Large Steps in Inverse Rendering of Geometry

BAPTISTE NICOLET, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
ALEC JACOBSON, University of Toronto, Canada
WENZEL JAKOB, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Inverse reconstruction from images is a central problem in many scientific and engineering disciplines. Recent progress on differentiable rendering has led to methods that can efficiently differentiate the full process of image formation with respect to millions of parameters to solve such problems via gradient-based optimization.

At the same time, the availability of cheap derivatives does not necessarily make an inverse problem easy to solve. Mesh-based representations remain a particular source of irritation: an adverse gradient step involving vertex positions could turn parts of the mesh inside-out, introduce numerous local self-intersections, or lead to inadequate usage of the vertex budget due to distortion. These types of issues are often irrecoverable in the sense that subsequent optimization steps will further exacerbate them. In other words, the optimization lacks robustness due to an objective function with substantial non-convexity.

Our method addresses both problems and converges to a high-quality mesh. Such robustness issues are commonly mitigated by imposing additional regularization, typically in the form of Laplacian energies that quantify and improve the smoothness of the current iterate. However, regularization introduces its own set of problems: solutions must now compromise between solving the problem and being smooth. Furthermore, gradient steps involving a Laplacian energy resemble Jacobi’s iterative method for solving linear equations that is known for its exceptionally slow convergence.

We propose a simple and practical alternative that casts differentiable rendering into the framework of preconditioned gradient descent. Our preconditioner biases gradient steps towards smooth solutions without requiring the final solution to be smooth. In contrast to Jacobi-style iteration, each gradient step propagates information among all variables, enabling convergence using fewer and larger steps.

Our method is not restricted to meshes and can also accelerate the reconstruction of other representations, where smooth solutions are generally expected. We demonstrate its superior performance in the context of geometric optimization and texture reconstruction.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:

1 INTRODUCTION

Differentiable rendering is an emerging tool for solving challenging inverse problems involving light transport. Methods in this area propagate derivatives through the full process of image formation to minimize a user-provided objective function defined on a high-dimensional space of scene parameters. The resulting derivatives encode the complex and ambiguous relationship of rendered pixels to light sources, the material, and the shape of observed objects. Physically based variants furthermore account for interreflection to compute derivatives due to indirectly observed objects. Compared to existing 3D reconstruction techniques, the main allure of differentiable rendering lies in its comparative lack of assumptions and potential to outperform standard methods in challenging situations.

Yet, the availability of derivatives is no panacea: gradient-based optimization of a non-convex objective can easily converge to local minima representing low-quality solutions. Notably, this situation almost always occurs when the scene contains mesh-based geometric representations. Optimizing a crude initial guess (e.g. a sphere) will necessarily require large-scale deformation, which manifests in the form of silhouette gradients that “pull” a sparse set of polygons into a target configuration, for example towards silhouettes observed in a reference photograph. A large gradient step could then turn part of the object inside-out, introduce numerous local self-intersections, or distort the triangulation so that the given vertex budget is not effectively used.

When the optimization relies on a pure rendering loss, such issues are essentially irrecoverable: geometric distortion is generally invisible in renderings, and unobserved inverted geometry does not generate gradients at all! Since there is no inherent correction mechanism, later gradient steps are likely to compound existing issues. Seen in another way, this fragility indicates that the non-convex objective is simply not amenable to gradient-based optimization. Such issues are not restricted to sparse shape-related gradients: for example, noisy derivatives of Monte Carlo methods can cause many similar problems.

A common way to mitigate such robustness issues is to make the optimization aware of the mesh quality, typically by imposing regularization in the form of Laplacian or bi-Laplacian smoothness energies. This certainly helps to stabilize the optimization, but it also introduces a new set of problems: solutions must now compromise between satisfying the original objective and being smooth. Regularization also requires a weighting factor, which adds a challenging hyper-parameter choice. In cases where the output contains both smooth and non-smooth regions, there may not be a good global setting for this parameter.

A third issue appears when first-order gradients are used to optimize a discrete Laplacian energy. In this case, each variable generates derivative terms that only affect variables in a local neighborhood: the 1-ring in the case of common discrete Laplacian operators for meshes. Such local information exchange along edges has been studied in iterative methods for solving sparse linear systems including the Jacobi and Gauss-Seidel methods. Both are known for their exceptionally slow convergence precisely due to this inherent locality.

We propose a simple and practical alternative to Laplacian regularization for differentiable rendering that is more robust, less sensitive to hyperparameter choices, and which accelerates convergence at equal time. Our method can be alternatively interpreted as casting differentiable rendering into the framework of Sobolev preconditioned gradient descent or as a re-parameterization of the input geometry resembling differential coordinates [Sorkine 2006]. Our preconditioned optimization biases gradient steps towards smooth solutions, but it does so without requiring the final solution to be smooth. In contrast to Jacobi-style iteration, each gradient step propagates information among all variables, enabling convergence using fewer and larger steps.

Our method solves a sparse linear system at every iteration, which has negligible cost compared to the primal and differential rendering phases. It works on any domain with a suitable discrete Laplacian operator, and we experimentally evaluate its performance in the context of geometric optimization and texture reconstruction.

The name of our submission is inspired by an influential article by Baraff and Witkin [1998] that pioneered the use of implicit timesteps to improve the robustness of cloth simulation. Our method admits a similar interpretation as an implicit timestep of a diffusion process.

2 BACKGROUND AND RELATED WORK

We now discuss relevant prior work and review fundamentals concerning the Laplacian and its applications.

2.1 Differentiable Rendering

Rendering is increasingly used as a sub-component within methods that rely on optimization to accomplish a particular task. For example, methods in the analysis-by-synthesis category [Patow and Pueyo 2003] reconstruct the shape and appearance of objects from images, using rendered candidate images to update a virtual scene representation. Embedded within a neural encoder-decoder architecture, a rendering step can convert latent scene representations into images that are consumed by convolutional layers [Genova et al. 2018]. The scene could itself be represented by a neural network that is queried many times during the rendering process [Mildenhall et al. 2020]. All of these applications involve high-dimensional parameter domains requiring first-order optimization, which has led to renewed interest in rendering methods with an explicit differentiation operation.

Suppose that a rendering algorithm is represented as a function $R$ that maps an input vector of scene parameters $x \in \mathbb{R}^n$ to a rendered image $y \in \mathbb{R}^p$. If $R$ accounts for global illumination, an individual scene parameter can often affect the entire image, hence the Jacobian matrix $J_R = \frac{\partial R}{\partial x} \in \mathbb{R}^{p \times n}$ is generally dense. Both of $n$ and $p$ could be in the millions, making explicit computation or storage of this matrix infeasible. Instead, differentiable rendering algorithms realize efficient matrix-vector products involving this matrix without ever constructing it. Typically, reverse-mode propagation [Griewank and Walther 2008] is desired, which converts an image-space derivative $\delta y$ into a parameter derivative $\delta x$ via the product $\delta x = J_R^T \delta y$. 

Work on differentiable rendering algorithms falls into two broad categories: starting with the early work of Loper et al. [2014] rasterization-based methods account for primary visibility and local shading [Kato et al. 2018; Liu et al. 2019; Ravi et al. 2020; Laine et al. 2020]. They achieve excellent performance but do not support indirect effect like shadows and interreflection. Another line of work targets derivatives of physically-based light transport. Visibility-induced discontinuities pose a challenge in physically-based rendering methods and require special treatment to avoid bias [Li et al. 2018; Loubet et al. 2019; Bangaru et al. 2020; Zhang et al. 2020]. Finally, integrals evaluated within rendering algorithms change following differentiation, and recent work [Zeltner et al. 2021; Zhang et al. 2021] has investigated specialized Monte Carlo sampling strategies that account for this.

This article focuses on shape reconstruction, where global illumination plays a lesser role, and we therefore use the differentiable rasterizer of Laine et al. [2020] in our experiments. Section 4 also investigates the behavior of our method in a physically based renderer based on Monte Carlo integration.

2.2 The Laplace Operator

The Laplace operator is the workhorse of modern geometry processing, and familiarity is implicitly assumed by most literature in this area. Since the main audience of this article are researchers and practitioners in the area of (differentiable) rendering, we include a review of relevant definitions and properties.

The Laplacian $\Delta$ is one of the elementary differential operators; it arises in countless physical problems including the diffusion of heat, electrostatic potentials, and incompressible fluid flow. On an $n$-dimensional Euclidean domain, its definition reads

$$\Delta f = \frac{\partial^2 f}{\partial x_1^2} + \cdots + \frac{\partial^2 f}{\partial x_n^2}. \quad (1)$$

The Laplacian possesses a well-studied set of eigenvalues and eigenfunctions. On a 1D interval such as $[0, 1]$, the eigenfunctions are sinusoidal oscillations, whose frequency increases as $i \to \infty$:

$$\lambda_i = \pi^2 i^2, \quad f_i(x) = \sqrt{2} \cos(i \pi x). \quad (i = 0, \ldots) \quad (2)$$

They provide a convenient orthonormal frequency basis of the underlying domain, and this behavior also carries over to other kinds of Laplacian operators, e.g. on curved surfaces. This is the foundation of Fourier analysis on such general domains.

Two other aspects are relevant: the first eigenvalue $\lambda_0$ equals zero, which simply shows that $\Delta$ maps constant functions to zero. Next, the eigenvalue $\lambda_1$ has a quadratic dependence on the index $i$. In other words, $\Delta$ greatly amplifies the magnitude of high-frequency signals. Conversely, a hypothetical inverse operator $\Delta^{-1}$ would greatly attenuate the magnitude of high frequencies. This frequency-dependent attenuation will be a key ingredient of our method.

The Dirichlet energy $E(f)$ can be used to quantify the overall smoothness of a function $f$ on a Euclidean domain $\Omega$. It is defined as the integrated magnitude of the function’s gradient, i.e.,

$$E(f) := \frac{1}{2} \int_{\Omega} \|\nabla f\|^2 \, dx, \quad (3)$$

which can be cast into an inner product involving the Laplacian:

$$\frac{1}{2} \int_{\Omega} \nabla f : \nabla f \, dx = C - \frac{1}{2} \int_{\Omega} f : \Delta f \, dx = C - \frac{1}{2} \langle f, \Delta f \rangle. \quad (4)$$

where the effectively constant term $C$ depends only on the boundary conditions (i.e. $f(\partial \Omega)$ and $\nabla f(\partial \Omega)$); it is therefore usually ignored in definitions of the Dirichlet energy.

The heat equation is a partial differential equation of the form

$$\frac{df(t, x)}{dt} = \Delta_x f(t, x), \quad (5)$$

where the Laplacian is taken with respect to the spatial coordinate. This equation models the diffusion of heat within a solid material, as represented by a temporally and spatially varying function $f(t, x)$. As $t \to \infty$, $f$ becomes progressively smoother and eventually approaches an equilibrium. This equilibrium solution is uniquely defined and has the smallest possible Dirichlet energy.

2.3 Discrete Laplacians

Consider a polygonal mesh $M = (V, E)$ with a set of $n$ vertices and $m$ edges. A generalized discrete Laplacian operator $L$ on this mesh can be defined as follows:

$$(L)_{ij} = \begin{cases} -w_{ij}, & \text{if } (i, j) \in E \\ \sum_{(i, k) \in E} w_{ik}, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

The weights $w_{ij} \in \mathbb{R}$ discretize the first derivative along an edge, and the signed addition of multiple first derivatives within $L$ extends the notion of a second derivative to signals on $M$.

Analogous to the smooth Dirichlet energy $E(f) = -\frac{1}{2} \langle f, \Delta f \rangle$, we can now also define a Dirichlet energy in terms of $L$:

$$E(f) = \frac{1}{2} \langle f, Lf \rangle = \frac{1}{2} f^T L f. \quad (7)$$

It generalizes the notion of smoothness to discrete signals $f$ that are sampled at vertices of the curved domain $M$. The missing negation is due to a different sign convention used for graph Laplacians.

Various weights $w_{ij}$ can be chosen to obtain Laplacian operators with slightly different properties. The combinatorial Laplacian is based only on the topology of the input graph and sets $w_{ij} = 1$ for all edges $(i, j) \in E$. Previous works [Nealen et al. 2006; Botsch and Kobbelt 2004b] have observed its good properties for mesh optimization. The cotangent Laplacian [Pinkall and Polthier 1993; Dziuk 1988] is derived by integrating the Laplace-Beltrami operator over Voronoi regions yields a more accurate discretization of the mean curvature flow, albeit with adverse effects on mesh quality [Kazhdan et al. 2012]. We refer to Botsch et al. [2010] for a review of Laplacian mesh processing.
2.4 Sobolev Preconditioning


Our method is also related to Active Surface Models that optimize a deformable mesh subject to extrinsic forces and an intrinsic smoothness energy [Terzopoulos et al. 1988]. Motivated by this approach, Wickramasinghe et al. [2021] applied semi-implicit regularization to a graph neural network for surface reconstruction and volume segmentation. In contrast, our goal is to accelerate convergence without the typical compromises that regularization entails.

3 METHOD

Our goal is to determine a methodology for taking large descent steps in mesh optimization problems using differentiable rendering. To this end, we defer generalizations and modifications, and explain our proposed technique by considering a model problem:

$$\min_{x \in \mathbb{R}^{n \times 3}} \Phi(R(x)), \quad (8)$$

where $x \in \mathbb{R}^{n \times 3}$ collects mesh vertex positions along rows, and $\Phi$ is a loss function measuring the reconstruction accuracy of images produced by a renderer $R$ (we use a $L_1$ loss in our experiments). In principle, the function $\Phi$ could also examine its input in some other way, e.g., using a trained neural computation. The relevant property is that both $R$ and $\Phi$ are differentiable almost everywhere. We assume here that $R$ renders images with known camera poses.

3.1 Conventional Gradient Descent

Given initial mesh vertex positions, we may attempt to optimize the problem in Equation 8 by applying gradient descent:

$$x \leftarrow x - \eta \frac{\partial \Phi}{\partial x}, \quad (9)$$

where $\eta > 0$ is the scalar step length or learning rate.

Leveraging reverse-mode differentiation, the per-vertex gradients $\frac{\partial \Phi}{\partial x} \in \mathbb{R}^{n \times 3}$ can be efficiently computed (Section 2). For inducing high-frequency changes based purely on shading, conventional gradient descent can be sufficient to induce the necessary small displacements (see, e.g., Liu et al. [2018]). For reconstruction problems, the initial shape (e.g., a sphere) may be very far from optimal, resulting in silhouette mismatches. Differentiation of $R$ (e.g., via reverse-mode autodiff) then produces two very different gradient terms: shading gradients and silhouette gradients (see Fig. 2). Shading gradients account for discrepancies in surface normals and tend to be small and uniformly distributed when measured on mesh vertices.

Meanwhile, silhouette gradients account for gross discrepancies and concentrate large but sparse gradient terms on vertices composing the shape’s silhouette in each image (under mild assumptions, a set of size $O(\sqrt{n})$). Naively following silhouette gradients quickly leads to an unrecoverable tangled mesh (see Figure 1), where shading gradients have no opportunity to help improve. Decreasing the step-size $\eta$ will delay this process, but the mesh nevertheless becomes highly distorted by the time silhouettes are resolved enough for shading gradients to dominate.

The following toy example illustrates the problematic nature of sparse gradients in mesh optimization. Here, a 1D “mesh” made of evenly-spaced line segments on the interval $[-1, +1]$ is optimized to match a reference shape covering the smaller interval $[-1/2, +1/2]$. The first and third images show positions (vertex indices on the horizontal axis, positions on the vertical axis), and the second and third image show gradients due to silhouette misalignment.

The first step of gradient descent pulls the outermost vertices inward, but it moves them too far so that they end up within the “interior” of the 1D shape. A subsequent gradient step targets the next outer pair of vertices, and this pattern repeats to yield a tangled shape with multiple inverted elements.

3.2 Regularization

In geometry processing, the conventional approach to tame tangled meshes due to large variations is to append a Laplacian regularization objective, altering the original optimization in Equation 8 to:

$$\min_{x \in \mathbb{R}^{n \times 3}} \Phi(R(x)) + \frac{\lambda}{2} \tr(x^T L x), \quad (10)$$

where $L \in \mathbb{R}^{n \times n}$ is a sparse, symmetric positive definite Laplacian matrix that discretizes Dirichlet energy [Solomon et al. 2014]. Without loss of generality, $L$ also could be a power of the Laplacian, such
as squared Laplacian energy or bi-Laplacian [Botsch and Kobbelt 2004a; Jacobson et al. 2010]. The parameter $\lambda > 0$ balances between the original objective and regularization. Accordingly, the gradient descent update changes to:

$$
\mathbf{x} \leftarrow \mathbf{x} - \eta \left( \frac{\partial \Phi}{\partial \mathbf{x}} + \lambda L \mathbf{x} \right).
$$

(11)

For large $\lambda$, the regularization term dominates and the descent behaves as a forward Euler integration of mean curvature flow, known to be unstable even for mild step sizes [Deshpande et al. 1999]. For small $\lambda$, the effect of the regularization is, of course, diminished by this scale factor, but also by the same root cause of the forward Euler instability: the gradient contribution $L \mathbf{x} \in \mathbb{R}^{n \times 3}$ is a highly local action. Mesh Laplacians have the connectivity of an adjacency matrix. Consider multiplying $L$ against a set of positions $\mathbf{x}$ which are constant except for a “blip” at one vertex. The result will be zero everywhere except the immediate neighbors of the blip. Under favorable assumptions, $O(\sqrt{n})$ gradient descent iterations will be necessary to propagate information of any one vertex to all other vertices of a mesh.

Examining our toy 1D example, we observe oscillatory gradients in the second iteration that attempt to correct distortion introduced by the first step:

Appending a regularization term diffuses sparse gradients, but only locally and dependently on the mesh resolution.

### 3.3 Second-order Optimization

If Laplacian regularization is such a disappointment for gradient descent, then why is it so popular in geometry processing? The answer is in that subfield it is much more common to apply second-order optimization such as Newton’s method. As an important special case, if all other objective terms are quadratic convex functions of $\mathbf{x}$ — like for instance $\text{tr} (\mathbf{x}^T L \mathbf{x})$, then a single step of Newton’s method leads to the global optimum. What if we tried to apply Newton’s method to the regularized problem in Equation 10? The resulting update is given by

$$
\mathbf{x} \leftarrow \mathbf{x} - \eta \left( \frac{\partial^2 \Phi}{\partial \mathbf{x}^2} + \lambda L \right)^{-1} \left( \frac{\partial \Phi}{\partial \mathbf{x}} + \lambda L \mathbf{x} \right),
$$

(12)

where the Hessian of the regularization term is simply the Laplacian matrix $L$. Examining our toy 1D example, a single Newton iteration suffices to smoothly move all vertices into essentially the right place:

The second step is small in magnitude and relaxes the interior while correcting the endpoint positions. Unfortunately, the Hessian of the differentiable rendering term is far too complicated:

$$
\frac{\partial^2 \Phi}{\partial \mathbf{x}^2} = \frac{\partial^2 \Phi}{\partial \mathbf{R}^2} \frac{\partial \mathbf{R}}{\partial \mathbf{x}} + \frac{\partial \Phi}{\partial \mathbf{R}} \left( \frac{\partial \mathbf{R}}{\partial \mathbf{x}} \right)^2.
$$

(13)

These second-order terms are computationally expensive and delicate to compute. They are unavailable in many automatic differentiation systems. If the renderer $\mathbf{R}$ accounts for global illumination, transparency, or has differentiable lighting parameters, the resulting matrix $\frac{\partial^2 \Phi}{\partial \mathbf{x}^2}$ may be dense. Requiring $O(n^2)$ storage and $O(n^3)$ computation for linear solving per iteration, this is intractable for even modest mesh resolutions.

The role of $(\frac{\partial^2 \Phi}{\partial \mathbf{x}^2} + \lambda L)^{-1}$ can be understood as diffusing the (potentially sparse and concentrated) gradient updates over the entire domain. Our key idea is to achieve this diffusion without incurring the cost of a full-blown second-order optimization.

### 3.4 Modified Gradient Descent

Our proposed modification of the gradient descent update is to precondition the original objective gradients with a convex combination of the identity matrix $I$ and the mesh Laplacian $L$:

$$
\mathbf{x} \leftarrow \mathbf{x} - \eta (I + \lambda L)^{-p} \frac{\partial \Phi}{\partial \mathbf{x}}.
$$

(14)

where $p \geq 1$ further controls the amount of diffusion and later enables a useful reparameterization interpretation that invites tuning of momentum contributions. For now, it suffices to consider $p = 1$, though ultimately our final update rule will be more analogous to $p = 2$. Since $I + \lambda L$ is sparse, the associated linear system can be solved efficiently. In geometry processing, a standard choice for $L$ would be the cotangent Laplacian [Pinkall and Polthier 1995], which depends on edge lengths besides the discrete mesh connectivity. A potential disadvantage of this construction is that the coupling between Laplacian and evolving vertex positions can introduce singularities during descent [Kazhdan et al. 2012]. The combinatorial Laplacian lacks this dependence and furthermore has been shown to promote more regular tessellations [Nealen et al. 2006]. Counter to these observations, we did not observe singularities or noticeable qualitative differences between cotangent and combinatorial Laplacian in our experiments. Given the qualitative similarity, we prefer the combinatorial Laplacian mainly for its computational efficiency: thanks to the purely topological dependence, a potentially expensive factorization of Equation 14 can be reused across iterations.

The gradients resulting from our formulation diffuse the sparse silhouette gradients similar to the Newton step:

The overall step size is more approximate: a full step ($\eta = 1$) is usually ideal in a second-order method, whereas momentum or a line search to determine $\eta$ is needed in our case.

There are various ways of interpreting our proposed approach.
Quasi-Newton method. Our formulation can be viewed as a modification of the full Newton’s update in Equation 12, where the exact Hessian of the original objective in the gradient preconditioner is replaced with the identity L, i.e.,
\[ x \leftarrow x - \eta \left( I + \lambda L \right)^{-1} \frac{\partial \Phi}{\partial x} + \lambda L \mathbf{x}, \]  
(15)
the remaining difference being the presence of the regularization in the gradient term \( + \lambda L \mathbf{x} \). This modified update accelerates convergence of the regularized update from Equation (11) and will therefore still converge to a regularized solution. In contrast, our goal is to benefit from accelerated convergence without compromising on minimizing \( \Phi \), hence we furthermore replace the last term in parentheses with the pure rendering gradient \( \frac{\partial \Phi}{\partial x} \). Omitting this step retains the behavior of regularization, while stabilizing convergence with large step sizes. In this way, we may view our descent as a form of Sobolev or inverse-Laplacian preconditioning.

Diffusion reparameterization. A single implicit Euler step of coordinate-wise heat diffusion of any vector-valued quantity \( \mathbf{u} \in \mathbb{R}^{n \times 3} \) over the mesh as a graph has the form
\[ \arg \min_{\mathbf{x}} \frac{1}{2} \| \mathbf{x} - \mathbf{u} \|^2 + \lambda \frac{1}{2} \text{tr} \left( \mathbf{x}^\top L \mathbf{x} \right), \]  
(16)
whose solution is revealed by the Euler-Lagrange equation:
\[ \mathbf{x} = (I + \lambda L)^{-1} \mathbf{u} \]  
(17)
In this view, \( \lambda \) may be seen as the temporal duration of diffusion. For a fixed Laplacian \( L \), the Jacobian of \( \mathbf{x} \) as a function of the introduced variables \( \mathbf{u} \) is simply our inverted matrix expression:
\[ \frac{\partial \mathbf{x}}{\partial \mathbf{u}} = (I + \lambda L)^{-1}. \]  
(18)
Thus, an alternative interpretation of our update in Equation 14, is conventional gradient descent based on a reparameterized problem \( \min \Phi(R(x(u))) \) involving a single-step of implicit heat diffusion via the chain rule:
\[ \mathbf{u} \leftarrow \mathbf{u} - \eta \frac{\partial \Phi}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathbf{u}} \]  
(19)
Together with Equation 17, the corresponding update rule for the positions \( \mathbf{x} \) reduces to
\[ \mathbf{x} \leftarrow (I + \lambda L)^{-1} \left( \mathbf{u} - \eta \frac{\partial \Phi}{\partial \mathbf{x}} \right) = \mathbf{x} - \eta (I + \lambda L)^{-2} \frac{\partial \Phi}{\partial \mathbf{x}} \]  
(20)
which is equivalent to Equation 14 with \( p = 2 \).

These reparameterized coordinates \( \mathbf{u} \) bear some resemblance to the differential coordinates of Lipman et al. [2004] and Sorkine [2005] defined as \( \mathbf{u} = L \mathbf{x} \), which they then use in a least-squares setting. Descent can also be applied in the parameterization by \( \mathbf{x} = L^{-1} \mathbf{u} \), which corresponds to the asymptotic case \( \lambda \rightarrow \infty \). Special care must be taken to handle the rank deficiency of \( L \), e.g., constraining the position of one vertex per connected component [Lipman et al. 2004].

Having shown the equivalence of the reparameterized and preconditioned update rules above, working with the coordinates \( \mathbf{u} \) may appear circuitous. However, we will soon see that there are subtle differences that enable further quality improvements. Regardless of the preferred interpretation, we demonstrate substantial improvement over raw and Laplacian regularized gradient descent.

Compared to gradient descent, our method diffuses concentrated silhouette gradients and smoothly moves the entire mesh. Compared to Laplacian regularization, we do not modify the original objective function and consequently, we are less sensitive to the hyper-parameter \( \lambda \). This means that the step size \( \eta \) can be significantly larger without fear of instability or mesh tangling.

3.5 Momentum and Variance

Practical usage of gradient descent often benefits from introducing momentum terms. We similarly find that this noticeably improves the quality of our results. Let us consider some descent variants before arriving at our proposed update rule. In particular, we will apply momentum modifications to the update of the parameters \( \mathbf{u} \) which linearly control the mesh positions via Equation 17.

The classic momentum variant involves the following update rule:
\[ \mathbf{g} \leftarrow (I + \lambda L)^{-p} \frac{\partial \Phi}{\partial \mathbf{x}}, \]
\[ \mathbf{m}_1 \leftarrow \beta_1 \mathbf{m}_1 + (1 - \beta_1) \mathbf{g}, \]
\[ \mathbf{u} \leftarrow \mathbf{u} - \eta \frac{\mathbf{m}_1}{1 - \beta_1^p}, \]  
(21)
where \( \mathbf{g}, \mathbf{m}_1 \in \mathbb{R}^{n \times 3} \) represent the descent step and first-order moment estimate, \( 0 \leq \beta_1 \leq 1 \) controls the decay, the power \( k \) is the iteration number, and the division by \( 1 - \beta_1^p \) conducts de-biasing.

The widely used \textsc{Adam} optimizer [Kingma and Ba 2014] extends this update rule with a component-wise preconditioning scheme based on second-order moments:
\[ \mathbf{m}_2 \leftarrow \beta_2 \mathbf{m}_2 + (1 - \beta_2) \mathbf{g}^2, \]
\[ \mathbf{u} \leftarrow \mathbf{u} - \eta \frac{\mathbf{m}_1}{1 - \beta_1^p} \left( \sqrt{\frac{\mathbf{m}_2}{1 - \beta_2^p}} + \varepsilon \right), \]  
(22)
where matrix exponentiation, division, and the square root are appropriately component-wise.

We observed that aggressive steps resulting from this component-wise preconditioner tend to disturb the smoothness of the result. We therefore prefer a more uniform adaptation and refer to this update rule as Uniform\textsc{Adam}, which differs from \textsc{Adamax} [Kingma and Ba 2014] that applies the infinity norm over time. We state our final update rules incorporating this change in full for convenience:
\[ \mathbf{g} \leftarrow (I + \lambda L)^{-1} \frac{\partial \Phi}{\partial \mathbf{x}}, \]
\[ \mathbf{m}_1 \leftarrow \beta_1 \mathbf{m}_1 + (1 - \beta_1) \mathbf{g}, \]
\[ \mathbf{m}_2 \leftarrow \beta_2 \mathbf{m}_2 + (1 - \beta_2) \mathbf{g}^2, \]
\[ \mathbf{u} \leftarrow \mathbf{u} - \eta \frac{\mathbf{m}_1}{(1 - \beta_1^p) \sqrt{\frac{\mathbf{m}_2}{1 - \beta_2^p}}}, \]  
(23)
where \[ \mathbf{x}(\mathbf{u}) = (I + \lambda L)^{-1} \mathbf{u}, \]
and again, \( \mathbf{g}^2 \) is the component-wise square.
Fig. 3. We compare several first-order optimizers with a consistent choice of two diffusion steps ($p = 2$). The step size was adjusted to obtain the best quality at equal time in each case. (1) Vanilla gradient descent produces a comparably blurry result. (2) Momentum improves quality noticeably. (3) Aggressive component-wise preconditioning in the Adam algorithm disturbs the smoothness of the output. (4) Performing the optimization on a “differential coordinate” parameterization of the mesh improves this behavior, though some non-uniformity remains. (5,6) Our UniformAdam method produces the best results, with comparable quality on both domains.

3.6 Comparison of Gradient-based Optimizers

Figure 3 compares six different gradient-based optimization schemes. This comparison also reveals an interesting point: preconditioning based on second-order moments leads to differences between optimization on $x$ versus optimization of $u$ due to the component-wise division in Equations (22) and (23). The results shown in this paper are based on an optimization of the parametric representation $u$.

3.7 Remeshing

Our method is easily combined with a remeshing step applied once or periodically during the optimization. Figures 4 and 5 showcase results obtained with the technique of Botsch and Kobbelt [2004b], by isotropically remeshing the shape with a target edge length equal to half of the current average value. Isotropic remeshing is harmonious with our use of the combinatorial Laplacian $L$, as both operations prefer a regular surface tessellation. We remesh at manually specified timesteps and decrease the step size by a factor of 0.8 each time to improve stability; finding the optimal schedule and incorporating curvature adaptivity are both interesting directions for future work.

3.8 Numerical Solution

Our method requires the solution of a sparse linear system based on the connectivity of the input mesh. This system is strictly positive definite for finite $\lambda > 0$, and the addition of an identity leads to good numerical conditioning. We experimented with two approaches: the conjugate gradient method, and a direct solver based on a fill-in reducing sparse Cholesky factorization.

The conjugate gradient method builds on matrix-vector multiplications involving the system matrix. We use a sparse matrix representation, though we note that such a multiplication could in principle also be realized on top of an existing mesh data structure.

We use CHOLMOD [Chen et al. 2008] for the second approach, which exposes numerous strategies and heuristics: we use the simplicial method and nested dissection (NESDIS) ordering, which we found to produce factors with the greatest degree of sparsity. Our reliance on a combinatorial Laplacian is beneficial here, since the Cholesky factor can be reused across iterations. Remeshing steps must, however, update the factorization to account for the updated mesh connectivity. The overhead introduced by the factorization is on the order of a few optimization steps (see Table 1).

Solving linear systems using the direct solver was always multiple orders of magnitude faster in our experiments. However, we observed
Fig. 5. Our method can benefit from periodic re-meshing steps, particularly when the input shape is too coarse, or when it has an unsuitable mesh topology that leads to distortion. This figure shows an extreme case, where we recover the Stanford dragon starting from an icosahedron. Intermediate insets visualize the shape following remeshing steps.

Fig. 6. Large gradients steps can introduce considerable non-uniformity in this single-view optimization of a sphere towards a larger reference sphere. Our method (top row) converges to a high-quality solution within a few hundred iterations, while regularization produces meshes with unsatisfactory distortion for different strengths of the regularization parameter $\lambda$. The solution eventually collapses to a point when smoothness is prioritized too much. Plots of the individual loss terms reveal how the image loss is first optimized at the expense of regularization, which then catches up once the image loss has reached its minimum. Our method does not require this trade-off between two competing objectives.

in synthetic experiments that the computation time of the one-time factorization step grows super-linearly as a function of mesh size. This was not a concern in our geometry reconstructions—however, there could be a point where the conjugate gradient method becomes competitive, and we will therefore also evaluate its performance.

4 RESULTS

We now turn to results and remaining technical details.

Large timesteps. Figure 6 optimizes the 3D mesh of a sphere from a single viewpoint so that it matches a reference image of a larger sphere, which resembles the motivational 1D examples shown in Section 3. The geometry is purely emissive, which leads to rendered images that are essentially binary. We intentionally disable shading in this way to isolate the effect of the sparse silhouette gradients.

Our method converges within a few hundred iterations (top row). We then vary the $\lambda$ parameter of the regularization baseline (next 3 rows) to show configurations where there is too little, just enough, and too much regularization. With regularization, descent eventually converges to a spherical shape of the right size, but the geometry is highly non-uniform. The shape collapses to a point when the regularization term dominates the optimization objective.

The rendering- and Laplacian loss terms in the plots on the right show how silhouette-related changes dominate the first part of the optimization, which disturbs the uniformity of the initial mesh.
Table 1: Comparison of Mesh Reconstruction Methods

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Initial State</th>
<th>Regularized (L)</th>
<th>Regularized (L²)</th>
<th>Ours (λ = 19)</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bunny</td>
<td><img src="image" alt="Bunny" /></td>
<td><img src="image" alt="Bunny" /></td>
<td><img src="image" alt="Bunny" /></td>
<td><img src="image" alt="Bunny" /></td>
<td><img src="image" alt="Bunny" /></td>
</tr>
<tr>
<td>Bob</td>
<td><img src="image" alt="Bob" /></td>
<td><img src="image" alt="Bob" /></td>
<td><img src="image" alt="Bob" /></td>
<td><img src="image" alt="Bob" /></td>
<td><img src="image" alt="Bob" /></td>
</tr>
<tr>
<td>Cranium</td>
<td><img src="image" alt="Cranium" /></td>
<td><img src="image" alt="Cranium" /></td>
<td><img src="image" alt="Cranium" /></td>
<td><img src="image" alt="Cranium" /></td>
<td><img src="image" alt="Cranium" /></td>
</tr>
<tr>
<td>Planck</td>
<td><img src="image" alt="Planck" /></td>
<td><img src="image" alt="Planck" /></td>
<td><img src="image" alt="Planck" /></td>
<td><img src="image" alt="Planck" /></td>
<td><img src="image" alt="Planck" /></td>
</tr>
<tr>
<td>Suzanne</td>
<td><img src="image" alt="Suzanne" /></td>
<td><img src="image" alt="Suzanne" /></td>
<td><img src="image" alt="Suzanne" /></td>
<td><img src="image" alt="Suzanne" /></td>
<td><img src="image" alt="Suzanne" /></td>
</tr>
<tr>
<td>T-Shirt</td>
<td><img src="image" alt="T-Shirt" /></td>
<td><img src="image" alt="T-Shirt" /></td>
<td><img src="image" alt="T-Shirt" /></td>
<td><img src="image" alt="T-Shirt" /></td>
<td><img src="image" alt="T-Shirt" /></td>
</tr>
</tbody>
</table>

Fig. 7. Inverse shape reconstruction of targets with different topology and levels of detail. We compare the quality and convergence of a regularized differentiable renderer based on Laplacian (L) and bi-Laplacian (L²) smoothness energies to our method. Results were obtained at equal-time using optimized hyperparameters. The bottom three rows visualize image loss, Dirichlet energy, and geometric (Hausdorff) distance to the ground truth. Our method reliably converges to solutions that are simultaneously geometrically uniform and close to the reference. The plots in the second-to-last row show how our preconditioning approach gradually decreases smoothness to match the reference, while regularized differentiable rendering aggressively does so in the first few iterations.

Mesh reconstruction. Figure 7 provides a comprehensive comparison of methods in a reconstruction of six meshes using both shading and silhouette gradients and observation from multiple views (details in Table 1). In some cases, the initial state of the optimization was specially adapted to the topology of the desired output. For example, the Bob optimization uses a toroidal initialization. One and four holes have been cut into the spherical initializations of the Planck and T-shirt optimizations, respectively.
The first three rows depict initialization and regularization baselines. The fourth row shows results produced by our re-parameterized update rule from Equation (23). The last three rows visualize the decay (or increase!) of the image loss, Dirichlet energy, and average bidirectional Hausdorff distance over time. The descent step size was optimized on a per-shape basis to achieve the best reconstruction quality at equal time, and is provided for each scene in Table 1. We do not change the step size during the optimization in our experiments. For the regularization baselines, we additionally optimized the regularization weight $\lambda$ on a per-shape basis, while our method uses a fixed $\lambda$ in all cases. The regularization baselines obtain the best results using a traditional Adam optimizer with additional component-wise preconditioning, while the results of our method use UniformAdam (Section 3.5).

In general, we find that our method reliably converges to high-quality meshes, while the effects of regularization are more nuanced and problem-dependent. In addition to regularization via the combinatorial Laplacian ($L$), we also examine the behavior of a bi-Laplacian ($L^2$) that captures a stronger notion of smoothness. However, even with a tuned weight and step size, we found convergence at equal time inferior the original Laplacian $L$.

The second-to-last row depicting smoothness of solutions over time reveals striking differences in the convergence behavior of the various methods: starting with a maximally smooth initialization (e.g. a perfect icosphere), our method gradually decreases smoothness as needed to introduce spatial detail. Plots of regularization-based techniques begin with a vertical cliff that is introduced when geometric gradients attempt to pull the silhouette into place.

Influence of the number of viewpoints. Figure 8 compares the reconstruction quality of our method and a regularized baseline as the number of available viewpoints increases. Naturally, more views encode additional information that can be leveraged to reduce ambiguity and improve reconstruction quality. Each viewpoint contributes information about its respective set of silhouette vertices, which reduces the sparsity of these problematic gradients. That said, we observe noticeable changes and instability in baseline reconstructions even when the number of viewpoints is relatively large.

Influence of the $\lambda$ parameter. Our method also has a tunable parameter $\lambda$ that controls the implicit time step of the diffusion process.

Influence of the number of viewpoints. Figure 8 compares the reconstruction quality of our method and a regularized baseline as the number of available viewpoints increases. Naturally, more views encode additional information that can be leveraged to reduce ambiguity and improve reconstruction quality. Each viewpoint contributes information about its respective set of silhouette vertices, which reduces the sparsity of these problematic gradients. That said, we observe noticeable changes and instability in baseline reconstructions even when the number of viewpoints is relatively large.

Influence of the $\lambda$ parameter. Our method also has a tunable parameter $\lambda$ that controls the implicit time step of the diffusion process.
For $\lambda = \infty$, preconditioning can be seen to compute the steady-state solution of the heat equation, while $\lambda = 0$ disables our method. Figure 9 illustrates equal-iteration reconstruction results using different values of this parameter. We generally use values in the range 15-50. Extremely large values (or $\lambda = \infty$) significantly dampen high-frequencies, which can impede our method’s ability to reconstruct fine details.

Texture reconstruction using Monte Carlo sampling. We initially motivated our method as an acceleration scheme for taking large steps in mesh reconstruction based on differentiable rendering. Taking stock of its assumptions and requirements, we can now recognize that its operating range extends beyond this motivating case: any optimization involving sparse or noisy gradients potentially stands to benefit given a suitable Laplacian operator to quantify smoothness on an underlying domain. This includes albedo textures, displacement or normal maps, 3D volumes (e.g. participating media), temporally varying data, and potentially even alternative implicit geometric representations like signed distance functions.

For example, consider rendering a scene containing high-resolution textures: an individual Monte Carlo rendering will normally only observe a small and random subset of texels, whose derivatives are subject to further variance owing to the stochastic simulation. Handling such noisy gradient estimates has previously required the use of small optimization steps, multi-resolution optimization, and/or regularization.

Figure 10 analyzes the benefits of our method in this setting. We use Mitsuba 2 [Nimier-David et al. 2019] to optimize the textured back wall (1024×1024 pixels) of a Cornell box-like scene with diffuse inter-reflection. The texture starts out with a constant 50% gray value, and the optimization objective measures the difference between the rendered image and a reference image shown on the top left. Renderings use a resolution of 108 × 108 pixels at 4 samples/pixel, of which only 76 × 76 pixels show the back wall.

The texture resolution greatly exceeds that of the rendered view. While the final rendering should closely resemble the target (top left), we cannot expect to recover the original texture (bottom left) given this inherent loss of information. Due to the constant injection of variance, both naive and regularized optimization via ADAM converge slowly, producing texture reconstructions that are contaminated by severe amounts of fine-scale noise. Our approach attenuates this fine-scale noise before it reaches the texture, which accelerates convergence and produces an arguably more useful solution of this highly ambiguous problem.

Animated results. Please see the supplemental video for animated convergence visualizations of NEFFERTTI, CRANIUM, the Monte Carlo texture reconstruction, and several other results.

Implementation details. All experiments in this article were performed on a Linux Ryzen 3990X workstation using a TITAN RTX graphics card. We implemented our shape reconstruction pipeline on top of the nvidia@1 module differentiable rasterizer by Laine et al. [2020], along with a spherical harmonics shading model [Ramamoorthi and Hanrahan 2001] evaluated in PyTorch [Paszke et al. 2019].

Geometric reconstruction from images normally involves observations from multiple viewpoints to reduce ambiguity. During the optimization, these viewpoints could be randomly chosen as part of a stochastic gradient descent (SGD) procedure or rendered all at once. Even when random sampling is used, it can be beneficial to process groups of viewpoints in mini-batches. In this case, preconditioning is only necessary once and can be applied to the accumulated position gradients.

In our case, relatively few fixed viewpoints are used in experiments—concrete numbers for each scene are available in Table 1. At each iteration, we render the tentative shape reconstruction from this set of viewpoints in one large batch and compute the error as the mean absolute error (MAE) across pixels of all viewpoints.

Following initialization, all steps of the computation run on the GPU: for example, we upload the sparse Cholesky factor onto the GPU and use NVIDIA’s cuSPARSE library for sparse forward- and back-substitution via the routine csrsm2 solve [Naumov 2011]. One point worth noting here is that sparse matrix libraries including CHOLMOD frequently default to double precision arithmetic—for the portion that occurs on the GPU, we found it performance-critical to ensure the use of single precision arithmetic due to the roughly $16 \times$ lower double precision throughput on current NVIDIA GPUs.

Performance. Table 1 provides further information about the size of meshes in our experiments, the number of views, and cost of the one-time factorization step. Figure 11 decomposes the per-iteration cost of the regularization baseline and two implementations of our method into primal, adjoint, and regularization or preconditioning steps. The numbers show that the additional cost of preconditioning using a sparse Cholesky factorization is negligible compared to the (significant) time spent on the primal and adjoint rendering phases. Iterative solution of the linear system using conjugate gradients is feasible, albeit at significantly higher per-iteration cost.
Differentiable rendering is a promising new tool for solving challenging inverse problems in diverse disciplines that seek to derive understanding from images. Yet, anyone who has tried working with a differentiable renderer will recall their initial bitter disappointment: inversion of even a few nontrivial parameters leads to ambiguous and non-convex objectives, causing gradient-based optimization to simply explode or find absurd solutions leveraging this ambiguity.

To render this framework practical, we must inject knowledge about the expected nature of a solution, which has traditionally involved regularization that necessarily compromises on solving the original problem. Our method represents a large step towards robust geometry optimization using a preconditioner based on smoothness to alleviate issues that arise from the derivative of the visibility in a scene.

Our method is computationally cheap, easy to implement, and it represents a large step towards robust geometric optimization even for non-smooth data and non-convex objectives, causing gradient-based optimization to simply explode or find absurd solutions leveraging this ambiguity.

Our approach also shows how going to second order in a subset of the problem can greatly improve the robustness within an overall first-order optimization. Real-world usage of differentiable rendering requires simultaneous optimization of camera pose, geometry, and textures. Like a Dirichlet energy, this tightly couples the degrees of freedom: we could, e.g., change the color of a rendered pixel by adjusting the observed texel, moving mesh vertices or UV coordinates, or by rotating the camera. These relationships are not “perceived” by first-order descent, which must take tiny steps to navigate this complex optimization landscape. Generalizing such partial use of second-order optimization to other parameter combinations is a promising direction for future work.

## 5 CONCLUSION

Table 1. This table lists the number of optimized viewpoints, vertex count, the runtime cost of the Cholesky factorization, as well as both the baseline ($\eta_1$) and our ($\eta_2$) step sizes used for each of the various shapes shown in this article. For the Dragon optimization (Figure 5), we provide the mesh resolution at the start of the optimization and after the last remeshing step.

| Shape          | Views | $|V|$  | Fact. (ms) | $\eta_1$ | $\eta_2$ |
|----------------|-------|------|------------|----------|----------|
| Bob            | 16    | 2,145| 7          | 1e-2     | 7.5e-6   |
| Bunny          | 16    | 10,242| 52         | 1e-2     | 2.5e-5   |
| Cranium        | 16    | 10,242| 52         | 1e-2     | 1.25e-5  |
| Dragon - Start | 25    | 12   | 1          | 1e-2     | 2.5e-5   |
| Dragon - End   | 25    | 76,774| 492        | 1e-2     | 1.07e-6  |
| Nefertti       | 25    | 40,492| 285        | 1e-2     | 8e-7     |
| Planck         | 16    | 10,205| 54         | 1e-2     | 7.5e-6   |
| T-Shirt        | 16    | 10,252| 37         | 1e-2     | 7.5e-6   |
| Suzanne        | 13    | 10,242| 52         | 1e-2     | 5e-6     |

**ACKNOWLEDGMENTS**

The authors would like to thank Delio Vicini for early discussions about this project, Silvia Sellán for sharing her remeshing implementation and help for the figures, as well as Hsueh-Ti Derek Liu for his advice in making the figures.

**REFERENCES**


1. **Bob** Suzanne Cranium Nefertiti

Fig. 11. We compare the per-iteration cost of our method and a baseline using regularization. Preconditioning using a sparse Cholesky factorization only involves a small extra cost compared to the two phases of the differentiable renderer. Iterative solution of the linear system using conjugate gradients is significantly slower. On the positive side, this approach requires no precomputation of a factorization and is very easy to implement. The cost of the one-time Cholesky factorization is shown in Table 1.

### Table 1. This table lists the number of optimized viewpoints, vertex count, the runtime cost of the Cholesky factorization, as well as both the baseline ($\eta_1$) and our ($\eta_2$) step sizes used for each of the various shapes shown in this article. For the Dragon optimization (Figure 5), we provide the mesh resolution at the start of the optimization and after the last remeshing step.

| Shape          | Views | $|V|$  | Fact. (ms) | $\eta_1$ | $\eta_2$ |
|----------------|-------|------|------------|----------|----------|
| Bob            | 16    | 2,145| 7          | 1e-2     | 7.5e-6   |
| Bunny          | 16    | 10,242| 52         | 1e-2     | 2.5e-5   |
| Cranium        | 16    | 10,242| 52         | 1e-2     | 1.25e-5  |
| Dragon - Start | 25    | 12   | 1          | 1e-2     | 2.5e-5   |
| Dragon - End   | 25    | 76,774| 492        | 1e-2     | 1.07e-6  |
| Nefertti       | 25    | 40,492| 285        | 1e-2     | 8e-7     |
| Planck         | 16    | 10,205| 54         | 1e-2     | 7.5e-6   |
| T-Shirt        | 16    | 10,252| 37         | 1e-2     | 7.5e-6   |
| Suzanne        | 13    | 10,242| 52         | 1e-2     | 5e-6     |

**ACKNOWLEDGMENTS**

The authors would like to thank Delio Vicini for early discussions about this project, Silvia Sellán for sharing her remeshing implementation and help for the figures, as well as Hsueh-Ti Derek Liu for his advice in making the figures.

**REFERENCES**


