DOI: 10.1111/cgf.70159

COMPUTER GRAPHICS *forum*
Volume 0 (2025), number 0, e70159

Computing Skinning Weights via Convex Duality

J. Solomon[1] and O. Stein[2]

[1]Massachusetts Institute of Technology, Cambridge, USA
jsolomon@mit.edu
[2]University of Southern California, Los Angeles, USA
ostein@usc.edu

Abstract

We study the problem of optimising for skinning weights through the lens of convex duality. In particular, we show that the popular bounded biharmonic weight *(BBW) model for skinning is dual to a non-negative least-squares problem, which is amenable to efficient solution via iterative algorithms; the final weights are then recoverable via a closed-form expression. Our formulation maintains convexity and is provably equivalent to the original problem. We also provide theoretical discussion giving intuition for the dual problem in the smooth case. Our final algorithm, which can be implemented in a few lines of code, achieves efficient convergence times relative to generic quadratic programming tools applied to the primal problem, without nonconvex formulations, relaxations or specialised optimisation techniques.*

Keywords: animation, bounded biharmonic weight, skinning

CCS Concepts: • Computing methodologies → Shape analysis; • Theory of computation → Convex optimisation;

1. Introduction

Skinning is a central problem in geometry processing. Given a geometric domain (e.g., a region in the plane, a volume or a surface), skinning tools extract a set of non-negative weighting functions used to interpolate values from a few isolated points or regions to the rest of the domain smoothly. Computation of skinning weights is a key step in practical algorithms for deformation and animation, used to infer motion of a shape based on the motion of a few isolated parts of the domain articulated by an artist.

While most 3D modelling software includes an interface for artists to paint on skinning weights, the inefficiency of this process motivates a class of algorithms for automatic optimisation of skinning weights. These methods start with a discretisation of the domain—typically a triangle or tetrahedral mesh—as well as the locations of the handles and other optional guidance and solve a numerical problem to extract smooth weighting functions.

Automatic skinning weights vary in complexity and reliability. While a few nascent data-driven approaches attempt to infer weights from previous data, the vast majority of algorithms are *model-driven*, solving an optimisation problem for weights that are smooth, non-negative and artefact-free. The more terms and constraints added to the model, the better the weights might be in principle, but also the harder they are to compute. For example, simple harmonic weights [JMD*07] can be extracted using a quickly solved linear system of equations, but suffer from non-differentiability at the handles and hence undesirable artefacts in skinned deformations. Other models introduce inequality constraints, nonconvexity and auxiliary variables to improve quality at the cost of computation time. For example, non-negativity constraints ensure that if a handle is dragged in one direction, some points on the domain will not move in the opposite direction.

Among the many available models for automatic skinning weight computation, BBW [JBPS11] and variants are widely considered to be a gold standard for quality. The BBW formulation, however, is a convex quadratic programme with a large number of objective terms and constraints. Although convexity means a global optimum can be extracted given enough computation time, the cost of solving the full BBW problem is prohibitive in many applications, motivating approximations that drop constraints to accelerate computation (see Appendix A); implementations of BBW often completely forgo the partition of unity constraint and enforce it a posteriori as an approximation. One technique [WS21] introduces a faster means of optimising the biharmonic energy on the restricted class of weights that are harmonic with respect to an unknown (and possibly singular) metric, but it has to resort to a nonconvex formulation with a

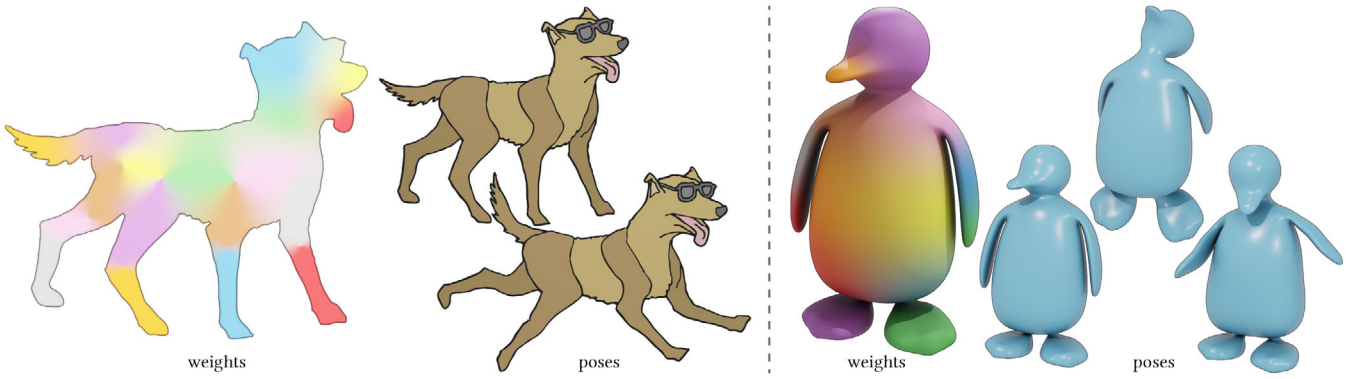

Figure 1: *Our method computes skinning weights by optimizing the bounded biharmonic energy over triangles (2D,* left*) or tetrahedra (3D,* right*), which can be used to animate characters. We visualise each skinning weight with a different colour. Unlike the original bounded biharmonic weight (BBW) of Jacobson et al. [JBPS11], we do not need an expensive generic large-scale constrained quadratic program solver.*

custom optimisation algorithm and a variety of parameters to achieve efficiency while avoiding local optima.

In this paper, we use convex duality to motivate an alternative approach to computing BBW. In particular, we show that BBW's dual is essentially a *non-negative least-squares* problem, which can be solved using simple iterative techniques. After optimising the dual, the skinning weights can be recovered using a few matrix multiplications. In contrast with previous work on accelerating BBW, our model is easily implemented in a few lines of code while achieving orders-of-magnitude acceleration over the original quadratic programming formulation. The resulting skinning weights (Figure 1) necessarily match the ones computed by BBW [JBPS11].

2. Related Work

While a vast literature in graphics and geometry processing provides many variations on skinning, deformation, and related problems, here we mention a few models directly linked to our work.

2.1. Skinning weights

A variety of models tackle the problem of automatically computing skinning weights. Here, we mention a few alternatives for computing *geometric skinning weights*, wherein weights are computed using an optimisation problem on a piece of geometry rather than, for example, learning weight functions from examples. We refer the reader to [WS21] for a broader discussion of alternatives and applications.

The simplest geometric weights are computed by solving a set of linear harmonic problems [JMD*07]. These weights are efficient to extract but become non-differentiable near the handles; they are ill-defined for point handles before discretisation. Diffusion-based weights [BP07] have similar drawbacks. Within the class of weights based on linear PDE, additional regularity can be added by using higher-order operators, for example, by solving the biharmonic [BK04] or even triharmonic [Tos08, JTSZ10] equations. Weights from these methods are relatively fast to compute but can be oscillatory or negative since the maximum principle is lost.

A landmark in automatic skinning, the BBW method [JBPS11] attempts to benefit from the regularity of higher-order operators while avoiding their oscillatory drawbacks. The BBW model minimises the weak form of the Bilaplace equation—that is, the integral of the Laplacian of the unknown weights squared pointwise—subject to inequality constraints that make the weights non-negative and linear constraints to make them sum to 1. The result is a convex quadratic program that couples together the unknown weight functions in a complex fashion that is slow to resolve; see Section 3 for mathematical discussion. Practical implementations of BBW drop constraints or use other approximations to accelerate computation. On the other hand, [JWS12] *adds* constraints to the BBW problem to avoid spurious local extrema in the optimised weights, at the cost of additional computation.

Most recently, [WS21] revisit the BBW problem with the goal of making it faster to optimise. They propose optimising the biharmonic energy in the space of weights that are harmonic with respect to an unknown metric, represented using one positive definite matrix per triangle. Using a custom parameterisation of the set of positive definite matrices, they solve their problem using unconstrained first-order optimisation. Their method requires custom parameterisation of the unknown tensors, uses a nonconvex model and includes several heuristics to avoid local optima. The authors conjecture that their space of quasi-harmonic weights includes the optimum of the BBW problem but are unable to prove this hypothesis for problems with more than two handles.

2.2. Convex duality

A rich and elegant theory of *duality* motivates many theoretical results and algorithms for constrained optimisation. Very roughly, duality suggests that members of a large class of optimisation problems admit 'dual' problems for different sets of variables whose optimal objective values bound one another and, under certain conditions, agree. In lucky circumstances, closed-form expressions can be used to find the optimal value of each primal variable given the solution to the dual problem.

The best-known examples of duality are articulated for linear programming, but more general convex programs can admit similar structures. While we do not provide a comprehensive introduction to the general theory here, we refer to [BV04, Roc70] for broad discussion. In Section 4, we use a minimax approach to deriving dual problems to derive an elegant alternative to solving the BBW problem; additional details are provided in that section.

2.3. Non-negative least-squares

The key result in this paper shows that BBW is equivalent in the dual to minimising an inhomogeneous convex quadratic form with a non-negativity constraint; that is, abstractly we can understand our problem as minimising a function of the form $x \mapsto \frac{1}{2}x^\top Ax - x^\top b$ subject to $x \geq 0$, where A is a positive semidefinite matrix. The best-known version of this problem is *non-negative least-squares* (NNLS), where an objective of the form $\frac{1}{2}\|Mx - d\|_2^2$ is equivalent to the former by expanding the square. When A is invertible, we can also write $\frac{1}{2}x^\top Ax - x^\top b = \frac{1}{2}\|L^\top x - L^{-1}b\|_2^2 + \text{const.}$, where we factor $A = LL^\top$, providing an equivalence between the problems. Indeed, many NNLS algorithms in reality only need access to $A = M^\top M$, making them directly applicable in our setting.

Given its connection to classical least squares, NNLS appears in a broad variety of statistical problems, motivating a long history of efficient NNLS optimisation algorithms. The best-known approach applies the active set method [LH74], iteratively solving least squares using submatrices of A corresponding to active constraints. This approach relies on efficient solution of linear subproblems, making it difficult to apply in large-scale settings. Interior point methods [BMM06] and the alternating direction method of multipliers (ADMM) [BPC*11] also can be applied to NNLS but often require linear solves. Since our A is defined by solving a sparse linear problem on a mesh, the most relevant class of algorithms to our setting relies only on multiplication by A, such as projected gradient descent [Pol15], non-monotonic descent [KSD13], exponentiated gradient descent [LS00], and unconstrained gradient descent after exponentiating the unknown [CMV22]. We also note multiplicative update methods like [SSL02], which rely on splitting A into a difference of non-negative matrices $A = A^+ - A^-$; these methods are not relevant to our setting since we do not have access to individual elements of A.

Our experiments in Section 9 use the approach of Chou et al. [CMV22], wherein the argument to the NNLS problem is replaced with the square of a generic argument, reducing NNLS to an unconstrained optimisation problem. They argue that this nonconvex method reaches a global optimum, specifically for NNLS.

3. Preliminaries and Notation

We begin by recalling the bounded biharmonic weights (BBW) model to establish notation used for our subsequent development.

3.1. Smooth formulation

We follow roughly the language of [JBPS11] in describing linear blend skinning. Skinning weights are computed on a geometric domain $\Omega \subseteq \mathbb{R}^d$, which could be a region of the plane ($d = 2$), surface ($d = 3$), or a volume ($d = 3$). A character receives a set of levers that can be manipulated to animate the character. In our formulation, these can be either

- *handles*, isolated points; or
- *bones*, straight line segments.

Each handle and each bone receives a weight function that determines how it influences every point on the domain. We denote these weights by functions $w_1(x), \ldots, w_m(x) : \Omega \to \mathbb{R}_+$. The artist can manipulate every handle and bone by applying a rotation R_i and a translation t_i to it. *Linear blend skinning* [MTLT89] then blends these transformations over the whole domain using the skinning weights. In particular, every $x \in \Omega$ gets transformed to $\bar{x} \in \mathbb{R}^d$ via

$$\bar{x} = \sum_{i=1}^{m} w_i(x)(R_i x + t_i)\ . \tag{1}$$

There are many standard desiderata for skinning weights:

- The w_i should sum to one pointwise (the 'partition of unity' property), so that if the same transformation is applied to every handle and bone, the object transforms rigidly.
- The w_i should be non-negative so that moving a handle does not result in part of the object moving in the opposite direction.
- The w_i should be as smooth as possible so that there are no abrupt discontinuity artefacts in the animation.
- The w_i should be associated with their respective handles or bones. This means the weights should be one on their respective handles/bones and zero on all others: Manipulating the lever i should not influence any of the other levers $j \neq i$.

BBW [JBPS11] achieve all these desiderata by combining them in a large constrained optimisation problem, where overall smoothness is achieved by minimising the biharmonic energy:

$$\begin{aligned} \min_{w_1(x),\ldots,w_m(x)} \quad & \frac{1}{2}\sum_{j=1}^{m}\int_\Omega (\Delta w_j(x))^2\, dA(x) & \\ \text{s.t.} \quad & w_j(x) = \delta_{jk} & \forall x \in H_k \\ & \sum_{j=1}^{m} w_j(x) \equiv 1 & \forall x \in \Omega \\ & w_j(x) \geq 0 & \forall x \in \Omega \\ & \nabla w_j(x) \cdot \hat{n}(x) \equiv 0 & \forall x \in \Gamma_N\ . \end{aligned} \tag{2}$$

Here, $H_k \subseteq \Omega$ is the k-th deformation handle/bone, and δ_{jk} is the Kronecker delta (this constraint makes sure that w_j is one on its associated handle/bone and zero on all others). $\Delta := -\nabla \cdot \nabla$ denotes the Laplacian operator. The set $\Gamma_N \subseteq \partial\Omega$ is a portion of the boundary on which we wish to enforce Neumann conditions; [SGWJ18] observe that the original BBW discretisation implicitly can enforce such a constraint. In our setting, we can add Neumann conditions or take $\Gamma_N = \varnothing$ to rely on natural boundary conditions.

Abstractly, we can understand (2) as a *quadratic program*, that is, an optimisation problem with a convex quadratic objective function and linear inequality constraints. Quadratic programs are amenable to elegant theoretical analysis in the smooth and finite-dimensional cases. After discretisation, they can be optimised to any fixed level

of precision in polynomial time [KTK79], but the degree of the polynomial can be prohibitive for practical applications.

3.2. Finite-dimensional formulation

In this paper, we are concerned with solving (2) when the degrees of freedom are associated with vertices of a triangular or tetrahedral mesh. Suppose our mesh has n vertices. We can model our unknown weights functions as columns of a matrix $W \in \mathbb{R}^{n\times m}$. We can understand discretisations of (2) as solving the following optimisation problem:

$$\begin{aligned} \min_W \quad & \tfrac{1}{2}\operatorname{tr}(W^\top B W) \\ \text{s.t.} \quad & W\mathbb{1}_m = \mathbb{1}_n \\ & W \geq 0 \\ & CW = D. \end{aligned} \tag{3}$$

Here, the objective $\frac{1}{2}\operatorname{tr}(W^\top BW) = \frac{1}{2}\sum_{j=1}^m w_j^\top B w_j$ is a quadratic form summed over the individual weight functions used to measure smoothness. We will assume that B is a positive semidefinite matrix, as is the case for the biharmonic energy since it is roughly the square of the Laplacian. The vector $\mathbb{1}_p$ denotes the vector whose elements are all 1, of length p. The constraint $CW = D$, where $C \in \mathbb{R}^{k\times n}$ and $D \in \mathbb{R}^{k\times m}$, abstracts away all remaining linear constraints in (2), namely the prescribed values at handles and bones. Note that k does not need to equal m; while each handle corresponds to one row in C, each bone yields many rows in C, as the Kronecker delta is enforced on many points discretely sampled on the bone. As expected, this formulation is a convex quadratic program.

We provide details of our choices of B, C and D for implementing BBW in Section 7; they match those of previous work.

4. Dual Formulation

In this section, we derive a dual to (3), which is easier to solve and yields identical skinning weights at optimality. As it is the central formula in our paper, we provide details of the duality argument here. Readers seeking the final formulation can skip to Section 4.3.

4.1. Minimax formulation

Our derivation of the dual of (3) follows a few steps. First, we re-express this formulation as a minimax problem:

$$\begin{aligned} \min_W \big[\max_{Y\geq 0, v} \big(\tfrac{1}{2}\operatorname{tr}(W^\top BW) + v^\top(\mathbb{1}_n - W\mathbb{1}_m) \\ - \operatorname{tr}(Y^\top W) + \chi(CW = D)\big)\big]. \end{aligned} \tag{4}$$

In this expression, we use χ to denote the convex indicator function

$$\chi(CW = D) = \begin{cases} 0 & \text{if } CW = D \\ \infty & \text{otherwise.} \end{cases}$$

We make a careful choice to enforce the $CW = D$ constraint using an indicator function instead of a Lagrange multiplier as is used for the partition of unity constraint, as it will lead to a particularly elegant dual formulation.

We pause to argue that (3) is equivalent to (4). Note that the internal maximisation problem takes on value ∞ any time W does not satisfy the constraints of (3) since:

- the $\chi(\cdot)$ term blows up if $CW \neq D$;
- the $v^\top(\mathbb{1}_n - W\mathbb{1}_m)$ term can be unbounded when $W\mathbb{1}_m \neq \mathbb{1}_n$, since the coefficient in front of any nonzero element can be amplified by v; and
- the $-\operatorname{tr}(Y^\top W)$ term can be similarly unbounded when $W \not\geq 0$ by using large values in $Y \geq 0$.

Hence, since the outer problem minimises the objective for W, choosing a W for which the inner objective is finite avoids an infinite objective value.

Since the objective of (3) is bounded below by zero and the constraint $W \geq 0$ admits strictly feasible points $W > 0$, by the "Slater conditions" in convex optimisation [Sla50], problem (4) admits a *strong dual*. In particular, we can safely swap the minimum and the maximum in (4) without affecting the optimal objective value.

4.2. Simplifying the dual

Abstractly the dual to (4) just swaps the order of the optimisation problems:

$$\begin{aligned} \max_{Y\geq 0, v} \Big[v^\top \mathbb{1}_n + \min_W \Big(\frac{1}{2}\operatorname{tr}(W^\top BW) - v^\top W\mathbb{1}_m \\ - \operatorname{tr}(Y^\top W) + \chi(CW = D)\Big)\Big]. \end{aligned} \tag{5}$$

That is, we now have a problem for (v, Y) whose objective is the optimal solution of a minimisation problem for W.

While this seems like a complicated expression, in fact it is a useful one: The inner problem for W as a function of (v, Y) is a quadratic problem with a linear constraint after eliminating the indicator function $\chi(\cdot)$ notation:

$$\begin{aligned} \min_W \quad & \tfrac{1}{2}\operatorname{tr}(W^\top BW) - v^\top W\mathbb{1}_m - \operatorname{tr}(Y^\top W) \\ \text{s.t.} \quad & CW = D. \end{aligned} \tag{6}$$

This problem has a quadratic objective function and no inequality constraints. Hence, using a Lagrange multiplier matrix Z, it can be solved by inverting the following linear system of equations:

$$\underbrace{\begin{pmatrix} B & C^\top \\ C & 0 \end{pmatrix}}_{Q^{-1}} \begin{pmatrix} W \\ Z \end{pmatrix} = \begin{pmatrix} v\mathbb{1}_m^\top + Y \\ D \end{pmatrix} \tag{7}$$

Using Q to denote the inverse of the matrix on the left-hand side, we have

$$\begin{pmatrix} W \\ Z \end{pmatrix} = \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{pmatrix} \begin{pmatrix} v\mathbb{1}_m^\top + Y \\ D \end{pmatrix}, \tag{8}$$

where Q_{ij} denote blocks of Q. Expanding the product yields closed-form expressions for W and Z:

$$\begin{aligned} W &= Q_{11}\big(v\mathbb{1}_m^\top + Y\big) + Q_{12}D \\ Z &= Q_{12}^\top\big(v\mathbb{1}_m^\top + Y\big) + Q_{22}D, \end{aligned} \tag{9}$$

Note in our algorithm we will never need to compute individual elements of the blocks of Q—a potentially dense inverse matrix—but rather *apply* Q by solving sparse linear systems like the one in (8).

The first line of (9) gives the optimal choice of W for the inner minimisation problem in (5), with the guarantee that $CW = D$ thanks to the use of the Lagrange multiplier Z. Hence, we can substitute this expression for W into (5) while removing the minimisation operation and the indicator $\chi(CW = D)$. After eliminating constants and simplifying, this substitution gives a new version of the dual problem:

$$\begin{aligned} \max_{v,Y} \quad & -\operatorname{tr}\left(D^\top Q_{12}^\top \left(v\mathbb{1}_m^\top + Y\right)\right) + v^\top \mathbb{1}_n \\ & -\frac{1}{2}\operatorname{tr}\left(\left(v\mathbb{1}_m^\top + Y\right)^\top Q_{11}\left(v\mathbb{1}_m^\top + Y\right)\right) \\ \text{s.t.} \quad & Y \geq 0. \end{aligned} \tag{10}$$

As a final simplification to (10), we observe that the problem for v with Y fixed is quadratic and unconstrained. Hence, we can recover v by setting the gradient of the objective with respect to v to zero:

$$\begin{aligned} 0 &= \nabla_v \Big[-\operatorname{tr}\left(D^\top Q_{12}^\top \left(v\mathbb{1}_m^\top + Y\right)\right) + v^\top \mathbb{1}_n \\ &\qquad -\frac{1}{2}\operatorname{tr}\left(\left(v\mathbb{1}_m^\top + Y\right)^\top Q_{11}\left(v\mathbb{1}_m^\top + Y\right)\right)\Big] \\ &= -Q_{12}D\mathbb{1}_m + \mathbb{1}_n - Q_{11}\left(Y + v\mathbb{1}_m^\top\right)\mathbb{1}_m \\ \Rightarrow Q_{11}v &= \frac{1}{m}[\mathbb{1}_n - (Q_{12}D + Q_{11}Y)\mathbb{1}_m] \end{aligned} \tag{11}$$

This expression allows us to eliminate v from (10).

4.3. Dual optimisation problem

After substituting the expression (11) for $Q_{11}v$ every time it appears in the simplified dual (10) and changing sign of the problem, we arrive at our final dual:

$$\boxed{\begin{aligned} \min_Y \quad & \frac{1}{2}\operatorname{tr}(P_m Y^\top Q_{11} Y) + \operatorname{tr}(R^\top Y) \\ \text{s.t.} \quad & Y \geq 0. \end{aligned}} \tag{12}$$

Here, we make some definitions to simplify the formula:

$$P_m \coloneqq I_m - \frac{1}{m}\mathbb{1}_m\mathbb{1}_m^\top \tag{13}$$

$$R \coloneqq Q_{12}DP_m + \frac{1}{m}\mathbb{1}_n\mathbb{1}_m^\top. \tag{14}$$

To summarise our argument, we have:

Proposition. *The optimal values of (3) and (12) are the same up to sign. Moreover, we can recover W from Y via the expression*

$$W = R + Q_{11}YP_m. \tag{15}$$

The expression for W above comes from substituting the closed-form value of v from (11) into (9), as follows:

$$\begin{aligned} W &= \frac{1}{m}[\mathbb{1}_n - (Q_{12}D + Q_{11}Y)\mathbb{1}_m]\mathbb{1}_m^\top + Q_{11}Y + Q_{12}D \\ &\qquad \text{by (9) and (11)} \\ &= \frac{1}{m}\mathbb{1}_n\mathbb{1}_m^\top + (Q_{11}Y + Q_{12}D)P_m \text{ by (13)} \\ &= R + Q_{11}YP_m \text{ by (14)}. \end{aligned}$$

Comparing (3) to (12), we see that the constraint set of (12) is comparatively much simpler. Moreover, the objective of (12) is a convex quadratic function, which can be understood as a least-squares problem for Y (see Section 2.3). Since (12) has a convex qua dratic objective with a single convex (non-negativity) constraint, it has no local optima; gradient-based optimisation procedures, including the one we propose in Section 6 hence are able to reach the global optimum.

Using $f(Y)$ to denote the objective function in (12), for convenience we include the derivative formula:

$$\nabla f(Y) = R + Q_{11}YP_m. \tag{16}$$

5. Smooth Interpretation

We presented the finite-dimensional duality argument in detail (Section 4), as it is the most relevant for computation of BBW. Duality theory, however, extends to certain convex functionals over infinite-dimensional function spaces, suggesting the possibility of carrying out a similar analysis on the smooth formulation (2). Despite its wide adoption in practice, the BBW problem is not well-studied theoretically, and little is known about solutions of the basic problem (2), including differentiability of the BBW weights; from this theoretical perspective, duality can begin to shed light into behaviour of the BBW model independently of its discretisation on a triangle mesh.

As Slater conditions apply only to the finite-dimensional case and existence/regularity questions for the smooth problem (2) are, to our knowledge, unstudied, we do not attempt to prove as rigorous a result as in Section 4.3 and leave such analysis to future work. By integrating by parts and performing a similar minimax swap, however, we conjecture that the dual to (2) is:

$$\begin{aligned} \min_{v(x),y_i(x):\Omega\to\mathbb{R}} \quad & \frac{1}{2}\sum_i \|v(\cdot) + y_i(\cdot)\|_{\Delta^{-2}}^2 - \int_\Omega v(x)\,dx \\ \text{s.t.} \quad & y_i(x) \geq 0 \;\forall i \in \{1,\ldots,m\}, x \in \Omega \end{aligned} \tag{17}$$

Here, the function $v(x)$ takes the place of vector v in our finite-dimensional formula (10), and the functions $y_i(x)$ take the place of the columns of Y; we could similarly eliminate $v(x)$ by solving a linear PDE. The norm $\|\eta(\cdot)\|_{\Delta^{-2}}^2$ is given by the inner product of η and the solution to the biharmonic equation $\Delta^2 g = \eta$ subject to the boundary conditions in (2). We recover the biharmonic weights $w_i(x)$ by solving the biharmonic equation with right-hand side $v + y_i$. An informal argument is given in Appendix C.

The reason to consider the smooth dual formula (17) is that it provides some intuition for our finite-dimensional problem (12). In particular, we can see that the biharmonic energy in the original BBW problem is replaced with an 'inverse biharmonic energy' in (17). We hypothesise that this change of variables and norms effectively preconditions our optimisation problem similar to Sobolev preconditioning [Neu85, RN95], helping explain the relatively efficient convergence of our numerical experiments.

6. Optimisation Algorithm

Abstractly, our dual (12) is a convex quadratic program that could be solved using standard tools like Mosek or Gurobi. In particular, 'unrolling' $Y \in \mathbb{R}^{n\times m}$ to a vector $y \in \mathbb{R}^{nm}$ yields a more familiar objective $\frac{1}{2}y^\top Ay + y^\top r$, where r unrolls the matrix R and A is constructed from P_m and Q_{11}. This strategy, however, is inefficient, as the matrix A would incorporate a Kronecker product of P_m and Q_{11}.

Since A above has a lot of repeated structure, it falls into the category of matrices that are 'easy to apply but difficult to store'. In particular, deconstructing the gradient formula (16) into linear and constant terms and matching it with the expression $\nabla_y(\frac{1}{2}y^\top Ay + r^\top y)$ yields the following:

$$\begin{array}{ccc} y \mapsto Ay & \longleftrightarrow & Y \mapsto Q_{11}YP_m \\ r & \longleftrightarrow & R. \end{array}$$

These formulas make it possible to directly apply iterative NNLS algorithms identified in Section 2.3 that only depend on *application* of the quadratic form.

For simplicity, our implementation uses the strategy suggested in recent work [CMV22] on NNLS, which enforces non-negativity by essentially squaring the unknown variable and then applies gradient descent; they argue this strategy reaches a global optimum for NNLS if it is initialised with a non-negative vector. We optimise the resulting unconstrained first-order problem using the L-BFGS algorithm to accelerate convergence (cf. [Sol15]).

7. Discretisation

Our method so far can be used for any quadratic program of the form (3). In our experiments, we make specific discretisation choices for B, C and D that largely align to prior work.

We discretise the biharmonic energy using the mixed finite element method [JTSZ10]. Let L be the cotangent discretisation of the Laplacian, and let M be a lumped mass matrix; we use the Voronoi lumped matrix implemented in LIBIGL [JP*18]. The biharmonic energy matrix B is then given by $B = L^\top M^{-1}L$, where the inverse mass matrix can be computed efficiently since M is diagonal. We rescale M by dividing by its largest entry, for numerical stability.

To incorporate the handle and bone constraints, we remesh the domain. Given a set of handle positions $h_1, \ldots, h_{k_h}$ and bone endpoints $(b_{1,1}, b_{2,1}), \ldots, (b_{1,k_b}, b_{2,k_b})$, we first sample each bone $i \in \{1, \ldots, k_b\}$ to get points $\hat{b}_{ij}$. We aim for ~ 2 points for every length ρ along a bone, where ρ is the average edge length of Ω, but enforce that there are at least two samples per bone. We then mesh the input boundary polyline of Ω (in 2D) or the input boundary triangle mesh of Ω (in 3D) to generate a triangle mesh or tetrahedral mesh, respectively, that contains the handles h_i and the computed samples $\hat{b}_{ij}$ as vertices that can be constrained.

The sparse matrix C and the matrix D are binary matrices that enforce the Kronecker delta constraints from (2). Neumann conditions do not need to be explicitly enforced using the constraint matrices, since the cotangent Laplacian L has zero Neumann boundary conditions baked in, which makes minimisers of the discrete biharmonic energy automatically have zero Neumann boundary conditions [SGWJ18].

8. Implementation

We implement our method in Python and C++ using LIBIGL [JP*18] and GPYTOOLBOX [SS*23]. The unconstrained optimisation problem resulting from our NNLS formulation coupled with [CMV22] is solved using L-BFGS as implemented in SCIPY [VGOS20] with random initialisation scaled so that the initial vector's squared entries sum to 1 in each column (default solver tolerance 10^{-6} and 100,000 maximum iterations). We generate triangle meshes using TRIANGLE [She02] and tetrahedral meshes using TETGEN [Si15].

We implement multiplication by Q_{11} and Q_{12} by using a sparse LU decomposition to factor the matrix $\left(\begin{smallmatrix} B & C^\top \\ C & 0 \end{smallmatrix}\right)$ once, after which we are able to apply its inverse efficiently throughout the optimisation procedure. The factorisation is done with SUPERLU [Li05] via SCIPY [VGOS20].

For comparisons, we reimplement classical BBW in Python using Mosek for optimisation [ApS24], and we use LIBIGL's Mosek-powered implementation of BBW without the partition of unity constraint [JP*23a]. We also use LIBIGL's BBW tutorial example [JP*23b], which uses a LIBIGL-internal active set solver and terminates after only 8 iterations even if the estimated solution is suboptimal. The output weights from all methods are clamped to [0,1] and renormalised to sum to 1 at every vertex as a post-processing step to account for numerical error.

The experiments are run on an Ubuntu Linux machine with an Intel i7-12700 processor and 32GB RAM, as well as on a general-use Linux cluster with 8 CPU cores and 64GB RAM.

9. Experiments

We test our algorithm on a variety of two- and three-dimensional examples (triangle and tetrahedral meshes, respectively) to demonstrate its effectiveness. Broadly, our experiments show that our convex dual provides a strong tradeoff between efficiency, stability and ease of implementation. Additional results can also be found in the teaser, Figure 1.

Figures 2 and 3 visualise our weight functions as colour maps on 2D regions and 3D surfaces. As expected, thanks to the biharmonic objective function, we recover smooth weights that conform to the handle/bone geometry; we do not observe ringing or other artefacts that might make the weights less useful.

Figure 2: *Visualisation of our weights on a variety of 2D meshes; each colour indicates the weight associated with a different handle/bone (shown on top).*

Figure 3: *Visualisation of our skinning weights on 3D models, each colour indicates the weight associated with a different handle/bone.*

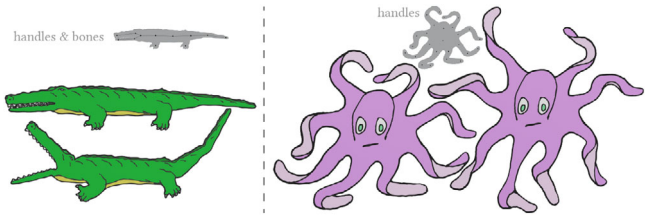

Figure 4: *Application of our skinning weights to linear blend skinning (1) for posing triangle meshes in 2D.*

Figures 4 and 5 show that our weights can be directly plugged into code for linear blend skinning (1) in 2D and 3D (see Section 3.1), respectively. Our weights yield smooth deformations that interpolate the handle/bone motions to the rest of the domain, identically to applications of past skinning methods.

Figures 6 and 7 confirm that our method agrees with the quadratic programming approach to optimising for BBW. In this experiment, we run a reference implementation of the primal problem and compare it to the output of our optimisation procedure; as expected, they are visually indistinguishable.

Since the original problem can be intractable for large meshes, as explained in Appendix A, a typical approximation when optimising for BBW is to drop the partition of unity constraint in (2), decoupling the weight functions in the optimisation problem. The weights can then be normalised pointwise a posteriori to sum to 1. This strategy is employed in the LIBIGL implementation. Figure 8 compares our BBW weights to this model. The non-PoU weights are slightly different and have higher biharmonic energy—and hence are in this

Figure 5: *Application of our skinning weights to linear blend skinning (1) for posing tetrahedral meshes in 3D.*

sense less smooth. In particular, dropping the PoU constraint yields weights that shift substantially along the legs of the octopus model.

Figure 9 provides detailed measurements about the progress of our solver for the crocodile example in Figure 4. As L-BFGS iterates, not only does the objective of the dual problem decrease, but also the constraint satisfaction of the corresponding primal weights improves. We measure this effect by applying (15) to each iterate, obtaining an estimate of W that before optimality still satisfies every linear constraint of the BBW problem (2) by construction. In the two plots, we see that the dual objective (12) (plotted in green) measured from the iterates decreases quickly and nearly monotonically, as would be expected since we apply BFGS directly to this problem. More interestingly, the primal objective measured from W in (3) (plotted in blue) and non-negativity of W (plotted in red) also improve steadily, even though these aspects are achieved indirectly by means of strong duality.

Table 1 compares the runtimes and final biharmonic energy values from our method, classical BBW (reimplemented by us), BBW without the partition of unity (PoU) constraint (BBWA in [WS21], using the LIBIGL Mosek implementation [JP*23a]), and [JP*23b]'s active set implementation of BBW without PoU with an iteration limit of 8, as used in the LIBIGL tutorial's BBW example. Our method is usually faster than classical BBW with similar energy values, and it consistently achieves lower energies than BBW without PoU and the fixed-iteration active-set BBW. While we have made a reasonable effort to match experimental conditions, we caution against drawing strong conclusions from this table, however, since parameters such as tolerances can influence runtime and final energy. Further timings with models featured in prior work are available in Appendix D.

10. Discussion and Conclusion

Our work suggests the power of convex duality in developing algorithms to solve variational problems in geometry processing. In this case, duality suggested a different quadratic program to solve, which is amenable to simple optimisation procedures.

In the future, we can consider many variations of the BBW problem. For example, [ZDL*14] suggest the possibility of an L^1 variant, which likely admits a similar dual but will not benefit from NNLS methods. We also can add constraints, as in [JWS12], to avoid local

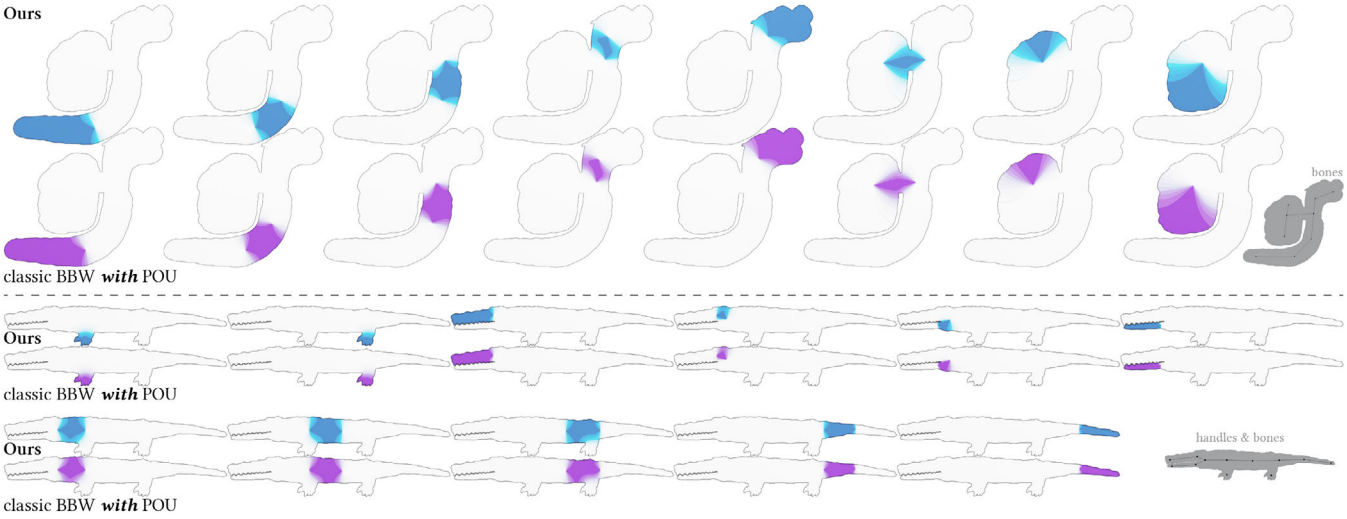

Figure 6: *Comparison between our weights and those computed by solving the BBW quadratic program [JBPS11], including a partition-of-unity constraint. As expected, the two are indistinguishable, but our approach requires less computation time. The largest pointwise weight difference between ours and classic BBW is* 0.029 *for the worm and* 0.013 *for the crocodile.*

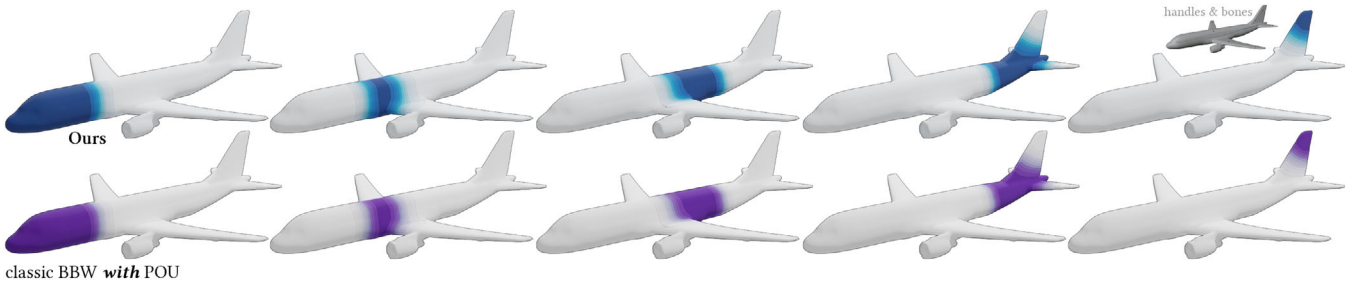

Figure 7: *Visualisation of a few selected skinning weights (and their classically computed counterparts) on a surface in 3D. Our weights are visually indistinguishable from the classical PoU weights. The largest pointwise weight difference between ours and classic BBW is* 0.015.

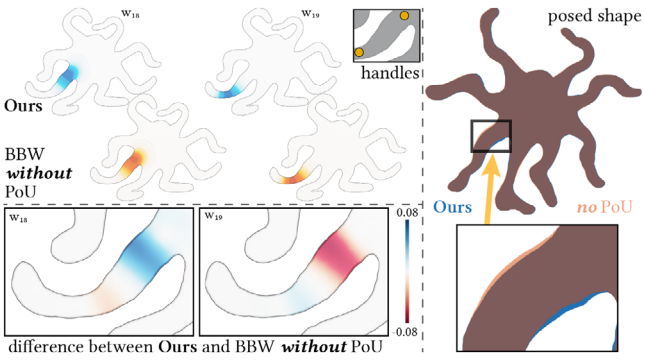

Figure 8: *Our solution agrees with classic BBW that enforces the partition of unity constraint (PoU), but differs from the classic BBW when PoU is dropped. Dropping the constraint, a choice recommended in literature that leads to speed-ups (BBWA in [WS21]), leads to visually different weights* (left)*. When a character is deformed with our weights and no-PoU weights, there is a visual difference (our and no-PoU results superimposed,* right*).*

optima in our weights or consider higher-order smoothness energies, as in [BK04], without changing our formulation significantly. The adjacent problem of optimising for barycentric coordinates may also admit a useful dual [NS12].

We also can explore optimisation strategies to recover skinning weights even more quickly. For example, [SRGB14] and others use the ADMM algorithm for convex problems in geometry processing, which may be possible in our setting. [WS21] suggest applying accelerated gradient descent methods like ADAM that are popular in machine learning [KB14], although in our preliminary experiments, this strategy without tuning did not outperform L-BFGS.

Our implementation is not parallelised, but our timings are already competitive with past work and fast enough for target applications. There are many species of parallelisation possible that could further accelerate our approach (and past work). While parallelising over weight functions is only possible when the POU constraint is dropped, both the POU and non-POU implementations can benefit from parallelising the numerical linear algebra operations involved in each iteration. We hypothesise that all major techniques

Table 1: *Runtimes (in seconds) and biharmonic energy values for our method with two different solver tolerances, classic BBW with the partition of unity constraint (PoU) reimplemented by us, BBW without PoU (BBWA [WS21]) implemented in* LIBIGL *using Mosek [JP*23a], and an active-set BBW without PoU with maximum number of iterations set to 8 [JP*23b].*

		Ours, tol 10^{-6}		Ours, tol $5 \cdot 10^{-8}$		classic BBW, PoU		classic BBW, ~~PoU~~		niter 8 BBW, ~~PoU~~	
	# verts	Time	Energy	Time	Energy	Time	Energy	Time	Energy	Time	Energy
Armadillo 3D 5 handles, 16 bones	31 641	**814.8**	85 745.2	957.9	**85 163.2**	†	†	†	†	104.3	85 618.62
Banana 3D 0 handles, 7 bones	21 087	**63.3**	**226.0**	98.1	**226.0**	3077.0	**226.0**	5047.9	**226.0**	7.3	226.1
Crocodile 2D 2 handles, 9 bones	13 912	**27.9**	14 026 579.0	56.2	14 022 707.4	2416.1	**14 022 001.4**	2106.2	14317 209.2	0.9	14 465 210.6
Dog 2D 6 handles, 12 bones	8099	**37.2**	2 708 773.3	63.8	2 708 121.6	2298.5	**2707 938.0**	858.4	2742181.4	0.7	2757786.0
Fish 2D 23 handles, 0 bones	12 829	**59.4**	21 690.9	104.0	21 675.5	5913.6	**21 673.7**	1803.5	23 438.3	1.3	23 418.8
Golem 3D 5 handles, 25 bones	6099	**78.5**	35 224.7	146.7	35 188.8	1248.1	**35 181.4**	665.5	36 970.8	4.3	36 966.1
Hand 3D 0 handles, 19 bones	16 658	**193.6**	143 253.9	231.3	**143 239.3**	†	†	7044.3	144 658.5	17.7	144 771.3
Crab 2D 32 handles, 11 bones	12 082	**143.9**	4 285 123.4	624.9	4 283 831.2	6776.5	**4 283 736.4**	2726.4	4 315 891.8	1.3	4 325 268.8
LIBIGL tutorial hand 3D [JP*23b] 0 handles, 20 bones.	7234	**140.2**	57 461.5	183.8	57 450.4	2907.9	**57 448.3**	703.0	59 610.1	2.9	59 567.2
Mushroom 3D 1 handle, 7 bones	50 931	165.3	9876.6	218.1	**9876.1**	†	†	**3.1**	781 507 229.6	105.8	10 003.6
Octopus 2D 26 handles, 0 bones	6923	**45.5**	16 133.0	72.1	**16 123.2**	213.7	16 123.5	790.73	17 596.4	1.2	17 685.3
Pants 2D 0 handles, 6 bones	2730	**1.1**	239 144.1	1.7	239 135.0	34.4	**239 120.1**	17.2	239 122.7	0.1	239 260.2
Penguin 3D 14 handles, 0 bones	55 405	**558.0**	157.5	696.5	**156.9**	†	†	†	†	791.1	164.9
Penguin 2D 9 handles, 4 bones	16 588	**62.2**	545 210.7	99.9	**545 175.3**	†	†	4612.5	545 740.7	1.6	546 171.9
Pigeon 3D 16 handles, 3 bones	42 319	**310.3**	**99.3**	439.6	**99.3**	†	†	†	†	34.5	103.8
Plane 3D 7 handles, 4 bones	15 164	**78.7**	**749.2**	109.9	**749.2**	7101.1	**749.2**	3598.0	749.7	6.4	749.8
Spot 3D 13 handles, 0 bones	36 585	**321.7**	221.9	389.7	**220.5**	†	†	†	†	403.9	225.2
Tree 3D 29 handles, 11 bones	58 406	**6054.9**	31 944.1	6409.8	**31 908.4**	†	†	†	†	265.3	32 541.4
Vacuum 2D 6 handles, 5 bones	8534	**29.9**	1 251 441.9	54.4	**1 251 416.3**	1122.1	1 251 426.4	590.5	1 251 963.5	0.6	1 252 814.9
Worm 2D 0 handles, 8 bones	9194	**5.2**	1 852 494.7	8.8	1 852 306.3	477.8	**1 852 275.3**	512.7	1 884 731.6	0.4	1 900 466.7

Note: The † symbol indicates a timeout (runtime limit of 2h), an out-of-memory error, failure of the Mosek system to be SPD, or other crash. The biharmonic energy values here are without the rescaling of the mass matrix that is used during optimisation. We **bold** the lowest timing/energy values, excluding the final column since it is not run to optimality.

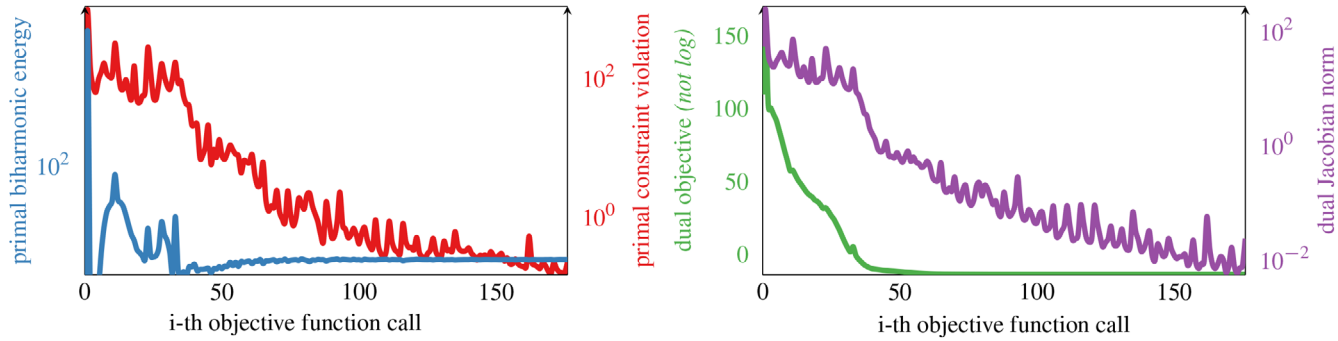

Figure 9: *Optimisation statistics for the crocodile example with tolerance* 10^{-6} *in log scale.* Left*: the biharmonic energy of the primal variable and its constraint violation* $\|\min(u, 0)\|_2$. Right*: the dual objective and Jacobian, used by the L-BFGS solver to solve the unconstrained dual problem.*

for skinning—with or without the POU constraint—can achieve nearly full CPU (or even GPU) usage by parallelising the matrix multiplies and linear solves rather than parallelising the loop over individual skinning weights functions.

Finally, although BBW and peers demonstrate strong performance across a broad variety of geometry processing tasks, there are still barriers to broad adoption in industry, where simpler alternatives like distance-based weights are popular. While our work reduces barriers related to the inefficiency of optimising state-of-the-art skinning weights models, a challenging topic for future work involves defining methods that do not rely on tetrahedral meshing and merging the domain into a single connected component.

Acknowledgements

We thank Alec Jacobson and Ana Dodik for scientific discussions and sharing data. We thank Daniella Levy for posing the hand in Figure 5. We thank Yu Wang, Pranav Jain, and Dylan Rowe for proofreading. We graciously acknowledge the creators and curators of assets used in this article [Jac13, The23, mat21, Den22, Max21, JP*18, Hol23, Ste23, NmC24, Yah19, Cra19, war24, DSSS25]. The MIT Geometric Data Processing Group acknowledges the generous support of Army Research Office grants W911NF2010168 and W911NF2110293, of National Science Foundation grant IIS-2335492, from the Air Force Office of Scientific Research, from the CSAIL Future of Data program, from the MIT–IBM Watson AI Laboratory, from the Toyota–CSAIL Joint Research Center, and from the Wistron Corporation. USC's Geometry and Graphics Group is generously supported by a gift from Adobe Inc. and graciously acknowledges the support of the National Science Foundation (award #2335493).

References

[ApS24] ApS M.: *MOSEK Optimizer API for Python 10.1.28*, 2024. URL: https://docs.mosek.com/latest/pythonapi/index.html.

[BK04] Botsch M., Kobbelt L.: An intuitive framework for real-time freeform modeling. *ACM Transactions on Graphics (TOG) 23*, 3 (2004), 630–634.

[BMM06] Bellavia S., Macconi M., Morini B.: An interior point newton-like method for non-negative least-squares problems with degenerate solution. *Numerical Linear Algebra with Applications 13*, 10 (2006), 825–846.

[BP07] Baran I., Popović J.: Automatic rigging and animation of 3D characters. *ACM Transactions on Graphics (TOG) 26*, 3 (2007), 72–es.

[BPC*11] Boyd S., Parikh N., Chu E., Peleato B., Eckstein J., et al.: Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning 3*, 1 (2011), 1–122.

[BV04] Boyd S. P., Vandenberghe L.: *Convex Optimization*. Cambridge University Press, 2004.

[CMV22] Chou H.-H., Maly J., Verdun C. M.: Non-negative least squares via overparametrization. *arXiv:2207.08437* (2022).

[Cra19] Crane K.: Spot, 2019. Downloaded modified version from odedstein-meshes github.com/odedstein/meshes/tree/master/objects/spot, originally from cs.cmu.edu/~kmcrane/Projects/ModelRepository.

[Den22] Dennisaurio: Odin Golem, 2022. URL: https://sketchfab.com/3d-models/odin-golem-1a10c5a5f23d41d6828b5bbd49c57edb.

[DSSS25] Dodik A., Sitzmann V., Solomon J., Stein O.: Robust biharmonic skinning using geometric fields, 2025. URL: https://arxiv.org/abs/2406.00238, arXiv:2406.00238.

[Hol23] Holinaty J.: Mushroom, 2023. Downloaded from odedstein-meshes github.com/odedstein/meshes/tree/master/objects/mushroom. Asset licensed under CC BY-NC 4.0.

[Jac13] Jacobson A.: *Algorithms and Interfaces for Real-Time Deformation of 2D and 3D Shapes*. PhD thesis, ETH Zürich, 2013.

[JBPS11] Jacobson A., Baran I., Popović J., Sorkine O.: Bounded biharmonic weights for real-time deformation. *ACM Transactions on Graphics 30*, 4 (2011), 78.

[JMD*07] Joshi P., Meyer M., DeRose T., Green B., Sanocki T.: Harmonic coordinates for character articulation. *ACM Transactions on Graphics (TOG) 26*, 3 (2007), 71–es.

[JP*18] Jacobson A., Panozzo D., et al.: libigl: A simple C++ geometry processing library, 2018. URL: https://libigl.github.io/.

[JP*23a] Jacobson A., Panozzo D., et al.: libigl mosek bounded biharmonic weights, 2023. URL: https://github.com/libigl/libigl/blob/main/include/igl/mosek/bbw.h.

[JP*23b] Jacobson A., Panozzo D., et al.: libigl tutorial 403: Bounded biharmonic weights, 2023. URL: https://github.com/libigl/libigl/blob/main/tutorial/403_BoundedBiharmonicWeights/main.cpp.

[JTSZ10] Jacobson A., Tosun E., Sorkine O., Zorin D.: Mixed finite elements for variational surface modeling. In *Computer Graphics Forum* (2010), vol. *29*, Wiley Online Library, pp. 1565–1574.

[JWS12] Jacobson A., Weinkauf T., Sorkine O.: Smooth shape-aware functions with controlled extrema. In *Computer Graphics Forum* (2012), vol. *31*, Wiley Online Library, pp. 1577–1586.

[KB14] Kingma D. P., Ba J.: ADAM: A method for stochastic optimization. *arXiv:1412.6980* (2014).

[KSD13] Kim D., Sra S., Dhillon I. S.: A non-monotonic method for large-scale non-negative least squares. *Optimization Methods and Software 28*, 5 (2013), 1012–1039.

[KTK79] Kozlov M. K., Tarasov S. P., Khachiyan L. G.: Polynomial solvability of convex quadratic programming. In *Doklady Akademii Nauk* (1979), vol. *248*, Russian Academy of Sciences, pp. 1049–1051.

[LH74] Lawson C. L., Hanson R. J.: Linear least squares with linear inequality constraints. *Solving Least Squares Problems* (1974), 158–173.

[Li05] Li X. S.: An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Transactions on Mathematical Software (TOMS) 31*, 3 (2005), 302–325.

[LS00] Lee D., Seung H. S.: Algorithms for non-negative matrix factorization. *Advances in Neural Information Processing Systems 13* (2000), 535–541.

[mat21] matousekfoto: Banana, 2021. URL: https://sketchfab.com/3d-models/banana-4d4c8d85a09b476c8136f0c4cd24c45a.

[Max21] Max Planck Society e.V via SMPL: Hand, 2021. Downloaded modified version from odedstein-meshes github.com/odedstein/meshes/tree/master/objects/hand, adapted from M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, M. Black "SMPL" (SIGGRAPH Asia 2015). Asset licensed under CC BY 4.0.

[MTLT89] Magnenat-Thalmann N., Laperrière R., Thalmann D.: Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics Interface '88* (CAN, 1989), Canadian Information Processing Society, pp. 26–33.

[Neu85] Neuberger J.: Steepest descent and differential equations. *Journal of the Mathematical Society of Japan 37*, 2 (1985), 187–195.

[NmC24] NmC BC: Pigeon #1, 2024. URL: https://sketchfab.com/3d-models/pigeon-1-640af0b61ea9454da08d4b75cb6bb6ae.

[NS12] Nieto J. R., Susín A.: Cage based deformations: A survey. In *Deformation Models: Tracking, Animation and Applications*. Springer, 2012, pp. 75–99.

[Pol15] Polyak R. A.: Projected gradient method for non-negative least square. *Contemporary Mathematics 636*, (2015), 167–179.

[RN95] Renka R. J., Neuberger J.: Minimal surfaces and Sobolev gradients. *SIAM Journal on Scientific Computing 16*, 6 (1995), 1412–1427.

[Roc70] Rockafellar R. T.: *Convex Analysis*. Princeton University Press, Princeton, 1970.

[SGWJ18] Stein O., Grinspun E., Wardetzky M., Jacobson A.: Natural boundary conditions for smoothing in geometry processing. *ACM Transactions on Graphics (TOG) 37*, 2 (2018), 1–13.

[She02] Shewchuk J.: Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry 22*, 1 (2002), 21–74.

[Si15] Si H.: TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software 41*, 2 (2015).

[Sla50] Slater M.: Lagrange multipliers revisited. In *Traces and Emergence of Nonlinear Programming (2014 reprint)*. Springer, 1950, pp. 293–306.

[Sol15] Solomon J.: *Numerical Algorithms: Methods for Computer Vision, Machine Learning, and Graphics*. CRC Press, 2015.

[SRGB14] Solomon J., Rustamov R., Guibas L., Butscher A.: Earth mover's distances on discrete surfaces. *ACM Transactions on Graphics (ToG) 33*, 4 (2014), 1–12.

[SS*23] Sellán S., Stein O., et al.: Gpytoolbox: A python geometry processing toolbox, 2023. URL: https://gpytoolbox.org/.

[SSL02] Sha F., Saul L., Lee D.: Multiplicative updates for non-negative quadratic programming in support vector machines. *Advances in Neural Information Processing Systems 15*, (2002), 697–704.

[Ste23] Stein O.: Penguin, 2023. Downloaded from odedstein-meshes github.com/odedstein/meshes/tree/master/objects/penguin. Asset licensed under CC BY 4.0.

[The23] The Stanford 3D Scanning Repository: Armadillo, 2023. Downloaded modified version from odedstein-meshes github.com/odedstein/meshes/tree/master/objects/armadillo. Original mesh by V. Krishnamurthy and M. Levoy "Fitting Smooth Surfaces to Dense Polygon Meshes" (SIGGRAPH 1996).

[Tos08] Tosun E.: *Geometric modeling using high-order derivatives*. PhD thesis, New York University, 2008.

[VGOS20] Virtanen P., Gommers R., Oliphant T. E., SciPy 1.0 Contributors: SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods 17* (2020), 261–272.

[war24] warmarine759: Tree, 2024. Downloaded modified version from odedstein-meshes github.com/odedstein/meshes/tree/master/objects/tree, originally from thingiverse.com/thing:3839198. Asset licensed under CC BY 4.0.

[WS21] Wang Y., Solomon J.: Fast quasi-harmonic weights for geometric data interpolation. *ACM Transactions on Graphics (TOG) 40*, 4 (2021), 1–15.

[Yah19] YahooJAPAN: Plane, 2019. Downloaded modified version from odedstein-meshes github.com/odedstein/meshes/tree/master/objects/plane, originally from thingiverse.com/thing:182252. Asset licensed under CC BY 3.0.

[ZDL*14] Zhang J., Deng B., Liu Z., Patanè G., Bouaziz S., Hormann K., Liu L.: Local barycentric coordinates. *ACM Transactions on Graphics (TOG) 33*, 6 (2014), 1–12.

Appendix A: Relaxation: Removing Partition of Unity

A common approximation to the bounded biharmonic weight (BBW) problem drops the partition of unity constraint, that is, the first constraint of (3). In its place, the constraint $W \leq 1$ is added to prevent weights from becoming too large. This adjustment leads to a relaxation of (3):

$$\begin{aligned} \min_{\hat{W}} \quad & \frac{1}{2}\operatorname{tr}(\hat{W}^\top B \hat{W}) \\ \text{s.t.} \quad & \hat{W} \geq 0 \\ & \hat{W} \leq 1 \\ & C\hat{W} = D. \end{aligned} \tag{A1}$$

Our earlier duality argument does not apply perfectly to this new box-constrained problem thanks to the introduction of a second inequality constraint $\hat{W} \leq 1$ relative to (3). As we derive in Appendix B, however, this problem still has an NNLS-style dual of the form

$$\begin{aligned} \min_{\hat{Y}} \quad & \frac{1}{2}\operatorname{tr}(\hat{Y}^\top \hat{Q} \hat{Y}) + \operatorname{tr}(\hat{R}^\top \hat{Y}) \\ \text{s.t.} \quad & \hat{Y} \geq 0. \end{aligned} \tag{A2}$$

Appendix B also gives a closed-form expression for recovering the weights $\hat{W}$ from $\hat{Y}$.

Table A1: *Effect of enforcing partition of unity (PoU) with our dual optimisation.*

	Ours, PoU		Ours, ~~PoU~~		Classic BBW, ~~PoU~~	
	Time	Energy	Time	Energy	Time	Energy
Dog 2d	39.1	2 708 773.3	51.6	2 743 932.6	851.6	2 742 181.4
Golem 3d	77.5	35 224.7	68.4	37 036.4	649.4	36 970.8

Note: We compare energies for two examples of skinning weights computed with our method *with* PoU (Section 4.3), our method *without* PoU (Appendix A), and classic BBW *without* PoU [JP*23a]. Not enforcing PoU for dual optimisations does not offer a significant time advantage for our dual method and is thus of little practical utility.

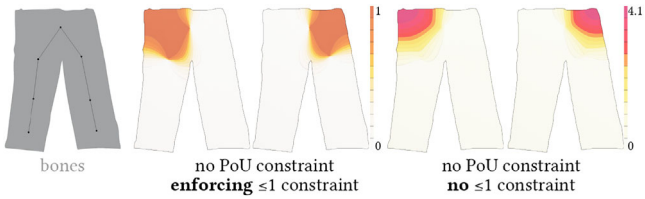

Figure A1: *If the partition of unity constraint is not enforced, weights have to be explicitly constrained to be* ≤ 1 (left)*; otherwise, the resulting weights can greatly exceed* 1 (right)*. Post-processing clamping and renormalisation have been turned off for this figure. Measuring the biharmonic energy after renormalisation and clamping gives* 239 195.8 *if* ≤ 1 *is enforced during optimisation, and* 245 288.4 *if it is not, that is, not enforcing the constraint during optimisation leads to* higher *energies after clamping and renormalisation.*

Unlike our previous dual (12), this new dual problem decouples over the columns of Y, as expected. On the other hand, since (A1) has twice as many inequality constraints as our original primal problem (3), the matrices in (A2) are twice as large. After computing $\hat{W}$, a posteriori we need to normalise each row of the matrix to sum to 1 to satisfy the partition of unity constraints needed for skinning-based applications, an operation that makes the rescaled $\hat{W}$ suboptimal for our original problem (3).

Table A1 shows that we can indeed optimise (A1) using our NNLS implementation. Doing so, however, offers no significant time advantage over our method *with* partition of unity included. Hence, since our full dual in Section 4.3 efficiently recovers the optimal solution of the original non-relaxed problem, we conclude that this relaxed formulation does not have much practical utility.

Figure A1 shows that enforcing the ≤ 1 constraint is really necessary; without it, the optimisation can return results with values that are way above 1. Clamping such weights to $[0, 1]$ and renormalising significantly increases the energy.

Appendix B: Dual Without Partition of Unity

If we remove the partition of unity constraint from the BBW problem, it decouples over columns w of the matrix $\hat{W}$. Hence, it is

sufficient to consider the following one-column-at-a-time problem:

$$\begin{aligned} \min_w \quad & \frac{1}{2} w^\top B w \\ \text{s.t.} \quad & w \geq 0 \\ & w \leq 1 \\ & Cw = d. \end{aligned} \tag{B1}$$

This problem can be expressed in minimax form as follows:

$$\min_w \max_{y,u \geq 0} \left[\frac{1}{2} w^\top B w - y^\top w - u^\top (\mathbb{1}_n - w) + \chi(Cw = d) \right]. \tag{B2}$$

The additional dual variable u is used to enforce the $w \leq 1$ constraint.

We once again dualise by swapping the min and the max. In this case, our inner linear problem for w becomes:

$$\begin{aligned} \min_w \quad & \frac{1}{2} w^\top B w - (y - u)^\top w \\ \text{s.t.} \quad & Cw = d. \end{aligned}$$

This problem is solved using a linear system of equations:

$$\begin{pmatrix} B & C^\top \\ C & 0 \end{pmatrix} \begin{pmatrix} w \\ z \end{pmatrix} = \begin{pmatrix} y - u \\ d \end{pmatrix} \tag{B3}$$

Using the notation in Section 4.2, we conclude:

$$w = Q_{11}(y - u) + Q_{12} d \tag{B4}$$

$$z = Q_{12}^\top (y - u) + Q_{22} d. \tag{B5}$$

Hence, the inner w optimisation problem has the following optimal objective value:

$$\begin{aligned} & \frac{1}{2} w^\top B w - (y - u)^\top w \\ & \quad = \frac{1}{2} w^\top (y - u - C^\top z) - (y - u)^\top w \text{ by (B3)} \\ & \quad = -\frac{1}{2} w^\top (y - u) - \frac{1}{2} d^\top z \text{ since } Cw = d \\ & \quad = -\frac{1}{2} \begin{pmatrix} w \\ z \end{pmatrix}^\top \begin{pmatrix} y - u \\ d \end{pmatrix} \\ & \quad = -\frac{1}{2} \begin{pmatrix} y - u \\ d \end{pmatrix}^\top Q \begin{pmatrix} y - u \\ d \end{pmatrix} \text{ by definition of } Q \\ & \quad = -\frac{1}{2} (y - u)^\top Q_{11} (y + u) - d^\top Q_{12}^\top (y - u) + \text{const.} \end{aligned}$$

Hence, after changing sign and eliminating constants, our dual problem can be written

$$\min_{y,u \geq 0} \frac{1}{2} (y - u)^\top Q_{11} (y - u) + (Q_{12} d)^\top (y - u) + \mathbb{1}_n^\top u. \tag{B6}$$

Define

$$\hat{Q} \coloneqq \begin{pmatrix} I_{n \times n} \\ -I_{n \times n} \end{pmatrix} Q_{11} \begin{pmatrix} I_{n \times n} \\ -I_{n \times n} \end{pmatrix}^\top$$

$$\hat{R} \coloneqq \begin{pmatrix} Q_{12} D \\ \mathbb{1}_n \mathbb{1}_m^\top - Q_{12} D \end{pmatrix}.$$

Then, we can vectorise and write our optimisation as an NNLS problem in $\hat{Y} \coloneqq \begin{pmatrix} Y^\top & U^\top \end{pmatrix}^\top$:

$$\begin{aligned} \min_{\hat{Y}} \quad & \frac{1}{2} \operatorname{tr}(\hat{Y}^\top \hat{Q} \hat{Y}) + \operatorname{tr}(\hat{R}^\top \hat{Y}) \\ \text{s.t.} \quad & \hat{Y} \geq 0. \end{aligned} \tag{B7}$$

We can recover the primal weights matrix W using (B4):

$$\hat{W} = Q_{11} \begin{pmatrix} I_{n \times n} & -I_{n \times n} \end{pmatrix} \hat{Y} + Q_{12} D.$$

Appendix C: Justification of Smooth Dual Formula

Denote by $\mathcal{S}$ the set of functions that satisfy the handle H_k and Neumann Γ_N conditions of (2). We can write a minimax formulation similar to (4) as follows:

$$\begin{aligned} \min_{w_i \in \mathcal{S}} \max_{y_i \geq 0, v(x)} & \frac{1}{2} \sum_{j=1}^{m} \int_\Omega \left[(\Delta^2 w_j - w_j \cdot (y_j + v) \right] dA \\ & + \int_\Omega v(x)\, dA(x) \end{aligned} \tag{C1}$$

Here, our unknown variables are primal skinning weights functions $w_i(x)$, non-negative dual functions $y_i(x) \geq 0$, and an unconstrained dual function $v(x)$.

Formally, we can swap the minimisation problem and the maximisation problem to obtain a dual. When we do so, we are left with the following unconstrained quadratic problem to recover each weight function $w_j(x)$ in terms of the functions $y_j(x)$, $v(x)$:

$$\begin{aligned} \min_{w_j(x)} \quad & \int_\Omega \left[\frac{1}{2} (\Delta^2 w_j - w_j \cdot (y_j + v) \right] dA \\ \text{s.t.} \quad & w_j(x) \equiv \delta_{jk} && \forall x \in H_k \\ & \nabla w_j(x) \cdot \hat{n}(x) \equiv 0 && \forall x \in \Gamma_N \end{aligned}$$

The two constraints here are the conditions that define $\mathcal{S}$. Taking the variational derivative of this expression with respect to w_j justifies that w_j solves the biharmonic equation $\Delta^2 w_j = y_j + v$ with appropriate boundary conditions. Substituting back into the objective of this minimisation yields the value $-\frac{1}{2} \|y_j + v\|^2_{\Delta^{-2}}$.

Plugging this objective for the w_j subproblem into (C1) and flipping sign to change maximisation to minimisation yields (17).

Appendix D: Additional Timing Experiments

We performed additional timing experiments, using meshes and handles from [DSSS25]. The results can be seen in Table D1.

Table D1: *Running time as well as final energy for our method with two different tolerances for a variety of 3D tetrahedral meshes by [DSSS25].*

	Ours, tol 10^{-6}		Ours, tol $5 \cdot 10^{-8}$	
	Time	Energy	Time	Energy
Bunny 10^{-3} approx 64k 133 228 verts, 671 168 tets	4230.30	4 520 115.30	6896.19	4 517 545.91
Bunny 10^{-3} 210 115 verts, 1 130 319 tets	4050.51	2 717 662.90	†	†
Cow 10^{-3} approx 64k 80 301 verts, 445 433 tets	2130.29	241 467.29	6896.19	4 517 545.91
Cow 10^{-3} 50 869 verts. 268 587 tets	1059.70	295 329.17	1828.28	294 860.44
Gear 10^{-3} approx 64k 56 273 verts, 309 589 tets	209.77	25 061.56	413.75	25 057.47
Gear 10^{-3} 31 831 verts, 164 244 tets	113.65	28 757.59	198.87	28 755.49
Penguin approx 64k 89 264 verts, 492 555 tets	243.76	244 904.56	473.11	244 867.46
Penguin 57 467 verts, 299 444 tets	162.93	294 151.62	328.49	294 062.57
Piggybank 10^{-3} approx 64k 90 106 verts, 501 846 tets	411.08	264 596.28	747.08	264 564.17
Piggybank 10^{-3} 80 954 verts, 447 000 tets	488.64	327 668.39	1085.33	327 480.85

Note: The † symbol indicates a timeout (runtime limit of 2h), an out-of-memory error, failure of the Mosek system to be SPD, or other crash. A regulariser of $10^{-9}\,I$ has been added to the system matrix Q; otherwise the same conditions as in Table 1 apply. Classic BBW (with PoU), as implemented by us, is unsuccessful for all of them.

[DSSS25]'s Table 1 features comparisons to other works that optimise the BBW energy to provide further context for our method's runtimes. The meshes and mesh names correspond directly to the ones used by [DSSS25]. We highlight here that the number of vertices or tetrahedra is not the most important factor in deciding the total runtime of the energy optimisation; factors like the choice of controls, as well as the difficulty of the geometry, are often more important.