Solving Partial Differential Equations via Learned Multigrid Smoothing

Bethany Witemeyer
bethanywitemeyer@tamu.edu
Texas A&M University
USA

David I.W. Levin
diwlevin@cs.toronto.edu
University of Toronto
Canada

Shinjiro Sueda
sueda@tamu.edu
Texas A&M University
USA

ABSTRACT

We present a method for increasing the accuracy of multigrid solvers by optimizing the entries in the smoother matrices. We focus on real-time applications where the wall clock time restrictions require the multigrid solver to run for a small, predetermined number of iterations. We additionally focus on physical systems that have a constant, linear system matrix. In doing so, we are able to modify the entries in the smoother matrices directly, which makes our method very easy to incorporate into existing multigrid solvers. We demonstrate our method using two examples: the heat equation and statics with linear elasticity. For the heat equation, we observe up to an 11.7x decrease in the average error for a single solve. For statics with linear elasticity, we observe up to a 41.7x decrease in the average error for a single solve.

CCS CONCEPTS

• **Computing methodologies** → **Physical simulation**; • **Mathematics of computing** → *Solvers*; *Partial differential equations.*

KEYWORDS

Multigrid, multigrid smoothers, PDE, optimization, self-supervised learning

ACM Reference Format:
Bethany Witemeyer, David I.W. Levin, and Shinjiro Sueda. 2026. Solving Partial Differential Equations via Learned Multigrid Smoothing. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Physically-based simulation is a powerful tool used in many areas of computer graphics, animation, and scientific computing [4, 63]. Partial differential equations (PDEs) are the driving force of physics simulation; however, closed-form solutions are rarely available for realistic geometries. Thus, physics simulation relies on numerical methods and efficient numerical solvers. Classical discretization methods—including finite difference, finite element, and finite volume approaches—transform PDEs into large sparse linear systems. As physics models get more accurate, these systems get larger and more computationally expensive to solve, quickly making solving the linear system the bottleneck in the simulation.

Many methods have been presented over the years to address the high computational cost of solving the large, sparse linear systems that arise in physically-based simulation [5, 12, 18]. In this paper, we look at multigrid methods, which combine a stationary iterative method, such as Jacobi or Gauss-Seidel, with a multilayered hierarchy of mesh resolutions [8]. Multigrid methods have already been shown to significantly improve the runtime of physics simulations [19, 38, 51, 54, 64, 66]. However there are two main drawbacks to multigrid methods that we seek to address in this paper.

First, multigrid solvers are iterative solvers and, unless they are run to convergence, the solution they compute is approximate. For real-time applications, the wall clock time available for solving the large, sparse linear system is typically too short to run an iterative solver to convergence. So, solvers will often be run for a predetermined number of iterations, resulting in an approximate solution. This problem is compounded for dynamic simulations as any error present at a given time step will get propagated to future time steps as the simulation progresses.

Second, the convergence rate of multigrid solvers is highly problem dependent. Multigrid solvers have many parameters, and when those parameters are tuned properly, multigrid methods can achieve near-optimal convergence rates. However, the tuning of those parameters is typically performed manually, and a set of parameters that works great for one model is not guaranteed to provide the same performance on another.

With the rise of machine learning techniques being applied to physically-based animation [11, 16, 17, 47, 50, 65], we ask the question: Can we optimize multigrid parameters automatically instead of manually? While we are not the first to ask this question, the majority of the existing work in this area requires replacing parts of the multigrid solver, or the solver in its entirety, with a neural network. We aim to generate optimal multigrid parameters in such a way that only minimal changes to an existing solver are required. Additionally, recent work has mainly explored learning inter-grid transfer operators such as prolongation and restriction, but the learning of smoothing operators themselves has received comparatively little attention.

In this paper, we propose a method for learning multigrid smoothers. Specifically, we learn multigrid smoothers in the context of real-time applications where the iteration counts for the multgrid solver are predetermined. Because the iteration counts are predetermined, we focus on improving the accuracy of the single multigrid solver rather than improving the convergence rate of the solver overall.

Conference acronym 'XX, June 03–05, 2026, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

We choose the Gauss-Seidel smoother as our base smoother and construct a differentiable multigrid solver.

We also focus solely on physics systems that are linear and have a system matrix that is constant over the course of the simulation. These systems can be time-varying or static, so long as the entries in the system matrix are not dependent on the solution variable(s). By focusing on these constant, linear systems, we are able to optimize the multigrid smoothers directly. This, in turn, enables our optimized smoothers to be easily utilized by existing multigrid solvers. We are simply optimizing the values of a sparse matrix, so using the optimized smoother is a simple drop-in replacement and the rest of the solver works exactly as before.

In the rest of this paper we will outline our lightweight training procedure for learning error-reducing smoothing weight and present a comprehensive evaluation of our method on the dynamic heat equation and static linear elasticity equations using a collection of triangle and tetrahedral meshes. We show that for the heat equation, this learned smoother achieves up to an 11.7x decrease in the average error for a single solve and additionally improves error accumulation over the course of a simulation. For statics with linear elasticity, our learned smoother achieves up to a 41.7x reduction in average solution error per solve. Both of these results are achieved under the same runtime budget as standard multigrid.

2 RELATED WORK

Physically-based animation has evolved significantly since its early foundations [57]. A seminal contribution by Baraff and Witkin [1] introduced implicit time stepping to the physically based animation community, enabling more efficient and stable results. Since then, various methods have been developed to speed up simulations, such as subspace dynamics [2, 14, 46], projective dynamics [6, 34, 35], and condensation [9, 56, 62]. Although Position-Based Dynamics [41] is a notable exception, most of these simulation methods rely on a large linear solve, the focus of our work.

The fast computation of solutions to these linear systems is an area of research that has also seen many methods presented over the years. Direct solvers, exemplified by libraries such as CHOLMOD [12] and SuperLU [31], have established high benchmarks for performance. However, while these factorization-based methods are robust, they can suffer from "fill-in"—the introduction of new nonzero elements—which can cause memory usage to scale prohibitively with system size. Consequently, iterative methods often become the preferred approach for such systems. Techniques such as Krylov subspace methods (e.g., Conjugate Gradients or Conjugate Directions) [49], stationary-point iterations, and multigrid schemes [8] offer a scalable alternative by avoiding matrix factorization. These methods trade the exactness of direct solvers for efficiency, approximating solutions through successive steps until a user-defined convergence threshold is met. Our approach falls under this category.

Physics-informed neural networks [48] are an alternative neural approach to solving PDEs [10] that avoids solvers entirely, replacing discretization and a linear solve with a machine learning training loop. However these methods often struggle maintaining complex boundary conditions, requiring additional data like distance fields [53] and can be significantly slower than standard approaches [11].

These limitations have motivated the integration of deep learning with iterative solvers, most notably through neural preconditioners designed to accelerate the convergence rate of conjugate gradient [32] and conjugate direction methods [26]. However, for elliptic Partial Differential Equations (PDEs), such as the Poisson equation common in physics simulations, multigrid methods [7, 8, 58] remain the theoretical gold standard, capable of solving these systems with optimal $O(N)$ complexity. This efficiency stems from multigrid's hierarchical structure, which addresses a fundamental limitation of CG: while CG is effective, its convergence often stalls due to persistent low-frequency (long-wavelength) errors. While multigrid is often used as a preconditioner for Krylov methods to resolve these errors, it is also a highly efficient standalone solver. In this work, we focus on optimizing the multigrid solver itself.

The efficiency and accuracy of a multigrid solver are highly problem-dependent, relying heavily on the interaction between its two primary components: the smoothers, which reduce high-frequency error, and the prolongation/restriction matrices, which transfer data between grid levels. The literature has a long history of analytical approaches that search for optimal parameters for these components. These range from early energy-minimization strategies [36, 43] to more recent sparse approximate inverse techniques [23, 55] and optimal polynomial smoothers [15]. However, these methods typically determine parameters by minimizing a fixed algebraic measure, such as the Frobenius norm of the approximate inverse error. This distinguishes them from learning-based methods, which can exploit the statistical patterns in a dataset to tailor the smoother to the specific problem distribution.

Conversely, machine learning techniques provide a flexible framework for deriving these components, bypassing the need for manual, theoretical derivation. While evolutionary strategies have been explored for automated design [44], recent focus has shifted toward differentiable learning-based optimizations. The existing literature has predominantly concentrated on optimizing the grid transfer operators (prolongation and restriction). Early works such as [27] and [20] focused on learning these matrices to improve coarse-grid correction, a trend that continues in recent graph-based approaches [24, 37, 59]. We differentiate our approach by focusing instead on the optimization of the multigrid smoothing operators.

A handful of works have proposed methods for learning multigrid smoothers. The majority of these works utilize convolutional neural networks (CNNs) to optimize the smoothers, noting that multigrid v-cycles and CNNs share a similar mathematical structure [21, 22]. While these methods effectively improve performance [13, 25, 39, 40], their reliance on CNNs restricts them to structured grids. In contrast, our method applies to unstructured meshes.

Most relevant to our approach are works that attempt to learn smoother parameters on unstructured data. However, these methods typically regress global scalars. For example, Wang et al. [60] and Kuznichov [30] learn weighting coefficients for Richardson or Jacobi iterations. Similarly, recent approaches like Liu et al. [33] utilize graph neural networks to correct solver errors, but do not explicitly optimize the sparse smoothing matrix. Rather than constraining the problem to a blanket weighting term, we propose to learn the smoother matrix values directly. This granular approach

enables each value in the matrix to change independently, allowing for a highly localized optimization that adapts to the specific geometry of the mesh. Furthermore, because the resulting smoother is simply a sparse matrix, our method acts as a drop-in replacement for standard algebraic smoothers.

3 METHODS

3.1 Background

Regardless of the model, spatial discretization scheme, or time integration method, most physically-based simulation frameworks require solving a system of linear equations:

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{1}$$

where $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{b} \in \mathbb{R}^n$ are known, and $\mathbf{x} \in \mathbb{R}^n$ is the unknown solution.

Multigrid solvers are built on a foundation of stationary iterative solvers [8]. There are two key observations that give rise to the multigrid structure. The first is that stationary iterative solvers, including the Gauss-Seidel solver, reduce high-frequency error quickly and low-frequency error slowly. Thus, in the context of multigrid solvers, the stationary iterative solver is typically called a *smoother*, and the solver's iteration matrix is called the *smoother matrix*. In this paper, we use a Gauss-Seidel solver, the details of which are presented in Alg. 1. In a standard Gauss-Seidel solver, the system matrix $\mathbf{A}$ is decomposed into three parts: a diagonal component $\mathbf{D}$, a lower triangular component $\mathbf{L}$, and an upper triangular component $\mathbf{U}$. These components combine such that $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$. The iteration matrix is defined to be $\mathbf{L} + \mathbf{D}$ (also denoted by $\text{TRIL}(\mathrm{A})$). For the sake of simplicity, we combine the lower triangular and diagonal components into a single matrix and call it $\mathbf{L}$.

The second key observation is that low-frequency error on a high-resolution mesh looks like high-frequency error on a low-resolution mesh. Given a mapping from the high-resolution mesh to the low-resolution mesh, we can restrict the error onto the low-resolution mesh, creating a smaller system that can be solved using any available technique for solving linear systems of equations. Common choices include solving the system directly, using an iterative solver, and recursively applying the multigrid solver. Once we have our low-resolution solution, we prolongate, or interpolate, it back to the high-resolution mesh and update the high-resolution solution. Optionally, we can then apply a few more iterations of the smoother to eliminate any high-frequency error introduced by the prolongation mapping.

Algorithm 1 Solves $\mathbf{A}\mathbf{x} = \mathbf{b}$ using the Gauss–Seidel method where $\mathbf{x}_0$ is the initial guess and n_{si} is the number of iterations.

1: **function** GaussSeidel($\mathbf{A}, \mathbf{b}, \mathbf{x}_0, n_{si}, \mathbf{L}$)
2: $\mathbf{x} \leftarrow \mathbf{x}_0$
3: $i \leftarrow 0$
4: **while** $i < n_{si}$ **do**
5: $\quad \mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}$
6: $\quad \mathbf{x} \leftarrow \mathbf{L}^{-1}\mathbf{r} + \mathbf{x}$
7: $\quad i \leftarrow i + 1$
8: **end while**
9: **return x**

Algorithm 2 Solves $\mathbf{A}\mathbf{x} = \mathbf{b}$ with initial guess $\mathbf{x}_0$ using a v-cycle with depth d_{max}. $R[\cdot]$ is a set of restriction matrices, with $R[d]$ being the restriction matrix mapping Level d to $d+1$. n_{si} is the number of smoother iterations.

1: **function** Multigrid($\mathbf{A}, \mathbf{b}, \mathbf{x}_0, R[\cdot], n_{si}, d_{max}, d, L[\cdot]$)
2: $\mathbf{L} \leftarrow L[d]$
3: $\mathbf{x} \leftarrow \text{GaussSeidel}(\mathbf{A}, \mathbf{b}, \mathbf{x}_0, n_{si}, \mathbf{L})$
4: $\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}$
5: $\mathbf{R} \leftarrow R[d]$
6: $\tilde{\mathbf{r}} \leftarrow \mathbf{R}\mathbf{r}$
7: $\tilde{\mathbf{A}} \leftarrow \mathbf{R}\mathbf{A}\mathbf{R}^T$
8: **if** $d < d_{max}$ **then**
9: $\quad \tilde{\mathbf{e}} \leftarrow \text{Multigrid}(\tilde{\mathbf{A}}, \tilde{\mathbf{r}}, \mathbf{0}, R[\cdot], n_{si}, d_{max}, d+1, L[\cdot])$
10: **else**
11: $\quad \tilde{\mathbf{e}} \leftarrow \tilde{\mathbf{A}}^{-1}\tilde{\mathbf{r}}$
12: **end if**
13: $\mathbf{e} \leftarrow \mathbf{R}^T\tilde{\mathbf{e}}$
14: $\mathbf{x}_0 \leftarrow \mathbf{x} + \mathbf{e}$
15: $\mathbf{x} \leftarrow \text{GaussSeidel}(\mathbf{A}, \mathbf{b}, \mathbf{x}_0, n_{si}, \mathbf{L})$
16: **return x**

This process is called a *v-cycle* and is presented in Alg. 2. To solve our low-resolution system, we recursively call our multigrid solver (line 9) until the system becomes small enough to solve quickly with a direct solver (line 11). For this paper, we define "small enough" as the system having fewer than one hundred (100) degrees of freedom.

We have intentionally not discussed our choice of prolongation and restriction matrices, that is, the mappings between multigrid levels. In order to retain the symmetric positive definiteness of our system matrices, we require the prolongation matrix to be the transpose of the restriction matrix, $\mathbf{P} = \mathbf{R}^T$, as shown in lines 7 and 13 of Alg. 2. However, because our method focuses on optimizing the multigrid smoothers, the prolongation and restriction matrices can be chosen to fit the simulation. Thus, we instead discuss our choice of these matrices in Sec. 4 with the rest of the simulation details.

3.2 Learning the Smoother Matrices

We seek to increase the accuracy of the multigrid solver by optimizing the Gauss-Seidel smoother matrices at each level. To that end, we expose the smoother matrices and place them in an array/list $L[\cdot]$ where $L[d]$ is the smoother matrix for level d of the multigrid solver (line 1 of both Algs. 1 and 2). This allows us to modify the values in the smoother matrices and then evaluate the effectiveness of those modifications in reducing the error of the solver.

We also fix the sparsity pattern of the smoother matrices to be that of the standard Gauss-Seidel smoother matrix, $\mathbf{L}$. Any non-zero value in $\mathbf{L}$ can be modified by the optimization process, and any zero value in $\mathbf{L}$ will remain zero. Multigrid solvers are most effective on systems that are large and sparse, and Gauss-Seidel solvers are often accelerated [29, 61]. By ensuring our sparsity pattern matches that of the standard Gauss-Seidel smoother, exchanging one matrix for the other remains a seamless process, regardless of any surrounding code. This makes our approach completely agnostic to the choice of the solver, allowing us to simply drop in the optimized smoother

matrix into the chosen solver. We also avoid adding computation time to the multigrid solver as a result of additional nonzeros in the sparse matrices, including time needed to rerun any solver acceleration algorithms.

During the optimization process, we fix all multigrid and system parameters except the smoother matrices. Prior to running the optimization, the system matrix $\mathbf{A}$, multigrid depth d_{max}, and restriction and prolongation matrices $\mathbf{R}$ and $\mathbf{P}$ are computed and exported for later use in the optimization process. Furthermore, to decrease the runtime of the differentiable multigrid solver, we precompute the $\mathbf{A}$ matrices at every level. This eliminates the need to compute the matrix product $\mathbf{RAR}^T$ every iteration.

Our goal is to increase the accuracy of the solution of the multigrid solver, so we define our optimization objective based on the solution error. More specifically, we define the loss as a function of the smoother matrices:

$$\mathcal{L}(L[\cdot]) = \text{MSE}(\mathbf{A}^{-1}\mathbf{B}, \textsc{Multigrid}(\mathbf{A}, \mathbf{B}, \cdots, 1, L[\cdot])), \quad (2)$$

where $\mathbf{B} \in \mathbb{R}^{n\times m}$ and m is the number of right-hand side vectors being solved simultaneously. Each column in $\mathbf{B}$ represents a training data sample, which we describe in the next section. The arguments $\mathbf{x}_0$, $R[\cdot]$, n_{si}, and d_{max} for $\textsc{Multigrid}(\cdots)$ are problem dependent. The recursion variable d is initialized to 1. To run the optimization, we implement a differentiable multigrid solver and look for a set of smoothers, $L_{opt}[\cdot]$, such that

$$L_{opt}[\cdot] = \arg\min_{L[\cdot]} \mathcal{L}(L[\cdot]). \quad (3)$$

We warm start the optimization by initializing our smoother matrices with the lower triangular matrix used by the standard, unoptimized Gauss-Seidel smoother. That is, we set $\mathbf{L} = \textsc{tril}(\mathbf{A})$ for each level of our multigrid solver prior to running the optimization. We also apply a ReLU activation layer to the diagonal elements of $\mathbf{L}$, adding an ϵ to ensure there are no zero values on the diagonal. Because the systems being solved at the coarse level(s) of the multigrid solver are residual systems, the distribution of right-hand sides for the coarse levels will be somewhat dependent on the results from the fine level(s). To ensure our optimized smoothers account for this coupling behavior, we optimize the smoothers at all levels of the multigrid solver simultaneously.

3.3 Training Data

Because we are structuring our optimization task as a self-supervised learning task, we need to generate data from which to learn. However, we should not use any right-hand side we want. Because PDEs typically model physical phenomena, the right-hand side vectors have a specific formulation, one that is based on the behavior being approximated. Thus, we will get the best results by training on right-hand side vectors that maintain that structure.

3.3.1 Heat Equation. Our first example application for this paper is the heat equation. For this application, the linear system takes the form

$$\left(\mathbf{M} + \alpha h \mathbf{G}^T \mathbf{W} \mathbf{G}\right) \mathbf{u}^{(k+1)} = \mathbf{M}\left(\mathbf{u}^{(k)} + h\mathbf{f}^{(k+1)}\right), \quad (4)$$

where $\mathbf{M} \in \mathbb{R}^{n\times n}$ and $\mathbf{W} \in \mathbb{R}^{n'\times n'}$ are the mass matrix and the area matrix of the mesh, respectively (and n and n' are the number of vertices and faces in the mesh, respectively), $\mathbf{G} \in \mathbb{R}^{n'\times n}$ is the gradient matrix of the mesh, $\mathbf{u} \in \mathbb{R}^n$ is the temperature vector, $\mathbf{f} \in \mathbb{R}^n$ is the amount of heat per unit time added by external heat sources, h is the time step used in the simulation, α is the thermal diffusivity of the material being modeled, and k denotes the current step in the simulation.

$\mathbf{M}$ is determined by the mesh being used in the simulation and will not change during the course of any simulation run on that mesh. Thus, to generate right-hand sides with the same formulation as the right-hand sides generated by the simulation, we generate vectors of the form $\mathbf{u}^{(k)} + h\mathbf{f}^{(k+1)}$ and multiply by $\mathbf{M}$.

To avoid overfitting to a particular type of heat simulation, we generate $\mathbf{u}^{(k)}$ randomly. Similarly, to avoid overfitting to a particular type of source function, we generate $\mathbf{f}^{(k+1)}$ randomly. We note that since both $\mathbf{u}^{(k)}$ and $h\mathbf{f}^{(k+1)}$ are vectors of random heat values, their sum will also be a vector of random heat values. Thus, it suffices to simply generate one set of random heat vectors, $\mathbf{u}^{(k)}$, and compute the right-hand side as $\mathbf{Mu}^{(k)}$. Any potential source functions will be accounted for in the randomness of the generated heat vector, and we maintain the structure defined by the physical system by multiplying the random temperature vector by $\mathbf{M}$.

3.3.2 Linear Elasticity. For our second example application, we test our method on statics systems with linear elasticity. These systems take the form

$$\mathbf{Kx} = \mathbf{f}, \quad (5)$$

where $\mathbf{K} \in \mathbb{R}^{3n\times 3n}$ is the stiffness matrix, $\mathbf{x} \in \mathbb{R}^{3n}$ is the displacement vector, and $\mathbf{f} \in \mathbb{R}^{3n}$ is the external force vector. For this system, the right-hand side consists of only one variable, $\mathbf{f}$. So, we simply generate random force vectors as our right-hand sides for optimizing on a statics system.

3.3.3 On-the-Fly Data Generation. While our system matrices and smoother matrices are sparse, our right-hand side vectors are not. As the meshes we use get larger, the amount of memory required to store an adequate number of right-hand side vectors for optimization quickly becomes intractable. So, we generate our data on the fly.

During each optimization iteration, we generate a set of right-hand side vectors according to the methods outlined above. Then, we compute the ground truth solution using a direct solver. We also pass the set of right-hand side vectors through our differentiable multigrid solver and compute the loss according to Eq. (2). After the iteration is complete, the data is discarded and a new set of random vectors is generated during each subsequent iteration.

3.3.4 Data Normalization. Depending on the units used and the physical constants, the values in $\mathbf{A}$ and $\mathbf{b}$ can be too small or too big for the optimization process to be effective. Therefore, we scale both quantities by the inverse of the largest eigenvalue of $\mathbf{A}$, which can be computed efficiently for sparse matrices [52]. By scaling $\mathbf{A}$ so that its largest eigenvalue is 1, we ensure that the spectral radius is controlled, preventing the error from exploding during iterations.

4 RESULTS

To test our method, we implement a differentiable multigrid solver in PyTorch [45]. The built-in autograd functions compute most of

Table 1: Mesh/multigrid specifics and wall clock training time for the heat equation.

Mesh	Vertices	Depth	Train Time (hr)
Sphere	2,562	4	0.854
Fish	10,786	5	0.441
Hammer	30,082	6	0.737
Scorpion	49,997	6	0.656
Cow	399,999	7	2.928
Nefertiti	1,009,118	8	7.323

the gradients along with [3] for a sparse, differentiable triangular solver. We additionally use cholespy [42] for the ground truth direct solver. We then optimize the smoother matrices using the methods outlined above. All tasks used the Adam optimizer [28] for 100,000 iterations. The heat equation optimization used a learning rate of 1e-4, and the optimization for linear elasticity used a learning rate of 1e-3 (both determined empirically). Additionally, all optimization tasks were run on an NVIDIA GeForce RTX 4090 GPU. All analysis of the optimized smoother matrices was done in MATLAB.

We choose our course level vertices according to the methods outlined in Xian et al. [64]. For our prolongation and restriction method, we use a piecewise constant formulation. We set both the number of v-cycles and smoother iterations to one (1) and train the model on a variety of meshes with sizes ranging from 2500 to one million vertices.

4.1 Heat Equation

We first evaluate our method using the heat equation. The sizes of each mesh used, the number of multigrid levels optimized for each mesh, and the corresponding wall clock training time are presented in Table 1. We note that even for our largest mesh with one million vertices, our method only takes a few hours to run.

4.1.1 Single Solve Improvement. To evaluate our optimized multigrid smoothers, we start by checking their performance on a single linear solve. We compute the error present in both the unoptimized multigrid solution and our optimized multigrid solution relative to the ground truth solution using the relative norm:

$$e = \frac{||\mathbf{u}_{mg} - \mathbf{u}_{gt}||}{||\mathbf{u}_{gt}||}, \tag{6}$$

where $\mathbf{u}_{mg}$ is the solution from one of the multigrid solvers, either unoptimized or optimized, and $\mathbf{u}_{gt}$ is the ground truth solution. We compute this metric on two data sets: a set of n random right-hand side vectors generated in the exact same way as our training data, and a set of n one-hot right-hand side vectors (both capped at 100,000 right-hand sides to prevent the analysis wall clock time from becoming intractable). The one-hot vectors are also generated using the same approach as the training data generation, only we replace the random vector with a one-hot vector (every value in the vector is the same except one, which is significantly higher than the other values). These tests check the ability of the solver to compute impulse-response like solutions. It is worth noting that both multigrid solvers have the exact same parameters, with only the smoother matrices being different.

Table 2: Average relative error compared to direct solve for a single multigrid solve using the unoptimized smoother matrix and our optimized smoother matrix for the heat equation.

Mesh	Random		One-Hot		Total
	Unopt.	Opt.	Unopt.	Opt.	Factor
Sphere	5.66e−2	1.16e−2	5.30e−2	2.81e−3	7.6
Fish	6.36e−2	8.92e−3	5.73e−2	1.40e−3	11.7
Hammer	5.29e−2	9.36e−3	4.09e−2	1.77e−3	8.4
Scorpion	5.83e−2	1.20e−2	4.53e−2	3.90e−3	6.5
Cow	4.13e−2	6.96e−3	3.15e−2	2.85e−3	7.4
Nefertiti	7.32e−3	2.58e−3	7.76e−3	6.22e−4	4.7

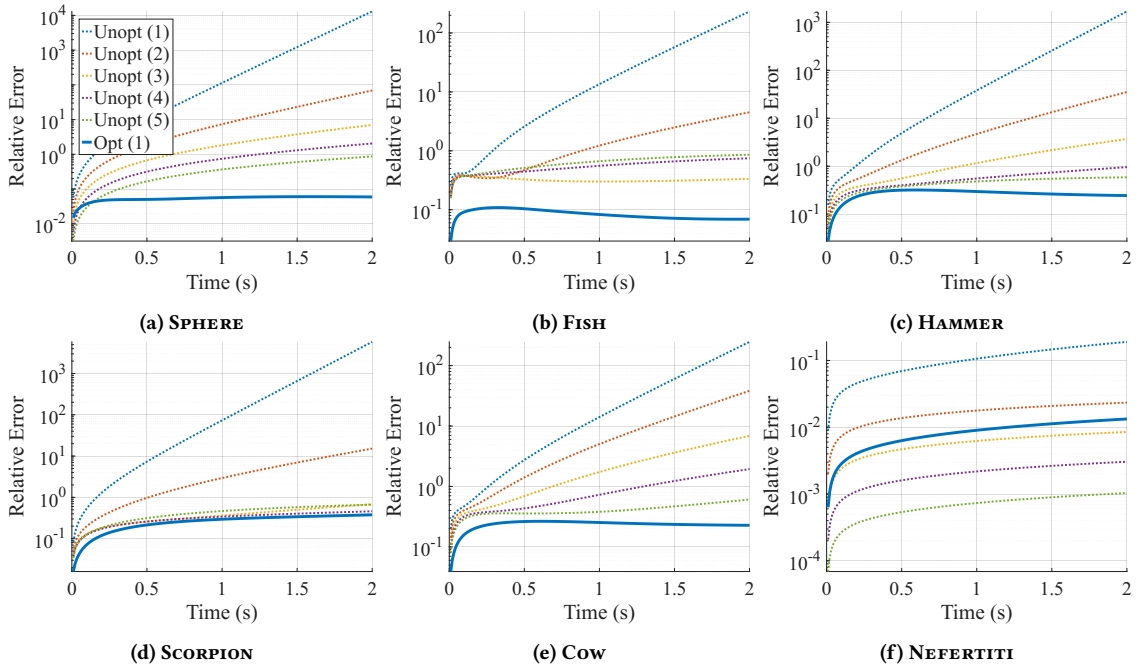
We average the relative errors for each set on each mesh and compute the overall accuracy improvement factors. The results are presented in Table 2. We observe that for every mesh we tested, using our optimized multigrid smoothers produced results that were 4.7 to 11.7 times more accurate than using the unoptimized multigrid smoothers. The performance of multigrid solvers is known to be highly problem dependent, so it follows that the effectiveness of optimizing multigrid smoothers would also be highly problem dependent.

It is also interesting to note that our method sees higher levels of accuracy improvement on the one-hot data set than on the random data set. The random data set contains both high- and low-frequency error, but the one-hot set contains mostly low-frequency error. This implies that our method is learning how to deal with areas of low-frequency error, something that stationary iterative solvers usually struggle with.

4.1.2 Simulation Results. We further test our method by using our optimized multigrid solver in a heat simulation framework. We first generate a ground truth simulation by running the simulation with a direct solver. We then run the same simulation with both the unoptimized multigrid solver and our optimized multigrid solver. We compute the error at each time step of the multigrid simulations relative to the same time step of the ground truth simulation (Eq. (6)). The plots of this error over time for the homogeneous heat equation ($\mathbf{f} = \mathbf{0}$) are presented in Fig. 1.

We observe that for each of the meshes, our optimized multigrid solver (denoted in each plot by the bolded blue line) accumulates significantly less error over the course of the simulation than the unoptimized multigrid solver with the same multigrid parameters (denoted in each plot by the dashed blue line), sometimes by several orders of magnitude. Not only that, the slope of the error accumulation for our method is lower as well, especially in the second half of the simulation where the error is largely low frequency (see Cow, for example). This means that in the same amount of computation time our optimized multigrid solver will not only produce a more accurate simulation, but it will also better handle simulations where there are periods of time with little change.

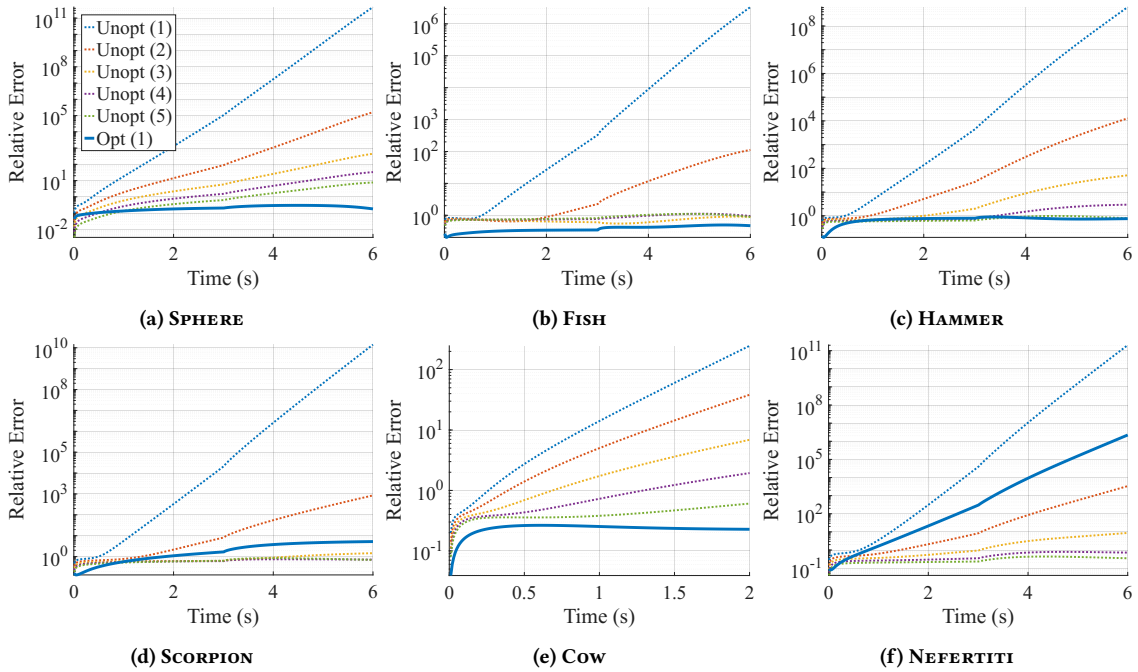
We also compare our optimized multigrid solver to the unoptimized multigrid solver run with additional smoother iterations (denoted in each plot by the not blue dashed lines). In all cases,



(a) Sphere (b) Fish (c) Hammer

(d) Scorpion (e) Cow (f) Nefertiti

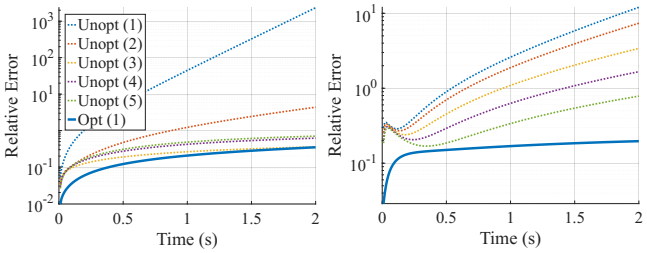
Figure 1: Homogeneous simulation results for each of our test meshes comparing our optimized multigrid solver to the unoptimized multigrid solver run with various smoother iteration counts. In the legend, 'Unopt' is unoptimized, and 'Opt' is optimized; the number in parenthesis is the number of smoother iterations.



(a) Sphere (b) Fish (c) Hammer

(d) Scorpion (e) Cow (f) Nefertiti

Figure 2: Non-homogeneous simulation results for each of our test meshes comparing our optimized multigrid solver to the unoptimized multigrid solver run with various smoother iteration counts. In the legend, 'Unopt' is unoptimized, and 'Opt' is optimized; the number in parenthesis is the number of smoother iterations.



(a) $\alpha = 0.4$ **(b)** $\alpha = 10.6$

Figure 3: Homogeneous simulation results for Scorpion trained with $\alpha = 0.4$ (tin) and $\alpha = 10.6$ (diamond). In the legend, 'Unopt' is unoptimized, and 'Opt' is optimized; the number in parenthesis is the number of smoother iterations.

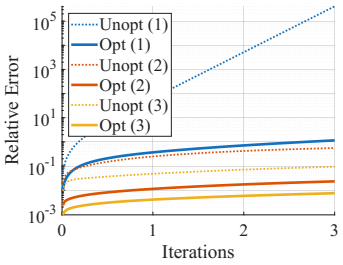


Figure 4: Homogeneous simulation results for Scorpion run with varying numbers of v-cycles. In the legend, 'Unopt' is unoptimized, and 'Opt' is optimized; the number in parenthesis is the number of v-cycles.

Table 3: Mesh and multigrid specifics for various meshes as well as wall clock training time for linear elasticity.

Mesh	Vertices	Depth	Train Time (hr)
Sphere	4,320	4	0.530
Fish	41,490	6	1.403

our optimized multigrid solver run with no additional smoother iterations outperforms the unoptimized multigrid solver run with one additional smoother iteration. In several cases (see Hammer, for example), our optimized multigrid solver outperformes the unoptimized multigrid solver run with several additional smoother iterations. This means that our solver will achieve the same level of accuracy faster than the unoptimized multigrid solver.

In Fig. 2, we use our optimized multigrid solver on non-homogeneous systems ($\mathbf{f} \neq \mathbf{0}$). In these simulations, we pick a single vertex and add heat to that vertex for three (3) seconds followed by removing heat from that vertex for three (3) seconds. We observe similar results to the homogeneous heat simulations in that our optimized multigrid solver consistently results in lower error accumulation than the unoptimized multigrid solver run with the same parameters and often accumulates less error than the unoptimized multigrid solver run with additional smoother iterations as well.

4.1.3 Varying Physical Parameters. All of the simulations mentioned above were run with the same set of physical parameters. Specifically, they were run with the thermal diffusivity constant corresponding to aluminum ($\alpha = 0.97$). To demonstrate that our method works with a variety of thermal diffusivity constants, we also optimize multigrid smoothers for Scorpion with $\alpha = 0.4$ and $\alpha = 10.6$ (corresponding to tin and diamond, respectively). The resulting error plots for these simulations are presented in Fig. 3. Just as with the aluminum simulation, our optimized multigrid solver accumulates less error over time than both the unoptimized multigrid solver run for the same number of smoother iterations and the unoptimized multigrid solver run for several additional smoother iterations.

4.1.4 V-cycle Generalization. While the main goal of our method is to reduce the error as much as possible for a predetermined set of multigrid parameters, sometime that improved accuracy still isn't enough to keep the accumulated error in a simulation from exploding (see, for example, Scorpion and Nefertiti in Fig. 2). For a standard multigrid solver, the accuracy of the solver can be easily improved by adding additional v-cycles to the solver. Yet, up to this point, we have only evaluated our optimized multigrid solver using the same number of v-cycles that the optimization was run with. In this section, we evaluate how well our optimized multigrid solver performs when adjusting the number of v-cycles.

Fig. 4 shows the error plots for Scorpion with both the unoptimized and optimized multigrid solvers run for one, two, and three v-cycles. We note that our optimized multigrid solver was optimized for only one (1) v-cycle. We first observe that the error for our optimized multgrid solver remains reasonable and decreases with the addition of additional v-cycles. This means that if the simulation needs to be run with a higher accuracy to prevent the accumulated error from exploding, our method can achieve this without requiring the smoother matrices to be reoptimized by simply adding additional v-cycles.

Additionally, we note that when comparing the results for the unoptimized and optimized multigrid solvers run for the same number of v-cycles, the simulation using our optimized multigrid solver has the higher accuracy. Finally, for this simulation, the error present in running our optimized multigrid solver with one v-cycle is roughly equal to the error present in running the unoptimized multigrid solver with two v-cycles. Thus, for the same level of accuracy, using our optimized multigrid solver would result in a simulation that was twice as fast.

4.2 Linear Elasticity

Finally, we apply our method to a statics system with linear elasticity. We test two meshes, Sphere and Fish, both of which were tetrahedralized for running a volumetric statics simulation. We use a piecewise constant mapping for these meshes as well, and the mesh and multigrid stats along with the wall clock training time are listed in Table 3. We pin a handful of vertices on each mesh to ensure the system matrix is well-posed, then run the optimization.

4.2.1 Single Solve Improvement. Just like with the heat equation, we start by analyzing our optimized smoother by looking at the error reduction of a single solve. We again calculate the relative

Table 4: Average relative error for various meshes compared to direct solve for a single multigrid solve using the unoptimized smoother matrix and our optimized smoother matrix.

Mesh	Random		One-Hot		Total
	Unopt.	Opt.	Unopt.	Opt.	Factor
Sphere	9.94e−1	3.70e−2	9.92e−1	1.06e−2	41.7
Fish	9.998e−1	8.42e−2	9.997e−1	8.19e−2	12.0

error according to Eq. (6) and average the result over a set of random right-hand sides and a set of one-hot right hand sides. The results are presented in Table 4.

Our method is still able to generate solution vectors that are multiple times more accurate than the unoptimized multigrid solver. We also note that this system had significant error present before optimization, yet our method was still able to produce smoothers that resulted in solutions within a reasonable accuracy for a real time iterative solver.

4.2.2 Simulation Results. We also test our optimized statics smoothers by using them in a simulation. Because we are running a statics simulation, there is only one simulation step, but we can see from the resulting plots, shown in Fig. 5, that our method produces results that are much more accurate to the ground truth than the unoptimized smoothers.

5 CONCLUSIONS

We present a method for optimizing Gauss-Seidel multigrid smoothers for real-time applications that can be easily dropped into an existing multigrid solver. We optimize the entries in the smoother matrices directly, meaning our method must be applied to physics systems for which the system is constant and linear. We test our method using the heat equation and statics with linear elasticity. For the heat equation, we observe up to an 11.7x improvement in accuracy for a single linear solve and reduced error accumulation over time when using our optimized solver in a simulation. For statics with linear elasticity, we observe up to a 41.7x improvement in accuracy for a single linear solve and simulation results that are much more accurate to the ground truth solution than when using the unoptimized solver.

For this paper, we chose a Gauss-Seidel smoother as our base smoother. However, our method is not restricted to a particular smoother. To use a different base smoother, one need only set up the new base smoother's iteration matrix as the optimization variable and run the rest of the optimization process exactly as described. As a proof of concept, we also optimized a block Jacobi smoother with a block size of three (3) for the heat equation on the Scorpion mesh and observed a 10.7x improvement in the accuracy of the multigrid solver.

5.1 Limitations and Future Work

One limitation of our method is that it is specific to the mesh, the boundary conditions, and the system being solved. Change any of those items, and the optimization must be run again. It is possible that deep learning techniques could address these limitations, and we leave that as future work. We also observed that the time required to optimize our solver can be greatly impacted by a mesh's quality and connectivity pattern.

Additionally, we discovered that our optimized smoother matrices do not perform well when used for a different (larger or smaller) number of smoother iterations than they were trained for. This is unfortunate because standard multigrid methods are able to use both additional v-cycles and additional smoother iterations to further reduce the error of the solution. Determining why our method does not and how to modify it so it does is an important area of future work.

Finally, this method is limited to applications with a constant, linear system matrix, and many common physical systems in computer graphics and animation are not constant and nonlinear. So, another important area of future work is to investigate how to expand our method to dynamic, nonlinear physical systems.

REFERENCES

[1] David Baraff and Andrew Witkin. 1998. Large steps in cloth simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. Association for Computing Machinery, New York, NY, USA, 43–54. https://doi.org/10.1145/280814.280821

[2] Jernej Barbič and Doug L James. 2005. Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM transactions on graphics (TOG)* 24, 3 (2005), 982–990.

[3] Theodore Barfoot, Ben Glocker, and Tom Vercauteren. 2024. *torchsparsegradutils: Sparsity-preserving gradient utility tools for PyTorch.* https://github.com/cai4cai/torchsparsegradutils

[4] Adam W Bargteil, Tamar Shinar, and Paul G Kry. 2020. An introduction to physics-based animation. In *SIGGRAPH Asia 2020 Courses*. 1–57.

[5] Jeff Bolz, Ian Farmer, Eitan Grinspun, and Peter Schröder. 2003. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM transactions on graphics (TOG)* 22, 3 (2003), 917–924.

[6] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective dynamics: fusing constraint projections for fast simulation. *ACM Trans. Graph.* 33, 4, Article 154 (jul 2014), 11 pages. https://doi.org/10.1145/2601097.2601116

[7] Achi Brandt. 1977. Multi-level adaptive solutions to boundary-value problems. *Mathematics of computation* 31, 138 (1977), 333–390.

[8] Achi Brandt and Oren E Livne. 2011. *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics, Revised Edition.* SIAM.

[9] Morten Bro-Nielsen and Stephane Cotin. 1996. Real-time Volumetric Deformable Models for Surgery Simulation using Finite Elements and Condensation. *Computer Graphics Forum* 15, 3 (1996), 57–66. https://doi.org/10.1111/1467-8659.1530057 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.1530057

[10] Shengze Cai, Zhicheng Wang, Sifan Wang, Paris Perdikaris, and George Em Karniadakis. 2021. Physics-informed neural networks for heat transfer problems. *Journal of Heat Transfer* 143, 6 (2021), 060801.

[11] Peter Yichen Chen, Jinxu Xiang, Dong Heon Cho, Yue Chang, G A Pershing, Henrique Teles Maia, Maurizio M Chiaramonte, Kevin Thomas Carlberg, and Eitan Grinspun. 2023. CROM: Continuous Reduced-Order Modeling of PDEs Using Implicit Neural Representations. In *International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=FUORz1tG8Og

[12] Yanqing Chen, Timothy A Davis, William W Hager, and Sivasankaran Rajamanickam. 2008. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software (TOMS)* 35, 3 (2008), 1–14.

[13] Yuyan Chen, Bin Dong, and Jinchao Xu. 2022. Meta-mgnet: Meta multigrid networks for solving parameterized partial differential equations. *Journal of computational physics* 455 (2022), 110996.

[14] Min Gyu Choi and Hyeong-Seok Ko. 2005. Modal warping: Real-time simulation of large rotational deformation and manipulation. *IEEE Transactions on Visualization and Computer Graphics* 11, 1 (2005), 91–101.

[15] Pasqua D'Ambra, Salvatore Filippone, and Panayot S Vassilevski. 2025. Optimal polynomial smoothers for parallel AMG. *Computers & Mathematics with Applications* 181 (2025), 1–14.

[16] Yutao Feng, Yintong Shang, Xuan Li, Tianjia Shao, Chenfanfu Jiang, and Yin Yang. 2023. PIE-NeRF: Physics-based Interactive Elastodynamics with NeRF. *arXiv preprint arXiv:2311.13099* (2023).

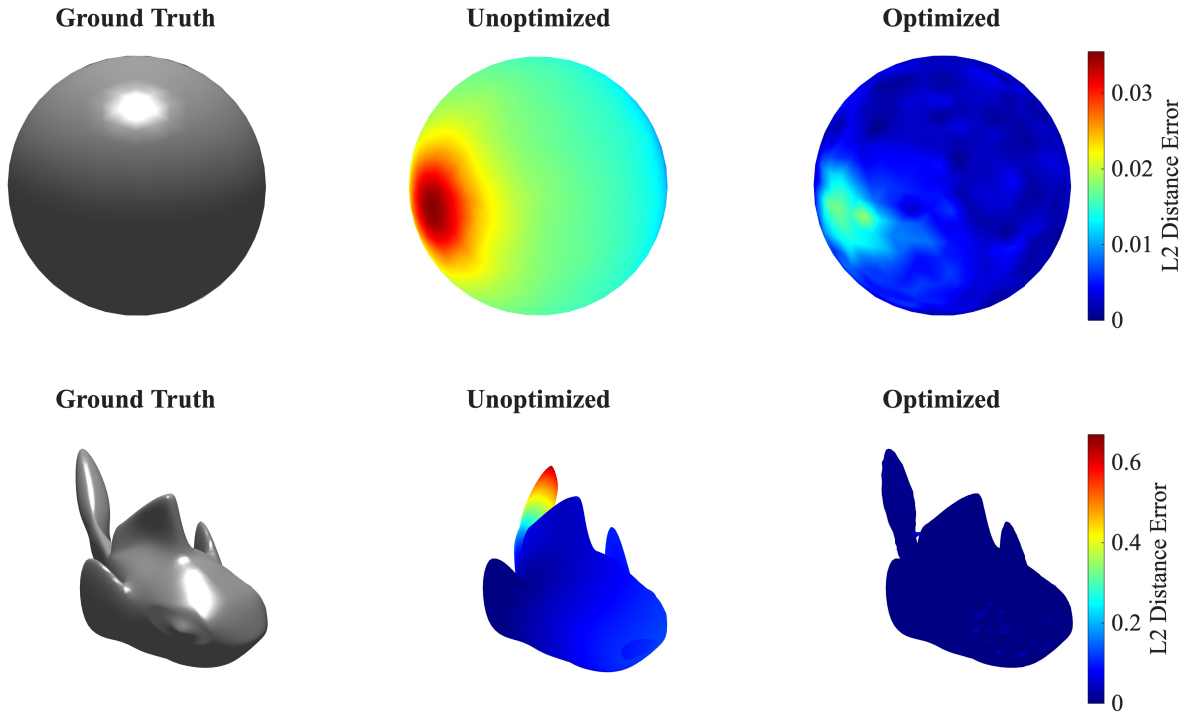


Figure 5: Static simulation results comparing our optimized multigrid solver to the unoptimized multigrid solver. The top row shows results for Sphere (radius 1) and the bottom row shows results for Fish (size $1.4 \times 3.2 \times 1.7$). For each mesh, we compare the ground truth solution (left), the unoptimized solver (middle), and the optimized solver (right), using color to represent the error in the multigrid results.

[17] Lawson Fulton, Vismay Modi, David Duvenaud, David IW Levin, and Alec Jacobson. 2019. Latent-space dynamics for reduced deformable simulation. In *Computer graphics forum*, Vol. 38. Wiley Online Library, 379–391.

[18] Gene H Golub and Charles F Van Loan. 2013. *Matrix computations*. JHU press.

[19] Nolan Goodnight, Cliff Woolley, Gregory Lewin, David Luebke, and Greg Humphreys. 2005. A multigrid solver for boundary value problems using programmable graphics hardware. In *ACM SIGGRAPH 2005 Courses*. 193–es.

[20] Daniel Greenfeld, Meirav Galun, Ronen Basri, Irad Yavneh, and Ron Kimmel. 2019. Learning to optimize multigrid PDE solvers. In *International Conference on Machine Learning (ICML)*. PMLR, 2415–2423.

[21] Eldad Haber, Lars Ruthotto, Elliot Holtham, and Seong-Hwan Jun. 2018. Learning across scales-multiscale methods for convolution neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[22] Juncai He and Jinchao Xu. 2019. MgNet: A unified framework of multigrid and convolutional neural network. *Science China Mathematics* 62, 7 (2019), 1331–1354.

[23] Yunhui He, Jun Liu, and Xiang-Sheng Wang. 2023. Optimized sparse approximate inverse smoothers for solving Laplacian linear systems. *Linear Algebra Appl.* 656 (2023), 304–323.

[24] Ru Huang, Kai Chang, Huan He, Ruipeng Li, and Yuanzhe Xi. 2023. Reducing Operator Complexity in Algebraic Multigrid with Machine Learning Approaches. *arXiv preprint arXiv:2307.07695* (2023).

[25] Ru Huang, Ruipeng Li, and Yuanzhe Xi. 2022. Learning optimal multigrid smoothers via neural networks. *SIAM Journal on Scientific Computing* 45, 3 (2022), S199–S225.

[26] Ayano Kaneda, Osman Akar, Jingyu Chen, Victoria Alicia Trevino Kala, David Hyde, and Joseph Teran. 2023. A Deep Conjugate Direction Method for Iteratively Solving Linear Systems. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 15720–15736. https://proceedings.mlr.press/v202/kaneda23a.html

[27] Alexandr Katrutsa, Talgat Daulbaev, and Ivan Oseledets. 2017. Deep multigrid: learning prolongation and restriction matrices. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*. 1–9. Also arXiv:1711.03825.

[28] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.).

[29] David P Koester, Sanjay Ranka, and Geoffrey C Fox. 1994. A parallel Gauss-Seidel algorithm for sparse power system matrices. In *Supercomputing'94: Proceedings of the 1994 ACM/IEEE Conference on Supercomputing*. IEEE, 184–193.

[30] Dmitry Kuznichov. 2022. Learning Relaxation for Multigrid. *arXiv preprint arXiv:2207.11255* (2022).

[31] Xiaoye S Li. 2005. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Transactions on Mathematical Software (TOMS)* 31, 3 (2005), 302–325.

[32] Yichen Li, Peter Yichen Chen, Tao Du, and Wojciech Matusik. 2023. Learning preconditioners for conjugate gradient PDE solvers. In *International Conference on Machine Learning*. PMLR, 19425–19439.

[33] Haoxiang Liu et al. 2025. Vertex-based Graph Neural Solver and its Application to Linear Elasticity Equations. *arXiv preprint arXiv:2501.xxxxx* (2025).

[34] Tiantian Liu, Adam W Bargteil, James F O'Brien, and Ladislav Kavan. 2013. Fast simulation of mass-spring systems. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 1–7.

[35] Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. 2017. Quasi-newton methods for real-time simulation of hyperelastic materials. *Acm Transactions on Graphics (TOG)* 36, 3 (2017), 1–16.

[36] Oren E Livne and Achi Brandt. 2012. Lean algebraic multigrid (LAMG): Fast graph Laplacian linear solver. *SIAM Journal on Scientific Computing* 34, 4 (2012), B499–B522. Published online 2012, often cited as 2013 or arXiv:1108.1310.

[37] Ilay Luz, Meirav Galun, Haggai Maron, Ronen Basri, and Irad Yavneh. 2020. Learning algebraic multigrid using graph neural networks. In *International Conference on Machine Learning*. PMLR, 6489–6499.

[38] Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. 2011. Efficient elasticity for character skinning with contact and collisions. In *ACM SIGGRAPH 2011 Papers* (Vancouver, British Columbia, Canada) *(SIGGRAPH '11)*. Association for Computing Machinery, New York, NY, USA, Article 37, 12 pages. https://doi.org/10.1145/1964921.1964932

[39] Changyu Meng and Yongming Liu. 2023. A multigrid finite element neural network for efficient material response prediction. In *AIAA SCITECH 2023 Forum*. 0770.

[40] Xinyu Meng, Wing Kam Liu, et al. 2025. MFEA-Net: A pixel-adaptive multigrid finite element analysis neural network for efficient material response prediction. *Computer Methods in Applied Mechanics and Engineering* 418 (2025), 116556.

[41] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. *Journal of Visual Communication and Image Representation* 18, 2 (2007), 109–118.

[42] Baptiste Nocolet, Wenzel Jakob, and Jonathan Schmidt. 2026. cholespy: An easily integrable Cholesky solver on CPU and GPU. https://github.com/rgl-epfl/cholespy Accessed: 2026-01-28.

[43] Luke N Olson, Jacob B Schroder, and Ray S Tuminaro. 2011. A general interpolation strategy for algebraic multigrid using energy-minimization. *SIAM Journal on Scientific Computing* 33, 2 (2011), 966–991.

[44] S Parthasarathy et al. 2025. Towards Automated Algebraic Multigrid Preconditioner Design Using Genetic Programming for Larger-Scale Laser Beam Welding Simulations. *Applications in Engineering Science* (2025), 100200.

[45] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in neural information processing systems*. Curran Associates, Inc., 8024–8035.

[46] A. Pentland and J. Williams. 1989. Good vibrations: modal dynamics for graphics and animation. *SIGGRAPH Comput. Graph.* 23, 3 (jul 1989), 207–214. https://doi.org/10.1145/74334.74355

[47] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. 2021. Learning Mesh-Based Simulation with Graph Networks. In *International Conference on Learning Representations*.

[48] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics* 378 (2019), 686–707.

[49] Yousef Saad. 2003. *Iterative methods for sparse linear systems*. SIAM.

[50] Igor Santesteban, Miguel A. Otaduy, Nils Thuerey, and Dan Casas. 2022. ULNeF: Untangled Layered Neural Fields for Mix-and-Match Virtual Try-On. In *Advances in Neural Information Processing Systems, (NeurIPS)*.

[51] Lin Shi, Yizhou Yu, Nathan Bell, and Wei-Wen Feng. 2006. A fast multigrid algorithm for mesh deformation. *ACM Transactions on Graphics (TOG)* 25, 3 (2006), 1108–1117.

[52] G. W. Stewart. 2002. A Krylov–Schur Algorithm for Large Eigenproblems. *SIAM J. Matrix Anal. Appl.* 23, 3 (2002), 601–614. https://doi.org/10.1137/S0895479800371529

[53] Natarajan Sukumar and Ankit Srivastava. 2022. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Computer Methods in Applied Mechanics and Engineering* 389 (2022), 114333.

[54] Rasmus Tamstorf, Toby Jones, and Stephen F McCormick. 2015. Smoothed aggregation multigrid for cloth simulation. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 1–13.

[55] Wei-Pai Tang and Wing Lok Wan. 2000. Sparse approximate inverse smoother for multigrid. *SIAM J. Matrix Anal. Appl.* 21, 4 (2000), 1236–1252.

[56] Yun Teng, Mark Meyer, Tony DeRose, and Theodore Kim. 2015. Subspace condensation: full space adaptivity for subspace deformations. *ACM Trans. Graph.* 34, 4, Article 76 (jul 2015), 9 pages. https://doi.org/10.1145/2766904

[57] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. 1987. Elastically deformable models. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. 205–214.

[58] U. Trottenberg, C. W. Oosterlee, and A. Schüller. 2001. *Multigrid*. Academic Press.

[59] Fan Wang, Xiang Gu, Jian Sun, and Zongben Xu. 2023. Learning-based local weighted least squares for algebraic multigrid method. *J. Comput. Phys.* 493 (2023), 112437.

[60] Zhen Wang, Yun Liu, Chen Cui, and Shi Shu. 2024. Momentum-Accelerated Richardson(m) and their Multilevel Neural Solvers. *arXiv preprint arXiv:2412.08076* (2024).

[61] Zhendong Wang, Longhua Wu, Marco Fratarcangeli, Min Tang, and Huamin Wang. 2018. Parallel Multigrid for Nonlinear Cloth Simulation. *Computer Graphics Forum* (2018). https://doi.org/10.1111/cgf.13554

[62] Nicholas J. Weidner, Theodore Kim, and Shinjiro Sueda. 2020. ConJac: Large Steps in Dynamic Simulation. In *Proceedings of the 13th ACM SIGGRAPH Conference on Motion, Interaction and Games* (Virtual Event, SC, USA) *(MIG '20)*. Association for Computing Machinery, New York, NY, USA, Article 6, 8 pages. https://doi.org/10.1145/3424636.3426901

[63] Andrew Witkin. 1997. Physically based modeling: principles and practice constrained dynamics. *Computer graphics* 9 (1997), 27.

[64] Zangyueyang Xian, Xin Tong, and Tiantian Liu. 2019. A scalable galerkin multigrid method for real-time simulation of deformable objects. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–13.

[65] Tianyi Xie, Zeshun Zong, Yuxin Qiu, Xuan Li, Yutao Feng, Yin Yang, and Chenfanfu Jiang. 2023. Physgaussian: Physics-integrated 3d gaussians for generative dynamics. *arXiv preprint arXiv:2311.12198* (2023).

[66] Yongning Zhu, Eftychios Sifakis, Joseph Teran, and Achi Brandt. 2010. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Transactions on Graphics (TOG)* 29, 2 (2010), 1–18.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009