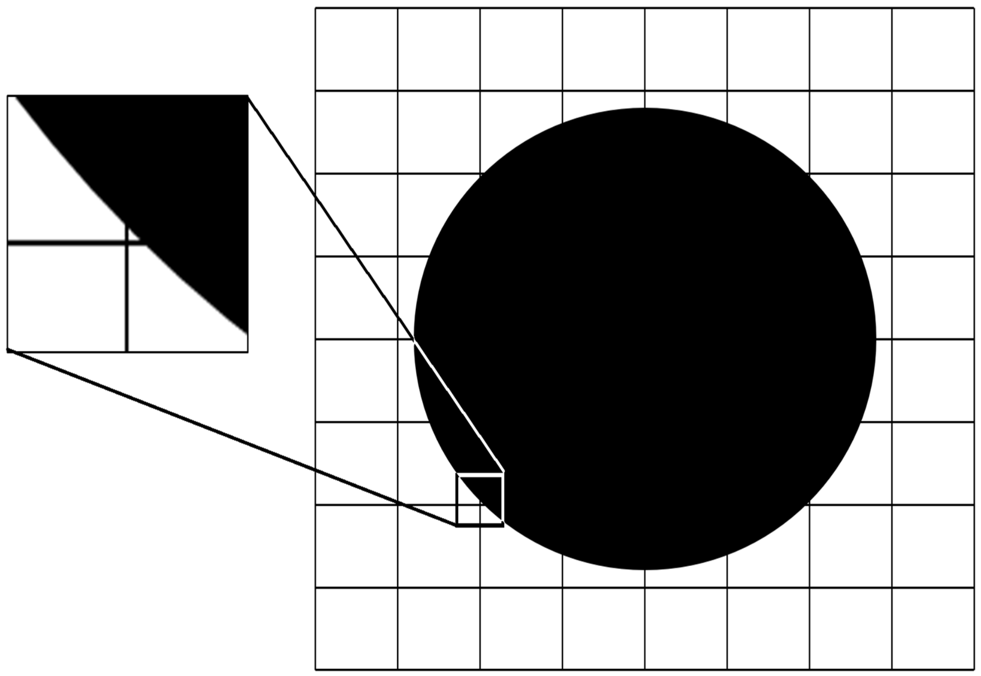
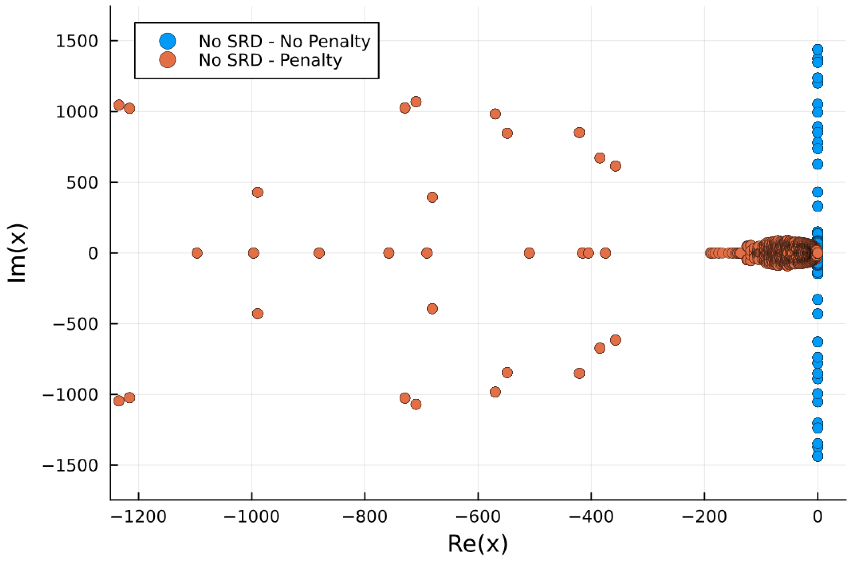


Figure 5. The mesh and embedded object for the eigenvalue experiments, with one of smallest cells highlighted.

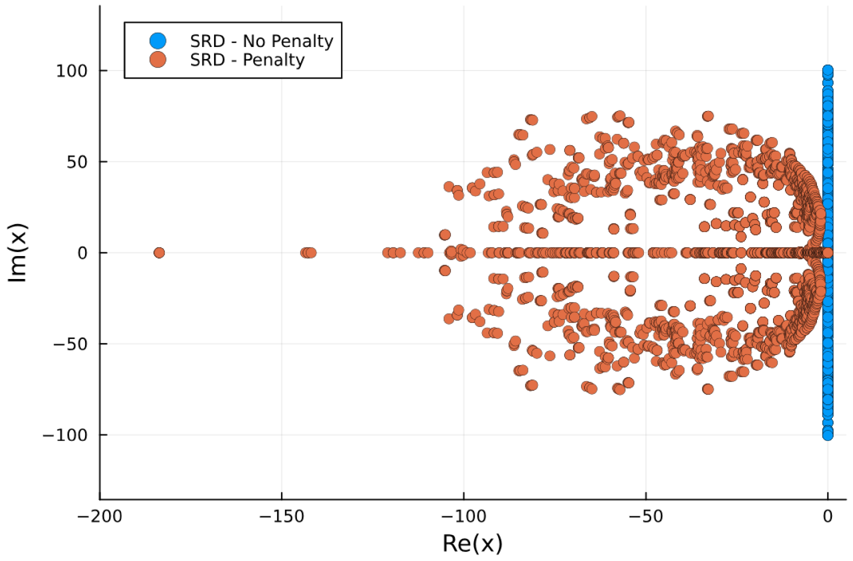
Of interest in these experiments is the sign of the imaginary component of the eigenvalues and the maximum eigenvalue magnitude. The imaginary component of the eigenvalues reflects the energy conservation status of the scheme: positive imaginary component reflects increasing energy, zero imaginary component energy conservation, and negative imaginary component energy dissipation. The maximum eigenvalue magnitude meanwhile dictates the CFL condition of the scheme; the smaller the magnitude, the more gracious the CFL condition.

Figure 6a shows the eigenvalues of the RHS operator without state redistribution for the original skew-symmetric energy conservative formulation and the penalized energy stable formulation. With neither state redistribution nor penalty terms, the eigenvalues of the RHS operator fall solely on the real axis. When penalty terms are added, the eigenvalues' imaginary components are pushed into the left-half plane but maintain a large maximum magnitude. These results confirm the respective energy conservation and stability of these two schemes.

Figure 6b shows the eigenvalues of the combined RHS-state redistribution operator with and without penalty terms. In both instances state redistribution is able to shrink the maximum eigenvalue magnitude and thus improve the CFL condition of the scheme. Table 1 summarizes the changes in maximum eigenvalue magnitude both with and without state redistribution.



(a) Eigenvalues without state redistribution.



(b) Eigenvalues with state redistribution.

Figure 6. Eigenvalues of the discretization operator with and without penalty terms and state redistribution. With penalty terms, the maximum magnitudes of the eignevalues' real components were 183 with state redistribution and 2157 without state redistribution. Without penalty terms, the maximum magnitudes of the real component of the eigenvalues were 6.07×10^{-8} with state redistribution and 1.24×10^{-13} without state redistribution.

Table 1. Maximum eigenvalues for the RHS operator with and without state redistribution and penalty terms.

	No Penalty	Penalty
No State Redistribution	1438	2157
State Redistribution	100	183
Ratio	14.33	11.74
For reference:	Volume ratio=940	Length ratio=21.7

3.3. Pacman Benchmark

Next we compare our solver's solution with the "Pacman" benchmark given in [40] to test the accuracy of our scheme. This benchmark provides an analytic solution to the acoustic wave equation for multiple wave sources scattering around a circle with an angular section removed, an object that resembles the video game character Pacman. This benchmark is of particular interest as it features a non-trivial embedded object and wave field while supplying an analytic solution to test simulations against. Here, we will only consider the case of a incoming plane wave originating from outside the pacman "mouth".

For our simulation, we take the circle radius to be $R = 1$ from which we remove the angular segment from $\theta = -\pi/6$ to $\theta = \pi/6$. We simulate the resulting pacman object on the Cartesian domain $x \in [-3.3, 3]$, $y \in [-3, 3]$ and a single 33×33 mesh. The offset in the lower x boundary is to prevent split cut elements, where a single background Cartesian element is cut into multiple cut elements [50], which our code does not currently support.

As given in [40], the analytic solution to the wave field is given in polar coordinates by infinite sums of either Hankel or Bessel functions depending on whether the point of interest is outside $(p^{(I)}, v_\theta^{(I)}, v_r^{(I)})$ or inside of the pacman mouth $(p^{(II)}, v_\theta^{(II)}, v_r^{(II)})$. The solution in the region outside the pacman mouth is given by:

$$p^{(I)}(r,\theta) = \sum_{n=0}^{\infty} \left(a_n^A \sin(n\theta) + a_n^S \cos(n\theta)\right) H_n^{(2)}(kr), \tag{45}$$

$$v_r^{(I)}(r,\theta) = \frac{\mathrm{i}}{Z_0} \sum_{n=0}^{\infty} \left(a_n^A \sin(n\theta) + a_n^S \cos(n\theta)\right) H_n^{'(2)}(kr), \tag{46}$$

$$v_\theta^{(I)}(r,\theta) = \frac{\mathrm{i}}{krZ_0} \sum_{n=0}^{\infty} n \left(a_n^A \cos(n\theta) - a_n^S \sin(n\theta)\right) H_n^{(2)}(kr). \tag{47}$$

Similarly the solution in the region inside the pacman mouth is given by:

$$p^{(II)}(r,\theta) = \sum_{n=0}^{\infty} b_n^A J_{(n+1/2)N}(kr) \sin\left(\left(n+\frac{1}{2}\right)N\theta\right) + b_n^S J_{nN}(kr)\cos(nN\theta), \tag{48}$$

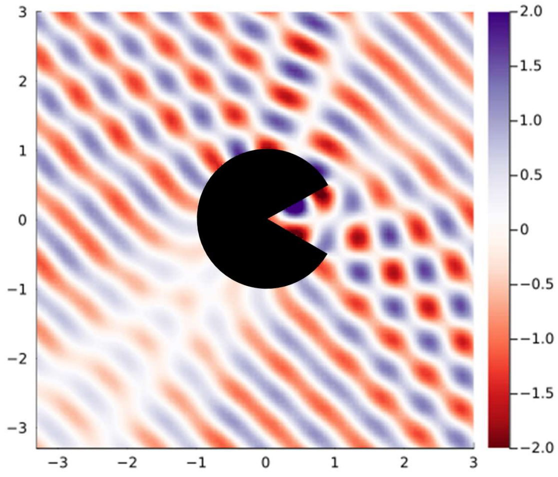
$$v_r^{(II)}(r,\theta) = \frac{\mathrm{i}}{Z_0}\sum_{n=0}^{\infty} b_n^A J'_{(n+1/2)N}(kr) \sin\left(\left(n+\frac{1}{2}\right)N\theta\right) + b_n^S J'_{nN}(kr)\cos(nN\theta), \tag{49}$$

$$v_\theta^{(II)}(r,\theta) = \frac{\mathrm{iN}}{krZ_0}\sum_{n=0}^{\infty} (n+1/2)\, b_n^A J_{(n+1/2)N}(kr) \cos\left(\left(n+\frac{1}{2}\right)N\theta\right) - nb_n^S J_{nN}(kr)\sin(nN\theta), \tag{50}$$

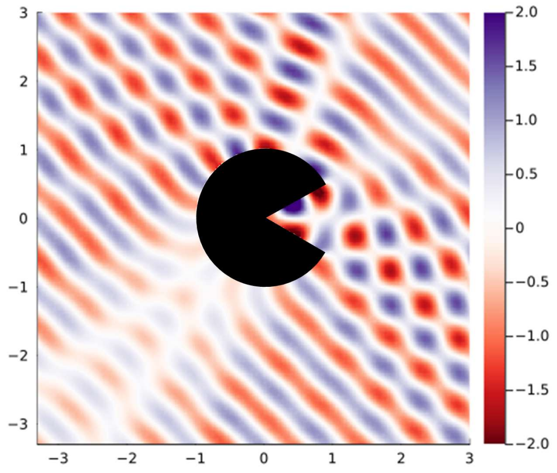
where $H_n^{(2)}$ is the n^{th} Hankel function of the second kind, J_n is the n^{th} Bessel function, $N = 6$ is the wedge number for defining the pacman mouth angle, and the coefficients $a_n^A, a_n^S, b_n^A, b_n^S$ are calculated as described in [40]. To compare this solution to our simulation, we truncate this sum to 100 terms. For our simulation we take $\rho = c_0 = 1$ and $k = 9$ which then dictates $Z_0 = 1$, $f = 1.432$, and $\omega = 9$. The plane wave is introduced from the upper right corner of the domain at an angle of $\pi/4$.

To introduce the plane wave in our simulation, we use the analytic solution as the boundary conditions on the Cartesian domain boundaries and zero-velocity boundary conditions on the pacman boundaries. The analytic solution at $t = 0$ is used as the initial condition for our solver. We ran the simulation up to an end time of $t = 1$ (approximately 1.43 periods) for degree $N = 1, 2, 3, 4$ polynomials using the `Tsit5` adaptive time integration scheme [51] provided in `OrdinaryDiffEq.jl` [52] and an initial time step of $\Delta t = 10^{-4}$.

Figure 7 shows the analytic and $N = 4$ solution. While the two seem indistinguishable, there is in fact large localized error present in the solved solution.



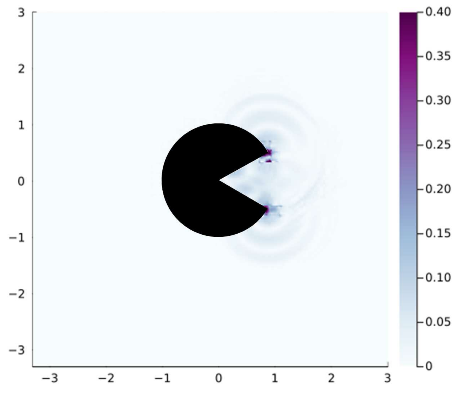
(a) Analytic solution.



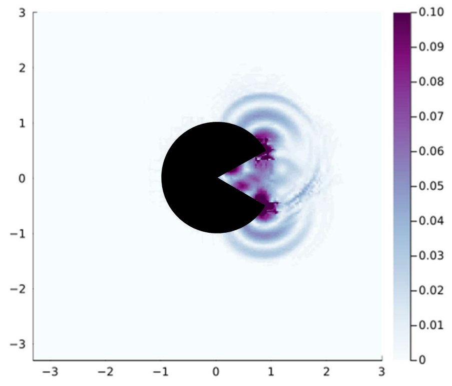
(b) Solver solution for $N = 4$.

Figure 7. Pressure field for the (a) analytic and (b) simulated solutions to the pacman wave scattering problem.

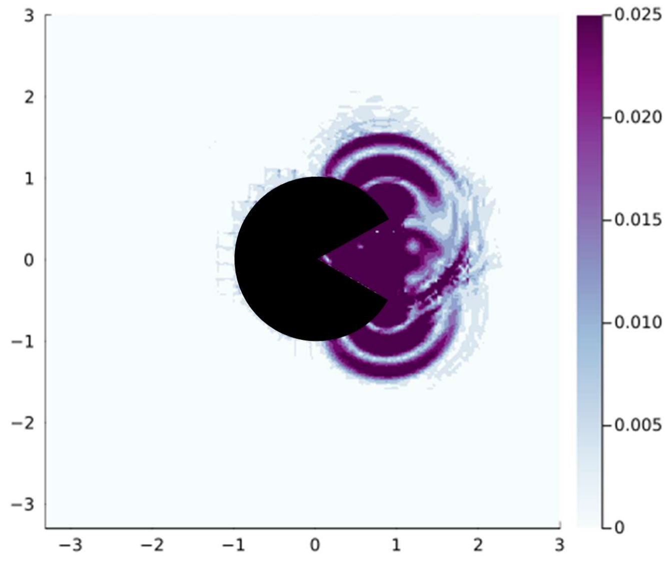
As described in more detail in Appendix 6.1, we represent embedded boundaries using explicit parameterizations. Due to this representation corners are represented exactly without smoothing. The sharpness of the convex corners is a source of localized error due to their lack of differentiability. Figure 8 shows the pointwise ℓ_2 error in $(p, \boldsymbol{u})$ around the pacman object (clipped to various plotting thresholds). The actual maximum magnitude of the pointwise error is 0.96. As can be seen in Figure 8a (whose color scale saturates at 0.4), the largest error values are highly localized and propagate outward from the pacman mouth's corners (Figure 8b). Besides the error at the corners very low magnitude error can also be seen all around the embedded object in Figure 8c. This low-level error around the object may be due to state redistribution and its dissipative nature.



(a) Threshold = 0.4.



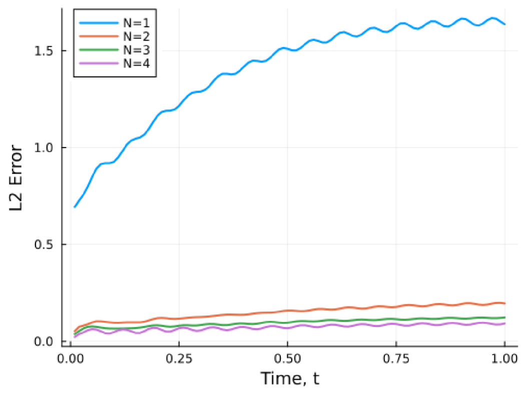
(b) Threshold = 0.1.



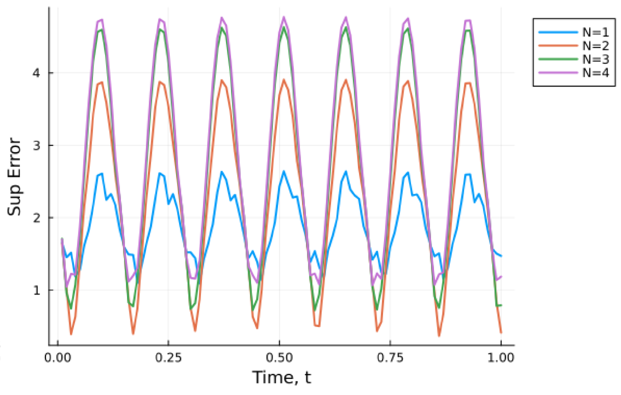
(c) Threshold = 0.025.

Figure 8. Pointwise error at $t = 1$ for the $N = 4$ solution with the plotting scale clipped to various threshold values.

Figure 9 shows the L_2 and L_∞ error over time. Notably, the L_∞ exhibits cyclic behavior; we theorize this is due to the corner effects of the pacman mouth. The L_2 errors also reflect this oscillatory behavior but still displays the expected long term behavior.



(a) L_2 error.



(b) L_∞ error.

Figure 9. L_2 and L_∞ error versus time for the pacman benchmark. Note that despite the disturbance caused by the mouth corners the L_∞ error remains bounded.

3.4. Fish Simulation

For our final numerical experiment we consider multiple embedded objects on a high-resolution mesh to test the versatility of our scheme. For this simulation, we embed 10 “fish” arranged in a triangle pattern on the Cartesian domain $[-1, 1]^2$ over which we impose a 120×120 mesh. Figure 10 shows the arrangement of fish in the Cartesian domain.

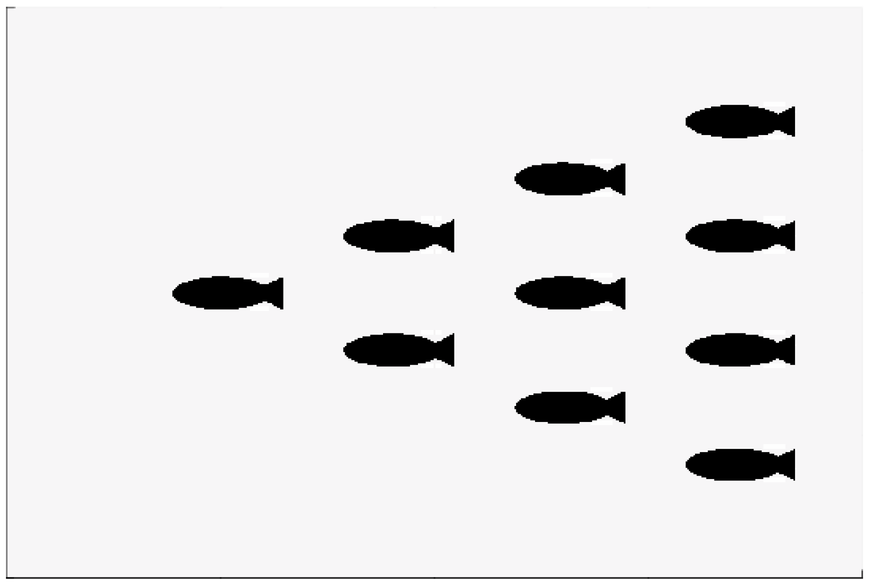
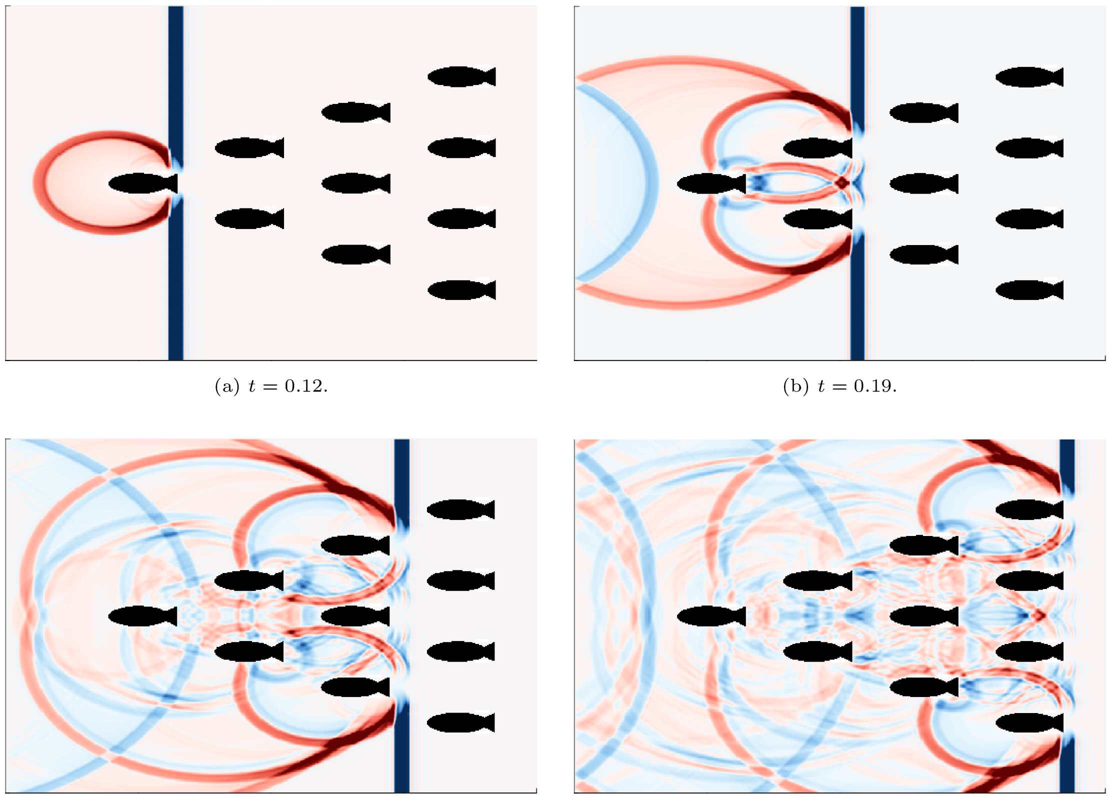


Figure 10. Embedded objects for the fish simulation. Note the fish contain sharp concave and convex corners.

We impose zero-pressure boundary conditions at the right Cartesian boundary ($x = 1$) and fish. On the top and bottom Cartesian domain boundaries ($y = -1, 1$) we impose extrapolation boundary conditions. For times $t \in (0, 0.05]$, the pressure on the left Cartesian boundary ($x = -1$) is set to $p = 2$ to generate a right-moving plane wave; for $t > 0.05$ zero-pressure conditions are enforced on this boundary instead. At time $t = 0$ both pressure and velocity fields are zero. We ran the simulation up to an end time of $t = 6$ using the `Tsit5` adaptive time integration scheme [51] provided in `OrdinaryDiffEq.jl` [52] and an initial time step of $\Delta t = 10^{-4}$. Figure 11 shows snapshots of the solution as the plane wave passes each column of fish at times $t = 0.12, 0.19, 0.26$, and 0.32.



(a) $t = 0.12$.

(b) $t = 0.19$.

(c) $t = 0.26$.

(d) $t = 0.32$.

Figure 11. Pressure field at various times showing the reflection and scattering of the initial pressure wave off and through the fish.

4. Conclusions

We have proven that state redistribution can be added to an energy stable DG formulation without affecting the scheme's energy stability. We implemented a high-order energy stable DG solver with state redistribution for the acoustic wave equation on 2D cut meshes. To accommodate the loss of polynomially exact quadrature rules on cut elements we use a skew-symmetric DG formulation which is energy conservative under arbitrary quadrature and penalize the jumps along the interfaces of elements to add energy dissipation. Our code for constructing cut meshes and their associated DG and state redistribution operators are given in the registered Julia packages `PathIntersections.jl` and `StartUpDG.jl` [53, 54]. All code for our experiments is provided in this paper's accompanying GitHub repository [55]. Our codes represent embedded boundaries using explicit parameterizations which allows exact computation of normal vectors on the embedded boundaries and exact representation of sharp features. As shown in our numerical results, the presence of sharp geometry features can be a source of error due to their lack of differentiability. We have numerically verified our solver's expected convergence rates, energy stability, and relaxation of the CFL condition via state redistribution.

5. Acknowledgments

Christina Taylor and Jesse Chan gratefully acknowledges support from National Science Foundation under awards GRFP-1842494 and DMS-1943186. The authors also thank Dr. Marsha Berger and Dr. Andrew Giuliani for helpful discussions on state redistribution and constructing cut meshes.

6. Appendix: Implementation Details

6.1. Embedded Boundary Representation

To represent the embedded boundaries, we break with much of the Embedded Boundary method literature and use explicit parameterizations in place of the more common level-set functions. In the absence of a level-set/signed-distance function we determine which regions to include/exclude via a right-hand rule. The code to generate our cut mesh and its interpolation, quadrature, and state redistribution operators are provided in the respective Julia packages `PathIntersections.jl` and `StartUpDG.jl` [53, 54].

Given a parameterization of a curve $\mathcal{C}_i : [0, 1] \to \mathbb{R}^2$ with parameter $s \in [0, 1]$, our routines use a step-based algorithm to calculate mesh-curve intersections. This algorithm walks/"steps" along the curve from $s = 0$ to $s = 1$ sensing for intersections with the mesh and calculating the corresponding s values as needed. This setup also allows users to define arbitrary "stop points". Stop points can denote junctions in piecewise curves or points of interest on the curve to be used alongside intersection points to define the faces of cut elements. The intersections and stop points are then used to determine cut and excluded elements and build explicit piecewise representations of cut elements' boundaries. These routines are given in the Julia package `PathIntersections.jl`. More information on the routines can be found in the `PathIntersections.jl` documentation [53]. An example of the results of the intersection routine, including stop points, are shown in Figure 12 for the pacman mesh.

The use of explicit parameterizations, while not extensible to 3D in our current implementation, comes with a number of benefits. Explicit parameterizations allow us to exactly represent sharp features and the curved boundaries of cut elements and exactly evaluate normal and tangent vectors via the use of automatic differentiation as provided in `ForwardDiff.jl` [56]. These normal vectors are used elsewhere in our codes for constructing surface and volume quadratures.

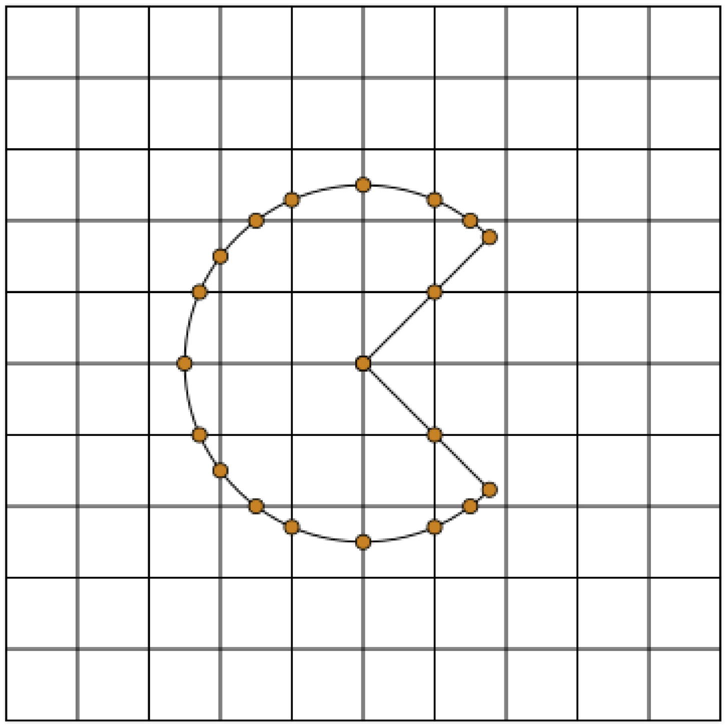


Figure 12. Example of the mesh-curve intersections calculated by `PathIntersections.jl` for the pacman mesh. Note the stop points at the corners of the pacman mouth are included with the intersection points for use in defining element faces.

6.2. Construction of Merge Neighborhoods

To determine when stabilization is needed for cut elements, we follow the convention in the original presentation of state redistribution in [1] and stabilize cut elements whose volume is less than one-half the volume of the Cartesian elements. We can calculate the approximate volume of the cut elements using their volume quadrature rules applied to the constant function $f(x, y) = 1$.

To determine merge neighborhoods, we break with convention, which typically dictates merging with the nearest neighboring element in the direction normal to the embedded boundary. Instead, we use a greedy, volume-based approach. For a given cut element in need of merging, we compute its face neighbors. The neighbor that yields that largest increase in volume is added to the cut element's merge neighborhood. This is greedy process is repeated, now using the neighboring elements of the entire merge neighborhood, until the total volume of the cut neighborhood is at least threshold volume. Notably, the outcome of this process is affected by the order in which neighboring elements are found as multiple neighboring elements may have the same volume. In the case of such ties, the first element to be listed is used for merging.

There are other, more advanced ways of computing merge neighbors, such as incorporating the information on the outward normals of the cut boundary as previously mentioned. In many cases our greedy scheme, which will always favor merging with Cartesian elements over cut elements, resembles the merge neighborhoods that would be found via a normal-based scheme. This is especially true in the meshes we consider, where most cut elements have Cartesian or near Cartesian neighbors. In such meshes the greedy nature of our scheme makes it efficient with little impact on quality. For more complex meshes where cut elements have no Cartesian/near-Cartesian neighbors our scheme may result in oddly shaped/oblong merging neighborhoods which may impact the extent to which the CFL condition is relaxed.

6.3. Construction of Volume Quadrature on Cut Elements

To construct volume quadrature rules on cut elements we use the divergence theorem to first compute target integrals via the surface quadrature. We then compute approximate Fekete points to use as quadrature nodes and solve for quadrature weights over those nodes using the target integral values. This approach is a form of moment fitting as described in [42].

6.3.1. Computing the Target Integrals

On each cut element D^k we seek a high-accuracy volume quadrature rule for integrating an arbitrary polynomial function $g(x,y) \in \mathbb{P}^{2N}(D^k)$. To calculate volume quadrature rules we use an orthogonal 2D Legendre polynomial basis. Since these basis functions are defined on $[-1,1]^2$ we first map $(x,y) \in D^k$ to $(\hat{x},\hat{y}) \in \hat{D} = [-1,1]^2$. This mapping consists of shifting the centroid of the cut element, which we compute as the average of the face quadrature nodes, to the origin and then scaling by the maximum distance between the centroid and face nodes:

$$\hat{x} = \frac{x - \bar{x}_0}{\max_i \sqrt{(x_i - \bar{x}_0)^2, (y_i - \bar{y}_0)^2}}, \quad \bar{x}_0 = \frac{1}{n_q} \sum_{i=0}^{n_q} x_i, \tag{51}$$

$$\hat{y} = \frac{y - \bar{y}_0}{\max_i \sqrt{(x_i - \bar{x}_0)^2, (y_i - \bar{y}_0)^2}}, \quad \bar{y}_0 = \frac{1}{n_q} \sum_{i=0}^{n_q} y_i, \tag{52}$$

where (x_i, y_i) are the surface quadrature nodes on the faces of the element. We then define the functional F that maps g to a vector field $F(g)$ with $\nabla \cdot F(g) = g$ using

$$F(g)(\hat{x},\hat{y}) = \begin{bmatrix} \frac{1}{2}\int_{-1}^{\hat{x}} g(s,\hat{y}) \, \mathrm{d}s \\ \frac{1}{2}\int_{-1}^{\hat{y}} g(\hat{x},r) \, \mathrm{d}r \end{bmatrix}. \tag{53}$$

The divergence theorem then states that:

$$\int_{\hat{D}} g \, \mathrm{d}A = \int_{\hat{D}} \nabla \cdot F(g) \, \mathrm{d}A = \int_{\partial\hat{D}} (F(g) \cdot \hat{\boldsymbol{n}}) \, \mathrm{d}s. \tag{54}$$

By using explicit parameterizations of the embedded boundaries we have access to exact normal vectors which allows us to exactly map line quadrature rules to the edges of cut cells. However, given the embedded boundaries are not (in general) polynomially defined, the resulting surface quadrature rules are not polynomially exact, though they may be high accuracy. Via (54) we use a highly accurate surface quadrature rule ($4(N+1)$ Gaussian quadrature nodes, where N is the solution polynomial degree) to calculate target volume integral for each basis function, ψ_i, to yield the vector of target integrals, $\boldsymbol{b}$

$$\boldsymbol{b} = \begin{bmatrix} \int_{\hat{D}} \psi_0 \, \mathrm{d}A \\ \vdots \\ \int_{\hat{D}} \psi_N \, \mathrm{d}A \end{bmatrix} = \begin{bmatrix} \int_{\partial\hat{D}} (F(\psi_0) \cdot \hat{\boldsymbol{n}}) \, \mathrm{d}s \\ \vdots \\ \int_{\partial\hat{D}} (F(\psi_N) \cdot \hat{\boldsymbol{n}}) \, \mathrm{d}s \end{bmatrix}. \tag{55}$$

6.3.2. Computing Volume Quadrature Nodes

To compute quadrature nodes, we start by sampling a large number of equally spaced points in the cut element, $\{(x_k, y_k)\}_{k=1}^{N_s}$. We use these points to build a tall, rectangular Vandermonde matrix $\boldsymbol{V}$

$$\boldsymbol{V} = \begin{bmatrix} \psi_0(x_1, y_1) & \cdots & \psi_N(x_1, y_1) \\ \psi_0(x_2, y_2) & \cdots & \psi_N(x_2, y_2) \\ \vdots & \ddots & \vdots \\ \psi_0(x_{N_s-1}, y_{N_s-1}) & \cdots & \psi_N(x_{N_s-1}, y_{N_s-1}) \\ \psi_0(x_{N_s}, y_{N_s}) & \cdots & \psi_N(x_{N_s}, y_{N_s}) \end{bmatrix}. \tag{56}$$

We apply QR factorization with pivoting to the matrix $\boldsymbol{V}^T$. Since our integrand is total degree $2N$, we take the $(2N+1)(2N+2)/2$ square matrix of pivot columns of $\boldsymbol{V}^T$ to produce a reduced, square Vandermonde matrix $\tilde{\boldsymbol{V}}^T$. As defined in [41], the sampling points corresponding to the pivot columns of the QR factorization are approximate Fekete points. Fekete points are interpolation points that ensure the reduced Vandermonde matrix $\tilde{\boldsymbol{V}}^T$ is well conditioned. With the vector of target integrals and the reduced Vandermonde matrix we can then solve the linear system

$$\tilde{\boldsymbol{V}}^T \boldsymbol{w} = \boldsymbol{b} \tag{57}$$

for $\boldsymbol{w}$, the vector of quadrature weights.

While our process for generating volume quadrature rules on cut elements is very practical and utilizes robust routines like QR factorization it has two major drawbacks: it can return negative quadrature weights and, on extremely small elements, poorly chosen nodes. We discuss both of these issues next.

6.3.3. Conditioning of the Volume Quadrature Weights on Cut Elements

There are methods for generating purely positive quadrature weights, such as in [57, 58], but they are more complicated and computationally expensive to compute. Here, the properties of our formulation do not require positive quadrature weights so we accept the possibility of negative quadrature weights but acknowledge their impact on numerical round-off.

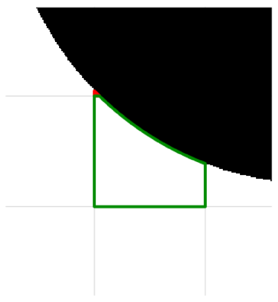
In Table 2 we list the conditioning of these quadrature weights for the circle mesh used in eigenvalue experiment and the fish mesh for polynomial degree $N = 1, 2, 3, 4$. These condition numbers are computed using the formula

$$\kappa_N = \left(\sum_{i=1}^{m_N} |w_i| \right) \left(\sum_{i=1}^{m_N} w_i \right)^{-1} \tag{58}$$

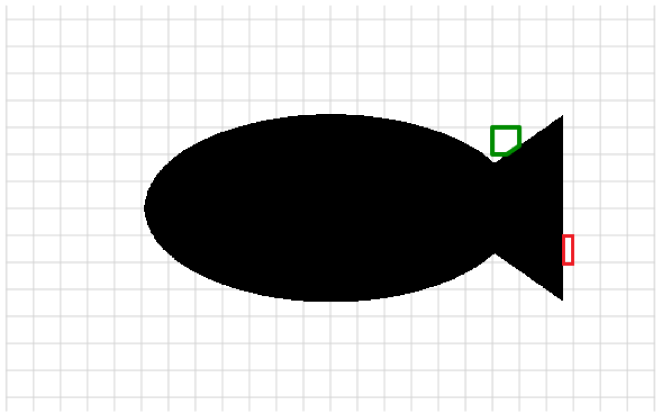
where w_i are the quadrature weights and m_N is the number of quadrature points for the degree N scheme. Note if all quadrature weights are non-negative the conditioning number will be 1. Figure 13 shows the cut elements with the best and worst conditioned quadrature weights for $N = 4$ in each mesh while Figure 14 shows the spread of conditioning numbers for every cut element in the fish mesh.

Table 2. Best and worst cut element quadrature weight conditioning numbers for the circle and fish meshes.

		Circle Mesh		**Fish Mesh**	
N	m_N	Best	Worst	Best	Worst
1	7	1	2.24163	1	2.17636
2	16	1.19370	2.02327	1.03374	1.96366
3	29	1.10240	2.63219	1.00875	2.64212
4	46	1.08764	3.19112	1.02858	6.15015



(a) Circle mesh.



(b) Fish mesh.

Figure 13. Cut elements with the best (green) and worst (red) conditioned quadrature weights from the for $N = 4$ schemes on the circle and fish meshes. In the case of the fish mesh, while the cut elements around each fish is analogous to the cut elements around other fish, analogous cut elements are slightly different. The best and worst elements actually occur on different fishes.

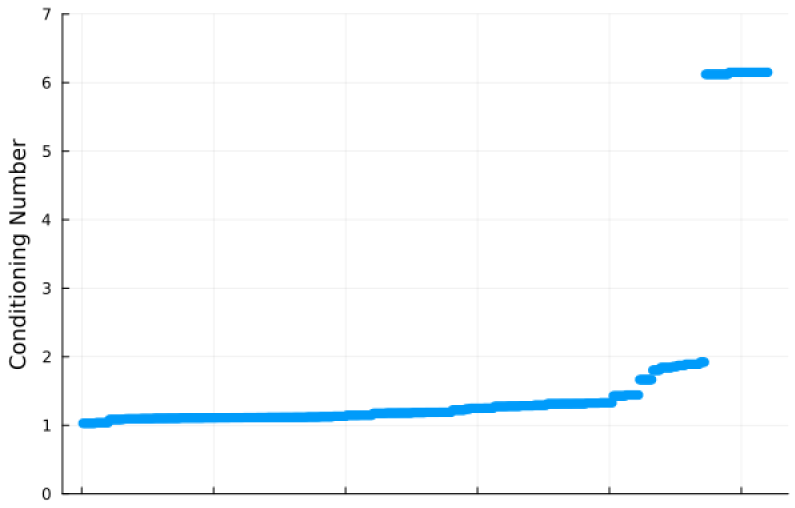


Figure 14. Cut element quadrature weight conditioning numbers for the fish mesh for $N = 4$ in order from best to worst. Of the 520 cut elements in this mesh 48 (9.2%) have a conditioning number greater than 6. All other conditioning numbers are less than 2.

6.3.4. Poorly Chosen Quadrature Nodes

The second, more problematic issue of poorly chosen quadrature nodes arises with extremely small elements. Our routine finds the sampling points closest to true Fekete points. As a result the quality of the final approximate Fekete points depends on the resolution of the sampling points. We generate sampling points via an evenly spaced grid of points on the background Cartesian element from which we remove points that are not in the cut element. If the cut element is extremely small, very few of the generated points will actually be in the cut element and thus represented in $\boldsymbol{V}^T$, from which the Fekete points are calculated. This issue is reflected in the rank of $\boldsymbol{V}^T$: if there where not a sufficient number of well-positioned points, $\boldsymbol{V}^T$ will be low-rank.

To address this issue, we check the conditioning of $\boldsymbol{V}^T$ and resample points on a finer grid until $\boldsymbol{V}^T$ has sufficient rank. This resampling likely influences the clustering of conditioning number values seen in Figure 14. However, even with this safe guard, the conditioning of the quadrature rules on extremely small cut cells may be poor. There are alternative methods for generating volume quadrature such as work by Saye in [57] and [58]. However, for this proof of concept work we used this approach as it is relatively straightforward in comparison to these other methods.

References

[1] M. Berger, A. Giuliani, A state redistribution algorithm for finite volume schemes on cut cell meshes, Journal of Computational Physics 428 (2021) 109820. `doi:10.1016/j.jcp.2020.109820`.

[2] C. M. Dafermos, Hyperbolic Conservation Laws in Continuum Physics, A Series of Comprehensive Studies in Mathematics, Springer-Verlag Berlin Heidelberg, 2009. `doi:10.1007/978-3-642-04048-1`.

[3] T. Warburton, A low-storage curvilinear discontinuous Galerkin method for wave problems, SIAM Journal on Scientific Computing 35 (4) (2013) A1987–A2012. doi:10.1137/120899662.

[4] Z. Wang, K. Fidkowski, R. Abgrall, F. Bassi, D. Caraeni, A. Cary, H. Deconinck, R. Hartmann, K. Hillewaert, H. Huynh, N. Kroll, G. May, P.-O. Persson, B. van Leer, M. Visbal, High-order CFD methods: current status and perspective, International Journal for Numerical Methods in Fluids 72 (8) (2013) 811–845. doi:10.1002/fld.3767.

[5] M. R. Visbal, D. V. Gaitonde, High-order-accurate methods for complex unsteady subsonic flows, AIAA Journal 37 (10) (1999) 1231–1239. doi:10.2514/2.591.

[6] M. Ainsworth, Dispersive and dissipative behaviour of high order discontinuous Galerkin finite element methods, Journal of Computational Physics 198 (1) (2004) 106–130. doi:10.1016/j.jcp.2004.01.004.

[7] M. Berger, Chapter 1 - Cut cells: Meshes and solvers, in: R. Abgrall, C.-W. Shu (Eds.), Handbook of Numerical Methods for Hyperbolic Problems, Vol. 18 of Handbook of Numerical Analysis, Elsevier, 2017, pp. 1–22. doi:10.1016/bs.hna.2016.10.008.

[8] W. H. Reed, T. R. Hill, Triangular mesh methods for the neutron transport equation, Tech. Rep. LA-UR-73-479, Los Alamos Scientific Lab., N. Mex.(USA) (1973).

[9] J. J. Quirk, An alternative to unstructured grids for computing gas dynamic flows around arbitrarily complex two-dimensional bodies, Computers & Fluids 23 (1) (1994) 125–142. doi:10.1016/0045-7930(94)90031-0.

[10] D. Ingram, D. Causon, C. Mingham, Developments in Cartesian cut cell methods, Mathematics and Computers in Simulation 61 (3) (2003) 561–572, mODELLING 2001 - Second IMACS Conference on Mathematical Modelling and Computational Methods in Mechanics, Physics, Biomechanics and Geodynamics. doi:10.1016/S0378-4754(02)00107-6.

[11] M.-H. Chung, Cartesian cut cell approach for simulating incompressible flows with rigid bodies of arbitrary shape, Computers & Fluids 35 (6) (2006) 607–623. doi:10.1016/j.compfluid.2005.04.005.

[12] B. Schott, C. Ager, W. A. Wall, Monolithic cut finite element–based approaches for fluid-structure interaction, International Journal for Numerical Methods in Engineering 119 (8) (2019) 757–796. doi:10.1002/nme.6072.

[13] H. Udaykumar, R. Mittal, W. Shyy, Computation of solid–liquid phase fronts in the sharp interface limit on fixed grids, Journal of Computational Physics 153 (2) (1999) 535–574. doi:10.1006/jcph.1999.6294.

[14] H. Udaykumar, H.-C. Kan, W. Shyy, R. Tran-Son-Tay, Multiphase dynamics in arbitrary geometries on fixed Cartesian grids, Journal of Computational Physics 137 (2) (1997) 366–405. doi:10.1006/jcph.1997.5805.

[15] N. Berre, M. E. Rognes, A. Massing, Cut finite element discretizations of cell-by-cell EMI electrophysiology models (2023). arXiv:2306.03001.

[16] J. Chan, Z. Wang, A. Modave, J.-F. Remacle, T. Warburton, GPU-accelerated discontinuous Galerkin methods on hybrid meshes, Journal of Computational Physics 318 (2016) 142–168. doi:10.1016/j.jcp.2016.04.003.

[17] Z. Xie, An implicit Cartesian cut-cell method for incompressible viscous flows with complex geometries, Computer Methods in Applied Mechanics and Engineering 399 (2022) 115449. doi:10.1016/j.cma.2022.115449.

[18] S. May, M. Berger, A mixed explicit implicit time stepping scheme for Cartesian embedded boundary meshes, in: J. Fuhrmann, M. Ohlberger, C. Rohde (Eds.), Finite Volumes for Complex Applications VII-Methods and Theoretical Aspects, Springer International Publishing, Cham, 2014, pp. 393–400. doi:10.1007/978-3-319-05684-5_38.

[19] S. May, M. Berger, F. Laakmann, Accuracy considerations of mixed explicit implicit schemes for embedded boundary meshes, Proceedings in Applied Mathematics and Mechanics 19 (1) (2019) e201900411. doi:10.1002/pamm.201900411.

[20] S. Schoeder, S. Sticko, G. Kreiss, M. Kronbichler, High-order cut discontinuous Galerkin methods with local time stepping for acoustics, International Journal for Numerical Methods in Engineering 121 (13) (2020) 2979–3003. doi:10.1002/nme.6343.

[21] S. Sticko, G. Ludvigsson, G. Kreiss, High-order cut finite elements for the elastic wave equation, Advances in Computational Mathematics 46 (3) (2020) 45. doi:10.1007/s10444-020-09785-z.

[22] P. Fu, G. Kreiss, High order cut discontinuous Galerkin methods for hyperbolic conservation laws in one space dimension, SIAM Journal on Scientific Computing 43 (4) (2021) A2404–A2424. doi:10.1137/20M1349060.

[23] C. Gürkan, S. Sticko, A. Massing, Stabilized cut discontinuous Galerkin methods for advection-reaction problems, SIAM Journal on Scientific Computing 42 (5) (2020) A2620–A2654. doi:10.1137/18M1206461.
URL https://doi.org/10.1137/18M1206461

[24] S. Sticko, G. Kreiss, A stabilized Nitsche cut element method for the wave equation, Computer Methods in Applied Mechanics and Engineering 309 (2016) 364–387. doi:10.1016/j.cma.2016.06.001.

[25] F. de Prenter, C. Lehrenfeld, A. Massing, A note on the stability parameter in Nitsche's method for unfitted boundary value problems, Computers & Mathematics with Applications 75 (12) (2018) 4322–4336. doi:10.1016/j.camwa.2018.03.032.

[26] C. Engwer, S. May, A. Nüßing, F. Streitbürger, A stabilized DG cut cell method for discretizing the linear transport equation, SIAM Journal on Scientific Computing 42 (6) (2020) A3677–A3703. doi:10.1137/19M1268318.

[27] G. Birke, C. Engwer, S. May, F. Streitbürger, DoD Stabilization of linear hyperbolic PDEs on general cut-cell meshes, Proceedings of Apllied Mathematics and Mechanics 23 (1) (2023) e202200198. doi:10.1002/pamm.202200198.

[28] S. May, F. Streitbürger, DoD Stabilization for non-linear hyperbolic conservation laws on cut cell meshes in one dimension, Applied Mathematics and Computation 419 (2022) 126854. doi:10.1016/j.amc.2021.126854.

[29] F. Streitbürger, C. Engwer, S. May, A. Nüßing, Monotonicity considerations for stabilized DG cut cell schemes for the unsteady advection equation, in: F. J. Vermolen, C. Vuik (Eds.), Numerical Mathematics and Advanced Applications ENUMATH 2019, Springer International Publishing, Cham, 2021, pp. 929–937. doi:10.1007/978-3-030-55874-1_92.

[30] M. Berger, R. LeVeque, Cartesian meshes and adaptive mesh refinement for hyperbolic partial differential equations, in: Proceedings of the Third International Conference on Hyperbolic Problems. Uppsala, Sweden. June 1990, 1990.

[31] C. Helzel, M. J. Berger, R. J. Leveque, A high-resolution rotated grid method for conservation laws with embedded geometries, SIAM Journal on Scientific Computing 26 (3) (2005) 785–809. doi:10.1137/S106482750343028X.

[32] M. J. Berger, C. Helzel, R. J. LeVeque, H-box methods for the approximation of hyperbolic conservation laws on irregular grids, SIAM Journal on Numerical Analysis 41 (3) (2003) 893–918. doi:10.1137/S0036142902405394.

[33] M. Berger, C. Helzel, A simplified h-box method for embedded boundary grids, SIAM Journal on Scientific Computing 34 (2) (2012) A861–A888. doi:10.1137/110829398.

[34] I.-L. Chern, P. Colella, A conservative front tracking method for hyperbolic conservation laws, LLNL Rep. No. UCRL-97200, Lawrence Livermore National Laboratory 51 (1987) 83–110.

[35] P. Colella, D. T. Graves, B. J. Keen, D. Modiano, A Cartesian grid embedded boundary method for hyperbolic conservation laws, Journal of Computational Physics 211 (1) (2006) 347–366. doi:10.1016/j.jcp.2005.05.026.

[36] M. Berger, A note on the stability of cut cells and cell merging, Applied Numerical Mathematics 96 (2015) 180–186. doi:10.1016/j.apnum.2015.05.003.

[37] B. Muralidharan, S. Menon, A high-order adaptive Cartesian cut-cell method for simulation of compressible viscous flow over immersed bodies, Journal of Computational Physics 321 (2016) 342–368. doi:10.1016/j.jcp.2016.05.050.

[38] M. Kirkpatrick, S. Armfield, J. Kent, A representation of curved boundaries for the solution of the Navier–Stokes equations on a staggered three-dimensional Cartesian grid, Journal of Computational Physics 184 (1) (2003) 1–36. doi:10.1016/S0021-9991(02)00013-X.

[39] D. Cecere, E. Giacomazzi, An immersed volume method for large eddy simulation of compressible flows using a staggered-grid approach, Computer Methods in Applied Mechanics and Engineering 280 (2014) 1–27. doi:10.1016/j.cma.2014.07.018.

[40] H. Ziegelwanger, P. Reiter, The PAC-MAN model: Benchmark case for linear acoustics in computational physics, Journal of Computational Physics 346 (2017) 152–171. doi:10.1016/j.jcp.2017.06.018.

[41] A. Sommariva, M. Vianello, Computing approximate Fekete points by QR factorizations of Vandermonde matrices, Computers & Mathematics with Applications 57 (8) (2009) 1324–1336. `doi:10.1016/j.camwa.2008.11.011`.

[42] P. J. Davis, A construction of nonnegative approximate quadratures, Mathematics of Computation 21 (100) (1967) 578–582. `doi:10.2307/2005001`.

[43] J. S. Hesthaven, T. Warburton, Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications.

[44] J. Chan, L. C. Wilcox, On discretely entropy stable weight-adjusted discontinuous Galerkin methods: curvilinear meshes, Journal of Computational Physics 378 (2019) 366–393. `doi:10.1016/j.jcp.2018.11.010`.

[45] J. Chan, R. J. Hewett, T. Warburton, Weight-adjusted discontinuous Galerkin methods: Wave propagation in heterogeneous media, SIAM Journal on Scientific Computing 39 (6) (2017) A2935–A2961. `doi:10.1137/16M1089186`.

[46] A. Giuliani, A. Almgren, J. Bell, M. Berger, M. Henry de Frahan, D. Rangarajan, A weighted state redistribution algorithm for embedded boundary grids, Journal of Computational Physics 464 (2022) 111305. `doi:10.1016/j.jcp.2022.111305`.

[47] J. Nordström, A. R. Winters, Stable filtering procedures for nodal discontinuous Galerkin methods, Journal of Scientific Computing 87 (1) (2021) 17. `doi:10.1007/s10915-021-01434-x`.

[48] T. Lundquist, J. Nordström, Stable and accurate filtering procedures, Journal of Scientific Computing 82 (1) (2020) 16. `doi:10.1007/s10915-019-01116-9`.

[49] J. L. W. V. Jensen, Sur les fonctions convexes et les inégalités entre les valeurs moyennes, Acta Mathematica 30 (none) (1906) 175 – 193. `doi:10.1007/BF02418571`.

[50] M. Tao, C. Batty, E. Fiume, D. I. W. Levin, Mandoline: robust cut-cell generation for arbitrary triangle meshes, ACM Transactions on Graphics 38 (6) (nov 2019). `doi:10.1145/3355089.3356543`.

[51] C. Tsitouras, Runge–Kutta pairs of order 5(4) satisfying only the first column simplifying assumption, Computers & Mathematics with Applications 62 (2) (2011) 770–775. `doi:10.1016/j.camwa.2011.06.002`.

[52] C. Rackauckas, Q. Nie, Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in Julia, Journal of Open Research Software 5 (1) (2017) 15–15. `doi:10.5334/jors.151`.

[53] PathIntersections.jl.
URL `https://github.com/cgt3/PathIntersections.jl`

[54] StartUpDG.jl.
URL `https://github.com/jlchan/StartUpDG.jl`

[55] Reproducability repository for simulation codes.
URL `https://github.com/cgt3/paper-2024-srd-wave`

[56] J. Revels, M. Lubin, T. Papamarkou, Forward-mode automatic differentiation in Julia (2016). `arXiv:1607.07892`.

[57] R. I. Saye, High-order quadrature methods for implicitly defined surfaces and volumes in hyperrectangles, SIAM Journal on Scientific Computing 37 (2) (2015) A993–A1019. `doi:10.1137/140966290`.

[58] R. I. Saye, High-order quadrature on multi-component domains implicitly defined by multivariate polynomials, Journal of Computational Physics 448 (2022) 110720. `doi:10.1016/j.jcp.2021.110720`.