A GPU-based Multi-level Algorithm for Boundary Value Problems

J. T. Becerra-Sagredo, F. Mandujano and C. Málaga

October 5, 2016

Abstract

A novel and scalable geometric multi-level algorithm is presented for the numerical solution of elliptic partial differential equations, specially designed to run with high occupancy of streaming processors inside Graphics Processing Units (GPUs). The algorithm consists of iterative, superposed operations on a single grid, and it is composed of two simple full-grid routines: a restriction and a coarsened interpolation-relaxation. The restriction is used to collect sources using recursive coarsened averages, and the interpolation-relaxation simultaneously applies coarsened finite-difference operators and interpolations. The routines are scheduled in a saw-like refining cycle. Convergence to machine precision is achieved repeating the full cycle using accumulated residuals and successively collecting the solution. Its total number of operations scale linearly with the number of nodes. It provides an attractive fast solver for Boundary Value Problems (BVPs), specially for simulations running entirely in the GPU. Applications shown in this work include the deformation of two-dimensional grids, the computation of three-dimensional streamlines for a singular trifoil-knot vortex and the calculation of three-dimensional electric potentials in heterogeneous dielectric media.

Parallel multigrid; GPU; CUDA
PACS: 65F10 65N22 65N55 65Y05 65Y10

1 Introduction

The multi-grid algorithm [4, 27] is one of the fastest methods for solving linear and non-linear systems of equations derived from a variety of problems, like numerical discretizations of partial differential equations and non-linear variational problems [18, 21]. The main idea is to accelerate the convergence of an iterative method using a hierarchy of nested grids where discretizations perform resolution-dependent corrections that are
passed between levels using interpolations. Multi-grid has the main advantage over other methods that when used to solve the problems with a given accuracy, its number of operations often scale linearly with the number of discrete nodes used. It has been widely studied and optimized for sequential operation, providing very fast convergent algorithms for a wide variety of applications [6, 22, 28, 5, 23, 19].

Multi-grid algorithms can be roughly divided in two groups: geometric multi-grid (GMG) and algebraic multi-grid (AMG). GMG works over a regular grid which can be perfectly subdivided into lower resolution grids, whereas AMG works over unstructured grids without a fixed neighbor node structure. GMG methods are more efficient than AMG on structured problems, since they take advantage of the fixed geometric representation, which is likewise thought to limit their applicability. However, they can handle some complex geometries by implementing boundary conditions over immersed interfaces [1].

Early works on multi-grid for GPUs date back to 2003, when GPUs started to outperform CPUs and control over the operations and memories of the GPU were available using programmable vertex shaders [3, 12]. These implementations are many-core maps of a classic multi-grid algorithm using recursive V-cycles. Additional interesting works in this direction are found in [26, 16, 13, 25], and more recently using Nvidia’s compute unified device architecture (CUDA) in [20, 8, 7, 10, 24, 17].

A consequence of the use of a classic multi-grid in many-core architectures is the appearance of performance penalties for coarse grids, where the number of independent operations is reduced and memory latencies cannot be hided using multithreading. In order to avoid these penalties, a hybrid approach is preferred by many authors, see [11, 9], where the CPU is used for serial matrix inversions over coarse grids, and the GPU for computing fine-grid relaxations. This is a natural way to avoid the low parallelism of the coarse grids but with the penalty of the memory exchange between the random access memory (RAM) of the CPU and that of the GPU, which can be several orders of magnitude lower in bandwidth than the GPU fast off-chip memory.

In this work we present a novel geometric multi-level algorithm that performs fully parallel operations over single grid data without the need of multiple grids. The algorithm is designed to work entirely inside the GPU without data transfers to the CPU. It is implemented with fine-grain parallelism, using a thread per grid node, such that its performance scales with every new generation of many-core architectures. It has only two simple full-grid many-core kernel routines: the relaxation-interpolation and the restriction. The routines are scheduled in a refining cycle, from the coarsest to the finest scales. Convergence of the solution of the linear system to machine precision is achieved repeating the cycle using the accumulated residual and collecting the overall result. A novel radial finite difference stencil is proven to provide convergence when used for the discretization of the heterogeneous Helmholtz-Poisson equation.

The algorithm is presented in section 2 and results are presented in section 3. The
results include convergence tests, and some applications to two-dimensional grid de-
formation, three-dimensional streamlines of a singular trifoil-knot vortex and three-
dimensional electric potentials in heterogeneous dielectric media.

2 The Algorithm

Let the elliptic partial differential equation be denoted by

$$\mathcal{L}(u) = f,$$

where $\mathcal{L}$ is a spatial partial differential operator in a bounded, cubic domain $\Omega \in \mathbb{R}^2$ or
$\mathbb{R}^3$, with Dirichlet, Neumann or mixed boundary conditions for $u$ at the boundary $\partial \Omega$.

The solution of (1) is equivalent to the steady state solution of the pseudo-time de-
pendent problem

$$\frac{\partial u}{\partial \tau} = (\mathcal{L}(u) - f),$$

with respective boundary conditions, starting, for example, from null fields.

The time explicit numerical integration of (2) requires a large amount of steps to
reach a steady state due to the slow transfer of low frequency components of the solution
[4]. Therefore, analogous to classic multi-grid, we propose to solve first (2) for the low
frequency components over a coarse subset of the entire mesh and interpolate the result
to the rest, repeating the process using the advanced solution data in finer subsets. For
this, as in the case of GMG, we need a grid that allows nesting, with $N = 2^n + 1$ points
per dimension, and $n \in \mathbb{N}$. The nested subset of level $\nu$ has $2^{n-\nu} + 1$ grid points and
spacing of $2^\nu h$, such that $\nu = 0, 1, \ldots, n - 1$.

But instead of using residuals and errors to collect the solution over an extended set
of multiple nested grids, like in a classic multi-grid algorithm, we have chosen to su-
perpose operations over a single grid data, fully in parallel. Obtaining the solution and
interpolating it to the rest of the mesh, almost simultaneously. Notice that this strategy
only makes sense for a fully parallel implementation and provides a completely differ-
ent method to a classic multi-grid. What we call the single-grid multi-level algorithm
(SGML) contains only two full-grid many-core kernel routines: the restriction and the
relaxation-interpolation, organized in a refining cycle with recurrences. This strategy
allows for the occupation all the streaming processors of the GPU, without the penalties
associated to low number of threads on coarse grids, and without communications to
the CPU.

The restriction is a local volume average of given fields, such as sources and co-
efficients, necessary to correctly solve the problem for a given level $\nu$. The idea is to
represent the fields in a coarse subset of the mesh by widening the support of interpo-
lations given by tensor products of linear interpolations, known as hat functions $\wedge(x)$.
1: \( g_1 = f \)
2: do \( m \) from 0 to \( \nu \)
3: \( \lambda = 2^m \)
4: for every \((i, j, k)\)
5: \( g_2 = \mathcal{R}_\lambda(g_1) \)
6: \( g_1 = g_2 \)

Algorithm 1. The restriction(\( \nu \)).

The sum of the neighboring sources, evaluated using the widened hat functions, can be done directly or using a simple multi-dimensional geometric reduction, consisting of a sequence of nested sums, from the finest grid till the desired level \( \nu \), as shown in Algorithm 1. In line 5 of Algorithm 1, \( \mathcal{R}_\lambda(f) \) is the weighted average of neighbors at distance \( \lambda h \). In two dimensions

\[
\mathcal{R}_\lambda(f_{i,j}) = \frac{1}{4} \sum_{p=-1}^{1} \sum_{q=-1}^{1} \left( \frac{-p}{2} \right) \left( \frac{-q}{2} \right) f_{i+p, j+q, \lambda},
\]

where \( \wedge(\pm 0.5) = 0.5 \), and in three dimensions

\[
\mathcal{R}_\lambda(f_{i,j,k}) = \frac{1}{8} \sum_{p=-1}^{1} \sum_{q=-1}^{1} \sum_{r=-1}^{1} \left( \frac{-p}{2} \right) \left( \frac{-q}{2} \right) \left( \frac{-r}{2} \right) f_{i+p, j+q, k+r, \lambda}.
\]

In the SGML, we choose the application of (2) over a coarse subset of the grid, with an explicit discretization using maximum stable pseudo time step and radial finite differences for \( \mathcal{L}(u) \), denoted by \( \delta \mathcal{L}(u) \). The relaxation-interpolation performs two operations in parallel for different sets of grid points as shown in Algorithm 2. The relaxation subset grid is defined by a multi-dimensional power of two module (line 3 of Algorithm 2), and if a node belongs to this subset, then the relaxation operation \( \mathcal{D}_\lambda \) is applied (line 4 of Algorithm 2). \( \mathcal{D}_\lambda \) represents the discretization of equation (2) in pseudo-time, with restricted sources and coefficients, using a forward Euler advance with stable time step and finite differences approximation of the spatial operator \( \mathcal{L} \) with grid spacing \( \lambda h \). The stability condition for \( \mathcal{L} = \nabla^2 \), with second order finite differences \( \delta \mathcal{L}(u) \), is a pseudo-time step \( \delta \tau < (2^\nu h)^2 / 2 \).

The interpolation operator \( \mathcal{I}_\lambda \) (line 6 of Algorithm 2) acts over the complement of the relaxation subset grid and uses the tensor product of hat functions \( \wedge(x) \) with support \( 2\lambda h \) to interpolate the variation \( \delta u = \mathcal{D}_\lambda(u_1) - u_1 \), obtained for those nodes in the relaxation subset grid at the previous iteration.

The exploration of refining cycles has provided us with the fast convergence cycle described in Algorithm 3 and sketched in Figure 1. It’s a saw-like pattern that is resemblant, but not equivalent, to what is known as a full multi-grid cycle [4]. The number
Algorithm 2. The relaxation-interpolation($\nu$).

1: $\lambda = 2^\nu$
2: $u_1 = u$
3: if $(i, j, k) = (0, 0, 0) \mod \lambda$
4: $u = \mathcal{D}_\lambda(u_1)$
5: else
6: $u = \mathcal{I}_\lambda(\delta u_1)$

Algorithm 3. The single cycle.

1: $n = \log_2(N - 1)$
2: do $v_1$ from $n-1$ to 0
3: do $\nu$ from $v_1$ to $v_1 - 1$
4: restriction($\nu$) [Algorithm 1]
5: do min($n_r$, $2^{n-v_1}$) times relaxation-interpolation($\nu$) [Algorithm 2]
6: do $\nu = v_1$
7: restriction($\nu$) [Algorithm 1]
8: do min($n_r$, $2^{n-v_1}$) times relaxation-interpolation($\nu$) [Algorithm 2]
9: do $\nu = 0$
10: do min($n_r$, $2^n$) times relaxation-interpolation(0)

Figure 1: Single cycle with doubling relaxation-interpolations for a $33^2$ mesh. The numbers beside the points indicate the number of relaxation-interpolations per level for full doublings. During the recurrence of cycles, this number can be bounded by a maximum number of relaxations $n_r$. 
Figure 2: Scheme for the radial finite differences approximation of the divergence of the gradient (5) at coordinates \((i,j,k)\). Individual radial derivatives are computed with all the 26 neighbors and integrated over the area of the sphere with radius \(h/2\) using the divergence theorem.

of relaxation-interpolations doubles for every level \(\nu\) and can be limited by a maximum number of repetitions \(n_r\) (see lines 5, 8 and 10 of Algorithm 3).

The convergence achieved in a single cycle is extended to a desired tolerance by recurring the cycles using the residual after each cycle as source term and adjusting the boundary conditions. The cycles are repeated, first substituting the sources with the residual \(r_1 = f - \delta \mathcal{L}(u_0)\), and so on successively, solving each recurrence for \(e_i\) in \(\delta \mathcal{L}(e_i) = r_i\), where the residual of cycle \(i\) is \(r_i = r_{i-1} - \delta \mathcal{L}(e_{i-1})\). If we define \(r_0 = f\) and \(u_0 = e_0\), then \(u = \sum_{i \geq 0} e_i\) is the solution of the algebraic system to machine precision for enough iterations.

Dirichlet boundary conditions are imposed in the variation \(\delta u\) for the first cycle. Neumann boundary conditions require the use of images in the restriction routine, the imposition of the derivative at the boundary substituting data lying outside of the domain, and the integral of the sources over the whole domain to be zero. The latest condition is necessary to avoid the migration of the solution, given that the Neumann problem has multiple shifted solutions.

3 Numerical examples

Consider the Helmholtz equation in heterogeneous media with \(\mathcal{L} = \nabla \cdot (\sigma \nabla) + a\). Convergence of the algorithm has been achieved using a discretization with a 9-point stencil in two dimensions or a 27-point stencil in three dimensions. The 27-point formula,
sketched in Figure 2, is given by
\[
\nabla \cdot (\sigma \nabla u)_{i,j,k} \approx \frac{3}{13h} \sum_{p=-1}^{1} \sum_{q=-1}^{1} \sum_{r=-1}^{1} \left( \sigma_{i+p,j+q,k+r} + \sigma_{i,j,k} \right) \frac{2}{h} \frac{(u_{i+p,j+q,k+r} - u_{i,j,k})}{l_{p,q,r}},
\]
where \( l_{p,q,r} \) is unity for \((\pm 1,0,0), (0,\pm 1,0), (0,0,\pm 1)\); \( \sqrt{2} \) for all combinations of \((\pm 1,\pm 1,0),(\pm 1,0,\pm 1),(0,\pm 1,\pm 1)\); and \( \sqrt{3} \) for all combinations of \((\pm 1,\pm 1,\pm 1)\).

### 3.1 Convergence

The SGML is tested for convergence, solving the two-dimensional Poisson equation
\[
\nabla^2 u = f
\]
on the unit square \( \Omega = [0,1] \times [0,1] \) with Dirichlet boundary conditions \( u = 0 \) on \( \partial \Omega \).

An analytic solution used in [4], is the polynomial
\[
u(x_1,x_2) = -x_1^2x_2^2(1-x_1^2)(1-x_2^2),
\]
corresponding to the source
\[
f(x_1,x_2) = -2(x_2^3(1-6x_1^2)(1-x_2^2) + x_1^3(1-6x_2^2)(1-x_1^2)).
\]

The error of the approximations is computed using the normalized \( L_1 \)-error
\[
\frac{||u - v_h||_{L_1}}{||u||_{L_1}} = \frac{1}{\int_{\Omega} |u|d\Omega} \int_{\Omega} |u - v_h|d\Omega.
\]
The recurrence of the cycles is shown in Figure 3 with maximum relaxations per level $n_r = 2$. The residual is computed using the maximum norm divided by the maximum norm of the sources. It shows a convergent saw pattern for every recurrence of the cycle, converging to machine precision. The $L_1$-error shows fast convergence for every cycle. Converging a little less than $10^{-2}$ for every cycle and stagnating to the discretization error. The diagnostic residual is the coarsened residual computed while applying the relaxation at every level with restricted sources and coefficients. This pseudo-residual is cost-less and is used to have control over the convergence of the algorithm in cases that the solution is unknown, by establishing a tolerance. Notice that for the full mesh, both residuals are the same. The examples shown in the following sections have reached a tolerance of $10^{-14}$.

The SGML algorithm is programmed with independent threads for every node in the grid, a straightforward formulation in many-core architectures for 100% parallel efficiency [15]. This has been observed in the numerical experiments shown in Figure 4. The runs are performed in GPUs with different number of processors.

### 3.2 Grid deformation

The deformation of grids is achieved using the force potential technique similar to that presented in [14]. A potential is obtained solving the Helmholtz equation

$$\nabla^2 u + au = f - \frac{1}{V} \int f d\Omega,$$

(10)
where $V$ is the area of $\Omega$, with Neumann boundary conditions $\frac{\partial u}{\partial n} = 0$.

The grid nodes are moved spatially with a forward Euler method, using equations of motion given by $\frac{dx}{dt} = v$, where the velocity

$$v = \frac{-\nabla u}{tf + \int f \, d\Omega},$$  \hspace{1cm} (11)$$
is computed using finite differences of the potential and with a pseudo-time $t$.

The source $f$ is a delta function over a parametrized curve, given by a set of points, not necessarily equidistant and resampled to equidistant points over the curve using the mesh spacing $h$. Z-splines [2] of first, third and fifth order are used to represent the continuous curve with the possibility of having discontinuous tangents. Tensor products of hat functions are used to represent a finite delta function source on the Cartesian grid. Examples for the deformation of a two-dimensional grid using the contours of a circle and Mexico are shown in Figure 5 with $a = 0.1$.

### 3.3 Streamlines of a singular trifoil-knot vortex

A three dimensional version of the SGML algorithm is used to solve the vector system

$$\nabla^2 \psi = -\omega,$$  \hspace{1cm} (12)$$
where $\psi \in \mathbb{R}^3$ is the stream vector potential and $\omega$ is the vorticity $\omega = \nabla \times v$, where $v$ is the velocity field.

A singular one-dimensional source of vorticity, pointing tangentially, is placed in the form of a trifoil-knot described by

$$\begin{align*}
x_1(\theta) &= r(\sin \theta + 2 \sin(2\theta)), \\
x_2(\theta) &= r(\cos \theta - 2 \cos(2\theta)), \\
x_3(\theta) &= -2r \sin(3\theta),
\end{align*}$$  \hspace{1cm} (13)$$
Figure 6: Lateral and frontal views of two streamlines around a singular trifoil-knot vortex colored with the scaled velocity magnitude \( v \) for a mesh of \( 257^3 \) nodes. One trajectory is trapped around the knot and the other orbits inside the box showing an average trifoil-orbit. A surface for \( v = 0.02 \) is shown in translucent grey.

where the parameter \( \theta \in [0,2\pi) \).

The velocity field can be obtained from the vector potential by
\[
\mathbf{v} = \nabla \times \mathbf{\psi}.
\]  

Streamlines can be integrated using the resulting velocity field with a RK-marching scheme provided in the visualization tool.

The solutions for the vector potential and the streamlines of a trifoil-knot vortex in a \([0,1]^3\) box are shown in Figure 6 for a \( 257^3 \) mesh and \( r = 0.2 \). Only two streamlines are shown: one close to the knot vortex, turning and traveling around the knot; and the second orbiting far from the singularity.

### 3.4 Heterogeneous coefficients

The heterogeneous Laplace equation
\[
\nabla \cdot (\sigma \nabla u) = 0,
\]  
is solved in three dimensions for two smooth coefficients
\[
\sigma(x_1,x_2,x_3) = 0.55 \pm 0.45 \tanh((r-0.2)/0.1),
\]

where \( r \) is the distance from the centerpoint of a \([0,1]^3\) domain. Dirichlet conditions are imposed on the top and bottom boundaries, set to \( u = 1 \) and \( u = -1 \), respectively, and Neumann conditions \( \partial u / \partial n = 0 \) are used for all the lateral faces.
Figure 7: Force lines produced by the electric potential $u$, solution to the heterogeneous Laplace equation (15), for the spherical variable coefficient of conductivity $\sigma$ (16), with higher (left) and lower (right) conductivity than its surroundings. The resolution is $129^3$ and the conductivity is represented by transparent surfaces from 0.2 to 0.8.

The problem represents an electric potential $u$ in a three dimensional capacitor filled with dielectric material of variable conductivity $\sigma$. Figure 7 shows the force lines $F = \nabla u$ produced by the electric potential $u$ for the given conductivity (16), which is a spherical distribution with lower (+) and higher (-) conductivity than its surroundings. The pseudo-time step is adjusted locally according to the local coefficient using the stability condition $\delta \tau < (2^N h)^2/2\sigma$.

4 Conclusion

The SGML algorithm is a novel and scalable iterative solver, designed to run entirely inside the GPU without memory transfers to the CPU. It is simple to implement and provides an attractive option for the solution of elliptic partial differential equations in simulations running entirely inside GPUs, stand-alone or as a system of hybrid hyperbolic-elliptic equations. In particular, we are including the SGML algorithm in incompressible fluid simulations and in computations of flows in porous media with an implicit formulation for the pressure, requiring the solution of a Helmholtz-Poisson equation for every time step. The main difference with a classic multi-grid algorithm is the use of a single grid data structure combined with fine-grain parallelism. The routines operate over full-grid data, keeping a constant high occupancy of the streaming processors in the GPU. The algorithm has been proven to scale with every new generation of many-core architectures. The choose of the finite difference operator is important. We have observed that simple finite difference molecules could lead to divergence of the algorithm
due to lack of correct information transmission for some directions of the mesh. Novel radial finite differences for the heterogeneous Helmholtz-Poisson equations have shown to provide convergence to machine precision for that particular equation. The overall residual is available from the algorithm data and can be used to monitor its convergence. Immersed interfaces could be used to obtain a solver for complex geometries without the need of special grids.

Acknowledgements

This work was partially supported by ABACUS, CONACyT grant EDOMEX-2011-C01-165873.

References

[1] L. Adams and Z. Li. The Immersed Interface/Multigrid Method for Interface Problems. *SIAM J. Sci. Comput.*, 24(2):463–479, 2002.

[2] J. Becerra-Sagredo, C. Málaga, and F. Mandujano. Moments Preserving and high-resolution Semi-Lagrangian Advection Scheme. *SIAM J. Sci. Comput.*, 38(4):A2141–A2161, 2016.

[3] J. Bolz, I. Farmer, E. Grinspun, and P. Shroeder. Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid. *ACM Trans. on Graphics (TOG)*, 22(3):917–924, 2003.

[4] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. Society for Industrial and Applied Mathematics, 2nd edition, 2000.

[5] A. Coco and G. Russo. Finite-difference ghost-point multigrid methods on Cartesian grids for elliptic problems in arbitrary domains. *Journal of Computational Physics*, 241:464–501, 2013.

[6] J. E. Dendy. Black box multigrid. *Journal of Computational Physics*, 48(3):366–386, 1982.

[7] E. Elsen, P. LeGresley, and E. Darve. Large Calculation of the Flow over a Hypersonic Vehicle Using a GPU. *J. Computational Physics*, 227(24):10148–10161, 2008.

[8] Z. Feng and P. Li. Multigrid on GPU: Tackling Power Grid Analysis on Parallel SIMT Platforms. *IEEE/ACM International Conference on Computer-Aided Design*, 2008.
[9] D. Göddeke and R. Strzodka. Cyclic Reduction Tridiagonal Solvers on GPUs Applied to Mixed Precision Multigrid. *IEEE Transactions on Parallel and Distributed Systems*, 22(1):22–32, 2010.

[10] D. Göddeke, R. Strzodka, J. Mohd-Yusof, P. McCormick, S. Buijsen, M. Grajewski, and S. Turek. Exploring weak scalability for FEM calculations on a GPU-enhanced cluster. *Parallel Computing*, 33(10-11):685–699, 2007.

[11] D. Göddeke, R. Strzodka, J. Mohd-Yusof, P. McCormick, H. Wobker, C. Becker, and S. Turek. Using GPUs to Improve Multigrid Solver Performance on a Cluster. *Int. J. Comput. Sci. Engin. (IJCSE)*, 4(1):36–55, 2008.

[12] N. Goodnight, C. Woolley, G. Lewin, D. Luebke, and G. Humphreys. A Multigrid Solver for Boundary Value Problems using Programmable Graphics Hardware. *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, 2003.

[13] L. Grady. A Lattice-Preserving Multigrid Method for Solving the Inhomogeneous Poisson Equations Used in Image Analysis, volume 5303 of *Lecture Notes in Computer Science*. Springer, 2008.

[14] M. Grajewski, M. Köster, and S. Turek. Numerical analysis and implementational aspects of a new multilevel grid deformation method. *J. App. Num. Math.*, 60(8):767–781, 2010.

[15] A. Grama, G. Karypis, V. Kumar, and A. Gupta. *Introduction to Parallel Computing*. Addison-Wesley, 2nd edition, 2003.

[16] H. Grossauer and P. Thoman. GPU-Based Multigrid: Real-Time Performance in High Resolution Nonlinear Image Processing, volume 5008. Springer, 2008.

[17] G. Haase, Liebmann M., C. Douglas, and G. Plank. A Parallel Algebraic Multigrid Solver on Graphics Processing Units. Number 5938 in Lecture Notes in Computer Science. Springer, 2010.

[18] W. Hackbusch. *Multigrid methods and applications*. Springer, 1985.

[19] Y. Jiang and X. Xu. Multigrid methods for space fractional partial differential equations. *Journal of Computational Physics*, 302:374–392, 2015.

[20] M. Kazhdan and H. Hoppe. Streaming Multigrid for Gradient-Domain Operations on Large Images. *ACM Trans. Graphics*, 27(3):1–10, 2008.

[21] S. F. McCormick. *Multigrid methods, Frontiers in applied mathematics*. Society for Industrial and Applied Mathematics, 2nd edition, 1994.
[22] B. Oskam and J. M. J. Fray. General relaxation schemes in multigrid algorithms for higher-order singularity methods. *Journal of Computational Physics*, 48(3):423–440, 1982.

[23] S. Popinet. A quadtree-adaptive multigrid solver for the Serre-Green Naghdi equations. *Journal of Computational Physics*, 302:336–358, 2015.

[24] T. Rehman, E. Haber, G. Pryor, J. Melonakos, and A. Tannenbaum. 3D non-rigid registration via optimal mass transport on the GPU. *Medical Image Analysis*, 13:931–940, 2009.

[25] T. Sangild and J. Mosegaard. *An Introduction to GPU Accelerated Surgical Simulation*, volume 4072 of *Lecture Notes in Computer Science*. Springer, 2006.

[26] R. Strzodka, M. Droske, and M. Rumpf. Image Registration by a Regularized Gradient Flow. Streaming Implementation in DX9 Graphics Hardware. *Computing*, 73(4):373–389, 2004.

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

[28] T. A. Zang, Y. S. Wong, and M. Y. Hussaini. Spectral multigrid methods for elliptic equations. *Journal of Computational Physics*, 48(3):485–501, 1982.