



**Utrecht  
University**

Game and Media Technology  
2022  
Thesis

---

# Simplification Directional Fields

Tom Broeders

---

Author: Tom Broeders  
Supervisor: dr. A. Vaxman,  
Department of Information and Computing Sciences, Utrecht University  
Second Examiner: prof.dr.ir. A.C. Telea,  
Department of Information and Computing Sciences, Utrecht University

## Abstract

In their recent paper, Liu et al. (2021) developed a new geometric multigrid solver that achieves better convergence than existing multigrid methods, and is orders of magnitude faster than conventional solvers. This was achieved by using a novel way to calculate the prolongation operator for scalar fields. In this work, we introduce similar multigrid solvers for directional fields, where we will use hierarchical simplification on directional fields to convergence to solutions quickly and accurately. Our main contributions are three new structure preserving prolongation operators that can transfer signals between the fine and the coarse multigrid levels for directional fields in discrete exterior algebra, face-based fields, and power fields. The prolongation operators can be defined as an interpolation from the fine to the coarse version of a collapsed 1-ring. We tested each of these operators with three different multigrid solvers on a most harmonic field problem for twelve different meshes with sizes ranging from  $13K$  to  $270K$  faces. Although the precomputation time for our operators is long, our prolongation operators in combination with our multigrid solvers are poised to outperform linear least squares by a significant margin when solving for larger meshes.

# Chapter 1

## Introduction

Directional fields are at the foundation of many problems on discrete surface meshes, such as directional flow, tangential flow, and mesh deformation. These problems require a solver that is both robust and structure preserving. In many cases however, these problems also need to be solved in real-time, which demands a solver that is quick as well.

For directional field problems on structured domains, it is common to use multigrid (MG) solvers. These solvers are accurate and are much faster than conventional least squares techniques. MG can precompute the system matrix using only the structure of the geometry. As a result, changes in the linear system do not require re-factorization and the same system matrix can be reused until the end of the application. This is ideal for dynamic real-time applications such as simulations. These solvers can also be combined with Discrete Exterior Calculus (DEC), which guarantees the solver to be structure preserving by having a well defined notion of linear operators such as divergence and curl. Unfortunately, MG solvers start to struggle when applied to unstructured domains such as arbitrary surface meshes.

As a result, traditional applications that need to solve directional field problems on unstructured surface meshes have been using direct solvers that use Cholesky decomposition for the full resolution of the mesh. Cholesky decomposition is very robust and works for any kind of system, but it scales poorly with the size of the mesh. Additionally, the decomposition often needs to be recalculated from scratch when the linear system changes. This forms a major bottleneck for dynamic real-time applications.

Recently however, a Galerkin geometric multigrid (GGMG) solver has been developed. This solver is capable of matching multigrid-like speeds on manifold surface meshes, while reaching better convergence than conventional solvers. This was achieved using a V-cycle technique in combination with a novel way of calculating the prolongation operator, using only the intrinsic geometry of the mesh. One shortcoming however, is that the prolongation operator was only defined for scalar fields, and not for directional fields.

In this thesis, we introduce three new, structure preserving prolongation operators for directional fields. We introduce a novel method for calculating the prolongation operator for DEC based directional fields, and we show that this method can easily be modified to create similar directional field prolongation operators for face-based fields and power fields. We show that these operators allow for large linear systems to be solved on manifold triangular surface meshes, at a significantly faster pace than conventional techniques. We also show that the accuracy of our technique is generally adequate, and can easily be tuned to the needs of the user at the cost of solve speed. This can be a worthwhile trade-off for many dynamic real-time applications, where solving directional field problems has always been a major bottleneck.

# Chapter 2

## Related work

Directional field problems are commonplace in geometry processing applications. Examples include tangential flow, mesh deformation, and mesh generation. There has therefore been a lot of research in this field in recent years. Relevant aspects of this research in tangent directional fields specifically has been covered by de Goes et al. (2016a). A more general review of relevant research in directional fields has been conducted by Vaxman et al. (2016).

### 2.1 Linear Least Squares

The standard method for solving linear problems is through the linear least squares (LLS) method. LLS is popular because it is both stable and accurate. However, it will generally result in huge sparse systems when applied to directional field problems. Solving such a system scales poorly with the size of the domain ( $O(n^2)$ ), which is often much too slow for real-time applications. Additionally, common physical applications such as solving the heat equation on a domain with a limited set of boundary conditions converge needlessly slowly. This is due to the local nature of LLS, where on each iteration a (temporary) solution on a given point in the domain only affects the points directly surrounding it.

A popular acceleration method for LLS solvers is Cholesky decomposition [Higham (1990)]. Cholesky decomposition can significantly reduce the size of the system matrix, speeding up the process of solving the system through LLS. As long as the system matrix doesn't change, the same decomposition can be used repeatedly, resulting in massive speedups. However, the decomposition itself is very slow ( $O(n^3)$ ). So when the system matrix does change, and the decomposition needs to be recalculated often, Cholesky decomposition can be slower than regular LLS. Techniques such as re-factorizations [Herholz and Sorkine-Hornung (2020)] and low-rank updates [Cheshmi et al. (2020)] can be utilized to update the Cholesky decomposition, rather than reconstructing it entirely. However, total reconstruction is the only option in many scenarios. Therefore, solving linear problems in general real-time and dynamic applications, such as simulations, requires a fundamentally different approach.

### 2.2 Multigrid

Multigrid methods [Brandt (1977)] is another popular technique for solving directional field problems. MG has been applied to solve optical flow fields by Enkelmann (1988) and it has been applied to solve Maxwell's equations by Hiptmair (1999). In a MG approach, the domain of the problem is



divided up into a number of structured grids, which range from coarse to fine. Movement between these grids can be facilitated by prolongation operators. The system matrix can then be constructed in an iterative manner, using only the internal geometry. Unlike Cholesky decomposition, the MG system matrix can be reused even when the external system changes. As a result, the MG technique is capable of solving complex systems in  $O(n)$ , which makes it ideal for high resolution domains. In some cases, MG can be used for unstructured domains as well. This was demonstrated for 3D finite element problems in solid mechanics by Adams (2002) and for 3D Euler equations on tetrahedral meshes by Mavriplis and Jameson (1987). It is also possible to apply MG to surface meshes, as demonstrated by Mavriplis (1995). These techniques use some form of interpolation to move between the coarse input mesh and the fine interpolated mesh. Unfortunately, these methods do not allow for movement from a fine input mesh to a decimated coarse mesh. This is mainly because edges are removed, rotated, and translated whenever a surface mesh is coarsened, making it very difficult to find appropriate prolongation operators between decimation steps.

## 2.3 Grid Subdivision and Prolongation

Another notable technique with a similar limitation is Subdivision Exterior Calculus (SEC) [de Goes et al. (2016b)]. This technique was developed to refine polygonal meshes without having to recalculate differential problems on the refined mesh. Instead, it uses subdivision techniques in combination with Whitney basis functions [Whitney (1957)] to prolong Discrete Exterior Calculus (DEC) [Crane et al. (2013)] fields from the coarse polygonal mesh to the fine subdivision surface. Alternatively, Subdivision Directional Fields (SDF) [Custers and Vaxman (2020)], combines Finite Element Methods (FEM) [Brenner and Scott (1994)] with ideas from DEC to provide a structure preserving subdivision procedure for face-based fields on surface meshes. These techniques use the assumption that the domain already has coarse degrees of freedom to construct the subdivision operators. However, this is a very limiting assumption since in many scenarios, the coarse degrees of freedom are not known.

This limitation was circumvented by the Galerkin Geometric Multigrid (GGMG) solver that was recently introduced by Liu et al. (2021). Similar to conventional MG techniques, this solver only requires the internal geometry to construct the system matrix, which means that the matrix can often be reused for the entire length of the application. Additionally, this solver introduces a novel way to calculate the prolongation operator for scalar fields, which permits the prolongation operator to be applied to unstructured curved surface meshes. This makes the GGMG solver much more versatile than most other MG methods. Moreover, this prolongation operator allows this solver to achieve better convergence than existing MG solvers.

# Chapter 3

## Theory

### 3.1 Discrete Exterior Calculus

DEC provides a simple and efficient mathematical framework for mesh processing procedures [Crane et al. (2013)]. In DEC, a surface mesh is treated as a chain complex where every  $k$ -simplex is assigned a unique  $k$ -form, forming a dual cochain complex. Consequently, the function field defined by these  $k$ -forms is only derived from the connections in the topology, and is therefore independent of any coordinate system. In addition,  $k$ -forms are anti-symmetric, meaning that if a  $k$ -simplex changes direction, the associated  $k$ -form changes sign.

In a practical sense, 0-forms are functions defined on the vertices of the mesh, 1-forms are functions defined on the edges of the mesh, and 2-forms are functions defined on the faces of the mesh. Every  $k$ -form has an associated  $k$ -field. As such, 0-fields are scalar fields defined on the vertices of the mesh, 1-fields are vector fields defined on the edges of the mesh, and 2-fields are volumetric fields defined on the faces of the mesh. DEC defines two operators that allows one to transform  $k$ -forms to  $k$ -fields and  $k$ -fields to  $k$ -forms. These are the sharp and flat operators respectively, formally known as the musical isomorphisms. In a practical sense, these operators don't do much. In a mathematical sense however, they are similar to the transpose operator in linear algebra, when transforming from a column vector to a row vector. It's a formality that allows us to distinguish between forms and fields.

DEC also allows for movement from  $k$ -forms to  $k + 1$ -forms with the differential operator  $d_k$ . In a general sense,  $d_k$  integrates a given  $k$ -form over a discrete surface mesh, resulting in a  $k + 1$ -form. In this thesis, we will only be using the  $d_0$  and  $d_1$  operators. These carry functions on vertices to functions on edges and functions on edges to functions on faces respectively. More precisely, if each edge gets a unique direction assigned, then  $d_0$  computes the difference between the values of the end vertex and the start vertex, and  $d_1$  adds the values of the boundary edges on a face according to their direction. Mathematically,  $d_0$  is a  $\mathcal{E} \times \mathcal{V}$  matrix such that

$$d_0 = \begin{pmatrix} 1 & 0 & 0 & \dots & -1 & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 & -1 \\ -1 & 0 & 0 & \dots & 0 & 1 & 0 \\ \vdots & & & \ddots & & \vdots & \\ 0 & 0 & -1 & \dots & 1 & 0 & 0 \\ 0 & 0 & 0 & \dots & -1 & 1 & 0 \\ 0 & 0 & 0 & \dots & 1 & 0 & -1 \end{pmatrix}.$$

Note that all rows add up to one, since every edge has exactly one start vertex and one end vertex. The  $d_1$  operator is similarly a  $\mathcal{F} \times \mathcal{E}$  matrix such that

$$d_1 = \begin{pmatrix} 1 & 0 & 0 & 1 & \dots & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & \dots & 0 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & \dots & 0 & 1 & 0 & 0 \\ & \vdots & & & \ddots & & & \vdots & \\ 0 & 0 & -1 & 0 & \dots & 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 0 & \dots & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & -1 \end{pmatrix}.$$

Here, all columns contain exactly two ones, since every edge is contained in exactly two faces. And every row contains exactly three ones, since every face contains exactly three edges.

Movement from  $k + 1$ -forms to  $k$ -forms is also possible through the boundary operator  $\partial_k$ , which is equal to the transpose of the differential operator  $d_k$  such that  $\partial_k = d_k^T$ . As such, the 0-form boundary operator takes edges to vertices and the 1-form boundary operator takes faces to edges. For simplicity, we will simply refer to the boundary operator simply as  $d_k^T$  in this thesis.

These definitions for the differential and the boundary operator are desirable because they provide the exact continuous integral. This can be proven using Stokes' theorem, which states that the integral over a domain is exactly equal to the signed sum on the boundary. Another notable consequence of these definitions is that  $d \cdot d = 0$ . This is another consequence of using Stokes' theorem, since the boundary of the boundary is always an empty set.

A consequence of this is that DEC inherently defines a de Rham complex on the surface mesh, which comes with many useful characteristics. Most notable among these is the notion that any exact form is automatically a closed form as well. Here, an exact form is a differential form that is the exterior derivative of some other differential form, and a closed form is a differential form whose exterior derivative is zero. On the other hand, closed forms are not necessarily exact forms as well. This depends on the topology of the mesh. For a prolongation operator to be structure preserving, exactness and closedness must be preserved after applying it to a differential form.

### 3.1.1 Primal Forms, Dual Forms, and the Hodge Star

In DEC, every (primal)  $k$ -form has a uniquely defined dual  $k$ -form. The primal 0-form has a dual 0-form that is the Voronoi areas of each vertex. The primal 1-form has a dual 1-form that is the 90 deg rotated edges. And the primal 2-form has a dual 2-form that is a point in the circumcenter of the face.

DEC allows movement between the primal and dual forms with the Hodge star operator, which is commonly denoted as  $\star_k$ . For 0-forms, the Hodge star is a diagonal matrix with the Voronoi areas of the mesh. For 1-forms, it is a diagonal matrix with ratio of dual-to-primal edge lengths. And for 2-forms—which are integrated in the primal setting—the hodge star contains *inverse* triangle areas. The relationship between primal 0-forms, 1-forms, and 2-forms with their respective dual forms is illustrated in Figure 3.1.

Because of the Hodge star operator is a de facto inverse of its respective  $k$ -form, it is often convenient to use it as a mass matrix. In other words, the  $\star_0$  operator is a good mass matrix for 0-forms because it perfectly divides the total area of the mesh among the vertices of the mesh, according to the Voronoi areas of the vertices. The  $\star_1$  operator is a good mass matrix for 1-forms because it encodes a de facto normalization procedure for every edge in the mesh. And finally the  $\star_2$  operator

is a good mass matrix for 2-forms because it connects every face with the inverse of its area, which encodes another de facto normalization procedure.

### 3.1.2 Hodge Decomposition

Every (primal) 1-form  $v$  on a closed mesh can be uniquely decomposed into an exact part, a coexact part, and a harmonic part:

$$\exists f \in \mathcal{V}, g \in \mathcal{T}, h \in \mathcal{H}, s.t. \quad v = d_0 f + \star_1^{-1} d_1^T \star_2 g + h.$$

Here,  $d_0 f$  encodes the *exact* component,  $\star_1^{-1} d_1^T \star_2 g$  encodes the *coexact* component, and  $h$  encoded the *harmonic* component. The exact component is what encodes the divergence of the field. This is as one can see:

$$d_0^T \star_1 v = d_0^T \star_1 d_0 f + 0 + 0.$$

The operator  $L_0 = d_0^T \star_1 d_0$  is the 0-form *Laplacian*. Equivalently,

$$d_1 v = d_1 \star_1^{-1} d_1^T \star_2 g$$

Where  $L_2 = d_1 \star_1^{-1} d_1^T \star_2$  is the colaplacian (or the 2-form laplacian) that connects between the coexact component  $g$  and the curl of the field  $d_1 v$ .

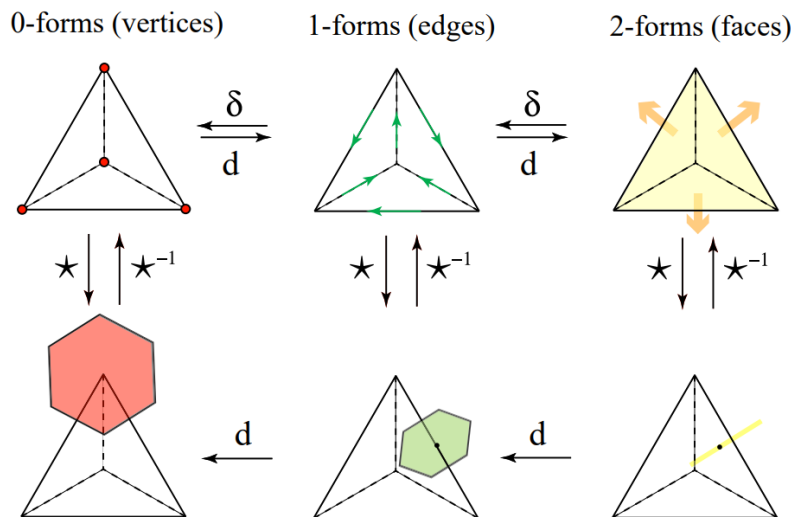


Figure 3.1: The primal and dual chain complexes. Original image from Desbrun et al. (2008).

## 3.2 Whitney Forms

One limitation of DEC is that a directional field is only defined on the edges of a given surface mesh. However, it is often necessary to have a continuous definition of the directional field over the entirety of the surface. On a triangular mesh, this can be achieved using Whitney forms. In short, Whitney forms are a barycentric interpolation of the three directional vectors on the triangle edges over the triangle surface. Keep in mind that we only define a 1-form over the surface mesh, and not a vector field. We must formally apply the flat operator to transform the 1-form into a vector

field before the barycentric interpolation can take place.

The appeal of using Whitney forms over other interpolation methods, is that the integral of the Whitney form over the boundary of the domain gives the original directional field back. This is essential when moving from 1-form to interpolated directional field to another 1-form.

### 3.3 Face-Based Fields and Power Fields

As an alternative to DEC based directional fields, one can also express directional fields as face-based fields and as power fields. Face-based fields [de Goes et al. (2016a)] are directional fields where every face on a mesh is assigned a 2D vector, which is tangent to the face and parameterized in the local basis. These vectors can be projected onto each triangle's half edges using projection operator

$$P_t = \begin{pmatrix} e_{t,12} \\ e_{t,23} \\ e_{t,31} \end{pmatrix}.$$

This operator creates a de facto 1-form, on half edges instead of regular edges. An example of projection operator  $P$  can be found in Figure 3.2. A halfedge-based field can be transformed back into a face-based field using either the pseudo-inverse of the projection operator or Whitney interpolation.

Power fields [Knöppel et al. (2013)] are essentially very similar to face-based fields. Only power fields are expressed in a complex number  $c$  raised to a power  $N$  per face instead of a vector. The complex number  $c^N$  can be transformed into a set of vectors by interpreting all  $N$   $N^{\text{th}}$  roots of  $c^N$  as vector  $v = (\text{Re}(c), \text{Im}(c))^T$ . Similar to face-based fields, power fields can be projected onto half edges using projection operator

$$\mathbf{P}_t^N = \begin{pmatrix} (\bar{\mathbf{e}}_{t,12})^N \\ (\bar{\mathbf{e}}_{t,23})^N \\ (\bar{\mathbf{e}}_{t,31})^N \end{pmatrix}.$$

Power field halfedge-forms can be transformed back into power fields using the pseudo-inverse of the projection operator.

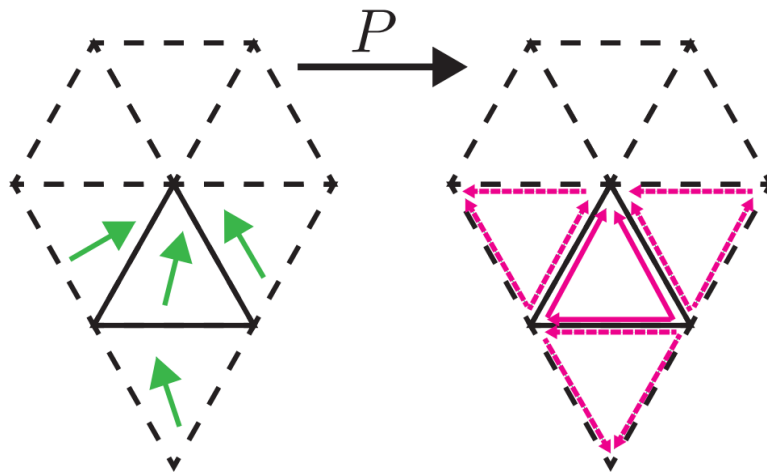


Figure 3.2: Projection from face-based field to halfedge-form. Edit of an original image from Custers and Vaxman (2020).

## 3.4 Multigrid

Classical uniform discretization procedures often produce surface meshes that are too fine in smooth areas of the geometry. This can cause unnecessary slowdowns when solving linear problems on such a mesh. Multigrid methods aim to solve this problem by introducing a sequence of grids with increasing resolution. We distinguish between algebraic multigrid (AMG) and geometric multigrid (GMG). In AMG, the grid levels are constructed independently of the geometry. In GMG however, each course level  $l + 1$  is constructed using the geometry of the finer level  $l$ . We shall be using mesh decimation steps to construct our grid levels, and as such we shall be using GMG. We will use GMG and MG interchangeably throughout this thesis.

In MG, linear problems can be solved by restricting the low frequency error on the coarser grids, and prolonging the high frequency error back to the finer grids. This process only has to be repeated until the remaining error is below some predetermined threshold. As a result, smooth areas of the mesh are solved very quickly, while detailed areas of the mesh get the time they need to converge properly.

The order in which restrictions, prolongations, Gauss-Seidel iterations, and LLS solves are performed strongly depends on the implementation of the solver. In fact, there exist many different kinds of multigrid solvers.

### 3.4.1 V-cycles

One popular multigrid solver is the V-cycle solver, which runs V-cycles until a predetermined level of convergence is reached. Such a V-cycle functions as follows. First, for a linear problem  $Ax = b$ , the finest grid solution  $x_0$  is guessed to be equal to zero. Then this solution  $x_0$  can be relaxed using a single iteration of the Gauss-Seidel method, after which a residual can be calculated as  $r_0 = b_0 - A_0x_0$ . This residual can subsequently be restricted by interpolating the system to the finer grid using  $b_1 = r_0$ ,  $x_1 = 0$ , and  $A_1 = P_1^T A_0 P_1$ , where  $P_1$  is a prolongation operator calculated beforehand. Once on the coarser grid, the same steps can be repeated again, until the coarsest grid level  $H$  is reached. This coarsest grid level can be solved exactly with LLS. With a solution for  $x_H$ , a solution for  $x_{H-1}$  can be obtained as  $x_{H-1} = x_{H-1,old} + P_H^T x_H$ . This solution for  $x_{H-1}$  can be relaxed one more time before returning the result to the finer grid. This process must be repeated until the finest grid is reached again. An illustration of the V-cycle algorithm can be found in Figure 3.3.

If the desired error is obtained, the process is finished. Otherwise more V-cycle can be run with the previous result as the initial guess for the finest grid. Using this method, a discrete system of  $n$  equations ( $n$  points on the finest grid) can be solved, to the desired accuracy, in  $O(n)$  operations.

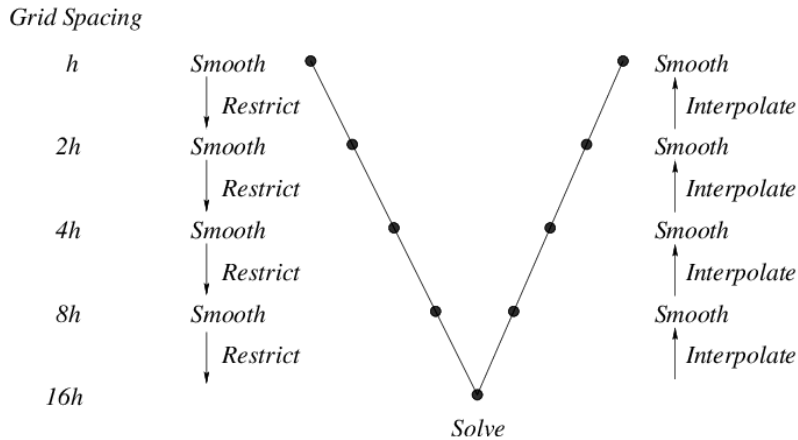


Figure 3.3: Basic illustration of the V-cycle algorithm. Original image from Moulton et al. (2008).

### 3.4.2 Nested Iterations

Another popular multigrid solver is the Nested Iterations solver. This solver is considered much simpler than V-cycles, but in many situations this solve is enough. Unlike V-cycles, the nested iterations solver starts at the coarsest level. Here, it solves the coarse system directly using LLS. Next, nested iterations iteratively prolongs the coarse solution to the next level, after which it will run a number of Gauss-Seidel iterations. The number of Gauss-Seidel iterations per level can be either a constant, or can be dynamically decided by running Gauss-Seidel iterations until the error stops decreasing. When the solver reaches the final level, it will run one final set of Gauss-Seidel iterations, after which it is finished.

### 3.4.3 Coarse Prolongation

The coarse prolongation solver strongly resembles the nested iterations solver, but without the Gauss-Seidel iterations. In other words, it solves the system on the coarsest level using LLS, and it prolongs this solution to the finest level directly.

## 3.5 Scalar Field Prolongation Operator

The most challenging part of applying MG is finding an appropriate prolongation operator  $P_l$  for each MG level  $l$ . For a long time this has formed a major road block for applying MG to problems on smooth surface meshes. Recently however, Liu et al. (2021) discovered a technique for finding such a prolongation operator for scalar fields (or 1-forms). To distinguish this prolongation operator from the DEC directional field (or 1-form), the face-based field, and the power field operators that we will introduce later on, we will refer to this operator as the  $P_0^l$  operator.

The  $P_0^l$  operator on a manifold surface mesh can be obtained as follows. If each level  $l + 1$  is obtained by collapsing a single edge on level  $l$ , then operator  $P_0^l$  that takes signals from level  $l$  to level  $l + 1$  must be a unit matrix for all vertices on the mesh, except in the (edge) 1-ring of the collapsed edge. This is a result of the fact that all vertices outside of the 1-ring of the collapsed edge remain completely unchanged. An illustration of such a 1-ring can be found in Figure 3.4.

Within this 1-ring, each  $l + 1$  vertex can be expressed in the barycentric coordinates of the  $l$  vertices. Because the mesh may be curved, this is not necessarily trivial however. Liu et al. developed

a joint flattening technique that works well for this application. This joint flattening technique projects the 1-ring before and after collapse on top of one another on a 2D plane. This is done by minimizing the joint distortion energy of both the 1-rings before and after collapse, while also ensuring the vertices on the outer rings overlap exactly.

With the barycentric coordinates, a matrix can be defined of size  $V_l \times V_{l+1}$ , where  $V_l$  is the number of  $l$  vertices and  $V_{l+1}$  is the number of  $l + 1$  vertices. Since most rows contain exactly a single unit and the remaining rows are expressed in barycentric coordinates, all rows sum up to exactly 1. As a result, the volume of the mesh is preserved by the  $P_0^l$  operator.

The total prolongation operator, that takes signals from the finest to the most course level, can be defined as  $P_0 = P_0^0 \cdot P_0^1 \cdot P_0^2 \cdot \dots \cdot P_0^H$ .

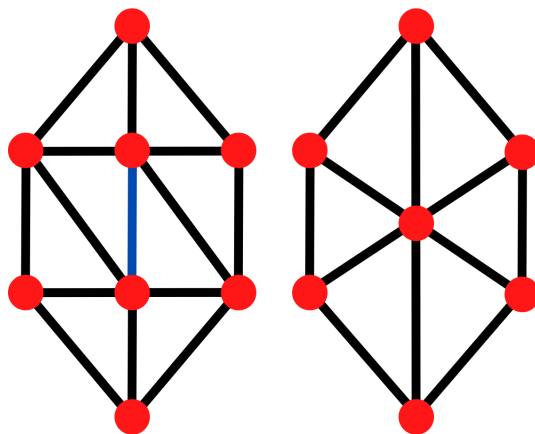


Figure 3.4: 1-ring edge collapse



# Chapter 4

## Directional-Field Prolongation Operators

In this chapter, we discuss how our three prolongation operators can be constructed. Note that we exclusively talk about the local versions of our prolongation operators. The construction of the global versions is discussed in chapter 5.

### 4.1 1-ring and Overlay

After every simplification step  $l$ , a coarse and fine planar 1-ring of the collapsed edge can be retrieved. These 1-rings can be considered small planar submeshes  $\mathcal{M}_C \subset \mathcal{M}^{l+1}$ , and  $\mathcal{M}_F \subset \mathcal{M}^l$  respectively. Note that we will not index the simplification level from this point on for simplicity. From the 1-rings, an overlay submesh can be created such that  $\Omega = \mathcal{M}_F \cup \mathcal{M}_C$ . An illustration of such an overlay mesh can be found in Figure 4.1. We will be using all three submeshes in the construction of our prolongation operators.

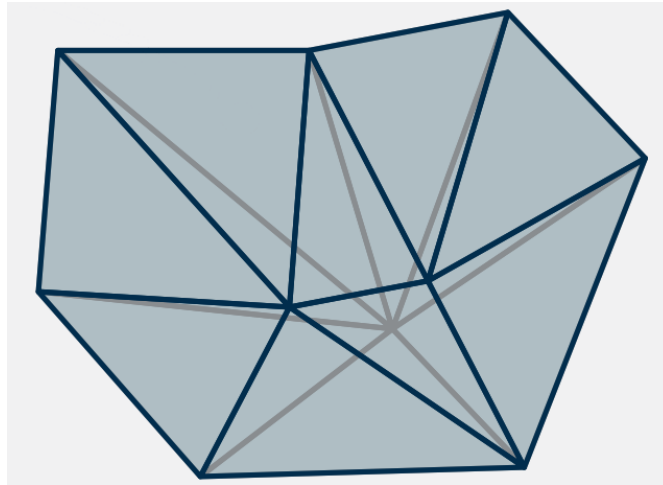


Figure 4.1: Coarse to fine 1-ring overlay. Coarse edges are marked in gray. Fine (sub-)edges and boundary edges are marked in dark blue. Original image from Liu et al. (2021).

### 4.2 DEC Prolongation

Our DEC 1-form prolongation operator takes coarse edges to fine edges as  $S_1 : \mathcal{E}_C \rightarrow \mathcal{E}_P$ . Here, we give a more detailed discussion on our implementation.

For a segment  $e_{ij}$  of a fine edge  $e_p$  which is overlaid on a coarse triangle  $t_{123}$ , we interpolate the vector field  $V_i$  and  $V_j$  using Whitney forms, and then integrate over  $e_{ij}$  to get:

$$z_{ij} = \frac{V_i + V_j}{2} \cdot e_{ij}.$$

Note the directionality. There is a simple way to compute this without explicitly going through the vector Whitney forms. We note that:

$$\begin{aligned} V_i = & z_{12} (B_1(i)\nabla B_2 - B_2(i)\nabla B_1) + \\ & z_{23} (B_2(i)\nabla B_3 - B_3(i)\nabla B_2) + \\ & z_{31} (B_3(i)\nabla B_1 - B_1(i)\nabla B_3). \end{aligned}$$

and similarly for  $V_j$ . We further note that  $\nabla B_k \cdot e_{ij} = B_k(j) - B_k(i)$ ,  $k = 0, 1, 2$  simply by Stokes' theorem. We then get that

$$\begin{aligned} \frac{V_i + V_j}{2} \cdot e_{ij} &= \frac{1}{2} z_{12} ((B_1(i) + B_1(j))(B_2(j) - B_2(i)) - (B_2(i) + B_2(j))(B_1(j) - B_1(i))) + z_{23} \dots \\ &= z_{12} (B_1(i)B_2(j) - B_2(i)B_1(j)) + z_{23} \dots \end{aligned}$$

For all segments  $e_{ij}$  of a fine edge  $e_p$ , with relative orientation determined by a sign  $s_{ij}$  to that fine edge, we set:

$$z_p = \sum_{ij \in p} s_{ij} z_{ij}.$$

In essence, our implementation of the  $S_1$  prolongation operator interpolates coarse edges to fine sub-edges, and sums fine sub-edges to fine edges. This implementation has some appealing properties.

- If the coarse 1-form is exact, the prolonged fine 1-form is also exact.
- The curl of the prolonged fine 1-form is equal to the curl of the coarse 1-form.
- Consequently, the prolonged closed 1-forms are also closed. Thus, harmonic 1-forms will be prolonged to a combination of harmonic 1-forms and some exact “noise” of high-frequency.
- Additionally, if the coarse 1-form is exact, every  $z_{ij} = f_j - f_i$  for a scalar PL function  $f$  defined on the entirety of  $\Omega$ . Consequently, when adding up all segments, it is easy to see that  $z_p$  is the differential of a function on the fine mesh where the vertex values are sampled from the PL function in a pointwise manner. That means our  $S_1$  commutes with  $S_0$ :

$$d_0 S_0 = S_1 d_0.$$

### 4.3 Face-Based Prolongation

We next show the implementation of our face-based prolongation operator  $S_\chi$ , which takes coarse face-based fields to fine face-based fields. First, we consider the projection operator  $P$ , which takes face-based field into halfedge forms, as discussed in section 3.3. Next, consider its pseudo-inverse  $P^\dagger$ , which does the inverse. In case  $\sum h_t \neq 0$  however, it basically filters out the average before finding

the face-based field. For a given face-based vector  $v_t$ , we have  $\sum h_t = 0$  by design. Therefore,  $P_t$  is of rank 2 (since  $\sum e_t = 0$ ).

After obtaining the halfedge form  $h_t$  from the coarse face-based field using  $P_t$ , we transform  $h_t$  to the plane just by copying. In fact, we only use that for subdivision (we never construct an explicit vector per-face in 2D). This means we can just do  $S_1$  in the plane, where instead of the 1-forms in a triangle we use the local halfedge form. Explicitly, we first sample the partial 1-forms on the fine edges using the direct form-to-form sampling formula, except we use the halfedge form in that coarse triangle. We then trivially make a fine halfedge-form from the fine 1-form by splitting the form into two: for a fine edge  $e$ , the fine 1-form  $z_e$  becomes  $h_e$  on the left of the edge, and  $-h_e$  on the right. Next, we transfer the forms to the 3D fine mesh  $\mathcal{F}$  by copying. For boundary edges, we only use the internal halfedge.

Lastly, we must project the fine halfedge-form back to zero sum. For this purpose we need to differentiate between two cases:

- The fine face is internal (there are exactly two like this in every local 1-ring). In this case, we simply use the filtering matrix

$$Q_I = P \cdot P^\dagger = \begin{pmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} \end{pmatrix},$$

which effectively just filters out the average of the halfedge form.

- The fine face has a boundary edge. For the sake of example, we determine it as the  $3^{rd}$  edge. Then, the projection has to preserve the tangent value on that edge, and we get the matrix:

$$Q_B = P \cdot P^\dagger = \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & 1 \end{pmatrix}$$

This effectively filters out the average, but only from the two internal edges of the boundary triangle.

We call the valid (zero-sum) halfedge-form subdivision operator  $S_h$ .  $S_{\mathcal{X}}$  is then designed as the composition of coarse vector to halfedge  $P$ , halfedge to halfedge refinement (generalization of  $S_1$ ), and fine halfedge to zero sum. Next,  $S_h \cdot Q$  can be chained together for many simplification steps until a major multigrid level is reached. When this happens, we proceed to transform the zero-sum halfedge form back to vector field. Specifically, we have that:

$$S_{\mathcal{X}} = (P^L)^\dagger \cdot S_h^L \cdot Q^L \dots S_h^0 \cdot Q^0 \cdot P^0.$$

That means that in general, we do not extract the actual face-based fields before reaching the final fine mesh, and we never need them for any intermediate level  $0 < l < L$ .

This definition of  $S_{\mathcal{X}}$  also has some useful properties:

- For exact fields, halfedge forms are equal on both sides of every edge-flap, which means the halfedge form is essentially equal to a 1-form. As a result, there is an equivalence of  $S_h$  with  $S_1$  for exact fields.

- Consequently, an exact commutation between  $S_{\mathcal{X}}$  and  $S_0$  exists where:

$$G \cdot S_0 = S_{\mathcal{X}} \cdot G,$$

where  $G$  is the DEC gradient operator.

## 4.4 Power Field Prolongation

For our power field prolongation operator  $S_P$ , which takes coarse  $N$  power fields to fine  $N$  power fields, we simply generalize  $S_{\mathcal{X}}$  to work with complex numbers. In fact,  $S_P$  is very similar to  $S_{\mathcal{X}}$  with two major differences:

1. Instead of using the Whitney-form formula for getting the value on a fine sub-edge, which might break on a power field, we simply unproject the coarse halfedges and project again. This means within coarse triangle  $t$ , on fine sub-edge  $ij$ , our prolongation formula is:

$$z_{ij}^N = \bar{e}_{ij}^N \cdot (P_t^N)^\dagger h_t^N,$$

where  $\bar{e}_{ij}$  is the complex conjugate of the complex edge  $e_{ij}$ . For a  $N = 1$  vector field, it merely reproduces the more sophisticated Whitney-base formula, but for power fields.

2. Instead of adding up the sub-edge contribution  $z_{ij}^N$  to the big fine edge  $e_p$ , we do the unprojection directly on the sub-edges, and we project again using the regular projection operator. Consequently, our  $Q$  is now:

$$Q^N = \begin{pmatrix} (\bar{e}_{12})^N \\ (\bar{e}_{23})^N \\ (\bar{e}_{31})^N \end{pmatrix} \cdot \begin{pmatrix} \bar{e}_{ij}^N \\ \dots \\ \dots \end{pmatrix}^\dagger$$

Note that  $Q$  is of dimensions  $3 \times \#\text{fineSubEdges}$ , and transforms the set of sub-edge forms  $z_{ij}$  into the proper  $h$  on the fine triangle.

Note that everything is still linear, the power is on the fixed edge vectors, and not our variables.

# Chapter 5

## Precomputation

Our prolongation operators are uniquely defined by the mesh they are created on. Therefore, our prolongation operators only need to be created once per mesh. This precomputation process is built up of several smaller algorithms chained together. Here, we will analyze these smaller algorithms and we will give a detailed discussion on how our total precomputation process works.

### 5.1 Qslim

When creating our prolongation operators, we coarsen the mesh one edge collapse at the time. This allows us to create a set of 1-ring pairs that can be used to generate our local prolongation operators as discussed in chapter 4. When collapsing edges however, it is essential that the (re)sampling error is kept at a minimum. This can be done by decimating the mesh using the Qslim algorithm [Micheal Garland (1997)].

In short, the Qslim algorithm assigns a quadric matrix  $Q_v$  to every vertex in the mesh. This matrix essentially captures the shape of the surrounding faces. Next, all valid vertex pairs must be selected. Although in principle, even unconnected vertices that are geometrically close together can be considered a pair, we only select pairs that are directly connected through an edge.

After the pairs are created, each pair must be assigned a collapsed vertex position  $\bar{v}$ . In the original Qslim algorithm, the position of this vertex is calculated using the inverse of the combined quadric. However, this resulted in a poor quality  $S_\chi$  operator for us. We have therefore decided to use midpoint positions instead, such that  $\bar{v} = (v_1 + v_2)/2$ . With the collapsed vertex positions, the error of the collapse can be calculated as  $e = \bar{v}^T(Q_{v_1} + Q_{v_2})\bar{v}$ . The list of pairs can consequently be sorted from lowest to highest error, such that pairs with a low collapse error are collapsed first. After a collapse has occurred, vertex  $v_1$  is moved to the position of  $\bar{v}$ , and vertex  $v_2$  is removed from the mesh. All references to  $v_2$  are updated to  $v_1$ .

One shortcoming of the default Qslim algorithm is that it doesn't prevent two neighbouring pairs from being collapsed in a short time span. When this happens, the two collapsed 1-rings overlap, which reduces the sparsity of the prolongation operators. We have made a modification that can help prevent this. After every collapse, we iterate over every pair with  $v_1$  or  $v_2$  in it, and we add a virtual cost  $cost_{max}$  to it. This ensures that surrounding pairs are placed at the end of the queue, reducing the number of overlapping 1-rings.

## 5.2 Overlay

After a pair is selected and the pre- and post-collapse 1-rings are created, an overlay can be constructed that bridges the gap between the two 1-rings. This overlay is a mesh that consists of what are essentially the building blocks of the two 1-rings. These are vertices, sub-edges, and sub-faces. Every sub-edge is mapped to a pre-collapse edge and either a post-collapse edge or a post-collapse face. Every sub-face is mapped to exactly one pre-collapse face and one post-collapse face. These maps are essential for transforming values from coarse 1-ring (half)edges to fine 1-ring (half)edges.

The algorithm we used to create the overlay converts all vertex coordinates from floating point numbers to rationals first. This ensures there are no floating point errors. It then places all outer vertices exactly on top of each other. From there, it finds all edge intersections. These can be used to create all the sub-edges and sub-faces, after which the maps are constructed.

## 5.3 Local to Global Operators

As detailed in chapter 4, all our prolongation operators are created on the local 1-ring. However, in order to use these operators on the full mesh, the local prolongation operators must be transformed into global prolongation operators. This process is relatively straight-forward. First, a map is created that transforms the indices of all local 1-ring (half)edges to the indices of global (half)edges. Then, a global  $N_{l+1} \times N_l$  sparse matrix is created, where  $N$  is the number of (half)edges in the mesh at level  $l$ . Using the map, the local 1-ring prolongation operator can then be placed directly inside of the global operator. Since all other (half)edges in the mesh remain untouched, identities should be placed on the remaining diagonal of the global operator. Keep in mind that potentially three edges are missing in the coarse mesh, so this introduces up to three 'jumps' in the fine edge to coarse edge pairs. The same is true for the fine face to coarse face pairs when working with halfedges.

## 5.4 The Total Process

In short, the precomputation process starts off by finding and ranking all collapsible vertex pairs from lowest to highest error. It then iteratively collapses a single pair and saves the coarse and fine 1-rings, along with their local-to-global index maps, in a queue. Note that these 1-rings are projected onto 2D using the same joint flattening technique discussed in the paper by Liu et al. (2021). Later, this queue is read in parallel. An overlay is created for every 1-ring pair, and the local prolongation operators are created. Next, the local-to-global index maps are used to transform the local prolongation operators into global prolongation operators. These are then stored in another list. Lastly, all prolongation operators per multigrid level are multiplied together to create a single  $S_{l \rightarrow l+1}$  operator. The set of all prolongation operators (one for each multigrid level), is then stored on disk for later use. In our implementation, every multigrid level takes a fine mesh to a coarse mesh that has exactly a fourth of the faces of the fine mesh. We create up to four multigrid levels total, where the coarsest level never has fewer than 1000 faces.

# Chapter 6

## Model Problems

In this thesis, we present one model problem to measure the performance of our prolongation operators with. For the sake of simplicity, this model problem will be the most harmonic field problem for DEC, face-based fields, and power fields. Because the most harmonic field problem is essentially the same system for all three frameworks, we only describe the system in full detail for DEC. For face-based fields and power fields, we only discuss the modifications needed to fit the system to these frameworks.

### 6.1 DEC Most Harmonic Field

Given a fine mesh  $\mathcal{M}^0$ , we first produce all discrete levels  $L$ , and the consequent  $S_1^L$  operator for prolonging between the levels. We give a subset of edges  $e_b \subset \mathcal{E}^0$  some prescribed values. We want to solve the harmonic interpolation problem (with soft constraints):

$$h = \operatorname{argmin}_{\lambda_D} \left( |d_1 h|^2 + |d_0^T \star_1 h|^2 \right) + \lambda_A |h_b - e_b|^2.$$

Here,  $|x|^2 = x^T M x$ , where  $M$  is the appropriate mass matrix  $\star_k$ . The energy  $|d_1 h|^2 + |d_0^T \star h|^2$  is the Dirichlet energy. As a quadratic form it can be written as:

$$|d_1 h|^2 + |d_0^T \star_1 h|^2 = h^T (d_1 \star_2 d_1^T + \star_1 d_0^T \star_0^{-1} d_0^T \star_1) h = h^T L_1 h$$

$L_1$  is the 1-form Laplacian (or Hodge Laplacian). Remember  $\star_2$  is already inverse triangle areas, and serves as the mass for integrated (primal) 2-forms. For the alignment part, we can write it succinctly as:

$$|h_b - e_b|^2 = |I_b h - e_b|^2 = h^T I_b^T \star_1 I_b h - h^T \star_1 I_b^T e_b.$$

where  $I_b : b \times \mathcal{E}^0$  is a matrix that just 'chooses' the values of  $b$  (it has 1 on the  $b_i$  column in each row, and 0 everywhere else).

On a given level  $L$ , the energy can be written as:

$$h^L = \operatorname{argmin}_{\lambda_D} \left( |d_1 S_1^L h|^2 + |d_0^T \star S_1^L h|^2 \right) + \lambda_A |(S_1^L h)_b - e_b|^2.$$

This is basically the same energy, except everything would be conjugated by  $S_1^L$ , and so with reduced degrees of freedom. Concretely, the system is now:

$$h^L = \operatorname{argmin} (\lambda_D (h^L)^T (S_1^L)^T L_1 S_1^L h^L + \lambda_A ((h^L)^T (S_1^L)^T I_b^T I_b S_1^L h^L - (h^L)^T (S_1^L)^T I_b^T e_b))$$

This can be abstracted to the following:

$$h = \operatorname{argmin} |Ah - b|^2.$$

Which culminates in the linear system:

$$A^T MAh = A^T Mb$$

where

$$A = \begin{pmatrix} d_1 \\ (d_0)^T \star_1 \\ I_b \end{pmatrix}, b = \begin{pmatrix} 0 \\ 0 \\ e_b \end{pmatrix}, M = \operatorname{blkdiag}(\lambda_D \star_2, \lambda_D \star_0^{-1}, \lambda_A \star_1).$$

This is the DEC most harmonic field system.

## 6.2 Face-Based Most Harmonic Field

It is not an entirely trivial task to generate a harmonic field for face-based fields. The common way is to find the null space of the operator

$$H = \begin{pmatrix} D_V \\ C_E \end{pmatrix},$$

where  $D_V$  is the face-based divergence operator and  $C_E$  is the face-based curl operator [de Goes et al. (2016a)]. Alternatively, and more robustly, one can take out the  $2g$  lowest eigenvectors of the square matrix  $H^T M_H H$ , with mass matrix:  $M_H = \operatorname{diag}(M_V^{-1}, M_E^{-1})$ . Here,  $M_V$  is the diagonal mixed elements vertex mass matrix

$$M_V(v, v) = \frac{1}{3} \sum_{t \in N(v)} A(t),$$

where  $N(v)$  is all faces adjacent to  $v$  (the 1-ring), and  $A(t)$  is the area of face  $t$ . And  $M_E$  is the diagonal mixed elements edge mass matrix

$$M_E(e, e) = \frac{1}{3} (A(t_1) + A(t_2)).$$

To solve the harmonic interpolation problem then, we solve for:

$$v = \operatorname{argmin} \lambda_H (v^T H^T M_H H v) + \lambda_A |v_b - u_b|^2$$

where  $b$  are the prescribed faces, and  $u_b$  are the prescribed vectors. From here, the same steps can be taken as in the DEC case to arrive at the most harmonic system matrix. Only using mixed elements operators instead of DEC operators. As such the face-based most harmonic field system is:

$$A^T MAh = A^T Mb$$

where

$$A = \begin{pmatrix} D_V \\ C_E \\ I_b \end{pmatrix}, b = \begin{pmatrix} 0 \\ 0 \\ u_b \end{pmatrix}, M = \operatorname{blkdiag}(\lambda_H M_V, \lambda_H M_E, \lambda_A M_F).$$



## 6.3 Power Field Most Harmonic Field

For a power field  $X$ , the most harmonic field problem can be constructed as

$$X = \operatorname{argmin} \sum_{e \in f, g} \frac{3|e|^2}{A(f) + A(g)} |X_f(\bar{e}_f)^N - X_g(\bar{e}_g)^N|^2 = W|\mathcal{D}^N X|^2,$$

where  $e$  is the *normalized edge vector*,  $A(t)$  is the area of triangle  $t$ ,  $W$  is the power field mass matrix, and  $\mathcal{D}^N$  is the power field divergence operator. Then, the power field Laplacian  $L_p^N = (\mathcal{D}^N)^\dagger W \mathcal{D}$ . The smoothest field problem then reduces to:

$$X = \operatorname{argmin} \lambda_S (X^\dagger L_p^N X) + \lambda_A |X_b - Y_b|^2,$$

where  $b$  is the set of prescribed faces, and  $Y_b$  is the prescribed power vector in them. To get the actual raw field, one can take all  $N^{\text{th}}$  roots of  $X$ .

This translates into a most harmonic field system  $AX = b$ , where

$$A = \begin{pmatrix} \lambda_S L_p^N \\ \lambda_A I_b \end{pmatrix}, b = \begin{pmatrix} 0 \\ Y_b \end{pmatrix}.$$

Note that we exclusively use power fields with  $N = 2$  in this thesis.

## 6.4 The Error Measure

In this thesis, we use two error measures. The normalized L2 error is defined as

$$er = \sqrt{\|E^T M E\| / \|x_{gt}^T M x_{gt}\|}, \quad E = x - x_{gt},$$

where  $M$  is the mass matrix of the system, and  $x_{gt}$  is the ground-truth solution for  $x$  as obtained from LLS. we will commonly refer to this measure simply as the error.

The second error measure we use is the normalized L2 energy. This measure is defined as

$$en = \sqrt{\|R^T M R\| / \|b^T M b\|}, \quad R = Ax - b.$$

We will commonly refer to this measure simply as the energy.

## 6.5 Solvers

For our most harmonic field systems, we use three multigrid solvers and one baseline LLS solver. More precisely, we use a coarse prolongation solver, a nested iterations solver, and a V-cycles solver, and we use a linear least squares solver in combination with Cholesky decomposition. Here, we briefly discuss these solvers.

### 6.5.1 Coarse Prolongation

The simplest solver is coarse prolongation, where we directly solve the system on the coarsest level using a LLS solver, and then prolong this solution directly to the finest level. This solver is likely to be very inaccurate, especially for large systems with large prolongation operators. But it will also be very fast, which may be a worth-while trade-off.

### 6.5.2 Nested Iterations

The second method is nested iterations. This is similar to coarse prolongation, where we first solve our system in the coarsest level using a LLS solver. Where this method differs is in the prolongation. Here, we only prolong one level at a time, and we apply a number of Gauss-Seidel iterations after every prolongation. The number of Gauss-Seidel iterations depends on the mesh and the operator. We keep running Gauss-Seidel until the normalized L2 error improvement per iterations drops below 0.1%. Because this cut-off point can be determined before running the solver, and because this cut-off point can also be a constant, we're not counting the calculation of this cut-off point in the prepare time or the solve time of nested iterations.

This solver will always be slower than the coarse prolongation solver, but it is also likely to be much more accurate. The Gauss-Seidel iterations are likely to strongly reduce the error introduced by the prolongation operators at every prolongation step.

### 6.5.3 V-cycles

The V-cycles method simply runs V-cycles until a certain error or time threshold is reached. It is not trivial to determine such a threshold since both low error and low time are desirable. We therefore graph the normalized L2 error of V-cycles against solve time. This allows us to compare V-cycles with the other solvers both in terms of error and solve time. The number of Gauss-Seidel iterations per multigrid level can strongly affect the convergence rate and solve time of the solver. We therefore determine the optimal number of V-cycle iterations per level per system in section 7.2.1.

With the optimal number of Gauss-Seidel iterations per level, V-cycles can rival nested iterations in both error and solve time.

### 6.5.4 Direct Fine Solve

Finally, we have a method that solves the model problems directly on the fine mesh. For this purpose, we use Cholesky decomposition to factorize the system, and we use regular and a LLS to solve the system. There are several different kinds of Cholesky decomposition. In this thesis, we use LDL decomposition. This algorithm is generally considered to be one of the fastest Cholesky decomposition methods. However, it still scales as  $O(n^3)$  where  $n$  is the size of the system. This means that the solve time of the Cholesky solver will rapidly grow with the size of the mesh. From this point, we commonly refer to the direct fine solver as the Cholesky solver.

# Chapter 7

## Results

In this chapter we demonstrate our three directional field prolongation operators on the meshes shown in Table 7.1. We show the prepare time, solve time, and the convergence of our operators when solving our most harmonic field problems with our three multigrid solvers: coarse prolongation, nested iterations, and V-cycles. All three are compared against the Cholesky solver. We also show the optimal settings for both V-cycles and nested iterations. Then we show the precomputation time necessary to generate our prolongation operators. Keep in mind that precomputation time is different from prepare time. In short, precomputation time is the time necessary to create our prolongation operators, whereas prepare time is the time necessary to prepare the solvers with a system matrix. After this, we take a more in-depth look at the relationship between the characteristics of the input mesh and quality of our prolongation operators. Lastly we present a brief comparison of our prolongation operators and the scalar field prolongation operator introduced by Liu et al.

### 7.1 Computation Time and Convergence

To show the quality of our prolongation operators, we solve our most harmonic system problem for all three operators using our Cholesky solver (CS), coarse prolongation solver (CP), nested iterations solver (NI), and V-cycle solver (VC). We consider the solution provided by the Cholesky solver to be the ground truth, and we compare the solutions provided by our other solvers against this.

The convergence of coarse prolongation and nested iterations for  $S_1$ ,  $S_\chi$ , and  $S_P$  can be found in Tables 7.8, 7.9, and 7.10 respectively. The computation times of the Cholesky solver, coarse prolongation, and nested iterations for  $S_1$ ,  $S_\chi$ , and  $S_P$  can be found in Tables 7.11, 7.12, and 7.13. All computation time results are averages of ten measurements. We have also split the computation time into a prepare part and a solve part to provide more insight. In the prepare part, the solvers are prepared with the system matrix. So, the Cholesky solver performs the factorization of the system matrix and the multigrid solvers coarsen the system matrix such that all multigrid levels have an appropriately coarsened system. The multigrid solvers also factorize the coarsest system matrix to facilitate the coarse direct solve.

The convergence of the V-cycle solver for  $S_1$ ,  $S_\chi$ , and  $S_P$  can be found in Figures 7.1, 7.2, and 7.3 respectively. Keep in mind that full convergence is often not necessary when using V-cycles. The appropriate cut-off point is determined by the needs of the particular application it is used on. Moreover, we have gathered indicative images of the solutions provided by our Cholesky solver and

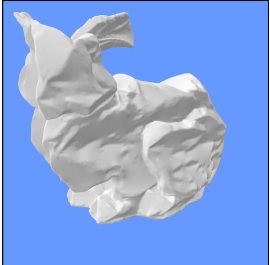

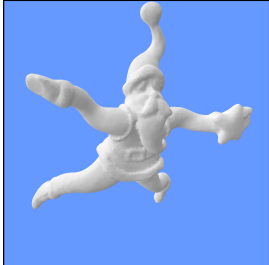



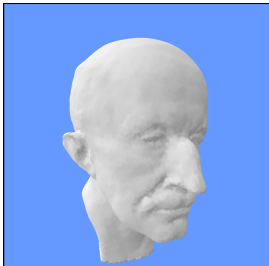
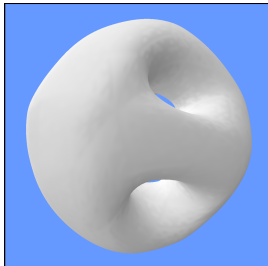


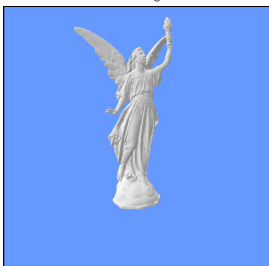
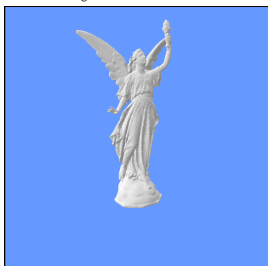
Deformed Bunny  278710 faces	Armadillo  174568 faces	Santa  151558 faces	Ramses  100000 faces
Chair  99968 faces	Owl  74538 faces	Old Face  55448 faces	Ball  31438 faces
Bunny Big  111364 faces	Bunny Small  13584 faces	Lucy  100000 faces	Lucy Remeshed  109592 faces

Table 7.1: Images, names, and sizes of all meshes used in this section.

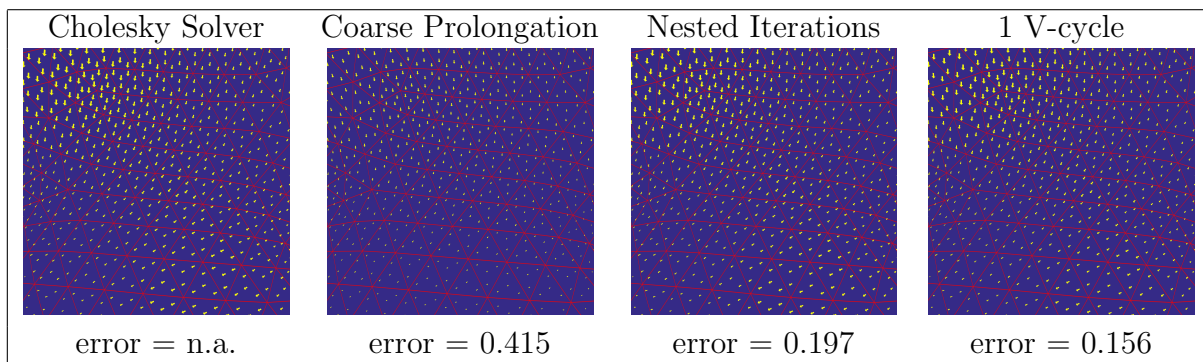


Table 7.2: Whitney interpolated solutions for the DEC most harmonic field problem on the armadillo mesh.

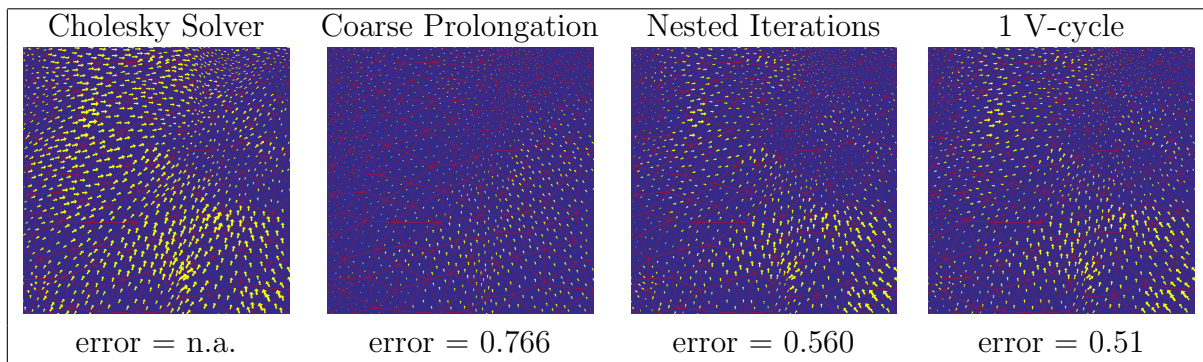


Table 7.3: Whitney interpolated solutions for the DEC most harmonic field problem on the owl mesh.

our multigrid solvers. Solutions for the DEC most harmonic field problem on the armadillo mesh and the owl mesh can be found in Figures 7.2 and 7.3. Solutions for the face-based most harmonic field problem on the deformed bunny mesh and the ball mesh can be found in Figures 7.4 and 7.5. And finally the solutions for the power field most harmonic field problem on the Ramses mesh and the small bunny mesh can be found in Figures 7.6 and 7.7. All solutions for the V-cycle solver were made with a single V-cycle. This makes the solve time comparable to that of the Cholesky solver and the coarse prolongation solver.

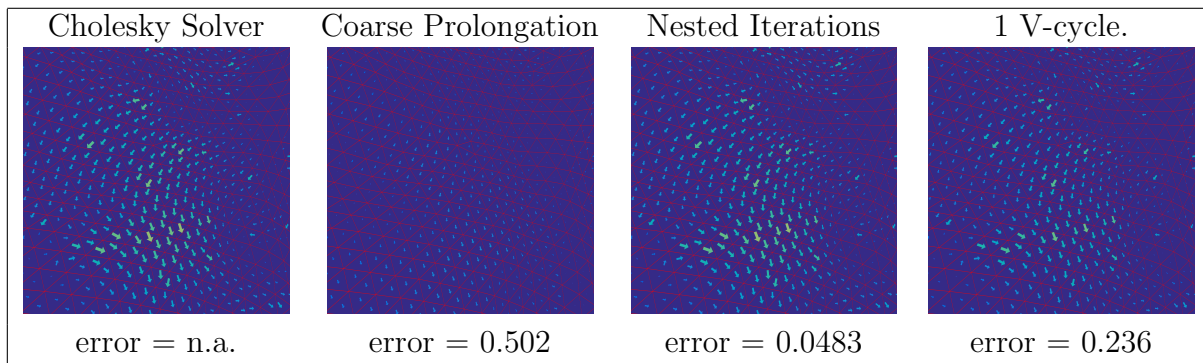


Table 7.4: Solutions for the face-based most harmonic field problem on the deformed bunny mesh.

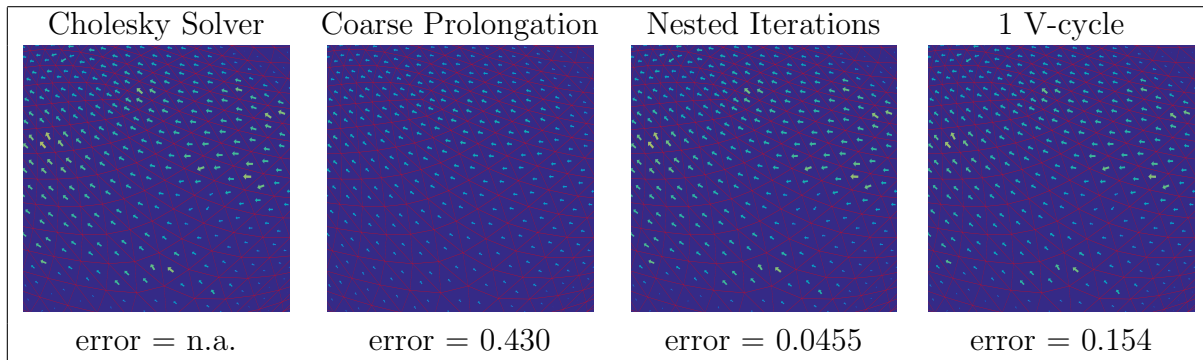
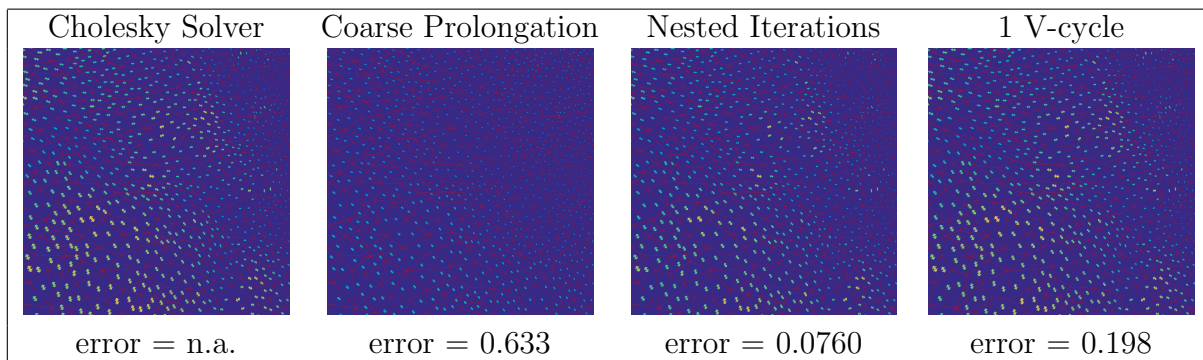
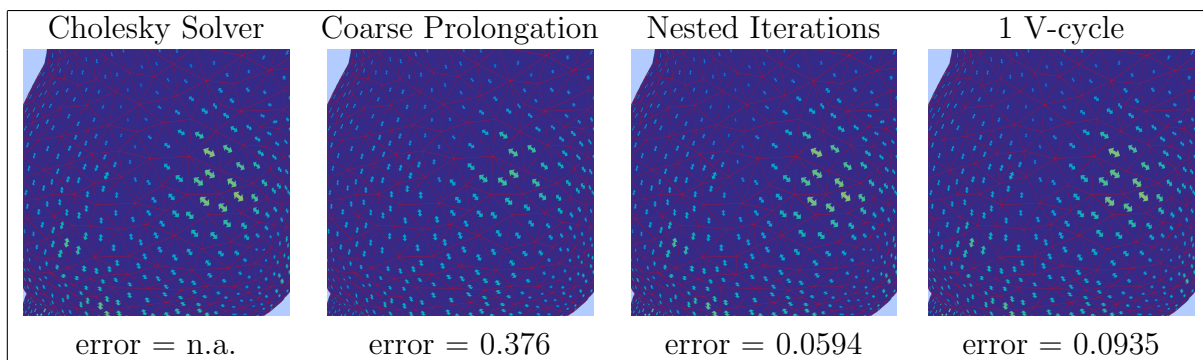


Table 7.5: Solutions for the face-based most harmonic field problem on the ball mesh.

Table 7.6: Solutions for the  $N = 2$  power field most harmonic field problem on the Ramses mesh.Table 7.7: Solutions for the  $N = 2$  power field most harmonic field problem on the small bunny mesh.

Mesh	CP Energy	CP Error	NI Energy	NI Error
Bunny Deformed	1.03	0.502	0.0178	0.276
Armadillo	1.02	0.415	0.0202	0.197
Santa	1.04	0.866	0.0438	0.783
Ramses	1.02	0.922	0.0460	0.809
Chair	1.02	0.849	0.0426	0.647
Owl	1.01	0.766	0.0315	0.560
Old Face	1.00	0.469	0.0169	0.218
Ball	0.917	0.485	0.0945	0.0693
Bunny Big	1.01	0.4743	0.0198	0.253
Bunny Small	0.941	0.500	0.0188	0.249
Lucy	1.02	0.956	0.0466	0.881
Lucy Remeshed	1.03	0.909	0.0398	0.826

Table 7.8: Coarse prolongation and nested iterations convergence of normalized L2 energy and normalized L2 error measures for the DEC most harmonic field problem.

Mesh	CP Energy	CP Error	NI Energy	NI Error
Bunny Deformed	0.972	0.502	0.00350	0.0483
Armadillo	0.958	0.675	0.00399	0.0494
Santa	0.960	0.637	0.00455	0.0454
Ramses	0.981	0.738	0.00489	0.0575
Chair	0.935	0.484	0.00627	0.0450
Owl	0.939	0.518	0.00489	0.0452
Old Face	failed	failed	failed	failed
Ball	0.905	0.430	0.00537	0.0455
Bunny Big	failed	failed	failed	failed
Bunny Small	0.909	0.446	0.00368	0.0433
Lucy	0.990	0.808	0.0109	0.101
Lucy Remeshed	0.903	0.431	0.00531	0.0386

Table 7.9: Coarse prolongation and nested iterations convergence of normalized L2 energy and normalized L2 error measures for the face-based most harmonic field problem. Our multigrid solvers can fail to solve face-based most harmonic field problems on certain meshes.

Mesh	CP Energy	CP Error	NI Energy	NI Error
Bunny Deformed	1.02	0.619	0.00648	0.0622
Armadillo	1.01	0.610	0.00563	0.0662
Santa	1.02	0.542	0.00776	0.0619
Ramses	0.998	0.633	0.00694	0.0760
Chair	0.993	0.498	0.00723	0.0656
Owl	1.01	0.419	0.00762	0.0496
Old Face	1.02	0.545	0.00617	0.0658
Ball	1.01	0.418	0.00669	0.0756
Bunny Big	1.02	0.583	0.00636	0.0607
Bunny Small	1.00	0.376	0.00586	0.0594
Lucy	0.998	0.525	0.00827	0.0682
Lucy Remeshed	1.01	0.686	0.00624	0.0663

Table 7.10: Coarse prolongation and nested iterations convergence of normalized L2 energy and normalized L2 error measures for the power field most harmonic field problem.

Mesh	CD Prepare	CD Solve	CP/NI Prepare	CP Solve	NI Solve
Bunny Deformed	12002ms	90ms	6106ms	24ms	1765mss
Armadillo	2269ms	42ms	3837ms	13ms	1114ms
Santa	1488ms	36ms	2557ms	10ms	953ms
Ramses	1186ms	24ms	2411ms	8ms	839ms
Chair	1611ms	25ms	2049ms	7ms	641ms
Owl	1167ms	20ms	1496ms	7ms	530ms
Old Face	854ms	15ms	941ms	5ms	336ms
Ball	560ms	9ms	938ms	4ms	201ms
Bunny Big	2412ms	30ms	2591ms	9ms	734ms
Bunny Small	59ms	2ms	255ms	2ms	68ms
Lucy	853ms	22ms	1924ms	8ms	739ms
Lucy Remeshed	1230ms	27ms	2192ms	8ms	714ms

Table 7.11: Prepare time and solve time for the DEC most harmonic field problem for our Cholesky solver, coarse prolongation solver, and nested iterations solver.



Mesh	CD Prepare	CD Solve	CP/NI Prepare	CP Solve	NI Solve
Bunny Deformed	80006ms	288ms	11465ms	36ms	2759ms
Armadillo	15858ms	130ms	6722ms	20ms	1748ms
Santa	11136ms	106ms	5079ms	17ms	1518ms
Ramses	8617ms	71ms	4236ms	13ms	1497ms
Chair	10995ms	73ms	3659ms	12ms	1078ms
Owl	9289ms	59ms	2897ms	10ms	726ms
Old Face	failed	failed	failed	failed	failed
Ball	4462ms	28ms	2088ms	6ms	341ms
Bunny Big	failed	failed	failed	failed	failed
Bunny Small	384ms	7ms	493ms	3ms	114ms
Lucy	6255ms	65ms	3488ms	11ms	1597ms
Lucy Remeshed	8679ms	76ms	3895ms	12ms	1080ms

Table 7.12: Prepare time and solve time for the face-based most harmonic field problem for our Cholesky solver, coarse prolongation solver, and nested iterations solver. Our multigrid solvers can fail to solve face-based most harmonic field problems on certain meshes.

Mesh	CD Prepare	CD Solve	CP/NI Prepare	CP Solve	NI Solve
Bunny Deformed	8420ms	77ms	7868ms	32ms	1773ms
Armadillo	1722ms	37ms	4871ms	19ms	1244ms
Santa	1124ms	30ms	2771ms	13ms	899ms
Ramses	773ms	20ms	3333ms	13ms	688ms
Chair	950ms	20ms	2100ms	10ms	601ms
Owl	780ms	16ms	1766ms	8ms	457ms
Old Face	632ms	12ms	1283ms	6ms	325ms
Ball	430ms	7ms	1207ms	5ms	178ms
Bunny Big	1708	26ms	4300ms	19ms	845ms
Bunny Small	46ms	2ms	345ms	2ms	68ms
Lucy	560ms	18ms	2394ms	10ms	688ms
Lucy Remeshed	784ms	21ms	2632ms	11ms	770ms

Table 7.13: Prepare time and solve time for the power field most harmonic field problem for our Cholesky solver, coarse prolongation solver, and nested iterations solver.

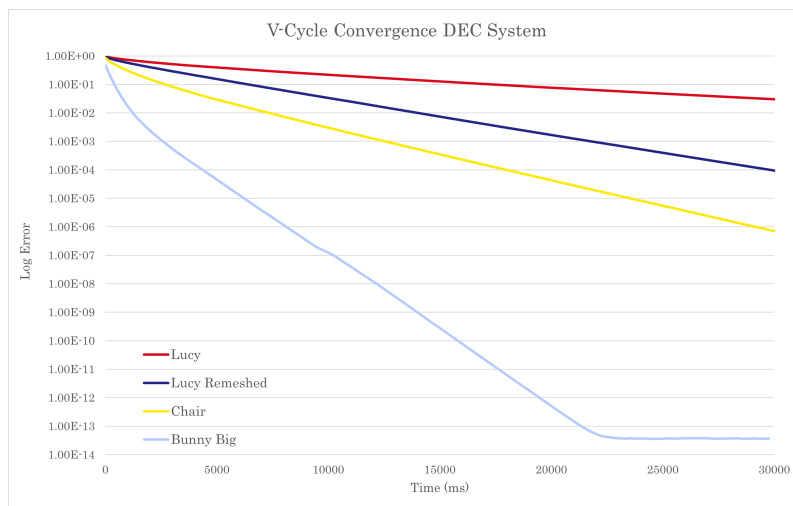


Figure 7.1: Convergence of the V-cycle solver for the DEC most harmonic field problem on the Lucy mesh, the remeshed Lucy mesh, the chair mesh, and the big bunny mesh.

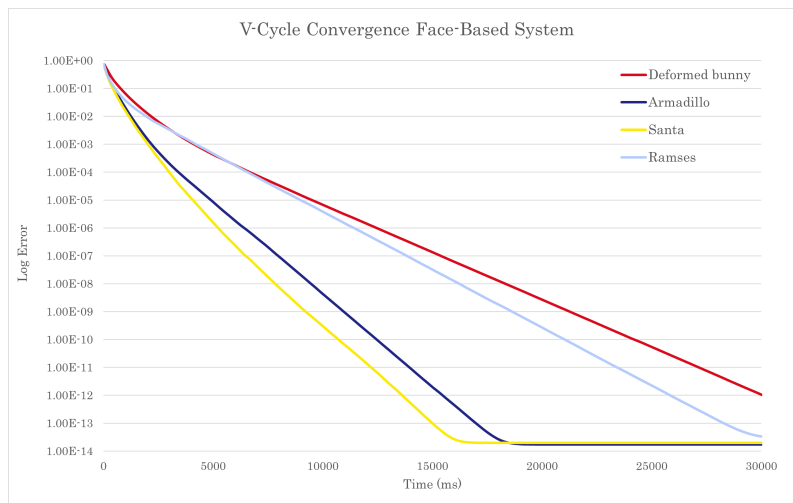


Figure 7.2: Convergence of the V-cycle solver for the face-based most harmonic field problem on the deformed bunny mesh, the armadillo mesh, the santa mesh, and the Ramses mesh.

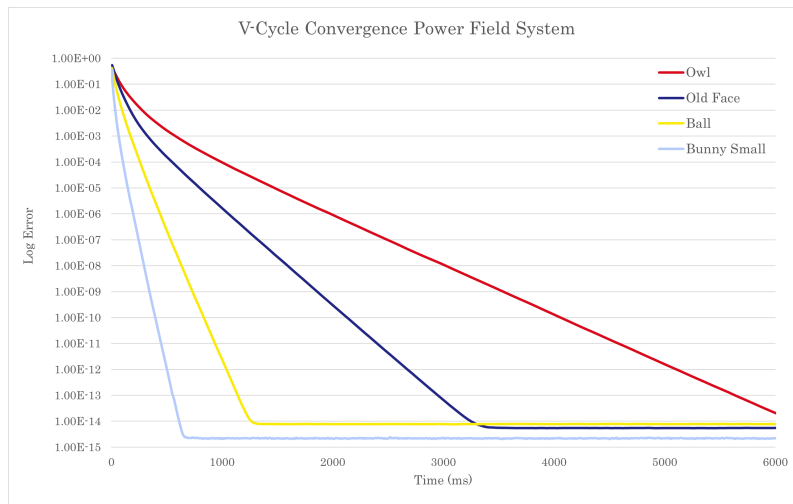


Figure 7.3: Convergence of the V-cycle solver for the power field most harmonic field problem on the owl mesh, the old face mesh, the ball mesh, and the small bunny mesh.

### 7.1.1 Prepare Time

The prepare time consists of the time necessary to prepare the solver with a system matrix. Recall that this is the time needed to factorize the system matrix for the Cholesky solver, and the time needed to coarsen the system matrix to all multigrid levels for the multigrid solvers. This also includes the time needed to factorize the coarsest system matrix for the multigrid solvers.

When looking at Tables 7.11, 7.12, and 7.13, it can be observed that our multigrid solvers are capable of taking up to 85.7% less prepare time than the Cholesky solver. This is a very significant improvement. However, in many cases multigrid is not able to reach this level of improvement. Moreover, in some scenarios multigrid performs significantly worse than Cholesky in terms of prepare time. From Figures 7.4, 7.5, and 7.6, it becomes clear that the performance difference between multigrid and Cholesky is strongly related to the size of the mesh. Plainly, the prepare time of the Cholesky solver scales as  $O(n^3)$  with the size of the system, whereas the prepare time of the multigrid solvers scale as  $O(n)$  with the size of the system. For large enough systems, multigrid will outperform Cholesky by an increasingly large margin. For smaller systems however, the Cholesky seems to easily outperform multigrid. This seems to be especially true for the DEC and power field most harmonic field systems.

Since our most harmonic systems are defined in sparse matrix form, the sparsity of the systems has a massive impact on the effective size of the system. In fact, the DEC most harmonic system matrix for the deformed bunny mesh has a size of  $418065 \times 418065$  and a sparsity ratio of 0.0026%. It is therefore reasonable to expect that other, more dense systems will require more computation time when solving with the Cholesky solver. The face-based most harmonic field system has a size of  $557420 \times 557420$  and a sparsity ratio of 0.0047%, and the power field most harmonic field system has a size of  $557420 \times 557420$  and a sparsity ratio of 0.0014%. When compared to Figures 7.4, 7.5, and 7.6, there is a strong correlation between effective system size and prepare time, for all solvers. So it is not unreasonable to note that working with most harmonic field problems was not the best choice for showing off our prolongation operators in conjunction with multigrid, as most harmonic field systems are very sparse. Testing out our techniques on denser systems will therefore be subject to future work.

Aside from system sparsity, it can also be noted that our multigrid prepare time is higher than it has to be because of the density of our prolongation operators. Since the prepare time of multigrid consist of coarsening the system matrix, the prepare time scales with the sparsity of our prolongation operators. We suspect that in the process of multiplying thousands of prolongation steps together, the final sparse prolongation operator gets filled with almost-zero entries. These do not have a significant impact on the quality of the prolongation operator, but they do significantly slow down multiplications with the operator. It is therefore a reasonable optimization to prune the final prolongation operators. Exactly where the cut-off point should be, what the effect on the quality of the operators will be, and what the effective performance gains will be are all subject to future work.

It should be noted however, that prepare time may not always be relevant. If the same system needs to be solved a large number of times, it can be worthwhile to have a higher prepare time if the solve time is lower. This can be true for simulations, where the same system on the same mesh needs to be solved every frame. Preparing the solver once, before the simulation starts, can take a long amount of time without negatively affecting the effective performance of the application. Of course, other applications exist where the system changes constantly. Here, the prepare time can be more important than the solve time, since it tends to be larger.

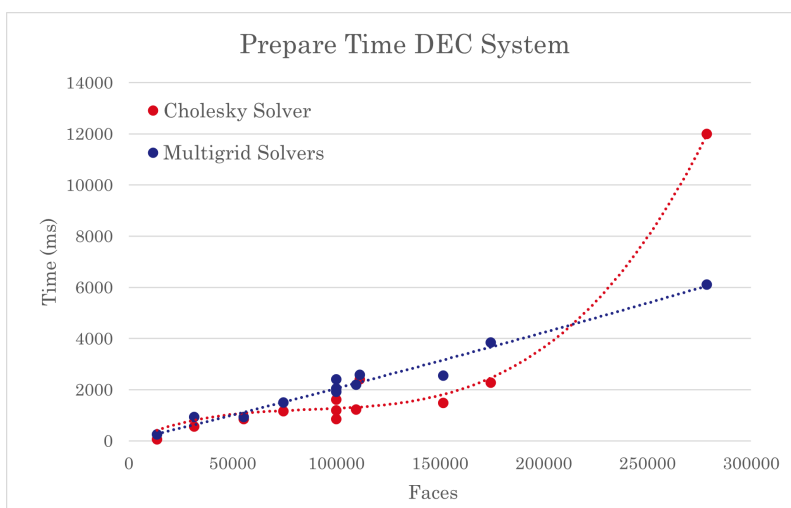


Figure 7.4: Cholesky and multigrid prepare time for the DEC most harmonic field problem.

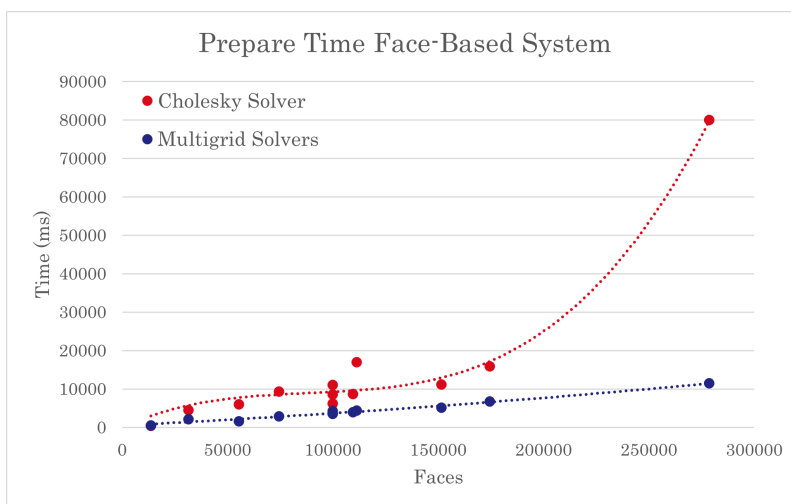


Figure 7.5: Cholesky and multigrid prepare time for the face-based most harmonic field problem.

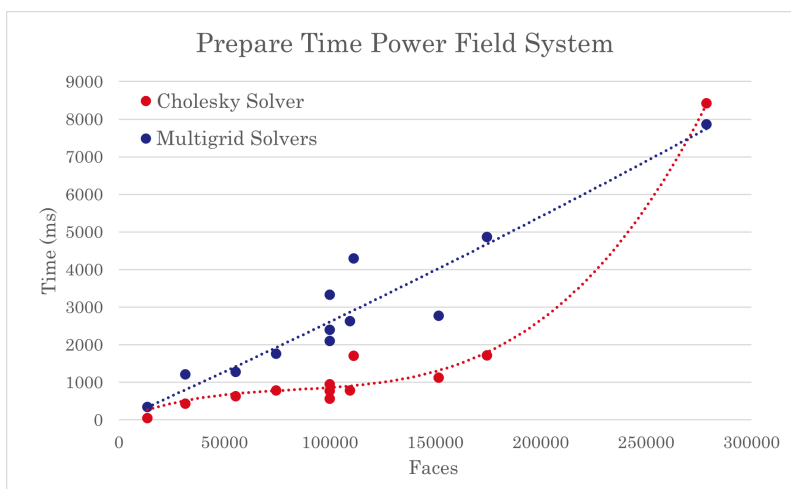


Figure 7.6: Cholesky and multigrid prepare time for the power field most harmonic field problem.

## 7.1.2 Solve Time

When looking at Tables 7.11, 7.12, and 7.13, it can be observed that our coarse prolongation operator is able to consistently outperform the Cholesky solver in terms of solve time. We can see a decrease in solve time of up to 87.5%. However, it can also be noted that our nested iterations solver is consistently slower than the Cholesky solver. We can see an increase in solve time of up to 131%.

From Figures 7.7, 7.8, and 7.9, it becomes clear that the reason for the high solve time of nested iterations is likely similar to that for the high prepare time. In short, Cholesky scales worse with the size of the system than nested iterations does, but the systems used are so sparse that nested iterations never gets the opportunity to outperform Cholesky. This is on top of the suspected high density of our prolongation operators, which also negatively affects the computation time of the coarse Gauss-Seidel iterations in the nested iterations solver.

Coarse prolongation performs much better than nested iterations and even Cholesky in terms of solve time. But from Tables 7.8, 7.9, and 7.10, it should be noted that coarse prolongation comes

with a much higher normalized L2 error and normalized L2 energy than nested iterations. This can be acceptable for some applications, but this is a trade-off.

The solve time of the V-cycle solver can be found in Figures 7.1, 7.2, and 7.3. Here, it can be noted that for the same normalized L2 error, V-cycles perform similarly to nested iterations in terms of solve time. The advantage of V-cycles is that one can reach errors orders of magnitude lower than with nested iterations if one is willing to let the solver run for longer. If one is content with relatively higher errors however, neither nested iterations nor V-cycles appears to be consistently the fastest. A more in-depth study into the performance of V-cycles and nested iterations will be left as future work.

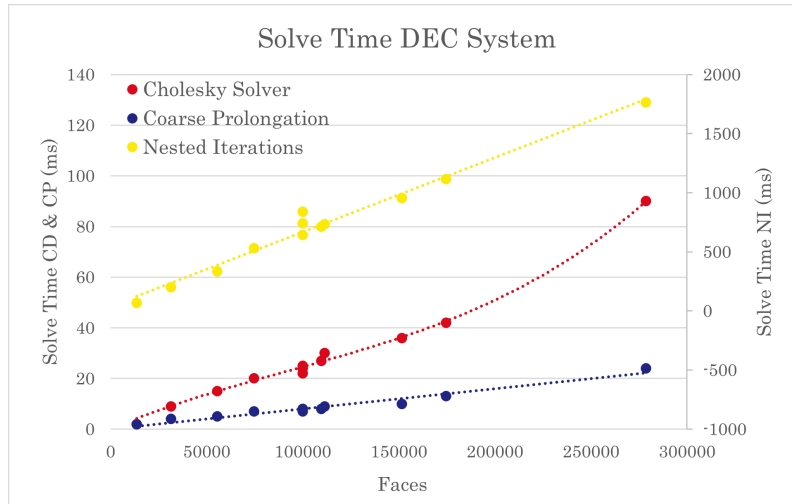


Figure 7.7: Cholesky, coarse prolongation, and nested iterations prepare time for the DEC most harmonic field problem.

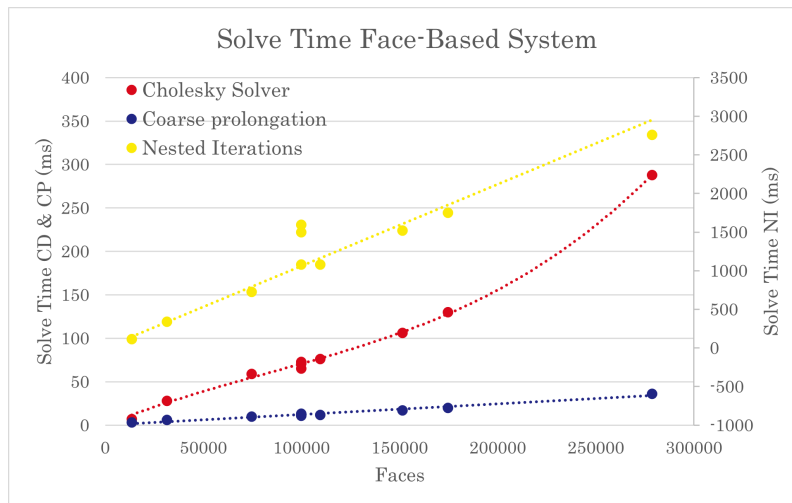


Figure 7.8: Cholesky, coarse prolongation, and nested iterations prepare time for the face-based most harmonic field problem.

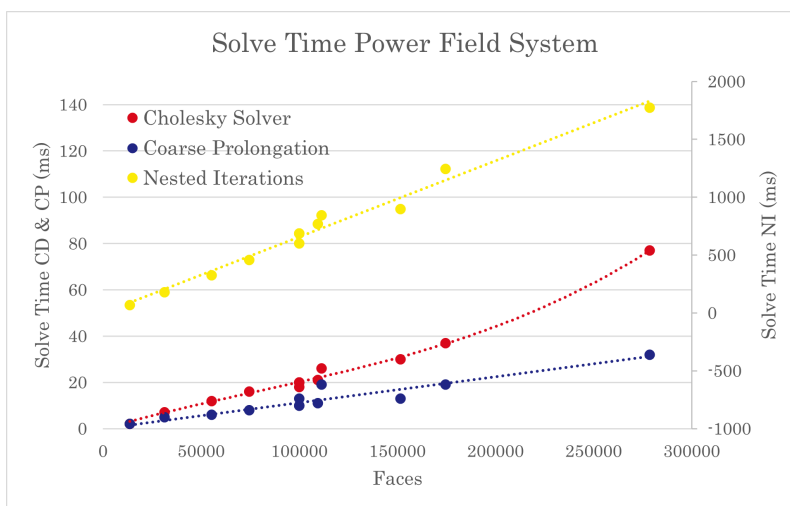


Figure 7.9: Cholesky, coarse prolongation, and nested iterations prepare time for the power field most harmonic field problem.

### 7.1.3 Convergence

From Tables 7.8, 7.9, and 7.10, it can be noted that nested iterations consistently outperforms coarse prolongation in terms of both energy and error convergence. As discussed in the previous section, this does come at the cost of solve time. As opposed to solve time, however, Figures 7.10 and 7.11 show that there is no correlation between convergence and the size of the mesh. This means that for a well behaved mesh, the convergence reached with a multigrid solver can be reasonably predicted beforehand. For example, the average normalized L2 energy of the DEC most harmonic field problem with the nested iterations solver is 0.0365, with a standard deviation of 0.0219.

When comparing V-cycle convergence from Figures 7.1, 7.2, and 7.3 to Tables 7.8, 7.9, and 7.10, it can clearly be observed that the V-cycle solver is capable of reaching much better convergence than coarse prolongation and nested iterations. As stated before however, this does come at the cost of solve time.

Our multigrid solvers can also be compared more practically through Tables 7.2, 7.3, 7.4, 7.5, 7.6 and 7.7. Here, it can be observed that for some meshed, coarse prolongation may provide a solution good enough for certain applications. Think of situations where only the general direction of the vector field matters. It can also be noted that nested iterations and a single V-cycle often provide solutions that look very similarly to each other. The extra time spent on nested iterations may therefore not always be worth it. Lastly, both nested iterations and V-cycles often produce solutions that look very similar to the solution provided by the Cholesky solver.

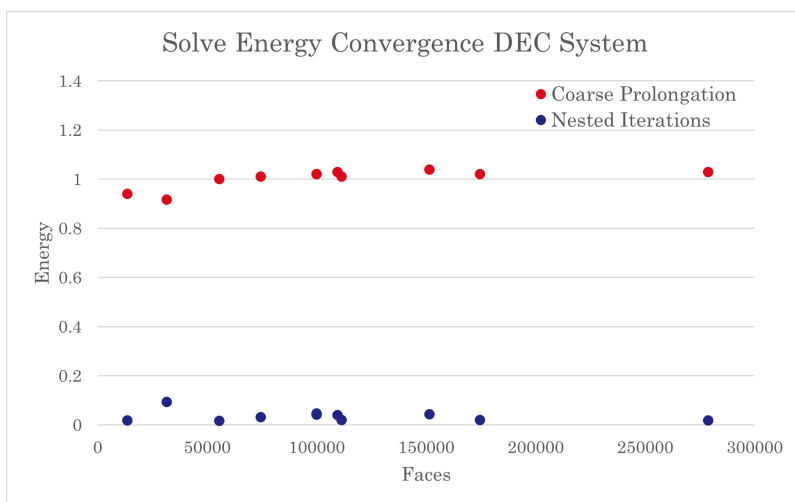


Figure 7.10: Normalized L2 energy convergence as a function of mesh size for the DEC most harmonic field problem.

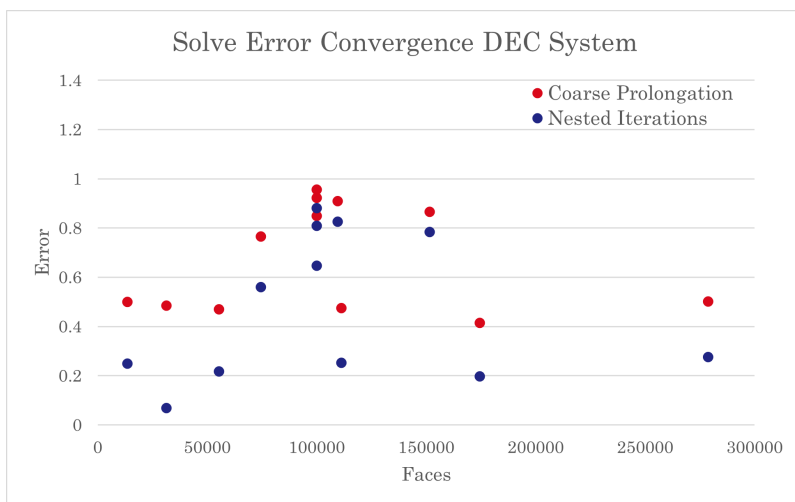


Figure 7.11: Normalized L2 error convergence as a function of mesh size for the DEC most harmonic field problem.

## 7.2 Solver settings

### 7.2.1 V-cycles

In Table 7.14, one can find the number of V-cycle iterations and the amount of time necessary to reach convergence for the DEC most harmonic field problem for the Deformed Bunny mesh. Here, we consider convergence reached when the normalized L2 error falls below 0.001. We used the deformed bunny mesh here because it is the largest we have, and so finding the optimal balance between convergence rate and solve time is most important here. From this, we can see that using 6 Gauss-Seidel iterations per V-cycle level is optimal in terms of computation time.

Similar results for the face-based most harmonic field problem and the power field most harmonic field problem can be found in Tables 7.15 and 7.16 respectively. From these it can be determined



Gauss-Seidel Steps	Convergence Steps	Time per Iteration	Convergence Time
1	148	137ms	20276ms
2	75	211ms	15825ms
3	50	285ms	14250ms
4	38	359ms	13642ms
5	31	433ms	13423ms
6	26	507ms	13182ms
7	23	580ms	13340ms
8	21	649ms	13629ms

Table 7.14: Convergence steps and time necessary to reach convergence as a function of Gauss-Seidel steps per V-cycle level for the DEC model problem on the Deformed Bunny mesh. Convergence is considered reached when the normalized L2 error drops below 0.001.

Gauss-Seidel Steps	Convergence Steps	Time per Iteration	Convergence Time
1	21	233ms	4893ms
2	12	375ms	4500ms
3	9	515ms	4635ms
4	8	634ms	5072ms
5	7	764ms	5348ms
6	6	881ms	5286ms
7	6	1024ms	6144ms
8	6	1130ms	6780ms

Table 7.15: Convergence steps and time necessary to reach convergence as a function of Gauss-Seidel steps per V-cycle level for the face-based field model problem on the Deformed Bunny mesh. Convergence is considered reached when the normalized L2 error drops below 0.001.

that using 2 Gauss-Seidel iterations per V-cycle level is optimal for both face-based fields and power fields. We have used these settings for all measurements done with V-cycles in this thesis.

## 7.2.2 Nested Iterations

The nested iterations solver can start from different multigrid levels. For example, it can start from multigrid level 2, and solve back to level  $L$  from there. If the lowest multigrid levels are too coarse, it can be advantageous to start the solver from a higher level. In some cases, the extra Gauss-Seidel steps are more expensive than a finer Cholesky solve. We therefore tested the optimal nested iterations starting level for this theses by measuring the prepare time and solve time for three different starting points. The results can be found in Table 7.17. Note that we once again chose to measure on the deformed bunny mesh, since optimal performance is most important for the largest systems. And since we fixed  $L = 3$ , we were only able to test starting levels 0, 1, and 2. From Table 7.17, it can be noted that prepare time increases greatly when starting from higher multigrid levels. It can also be noted that the solve time decreases slightly when starting from higher levels. The optimal starting level therefore depends on the use case. If a singular system needs to be solved several times, it can be optimal to start from level 1 or 2 in terms of total time spent. If a system only needs to be solved once however, it is optimal to start from level 0. We therefore start from level 0 for all measurements involving nested iteration throughout this thesis.

Gauss-Seidel Steps	Convergence Steps	Time per Iteration	Convergence Time
1	16	168ms	2688ms
2	9	259ms	2331ms
3	7	349ms	2443ms
4	6	438ms	2628ms
5	5	528ms	2640ms
6	5	613ms	3065ms
7	5	691ms	3455ms
8	5	785ms	3925ms

Table 7.16: Convergence steps and time necessary to reach convergence as a function of Gauss-Seidel steps per V-cycle level for the power field model problem on the Deformed Bunny mesh. Convergence is considered reached when the normalized L2 error drops below 0.001.

Starting Level	Error	Prepare Time	Solve Time
0	0.0633	8209ms	1741ms
1	0.0633	10131ms	1579ms
2	0.0633	34252ms	1200ms

Table 7.17: The effect of the starting level of nested iterations on the normalized L2 error, prepare time, and solve time. Measurements were made with the most harmonic model problem for power fields on the Deformed Bunny mesh.

### 7.3 Precomputation time

In Table 7.18, one can find the precomputation time necessary to generate our prolongation operators  $S_1$ ,  $S_\chi$ , and  $S_P$ . Note that all three are generated in this time, so this number can be reduced if only one or two of these operators are required. Also note that these operators only need to be calculated once per mesh. Once generated, the operators can be stored and reused whenever necessary.

Despite this, Figure 7.12 shows that the precomputation time scales quadratically with the size of the mesh. As a result, calculating the prolongation operators for exceptionally large meshes ( $> 1,000,000$  faces) can take an unreasonable amount of time. For example, generating the prolongation operators for the deformed bunny mesh took over 2.5 hours.

In principle, our algorithm scales linearly with the size of the mesh. However, it is likely the process of multiplying the sublevels of the operators together into a multigrid level that scale quadratically, as the operators become increasingly more dense over time. We believe that this problem can be mostly resolved by building the sublevels of our prolongation operators such that multiple sublevels can be prolonged at once. This can significantly reduce the number of matrix multiplications, therefore reducing the effect of the quadratic scaling. Implementing this improvement and testing it will be subject to future work.

### 7.4 Input mesh quality

Aside from the size of the mesh, solve time may also depend on the quality of the mesh. We have therefore solved the same system with our V-cycle solver on both the large bunny and the small bunny mesh. See Table 7.19 for the difference in resolution between these meshes. The results

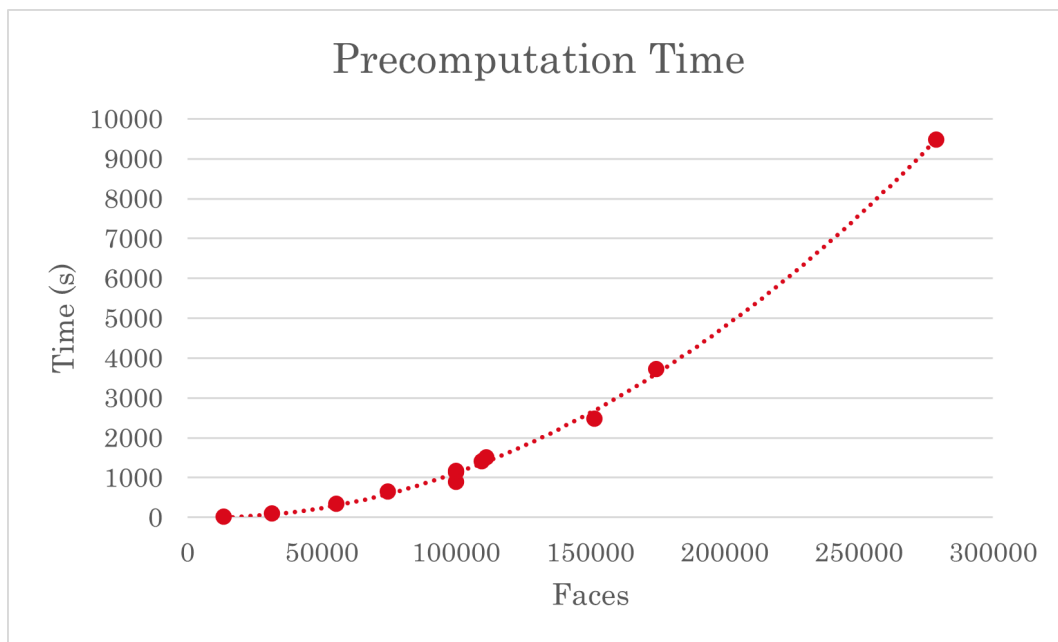


Figure 7.12: Precomputation time of  $S_1$ ,  $S_x$ , and  $S_P$  as a function of the size of the mesh.

Mesh	Faces	Time
Bunny Deformed	278710	9473576ms
Armadillo	174568	3717574ms
Santa	151558	2474853ms
Ramses	100000	1133211ms
Chair	99968	895905ms
Owl	74538	649464ms
Old Face	55448	343663ms
Ball	31438	101058ms
Bunny Big	111364	1500470ms
Bunny Small	13584	24528ms
Lucy	100000	1168947ms
Lucy Remeshed	109592	1405326ms

Table 7.18: Precomputation time of our prolongation operators for all meshes.

can be found in Figures 7.13 and 7.14. In terms of time, the small bunny mesh fully converges approximately 800ms, whereas the large bunny mesh takes more than 8000ms. This is to be expected since the large bunny mesh has a much larger system matrix. In terms of iterations however, the small bunny mesh also converges faster. This time by 10 iterations or 14%. This can be attributed to the lower resolution of the small bunny mesh. Since this mesh has a lower resolution, the resolution of the coarsened mesh is much closer to that of the original. This means the prolongation operator can more accurately transform between the coarse and the fine mesh. As a result, convergence is reached 14% faster.

This does not mean that smaller meshes are optimal, however. Since smaller meshes can be solved relatively quickly by a Cholesky solver, it is not necessarily worth while to use our multigrid method here. However, it would be interesting to see by how much the quality of our prolongation operators deteriorates as the mesh resolution gets finer. This is subject to future work.

The effect of sampling resolution can be described more clearly by comparing two meshes of approximately equal size, with a different sampling method. In Table 7.20, the original and remeshed Lucy meshes are compared side-by-side. The original Lucy mesh has many thin and long faces that accurately describe the horizontal curvature in the dress. In the remeshed version of Lucy however, the shape has been sampled more evenly, and the faces more closely resemble equilateral triangles. When comparing the convergence of these meshes in terms of time and iterations in Figures 7.15 and 7.16, it can be concluded that the remeshed version of Lucy converges much quicker than the original. The remeshed version of Lucy reaches a normalized L2 error of below 0.01 in approximately 15 seconds, while the original version of Lucy takes more than double this time. The coarse prolongation and nested iterations solvers also frequently reach a lower normalized L2 error and normalized L2 energy for the remeshed Lucy mesh than for the original Lucy mesh. In terms of solve time, nested iterations also generally performs better for the remeshed Lucy than for the original. The coarse prolongation solver however, does not solve significantly faster for either of the two meshes, and the Cholesky solver appears to solve faster for the original Lucy mesh. When looking at the prepare time, the original Lucy is always faster than the remeshed version. The same goes for the precomputation time. This would suggest that the prolongation operators for the original Lucy mesh are more sparse than those for the remeshed Lucy mesh.

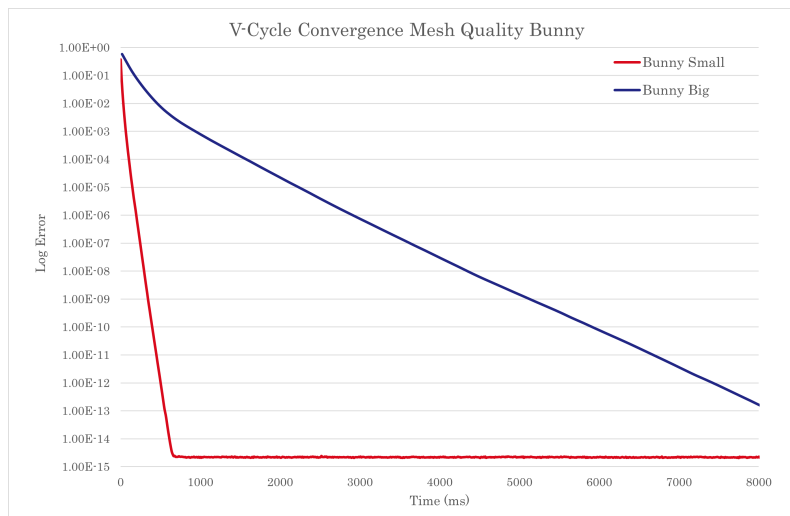


Figure 7.13: Convergence of the V-cycle solver as a function of time of the DEC most harmonic field problem on the large and small bunny meshes.

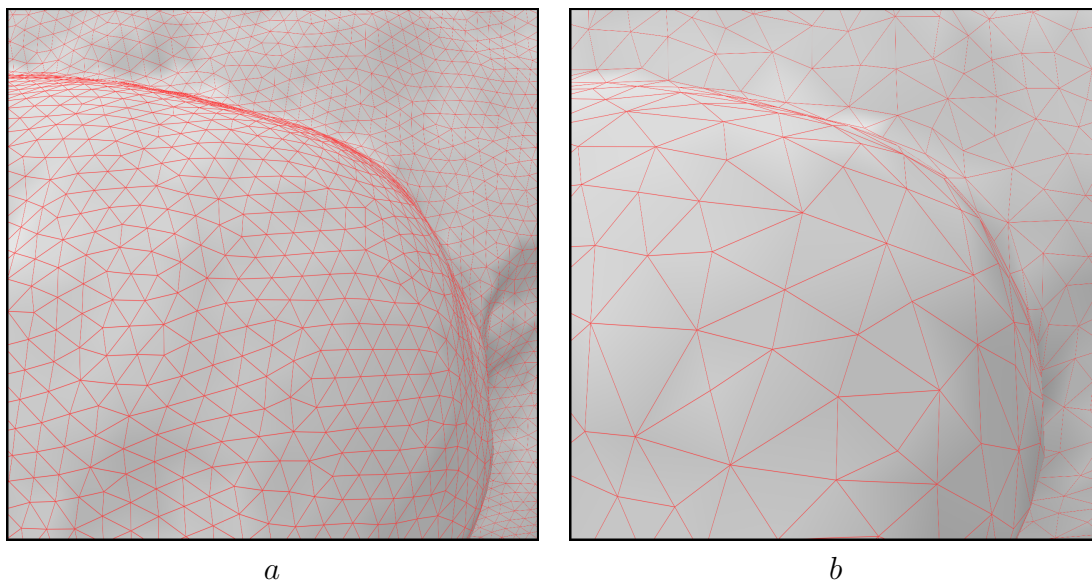


Table 7.19: Difference in resolution between the big (*a*) and small (*b*) bunny meshes.

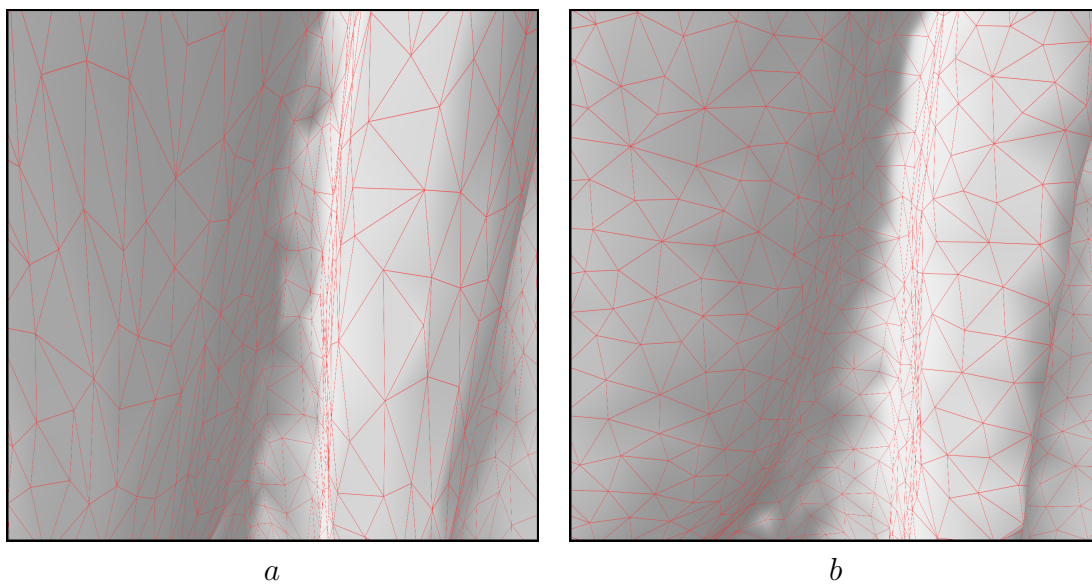


Table 7.20: Difference in sampling between the original (*a*) and remeshed (*b*) Lucy meshes.

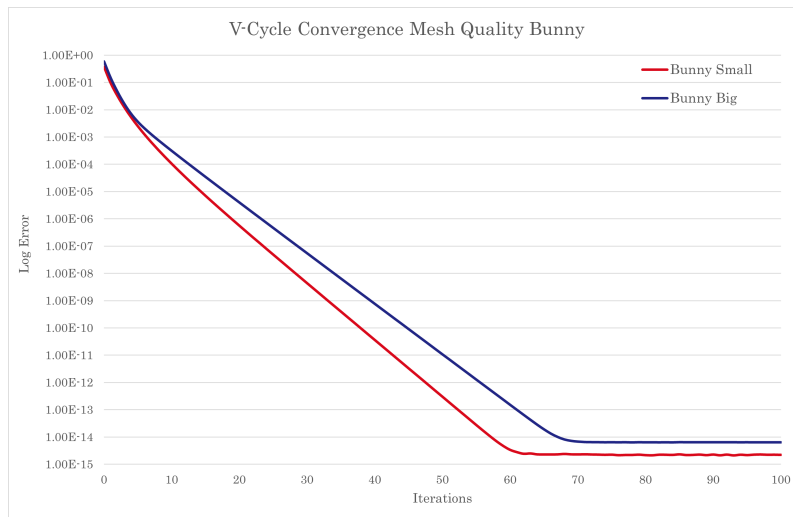


Figure 7.14: Convergence of the V-cycle solver as a function of iterations of the DEC most harmonic field problem on the large and small bunny meshes.

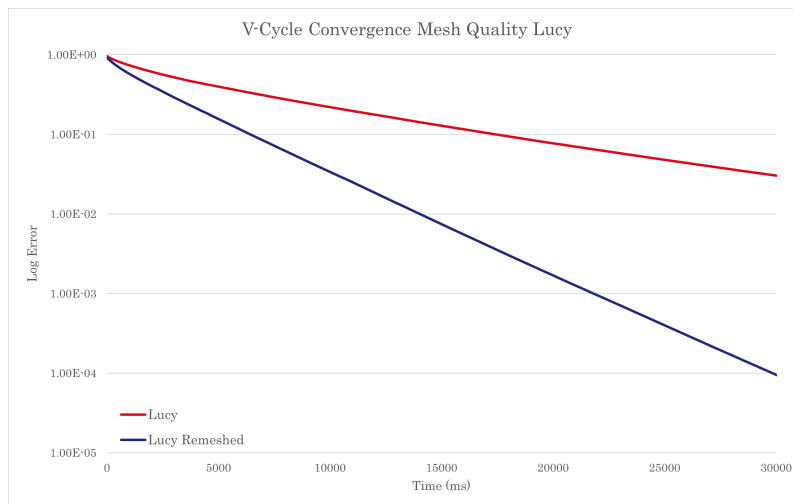


Figure 7.15: Convergence of the V-cycle solver as a function of time of the face-based most harmonic field problem on the normal and remeshed Lucy meshes.

## 7.5 Scalar Field Prolongation

The scalar field prolongation operator introduced by Liu et al. (2021) can solve a smoothing application (similar to our most harmonic field problem) on a mesh with over 400K vertices in less than 0.3 seconds using V-cycles. Our best comparison would be the face-based most harmonic field problem solved for the deformed bunny mesh in Figure 7.2. Even with a high error threshold like 0.01, our  $S_\chi$  operator with the V-cycle solver takes approximately 4 seconds to converge.

In terms of precomputation time, Liu et al. were able to create their  $S_0$  prolongation operator for a mesh with over 600K vertices in only 50 seconds. Although it is true that this is orders of magnitude faster than our method, the scalar field prolongation operator does not require a 1-ring overlay to compute. In fact, it only requires a few barycentric coordinates per 1-ring. This saves a

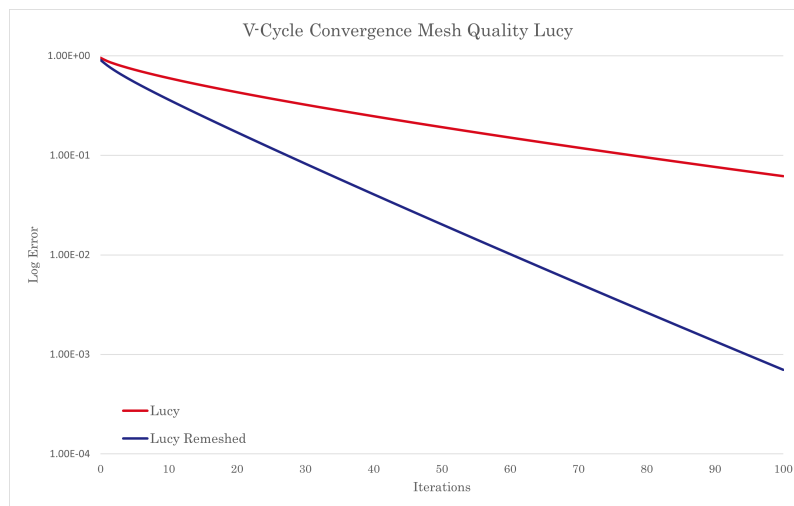


Figure 7.16: Convergence of the V-cycle solver as a function of iterations of the face-based most harmonic field problem on the normal and remeshed Lucy meshes.

lot of time, since the computation of the overlay mesh takes a lot of time. It might be possible to use a less robust edge-face intersection algorithm for the construction of our prolongation operators, but this is subject to future work. Regardless, our precomputation algorithm scales quadratically, while theirs scales linearly. This would be a another great point of improvement.

# Chapter 8

## Discussion

### 8.1 Conclusion

In this thesis, we have introduced three new, structure preserving prolongations operators for solving directional field problems. One for DEC vector fields, one for face-based fields, and one for power fields. Using these operators, multigrid solvers like coarse prolongation, nested iterations, and V-cycles are able to reach a sufficient degree of convergence in linear time. So although precomputation times are large, and although linear least squares combined with Cholesky decomposition consistently outperforms multigrid for smaller meshes and sparser systems, multigrid combined with our prolongation operators is poised to outperform linear least squares by a significant margin when solving for larger meshes.

### 8.2 Future Work

In this thesis, we have introduced three new prolongations operators for solving directional field problems. As these operators are state of the art, there is much that can still be improved upon. Most important of which is the density of the prolongation operators. As mentioned before, we suspect that our prolongation operators are much denser than they need to be. As a consequence, our multigrid solvers are also much slower than necessary. It is therefore of importance that the extent of this problem is researched, and a solution is found. Simply pruning the operators is a possibility we have thought of, but it is as of yet unclear to us where the pruning threshold should be, how this affects the quality of the operators, and how this affects the solve time of multigrid. Secondly, our  $S_\chi$  operator occasionally introduces zeroes on the diagonal of the coarse system matrix. This causes the Cholesky solver to fail on these operators. As a result, multigrid is not able to solve face-based field problems on some meshes. We are not sure why  $S_\chi$  fails for some meshes. Perhaps this operator is particularly sensitive to poorly decimated meshes, since  $S_\chi$  failed more often when using the default Qslim method. Another possibility is that we might have made a mistake in our edge collapse code. This should be investigated.

Another important factor is our precomputation time. We believe this can be vastly reduced. As of now, our precomputation time scales quadratically with the size of the mesh. However, the precomputation time of the scalar field prolongation operator introduced by Liu et al. (2021) scales linearly with the size of the mesh. We believe linear scaling should be possible for our operators as well. Currently, we calculate the global prolongation operators of every edge collapse, and we multiply all of these operators together. This introduces quadratic scaling to the precomputation



algorithm, since the multiplied matrices get denser at every step. By combining several independent local prolongation operators together directly into a single global prolongation operator, many multiplications can be prevented.

Furthermore, we believe the construction time for the local prolongation operators can be vastly reduced by changing our overlay algorithm. Currently, we use a robust overlay algorithm that employs rational numbers to create exact overlay-to-coarse and overlay-to-fine maps. For our purposes however, this may be more complex than necessary. We expect our local prolongation operators can be constructed by much simpler floating point fine-edge-to-coarse-face maps.

In this thesis, we have exclusively worked with closed meshes. In principle however, our prolongation operators should be able to work with boundary meshes as well. Whether this is true, and how boundary meshes affect the quality of our operators would be very useful to inspect.

Lastly, we have also exclusively used power fields with the power  $N = 2$ .  $S_P$  has to be built separately for every power. So it would be useful to find out how the power of  $S_P$  affects the quality of the operator, and how the power of the field affects the multigrid convergence and solve time. It would be interesting to know whether our method works better for lower power fields or for higher power fields.

# Chapter 9

## Acknowledgements

We thank Dr. Amir Vaxman (Utrecht University) for providing guidance throughout the thesis, and for providing the ground work for several theory sections. We also like to thank Anna Shtengel for writing the code for the face-based and power field most harmonic field problems, and for gathering the meshes used in the thesis. Finally, we like to thank Bente Maria van Son for proofreading and helping with the formatting of the graphs.

# Bibliography

- Mark Adams. 2002. Evaluation of three unstructured multigrid methods on 3D finite element problems in solid mechanics. *Internat. J. Numer. Methods Engrg.* 55, 5 (2002), 519–534.
- Achi Brandt. 1977. Multi-Level Adaptive Solutions to Boundary-Value Problems. *Math. Comp.* 31, 138 (1977), 333–390, 58 pages.
- Susanne C. Brenner and L. Ridgway Scott. 1994. *The Mathematical Theory of Finite Element Methods*. Springer, New York, NY, USA.
- Kazem Cheshmi, Danny M. Kaufman, Shoaib kamil, and Maryam Mahri Dehnavi. 2020. NASOQ: numerically accurate sparsity-oriented QP solver. *ACM Trans. Graph.* 39, 4 (2020), Article 96, 17 pages.
- Keenan Crane, Fernando de Goes, Mathieu Desbrun, and Peter Schröder. 2013. *Digital geometry processing with discrete exterior calculus*. Association for Computing Machinery, New York, NY, United States.
- Bram Custers and Amir Vaxman. 2020. Subdivision Directional Fields. *ACM Trans. Graph.* 1, 1 (2020), 20 pages.
- Fernando de Goes, Mathieu Desbrun, Mark Meyer, and Tony DeRose. 2016b. Subdivision Exterior Calculus for Geometry Processing. *ACM Trans. Graph.* 35, 4 (2016), Article 133, 11 pages.
- Fernando de Goes, Mathieu Desbrun, and Yiyong Tong. 2016a. Vector Field Processing on Triangle Meshes. *ACM SIGGRAPH* (2016).
- Mathieu Desbrun, Eva Kanso, and Yiyong Tong. 2008. *Discrete Differential Forms for Computational Modeling*. Birkhäuser Basel, Basel, 287–324.
- Wilfried Enkelmann. 1988. Investigations of multigrid algorithms for the estimation of optical flow fields in image sequences. *Computer Vision, Graphics, and Image Processing* 43, 2 (1988), 150–177.
- Philipp Herholz and Olga Sorkine-Hornung. 2020. Sparse cholesky updates for interactive mesh parameterization. *ACM Trans. Graph.* 39, 6 (2020), Article 202, 14 pages.
- Nicholas J. Higham. 1990. *Analysis of the Cholesky Decomposition of a Semi-Definite Matrix*. Oxford University Press, Oxford, UK.
- R. Hiptmair. 1999. Multigrid Method for Maxwell’s Equations. *SIAM J. Numer. Anal.* 36, 1 (1999), 204–225.

- Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. 2013. Globally Optimal Direction Fields. *ACM Trans. Graph.* 32, 4, Article 59 (jul 2013), 10 pages. <https://doi.org/10.1145/2461912.2462005>
- Hsueh-Ti Derek Liu, Jiayi Eris Zhang, Mirela Ben-Chen, and Alec Jacobson. 2021. Surface Multigrid via Intrinsic Prolongation. *ACM Trans. Graph.* 40, 4 (2021), Article 80, 13 pages.
- Dimitri Mavriplis and Antony Jameson. 1987. Multigrid solution of the Euler equations on unstructured and adaptive meshes. (08 1987).
- Dimitri J. Mavriplis. 1995. Multigrid Techniques for Unstructured Meshes.
- Paul S. Heckbert Micheal Garland. 1997. Surface Simplification Using Quadric Error Metrics. (1997).
- J. Moulton, Colin Fox, and Daniil Svyatskiy. 2008. Multilevel approximations in sample-based inversion from the Dirichlet-to-Neumann map. *Journal of Physics: Conference Series* 124 (08 2008), 012035. <https://doi.org/10.1088/1742-6596/124/1/012035>
- Amir Vaxman, Marcel Campen, Olga Diamanti, Daniele Panozzo, David Bommes, Klaus Hildebrandt, and Mirela Ben-Chen. 2016. Directional Field Synthesis, Design, and Processing. *Computer Graphics Forum* 35, 2 (2016).
- Hassler Whitney. 1957. *Geometric Integration Theory*. Princeton University Press, Princeton, NJ.