COORDINATE-FREE AUTOMATIC DIFFERENTIATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Automatic differentiation (AD) is a cornerstone of modern machine learning. Since all AD frameworks implicitly fix a basis of Euclidean space to define gradients and Jacobians, they natively process mathematical objects that are arrays of a fixed set of coordinates. This makes it awkward to express objects such as rotations, volume forms, and curvatures that arise, for example, in applications of AI to science. Bespoke workarounds lack reusability. We show that (i) a single, basis-independent derivative operator can represent many geometric differential operations, and (ii) this operator can be expressed entirely in terms of existing AD primitives. This reusable transformation can be composed with JAX batching, compilation, and hardware acceleration. Geometry enters as algebraic data. We experimentally demonstrate its reuse across different manifold operators, its composition through a spacetime inverse-problem pipeline, and its use in optimization over different rotor subgroups.

1 INTRODUCTION

Automatic differentiation (AD) plays a pivotal role in modern machine learning. On the one hand, it provides convenience and efficiency: forward computations efficiently and automatically compute gradients and tangents via the chain rule. On the other hand, it is a powerful programming abstraction that turns differentiation into a function transformation: if a function is implemented using differentiable primitives, it can be turned into another function that computes its local linearization, its adjoint, or higher-order variants, and these transformed functions can be composed with batching, compilation, and optimization. One-off calculations involving derivatives become reusable software objects (Baydin et al., 2018; Bradbury et al., 2018).

Current AD systems are built around coordinate arrays, which work well if derivatives are standard Euclidean Jacobians; this is often the case for the high-dimensional scalar objectives common in deep learning. Simple coordinate arrays do not, however, directly cover other geometric objects and operators, such as fields on manifolds, rotations and rigid motions, curvature, volume forms, divergence, curl, or covariant derivatives. These objects depend on additional geometric structure, such as a metric, an orientation, or a tangent space, none of which is carried by the differential that AD computes. Consequently, in domains where such objects are central, including physics, graphics, and engineering, differentiation is often not fully automatic. We address the problem by identifying a single, common source of the difficulty. Because the differential carries no geometric structure, that structure must be supplied by hand via a representation: Christoffel symbols in a chart, cotangent weights on a mesh, a stiffness matrix for a basis. Once this representation is fixed, the resulting operator is tied to it, and the same underlying operator must be re-derived for each new space. From a mathematical point of view, basis dependence is not inherent: coordinate-free formalisms, such as differential forms and exterior calculus, avoid this by describing geometric objects and operators independently of any particular coordinates.

We extend the AD philosophy of composable function transformations from flat-space AD to a fully coordinate-free formalism. The formalism used here is known as geometric algebra (Doran & Lasenby, 2003; Hestenes, 1999). It (i) provides a coordinate-free multivector derivative that incorporates all relevant metric and orientation information algebraically, and (ii) aligns with the geometric-algebra representations already used in parts of machine learning (Brandstetter et al., 2023; Brehmer et al., 2023; Ruhe et al., 2023b). Concretely, our contributions are:

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

- We show how the multivector derivative can be represented in terms of JAX primitives. It can therefore be implemented as a composable JAX transformation that supports forward and reverse mode AD, higher-order differentiation, and batched evaluation.
- By specializing this derivative, we obtain automatic computation of curl, divergence, covariant derivatives, shape operators, curvature tensors and related operators on any (pseudo-) Riemannian manifold, in any number of dimensions.
- We demonstrate (i) the framework's feasibility and (ii) reuse across applications in physics and machine learning, such as divergence-free flows on manifolds, electromagnetic inverse problems, and rotor-group optimization.

Just as `jax.grad` transforms a scalar function into its Euclidean gradient, our framework transforms a multivector function into its coordinate-free derivative. Because it is built entirely from JAX primitives, it composes with other JAX transformations and models.

2 Related Work

Automatic differentiation computes derivatives of programs by systematic application of the chain rule (Baydin et al., 2018; Griewank & Walther, 2008) and is implemented in modern systems such as JAX, PyTorch, and TensorFlow (Abadi et al., 2016; Bradbury et al., 2018; Paszke et al., 2019). More mathematical accounts make the role of coordinates explicit, and relate forward and reverse mode to tangent and cotangent constructions (Betancourt, 2018; Elliott, 2018). Our work keeps the composable transformations of modern AD, but replaces the Euclidean coordinate derivative with a coordinate-free geometric derivative.

Existing extensions of AD beyond plain Euclidean arrays include forward-mode AD for multivector-valued functions (Tingelstad & Egeland, 2017a;b; 2018), manifold and Lie-group AD methods that use e.g. charts or retractions to differentiate structured spaces (Deray & Solà, 2020; Solà et al., 2021; Sommer et al., 2016), and recent work on jet functors and Weil algebras in JAX (Sangha, 2025). In these methods, the geometry enters through a chosen local representation of the space being differentiated. In our method, the derivative is first defined algebraically on multivectors; manifold structure enters afterward through projection onto the tangent space.

Geometric algebra, or Clifford algebra, combines the inner and outer products into the geometric product (Hestenes, 1966; 1999). Geometric calculus develops the associated multivector derivative and recovers many operators of Riemannian differential geometry without charts or Christoffel symbols (Doran, 1994; Hestenes, 1968; Lasenby et al., 1993). It is closely related to other coordinate-free formalisms such as differential forms and exterior calculus, and complementary to discretized approaches such as DEC and FEEC (Arnold et al., 2010; Crane et al., 2013; Desbrun et al., 2005; Hirani, 2003). A broad ecosystem of GA libraries exists, e.g. (Bromborsky & Pygae community, 2019; Hadfield et al., 2013; Hoogendoorn, 2025; Kahlow, 2020a;b; Roelfs, 2025).

Geometric algebra in machine learning Geometric algebra has appeared in machine learning as a representation language for equivariant architectures, including Clifford neural layers, geometric Clifford algebra networks, the Geometric Algebra Transformer, and Clifford-steerable CNNs (Brandstetter et al., 2023; Brehmer et al., 2023; de Haan et al., 2024; Ruhe et al., 2023a;b; Spinner et al., 2024; Zhdanov et al., 2024). These works are not concerned with geometric calculus or automatic computation of differential operators; rather, they show that multivector representations are already useful in ML and provide natural downstream settings for our AD operators.

3 The Multivector Derivative

This section describes our approach. We define the multivector derivative and then translate this derivative into AD primitives. This allows us to compute a range of differential operators in "flat" spaces. In Section 3.5, we generalize the approach to manifolds via projections onto tangent spaces.

3.1 DEFINITIONS

We begin with an n-dimensional vector space V, with an inner product $\langle \cdot, \cdot \rangle$. The *outer product* $a_1 \wedge \cdots \wedge a_k$ of k vectors represents an oriented, k-dimensional volume element if the vectors are linearly independent (and vanishes otherwise), and is called a *k-blade*. A sum of k-blades is a grade k multivector (also called a k-vector). Geometric algebra adds an additional product operation to V, called the *geometric product* and given by

$$ab = \langle a, b \rangle + a \wedge b \qquad \text{for any } a, b \in V \ . \tag{1}$$

Observe this implies $\langle a, b \rangle = \frac{1}{2}(ab + ba)$ and $a \wedge b = \frac{1}{2}(ab - ba)$. Closing V under this product produces the *geometric algebra* $\Lambda(V)$. This is a vector space that contains the original vectors, but also oriented area elements, volume elements, and sums of these objects. If $b_1, \ldots, b_n$ is a basis of V, a basis of $\Lambda(V)$ can be obtained by forming every possible k-blade from the n basis vectors

$$b_J \ := \ b_{j_1} \wedge \ldots \wedge b_{j_k} \quad \text{for } J := \{j_1, \ldots, j_k\} \subset [n] \ .$$

We denote the subspace of grade-k elements $\Lambda^k(V)$, and the grade-k part of X by $\langle X \rangle_k$. The inner product $\langle \cdot, \cdot \rangle$ extends from V to all of $\Lambda(V)$ as the bilinear, scalar-valued form $\langle A, B \rangle := \langle AB \rangle_0$, and $\wedge$ extends by grade selection $A \wedge B := \langle AB \rangle_{p+q}$ for $A \in \Lambda^p(V), B \in \Lambda^q(V)$. We also define the *contraction* $A \rfloor B := \langle AB \rangle_{q-p}$, and is 0 if $q - p < 0$. These definitions are all standard; see e.g. Hestenes (1999); Macdonald (2017).

Any element X of $\Lambda(V)$ can now be represented by expanding it in the basis above as

$$X = \textstyle\sum_{J \in \mathcal{I}} X_J b_J \qquad \text{for some } \mathcal{I} \subset 2^{[n]} \ ,$$

for scalar coefficients X_J, called *coordinates*. We can always, of course, choose $\mathcal{I} = 2^{[n]}$ and set $X_J = 0$ if b_J does not contribute, but explicitly using a smaller set $\mathcal{I}$ can be useful to restrict attention to certain types of objects. For example, if we choose $\mathcal{I}$ as the set of singleton sets $\{1\}, \ldots, \{n\}$, then X is an element of the original vector space $V = \Lambda^1(V) \subset \Lambda(V)$.

Remark 1. We are interested in low dimensional geometry. In high dimensions, computational costs can become prohibitive, since the space $\Lambda(V)$ has dimension 2^n.

3.2 THE MULTIVECTOR DERIVATIVE

We consider *multivector functions* $F : \Lambda(V) \to \Lambda(V)$, or functions defined on the linear subspace $W \subset \Lambda(V)$ spanned by the basis blades $\{b_J\}_{J \in \mathcal{I}}$ for some index set $\mathcal{I} \subset 2^{[n]}$. A nondegenerate $\langle \cdot, \cdot \rangle$ on W implicitly defines a reciprocal basis $\{b^J\}$ satisfying $\langle b^J, b_K \rangle = \delta_{JK}$. The *multivector derivative* of F indexed by $\mathcal{I}$ is defined as

$$\partial_{\mathcal{I}} F := \textstyle\sum_{J \in \mathcal{I}} b^J \dfrac{\partial F}{\partial X_J} \qquad \text{for } \mathcal{I} \subset 2^{[n]} \text{ and } X = \textstyle\sum_{J \in \mathcal{I}} X_J b_J \ . \tag{2}$$

The multiplication in the sum is the geometric product. Four properties of $\partial_{\mathcal{I}}$ are worth emphasizing:

- $\partial_{\mathcal{I}}$ is coordinate-free. Eq. 2 depends only on this inner product (which defines the $\{b^J\}$), W, and the geometric product, but not on the particular choice of basis $\{b_J\}_{J \in \mathcal{I}}$ spanning W (Appendix A.1). The only requirement is that the inner product be non-degenerate on W.
- $\partial_{\mathcal{I}} F$ is intrinsic to $\Lambda(V)$, in contrast to the Jacobian matrix of a function on $\mathbb{R}^n$, which lives in $\mathbb{R}^{n^2}$. In fact, $\partial_{\mathcal{I}} F$ can be a defined as a projection $\mathrm{End}(\Lambda(V)) \to \Lambda(V)$.
- All differential operators relevant to our purposes are special cases of $\partial_{\mathcal{I}}$.
- As we will see below, $\partial_{\mathcal{I}}$ can be mapped to JAX primitives.

3.3 FROM THE DEFINITION TO AD PRIMITIVES

The derivative above can be implemented using AD primitives for Jacobian-vector products. In JAX, the relevant primitives are `jvp` and `vjp`. We stylize notation slightly, and write $\texttt{jvp}(F, X, Z)$ for `jvp(F, (X,), (Z,))`, etc. The Jacobian-vector product computed by forward-mode AD is the directional derivative, or differential applied to a tangent vector:

$$\texttt{jvp}(F, X, A) = dF_X[A] := \frac{d}{dt} F(X + tA)\Big|_{t=0}$$

of F at X in direction A. Reverse mode is more subtle; the vector-Jacobian product is meant to pull back an element of $\Lambda(V)^*$, a linear functional on $\Lambda(V)$. While this pullback is naturally defined (metric-free), AD stores this map as coordinate data where it assumes $\Lambda(V) = \Lambda(V)^*$. For more general settings, we require the *musical isomorphisms* $\flat : \Lambda(V) \to \Lambda(V)^*$ and $\sharp : \Lambda(V)^* \to \Lambda(V)$, which are the identity on coordinates only when all b_J are their own reciprocals. With these maps, we can define a "metric aware" vector-Jacobian product as:

$$\texttt{vjp}_{\langle\cdot,\cdot\rangle}(F, X, C) := \texttt{vjp}(F, X, C^\flat)^\sharp = dF_X^*[C] \quad \text{where} \quad \left\langle dF_X^*[C], A \right\rangle = \left\langle C, dF_X[A] \right\rangle . \tag{3}$$

With this reverse-mode primitive defined, we introduce the following elementary lemma, which shows how the multivector derivative arises from these primitives.

Lemma 2 *Let $F : \Lambda(V) \to \Lambda(V)$ be a differentiable function. Let $\{b_J\}_J$ be basis blades, with reciprocal blades $\{b^J\}_J$, and $\mathcal{I} \subset 2^{[n]}$. If $X = \sum_{J\in\mathcal{I}} X_J b_J$ ranges over the linear subspace of $\Lambda(V)$ defined by $\mathcal{I}$, then*

$$\partial_{\mathcal{I}} F(X) = \textstyle\sum_{J\in\mathcal{I}} b^J \, \texttt{jvp}(F, X, b_J) \, .$$

Similarly, if $\mathcal{O}$ indexes a set of basis blades in the codomain of F, then

$$\partial_{\mathcal{O}} F(X) = \textstyle\sum_{K\in\mathcal{O}} \texttt{vjp}_{\langle\cdot,\cdot\rangle}(F, X, b^K)\, b_K \, .$$

See Appendix A for the proof and Appendix B for implementation details.

Remark 3. If $\mathcal{I}$ contains all input directions and $\mathcal{O}$ all output directions present in F, then $\partial_{\mathcal{I}} = \partial_{\mathcal{O}}$, but costs may differ: Forward mode makes $O(|\mathcal{I}|)$ `jvp` calls, reverse mode $O(|\mathcal{O}|)$ `vjp` calls. If F is scalar, for example, then $|\mathcal{O}| = 1$ and a single `vjp` suffices, recovering the familiar reverse-mode advantage of standard `jax.grad`.

3.4 The Vector Derivative

We are particularly interested in the case $\mathcal{I}_1 := \{\{1\}, \ldots, \{n\}\}$, where all indices are singletons. In this case, $\partial_{\mathcal{I}_1}$ becomes the *vector derivative* $\partial_{\mathcal{I}_1} := \sum_{i=1}^n b^i \, \partial/\partial x_i$. If we take the geometric product of $\partial_{\mathcal{I}_1}$ and a grade-k function F, Eq. 1 shows that the result splits into exactly two grades,

$$\partial_{\mathcal{I}_1} F = \underbrace{\partial_{\mathcal{I}_1} \wedge F}_{\text{grade } k+1} + \underbrace{\partial_{\mathcal{I}_1} \rfloor F}_{\text{grade } k-1} \, . \tag{4}$$

The grade-raising part $\partial_{\mathcal{I}_1} \wedge F$ is the exterior derivative. If F is scalar valued, this is the ordinary gradient. If F is vector valued, $\partial_{\mathcal{I}_1} \wedge F$ is the curl-like component of F, and grade-lowering summand $\partial_{\mathcal{I}_1} \cdot F$ is the divergence-like component (Hestenes, 1968; Hitzer, 2002; Lasenby et al., 1993; Macdonald, 2017). The reason the case of the vector derivative is of primary importance is for computations on manifolds, as we expound on next.

3.5 From Flat Spaces to Manifolds

The approach developed so far assumes a "flat" ambient space. A simple construction generalizes it to curved manifolds. Let M be an m-dimensional manifold. We will represent M as a "vector manifold" (Hestenes, 2011; Hestenes & Sobczyk, 1984), with points $x \in M$ represented as vectors (in $\Lambda^1(V)$), and a tangent space of M at x given by a blade $I(x)$ of grade m.[1] The projection of a function $F : M \to \Lambda(V)$ onto the tangent space at x is given by

$$P(F)(x) := (F(x) \rfloor I^{-1}(x)) \rfloor I(x) \tag{5}$$

A grade k function F is called a *field* on M if $P(F) = F$. By projecting the vector derivative operator $\partial_{\mathcal{I}_1}$, we obtain the *intrinsic* derivative:

$$\partial_M := P(\partial_{\mathcal{I}_1}) \tag{6}$$

[1] Vector Manifolds motivate why the vector derivative is of primary concern for this work, as all functions on manifolds can now be viewed as functions from $M \subset \Lambda^1(V) \to \Lambda(V)$. One could in principle also attempt to study 'bivector manifolds', etc., but this does not embed cleanly into differential geometry.

Unfortunately, $\partial_M F$ is not necessarily a field, even if F is. This can be remedied by simply defining the *covariant derivative* to be the projection of $\partial_M F$:

$$\nabla_M F := P(\partial_M F) = P(P(\partial_{\mathcal{I}_1})F) \tag{7}$$

The relevance of these definitions is:

- As we will see in more detail in Section 4, standard differential operators on manifolds can be expressed in terms of ∂_M and ∇_M. The curl and divergence on M, for example, are of the form $\nabla_M \wedge F$ and $\nabla_M \rfloor F$. Both are nilpotent operators: $(\nabla_M \wedge)^2 = 0$ (the GA form of $d^2 = 0$ from differential forms) and dually $(\nabla_M \rfloor)^2 = 0$ (codifferential $\delta^2 = 0$). This ensures that the Hodge-De Rham Laplacian $\Delta_M = \nabla_M^2$ is *grade preserving*, as $(d + \delta)^2 = d^2 + d\delta + \delta d + \delta^2 = d\delta + \delta d$.
- The projector Eq. 5 is easy to compute in an AD framework.

In short, differential operations on manifolds reduce to AD primitives via Eq. 5 and the lemma above.

Remark 4. Compositions of P, ∂_M, and ∇_M give rise to the standard objects of differential geometry on M (intrinsic differential operators, Laplacians, and curvature tensors), with no Christoffel symbols, coordinate charts, or dimension-specific formulas required. A parallel integration theory pairs ∂_M with the directed volume field $\mathrm{d}X$ (the tangent space $I(x)$ weighted by the Riemannian measure $|\mathrm{d}X|(x) \in \mathbb{R}$) to give a single multivector-valued fundamental theorem subsuming the classical Stokes-type results (Hestenes & Sobczyk, 1984). Appendix B.3 develops these in full, showing the full 'API' for differential geometry.

4 Experiments

We present four sets of experiments. The first two concern manifold learning, using more standard differential geometric constructions. The second two go beyond differential geometry and show the application of our framework in physics and constrained optimization. These experiments are end-to-end demonstrations, not performance benchmarks. They show how a small collection of AD transformations can be reused and composed.

4.1 Explicit Manifold Learning

We can learn an explicit representation of a manifold by starting with a known simple one M_0 (a sphere, a disk, a flat square) and applying some smooth transformation to get a new manifold M_1. This set of experiments work within this paradigm to solve various learning problems.

Non-Euclidean tilings. A general recipe for constructing a divergence-free vector field on a manifold is immediate in our framework. Start with an arbitrary 2-form, project to get a 2-field, and then apply $\nabla_M \rfloor$ to get a divergence-free vector field, recalling that $(\nabla_M \rfloor)^2 = 0$. This generalizes the Euclidean neural conservation law construction (Richter-Powell et al., 2022). In the particular case of a 2-surface, any 2-field must be a scalar multiple of I_M, so we parameterize our field via $\phi_\theta I_M$ for a scalar valued neural network ϕ_θ, with the resulting divergence-free field given by $\nabla_M \rfloor (\phi_\theta I_M)$. We can apply these divergence free flows to learn tilings on manifolds; this has proven useful in, e.g., the design of mechanical meta-materials (Mirramezani et al., 2024). Our framework gives us a recipe to do this on any manifold: start with a fixed tiling, apply the flow, and then minimize a loss on the resulting tiles (ours is MSE against a target shape). On S^2 we deform an icosidodecahedron into a chiral, $\mathbb{Z}_5$-symmetric pinwheel target, see Fig. 1.

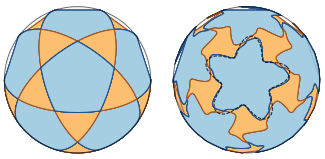


Figure 1: Icosidodecahedron $\to \mathbb{Z}_5$ pinwheel on S^2. Target in black.

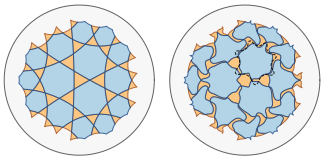


Figure 2: Rectified $\{3, 7\}$ tiling $\to D_7$ gear on H^2.

As a second example, on the Poincaré disk H^2 we deform a rectified $\{3, 7\}$ tiling into a D_7-symmetric gear, see Fig. 2. With the ambient Minkowski space $\mathbb{R}^{2,1}$ spanned by $\{e_1, e_2, e_n\}$, our framework immediately grants us access to the differential operators on H^2 once we write down the manifold transformation $\mathbb{R}^2 \to H^2$ given by $u \mapsto \cosh|u|\, e_n + (\sinh|u|/|u|)\, u$. The hyperbolic

metric is induced by the ambient Minkowski inner product restricted to the tangent space of H^2. The divergence free flow here preserves area with respect to this metric, which explains the distortion near the boundary. This also makes it impossible for the learned shape to fully align with the target shape. For complete details of the setup and training, see Appendix C.2.A.

Backprop through the Euler characteristic. As a fun learning demonstration enabled by our framework, consider the problem of learning to warp a flat square into a sphere. We can parameterize this warping as the flow φ_t^θ of a neural ODE (Chen et al., 2018) acting on our initial manifold M_0, and train it end to end so that the resulting manifold has certain geometric properties; see Appendix C.2.B for details. One way to characterize the sphere is by its constant curvature. A remarkable fact is that nearly all curvature-based properties of a manifold can be computed as compositions of ∂_M and I_M; Hestenes's so-called *shape* operator $S_M = \partial_M P$ measures how the projector changes as we move along the manifold, which turns out to encode a lot of information about the local geometry. In particular, scalar curvature R can be read off almost immediately for a manifold of any dimension, and equals twice the Gaussian curvature, $R = 2K$, on a surface. Training a flow to match the sphere's curvature results in learning a small section of the sphere (see Fig. 3). To warp the square into a full sphere, we can augment this training objective with a *differentiable* estimate of the Euler characteristic χ_M of M via the Gauss–Bonnet theorem for closed surfaces,

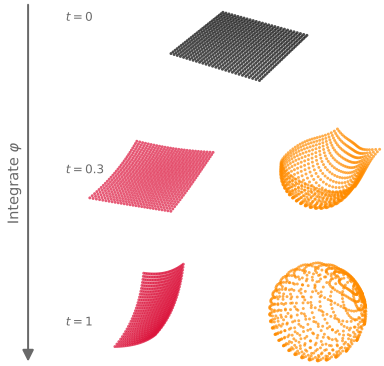


Figure 3: Flat square (black) flowed under curvature loss (red) vs. with the added Gauss–Bonnet term (orange).

$$\chi_M \;=\; \frac{1}{4\pi}\int_M R\,|\mathrm{d}X| \;\approx\; \frac{1}{4\pi}\sum_{i=1}^{N} R(x_i)\,|\mathrm{d}X|(x_i). \tag{8}$$

Again, this is not only easy to compute in our library, but differentiable, so we can *directly optimize* Eq. 8 to approach the Euler characteristic of the sphere $\chi_{S^2} = 2$. Of course, the square is not a closed manifold, and χ is invariant under diffeomorphisms (and hence the parameterized flow φ^θ). Nevertheless, this added loss is still able to guide the flow to something that is approximately closed, from which the Euler characteristic estimate is approximately 2 (see Fig. 3).

4.2 Implicit Manifold Learning

Explicit manifold learning (§4.1) requires a smooth invertible map φ between a known base manifold and the target, which restricts the types of architectures one can use. We can also learn an *implicit* representation, encoding the manifold as the support (or zero level set) of a learned field (Palmer et al., 2022; Park et al., 2019).

Minimal surfaces and the Biot–Savart kernel. Given a closed boundary Γ of dimension $n-k-1$ in $\mathbb{R}^n$, we learn the bounding $(n-k)$-submanifold M_θ as the set on which a learned k-form $\omega_\theta = \omega_\Gamma + \partial_{\mathcal{I}_1} \wedge \eta_\theta$ concentrates (extracted from ω_θ as a level set or ridge, Appendix C.3.A) for an arbitrary $(k-1)$-form η_θ parameterized by a neural network, refining the implicit-current representation of DeepCurrents (Palmer et al., 2022). The key observation is that the boundary constraint $\partial M_\theta = \Gamma$ can be made structural through the GA-native *Biot–Savart kernel*:

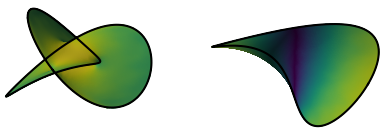


Figure 4: Learned minimal surfaces for trefoil (left) and a saddle (right).

$$\omega_\Gamma(x) = I^{-1}(x)\int_\Gamma \partial_{\mathcal{I}_1} G(x-y) \wedge d\Gamma(y), \quad \partial_{\mathcal{I}_1} \wedge \omega_\Gamma = \delta_\Gamma, \tag{9}$$

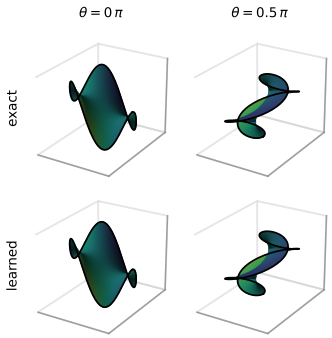


Figure 5: A holomorphic graph over a disk in $\mathbb{C}^2 \cong \mathbb{R}^4$: exact minimal surface (top) vs. surface learned by minimizing the mass norm (bottom), at two projection angles.

where G is the fundamental solution to $\Delta G = \delta$, also known as the Green's function of the Laplacian. With this choice, we guarantee that $\partial_{\mathcal{I}_1} \wedge \omega_\theta = \delta_\Gamma$, hence this parameterization satisfies $\partial M_\theta = \Gamma$ for any (n, k) pair. DeepCurrents shows this for $(n, k) = (3, 1)$, which we show is easily replicated in our framework; see Fig. 4. The problem of generalizing the training to other (n, k) is actually obstructed by the difficulty of computing the *mass norm* of the learned current, as we discuss next. Intuitively, the mass norm is a generalization of the measure of the manifold, which is why minimizing this quantity gives surfaces of minimal measure. Next, we discuss how the mass norm is computed pointwise for k-forms; from this we can train the network by minimizing the integral of the mass norm over some ambient domain containing our surface.

The general way to define the (pointwise) mass norm of a k-form ω is to write ω as a sum of blades and then take the sum of the multivector norms of these blades. The true mass norm is the infimum of this quantity over all such decompositions. For a 1-form ω, all vectors are blades hence the mass norm coincides with the multivector norm. For a 2-form ω, the infimum is attained by writing ω as a sum of *commuting* simple bivectors (Hestenes & Sobczyk, 1984): $\omega = \sum_{i=1}^{\lfloor n/2 \rfloor} \lambda_i B_i$ with $B_i B_j = B_j B_i$. The mass norm is then $\|\omega\| = \sum_i |\lambda_i|$. The essence of this norm is that it penalizes bivectors which are *not* pure subspaces (blades).

Thus, a bivector like $B = e_1 \wedge e_2 + e_3 \wedge e_4$ is penalized for being a superposition of two pure subspaces ($e_1 \wedge e_2$ and $e_3 \wedge e_4$) and has mass norm 2. In contrast, the bivector $B' = e_1 \wedge e_2 + e_3 \wedge e_1 = e_1 \wedge (e_2 - e_3)$ *is* a pure subspace having mass norm $\sqrt{2}$. Under the multivector norm, B and B' would have the same norm. Minimizing the mass norm thus pushes ω to be simple almost everywhere. Importantly, the mass norm on bivectors can be computed efficiently and is differentiable almost everywhere. Unfortunately, no closed form is known for the mass norm when $3 \leq k \leq n - 3$, so we only attempt the $n = 4, k = 2$ case. To get an interesting family of ground truth minimal surfaces for this codimension 2 setting, we can note that any holomorphic curve M in $\mathbb{C}^2$ is mass-minimizing (Federer, 1969), hence defines a minimal surface for any boundary contained on it. We show these comparisons in Figs. 5 and 6. For the exact surfaces and their parameters, more examples, and detailed metrics, refer to Appendix C.3.A.

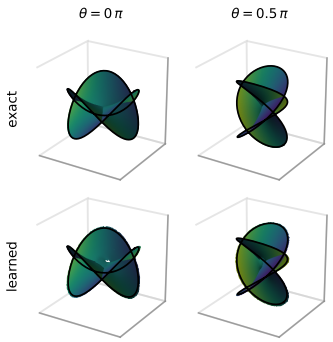


Figure 6: A branched holomorphic disk bounding the $(2, 3)$ torus knot, exact (top) vs. learned (bottom), as in Fig. 5.

4.3 Inverse Design in Spacetime

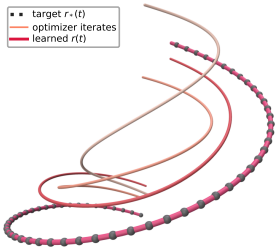


Figure 7: Recovered trajectory $r(t)$ (red) against the target helix, with intermediate training trajectories also plotted.

The spacetime algebra $\mathrm{Cl}(1, 3)$ is a useful model for electromagnetism (Doran & Lasenby, 2003; Hestenes, 1966). In this algebra, the electromagnetic field collapses into a single bivector $F = E + IB$ (electric E, magnetic B), the source 4-current is a single vector J, and the four classical Maxwell equations collapse to a single famous equation involving the multivector derivative: $\partial_{\mathcal{I}_1} F = J$. The grade decomposition splits this into the source half $\partial_{\mathcal{I}_1} \rfloor F = J$ and the homogeneous half $\partial_{\mathcal{I}_1} \wedge F = 0$ (no magnetic monopoles). Applying the multivector derivative once more gives charge conservation *for free*: $\partial_{\mathcal{I}_1} \rfloor J = 0$. The 4-potential factorization $F = \partial_{\mathcal{I}_1} \wedge A$ in turn exists because $(\partial_{\mathcal{I}_1} \wedge)^2 = 0$, and *gauge freedom* is the statement that A and $A + \partial_{\mathcal{I}_1} \wedge \phi$ produce the same F for any scalar ϕ. Picking a gauge (e.g. Lorenz, $\partial_{\mathcal{I}_1} \rfloor A = 0$) just selects a representative (usually $\phi = 0$).

To demonstrate our framework in an inverse design problem, consider the problem of recovering a charged particle's trajectory $r(t)$ from a sparse set of Faraday bivector observations $F(y_i)$ at spacetime points y_i.

To parameterize the particle's worldline, we use a small MLP which outputs a 3-velocity $v_\theta(t)$ in the lab frame such that $|v_\theta(t)| < 1$ (so the predicted worldline is timelike and future-directed at every step). Integrating $v_\theta(t)$ gives the spatial trajectory $r_\theta(t)$, and hence the worldline $x_\theta(t) = t\, e_0 + r_\theta(t)$. To get the Faraday bivector F at an observation point y, we first must find the retarded time $\tau_r(y)$ such that $y - x_\theta(\tau_r(y))$ is null, i.e., $(y - x_\theta(\tau_r(y)))^2 = 0$. Here $\tau_r(y)$ is the lab time at which the particle emits the signal received at y. We compute $\tau_r(y)$ via bisection search, which can be differentiated (with respect to r) via implicit function theorem. Once we have $\tau_r(y)$, we can compute the Liénard–Wiechert potential $A(y)$ and then the Faraday bivector $F(y) = \partial_{\mathcal{I}_1} \wedge A(y)$. The entire pipeline from $v_\theta(t)$ to $F(y)$ is differentiable, allowing us to backpropagate the relative MSE between predicted and observed Faraday bivectors to update the parameters of the MLP. The F field at a point is shown in Fig. 8 for a particle on a helical worldline, and the learned trajectory is shown in Fig. 7. Full details of the experiment, including the observation grid, input encoding, and optimizer schedule, are in Appendix C.4.

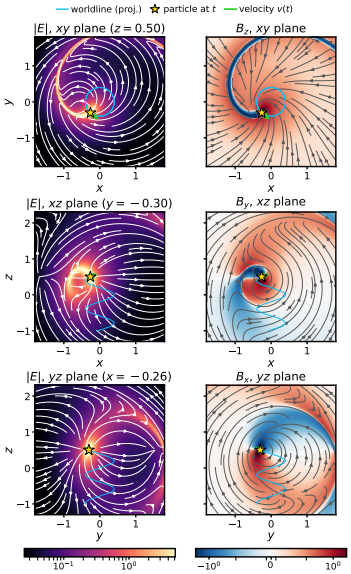


Figure 8: Projections of F field at t=2.

4.4 Optimization on Rotor Groups

Until now, all our examples have used the *vector* derivative $\partial_{\mathcal{I}_1}$;
we now show the utility of a different instance of the multivector derivative for optimization on rotor groups and the homogenous spaces they act on. While this is a well studied topic (Absil et al., 2008; Boumal, 2023; Deray & Solà, 2020; Solà et al., 2021), a GA-native medium allows for these spaces to be specified as data instead of code, making optimization procedures reusable and composable.

The *rotors* of a geometric algebra are even products of unit vectors $R = \prod_{i=1}^{2k} u_i$ satisfying $R\tilde{R} = 1$. It is clear that rotors form a group under the geometric product called the rotor group $\mathrm{Spin}(p,q)^+ \subset \mathrm{Cl}(p,q)$. Homogeneous spaces like the Stiefel manifold or the Grassmannian can be realized as $RX_0\tilde{R}$ for some fixed multivector X_0. Thus, optimizing a function over one of these spaces generally reduces to the problem of minimizing a function $f : G \to \mathbb{R}$ for G some subgroup of $\mathrm{Spin}(p,q)^+$.

To this end, we first need to specify G. One way to specify a group is via a set of generators. However, because G is a *Lie* group, we can instead provide generators of the corresponding Lie algebra $\mathfrak{g} := T_e G$. The utility of GA is that this Lie algebra will always be a *bivector* subspace, leading to a compact and uniform way to capture all matrix Lie groups (subgroups of $\mathrm{Spin}(p,q)^+$). With this choice of bivectors $\{B_k\}$, suppose that we currently have a rotor $g_t \in G$ and we would like to find an update U_t such that $f(g_t U_t) < f(g_t)$. This can be done with the *left trivialized differential* $df_e(g_t) \in \mathfrak{g}^*$, which can be implemented in terms of the multivector derivative as:

$$df_e(g) = \partial_{\mathcal{I}_{\text{even}}} f(g)\, g \in \mathfrak{g}^*, \tag{10}$$

Intuitively, we can think of the multivector derivative as an element of $T_g^* G$, which we then right multiply by g to get to the appropriate cotangent space at the identity, $\mathfrak{g}^*$. To turn this covector into an update direction we need a map $\sharp : \mathfrak{g}^* \to \mathfrak{g}$. For descent to work, the inner product inducing this map *must* be positive definite, meaning the usual multivector inner product won't work. While AD implicitly uses the Euclidean inner product on coordinates, our framework lets any positive-definite choice be made explicitly. Once we have mapped our element to $\mathfrak{g}$, exponentiating gives:

$$g_{t+1} = g_t \exp(-\eta\, df_e(g_t)^\sharp) \tag{11}$$

This update is valid for any Lie group embedded in $\mathrm{Cl}(p,q)$. Existing tools like `manopt` (Boumal et al., 2014) and `geoopt` (Kochurov et al., 2020) implement separate code for each manifold

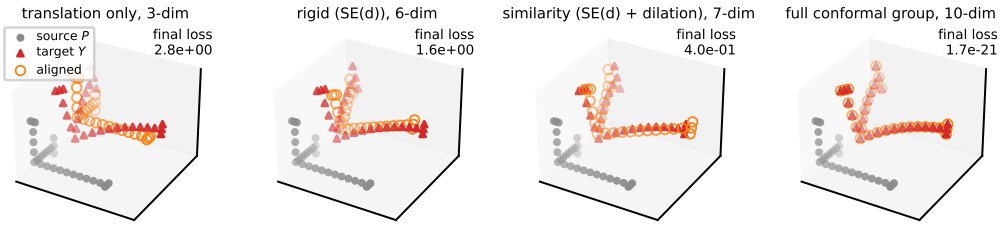


Figure 9: Vectorized optimization over different subgroups: Given an object (gray) and a target conformal transformation (red), we simultanously learn 4 rotors in *different* rotor groups corresponding to maskings of the 10-dimensional bivector space of $\mathrm{CGA}(3)$: translation (3D), rigid (6D), similarity (7D), and conformal (10D).

being optimized over, and additionally only work out of the box for Lie groups which are compact or the semidirect product of a compact Lie group with $\mathbb{R}^n$ (e.g., $SE(n) = SO(n) \ltimes \mathbb{R}^n$). `geotorch` (Lezcano-Casado, 2019) implements a similar trivialization procedure to ours, but even this library still assumes definite signature. See Appendix C.5 for further exposition.

With 'groups as data', we can use JAX's vectorization to simultaneously optimize an objective across *different* groups. Consider the chain $T(3) \subset SE(3) \subset \mathrm{Sim}(3) \subset \mathrm{Conf}(3) = \mathrm{Spin}(4,1)^+$, which are all rotor subgroups of the conformal geometric algebra $\mathrm{CGA}(3) = \mathrm{Cl}(4,1)$: each Lie algebra in the chain is a coordinate subspace of the 10-dimensional bivector algebra of the full conformal group. In Figure 9, we optimize the same point-cloud alignment loss on all four groups *at once*: each panel converges to the best approximation its subgroup admits (matching the centroid, Procrustes, and Umeyama closed forms), and the conformal panel recovers the target configuration exactly.

Next, we show how this same framework can support optimization over the Lorentz group of the spacetime algebra. Suppose that we are on a ship moving at some uniform *unknown* velocity and orientation, from which we can measure the (possibly noisy) electromagnetic field $\tilde{F}$. We can then try to recover the Lorentz transformation $R \in \mathrm{Spin}(1,3)^+$ that maps the lab frame to the ship frame by minimizing the loss between the mapped analytic field $RF_k\widetilde{R}$ and the measured field $\tilde{F}_k$. We plot the results of doing such an optimization in Figure 10. The optimization converges to the correct Lorentz transformation, and we can recover the ship's velocity from the optimized rotor R. See Appendix C.5 for full training details.

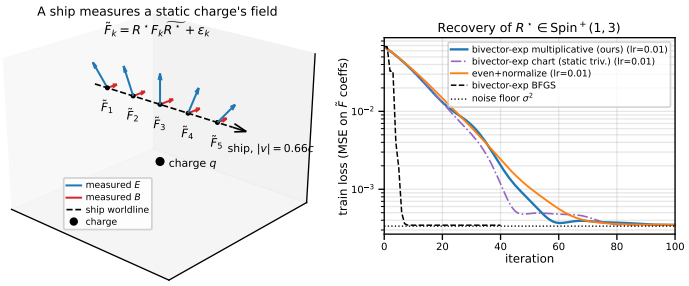


Figure 10: Recovering a Lorentz rotor from field measurements. *Left:* a ship crosses the lab frame at uniform velocity (dashed), past a static point charge, measuring the Faraday bivector $\tilde{F}_k$ in its own frame at events along its path, $\tilde{F}_k = R^\star F_k \widetilde{R^\star} + \epsilon_k$. In the ship frame the pure-electric Coulomb field tilts and compresses (electric part, blue) and acquires a circulating magnetic part (red). *Right:* training loss for recovering $R^\star$.

5 Discussion and Conclusion

The experiments illustrate that a small set of geometric operators can support several apparently different workflows. The same implementation of the multivector derivative, exterior derivative, and tangent projection solves problems ranging from manifold learning, inverse problems in spacetime, and optimization on rotor groups. In each case, the geometric structure enters through the selected index set $\mathcal{I}$, the multivector field being differentiated, and, on manifolds, the tangent pseudoscalar I_M. This is the main practical benefit of the approach: geometric differential operators can be exposed as reusable AD transformations rather than re-derived for each coordinate system or application.

AI USE STATEMENT

In this work, we used generative AI tools for three primary tasks: first, to collaborate and iterate on experiments. While early versions of all of the experiments were entirely human written, we found AI to increase our ability to iterate fast and find the right training settings to make things work (for example). Second, we used AI to find relevant sources in the literature. We have carefully reviewed all of these sources to ensure that no citations are hallucinated. Third, we used generative AI to help revise the paper, improve the precision of mathematical statements, and for formatting purposes. The authors take full responsibility for all claims made in the paper.

REPRODUCIBILITY STATEMENT

For reproducibility of this work, we have included a series of proofs in the appendix along with many examples of the API provided by our library, in addition to many experiment details such as architecture, optimizer, and citations to other work that we build off of. We hope to release the code publicly, with all experiments reproducible with single scripts that can be run on most hardware.

REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A system for large-scale machine learning. In *OSDI*, pp. 265–283, 2016.

P.-A. Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2008.

Douglas N. Arnold, Richard S. Falk, and Ragnar Winther. Finite element exterior calculus: From Hodge theory to numerical stability. *Bulletin of the American Mathematical Society*, 47(2):281–354, 2010.

Atılım Güneş Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

Michael Betancourt. A geometric theory of higher-order automatic differentiation. *arXiv preprint arXiv:1812.11592*, 2018.

Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Nicolas Boumal. *An Introduction to Optimization on Smooth Manifolds*. Cambridge University Press, 2023.

Nicolas Boumal, Bamdev Mishra, P.-A. Absil, and Rodolphe Sepulchre. Manopt, a Matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research*, 15(42):1455–1459, 2014.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Johannes Brandstetter, Rianne van den Berg, Max Welling, and Jayesh K. Gupta. Clifford neural layers for PDE modeling. In *International Conference on Learning Representations (ICLR)*, 2023.

Johann Brehmer, Pim de Haan, Sönke Behrends, and Taco Cohen. Geometric algebra transformer. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Alan Bromborsky and Pygae community. galgebra: Symbolic geometric algebra/calculus for SymPy, 2019. URL https://github.com/pygae/galgebra.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Keenan Crane, Fernando de Goes, Mathieu Desbrun, and Peter Schröder. Digital geometry processing with discrete exterior calculus. In *ACM SIGGRAPH 2013 Courses*, pp. Article 7, 2013.

Pim de Haan, Taco Cohen, and Johann Brehmer. Euclidean, projective, conformal: Choosing a geometric algebra for equivariant transformers. In *AISTATS*, pp. 3088–3096, 2024.

Jeremie Deray and Joan Solà. manif: A micro Lie theory library for state estimation in robotics applications. *Journal of Open Source Software*, 5(46):1371, 2020.

Mathieu Desbrun, Anil N. Hirani, Melvin Leok, and Jerrold E. Marsden. Discrete exterior calculus. *arXiv preprint math/0508341*, 2005.

Chris Doran. *Geometric Algebra and its Application to Mathematical Physics*. PhD thesis, University of Cambridge, 1994.

Chris Doran and Anthony Lasenby. *Geometric Algebra for Physicists*. Cambridge University Press, 2003.

Leo Dorst, Daniel Fontijne, and Stephen Mann. *Geometric Algebra for Computer Science: An Object-Oriented Approach to Geometry*. Morgan Kaufmann, 2007. ISBN 9780123694652.

Conal Elliott. The simple essence of automatic differentiation. *Proceedings of the ACM on Programming Languages*, 2(ICFP):70:1–70:29, 2018.

Herbert Federer. *Geometric Measure Theory*, volume 153 of *Grundlehren der mathematischen Wissenschaften*. Springer, 1969.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: Free-form continuous dynamics for scalable reversible generative models. In *International Conference on Learning Representations (ICLR)*, 2019.

Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, 2nd edition, 2008.

Hugo Hadfield, Eric Wieser, Alex Arsenovic, Robert Kern, and The Pygae Team. clifford: Numerical geometric algebra for Python, 2013. URL https://github.com/pygae/clifford.

David Hestenes. *Space-Time Algebra*. Gordon and Breach, 1966. 2nd ed., Birkhäuser, 2015.

David Hestenes. Multivector calculus. *Journal of Mathematical Analysis and Applications*, 24(2):313–325, 1968.

David Hestenes. *New Foundations of Classical Mechanics*. Kluwer Academic Publishers, Dordrecht, 2nd edition, 1999. ISBN 9780792355144.

David Hestenes. The shape of differential geometry in geometric calculus. In Leo Dorst and Joan Lasenby (eds.), *Guide to Geometric Algebra in Practice*, pp. 393–410. Springer, 2011.

David Hestenes and Garret Sobczyk. *Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics*. Fundamental Theories of Physics. Springer Dordrecht, 1984.

Anil N. Hirani. *Discrete Exterior Calculus*. PhD thesis, California Institute of Technology, 2003.

Eckhard M. S. Hitzer. Multivector differential calculus. *Advances in Applied Clifford Algebras*, 12(2):135–182, 2002.

Eelco Hoogendoorn. Numga: Geometric algebra in JAX and NumPy, 2025. URL https://github.com/EelcoHoogendoorn/numga. GitHub repository.

Robin Kahlow. JAXGA: Geometric algebra package for JAX, 2020a. URL https://github.com/RobinKa/jaxga. GitHub repository.

Robin Kahlow. TFGA: Geometric algebra package for TensorFlow 2, 2020b. URL https://github.com/RobinKa/tfga.

Max Kochurov, Rasul Karimov, and Serge Kozlukov. Geoopt: Riemannian optimization in PyTorch. *arXiv preprint arXiv:2005.02819*, 2020.

A. N. Lasenby, C. J. L. Doran, and S. F. Gull. A multivector derivative approach to Lagrangian field theory. *Foundations of Physics*, 23(10):1295–1327, 1993.

Mario Lezcano-Casado. Trivializations for gradient-based optimization on manifolds. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Alan Macdonald. A survey of geometric algebra and geometric calculus. *Advances in Applied Clifford Algebras*, 27:853–891, 2017.

Mehran Mirramezani, Anne S. Meeussen, Katia Bertoldi, Peter Orbanz, and Ryan P. Adams. Designing mechanical meta-materials by learning equivariant flows, 2024. URL https://arxiv.org/abs/2410.02385.

David Palmer, Dmitriy Smirnov, Stephanie Wang, Albert Chern, and Justin Solomon. Deepcurrents: Learning implicit representations of shapes with boundaries, 2022. URL https://arxiv.org/abs/2111.09383.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Jason Rader, Terry Lyons, and Patrick Kidger. Optimistix: modular optimisation in JAX and Equinox. *arXiv preprint arXiv:2402.09983*, 2024.

Jack Richter-Powell, Yaron Lipman, and Ricky T. Q. Chen. Neural conservation laws: A divergence-free perspective, 2022. URL https://arxiv.org/abs/2210.01741.

Martin Roelfs. kingdon: The willing kingdon clifford algebra library. *arXiv preprint arXiv:2503.10451*, 2025.

David Ruhe, Johannes Brandstetter, and Patrick Forré. Clifford group equivariant neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a.

David Ruhe, Jayesh K. Gupta, Steven De Keninck, Max Welling, and Johannes Brandstetter. Geometric Clifford algebra networks. In *International Conference on Machine Learning (ICML)*, pp. 29306–29337, 2023b.

Amandip Sangha. Jet functors and Weil algebras in automatic differentiation: A geometric analysis. *arXiv preprint arXiv:2510.14342*, 2025.

Joan Solà, Jeremie Deray, and Dinesh Atchuthan. A micro Lie theory for state estimation in robotics. *arXiv preprint arXiv:1812.01537*, 2021.

Hannes Sommer, Cédric Pradalier, and Paul Furgale. Automatic differentiation on differentiable manifolds as a tool for robotics. In *Robotics Research: The 16th International Symposium ISRR*, volume 114 of *Springer Tracts in Advanced Robotics*, pp. 505–520. Springer, 2016.

Jonas Spinner, Víctor Bresó, Pim de Haan, Tilman Plehn, Jesse Thaler, and Johann Brehmer. Lorentz-equivariant geometric algebra transformers for high-energy physics. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Lars Tingelstad and Olav Egeland. Automatic multivector differentiation and optimization. *Advances in Applied Clifford Algebras*, 27:707–731, 2017a.

Lars Tingelstad and Olav Egeland. Motor estimation using heterogeneous sets of objects in conformal geometric algebra. *Advances in Applied Clifford Algebras*, 27:2035–2049, 2017b.

Lars Tingelstad and Olav Egeland. Motor parameterization. *Advances in Applied Clifford Algebras*, 28:34, 2018.

Maksim Zhdanov, David Ruhe, Maurice Weiler, Ana Lučić, Johannes Brandstetter, and Patrick Forré. Clifford-steerable convolutional neural networks. In *International Conference on Machine Learning (ICML)*, pp. 61203–61228, 2024.

A PROOFS

Throughout this section, we let V be an n-dimensional real vector space with a nondegenerate symmetric bilinear form of signature (p, q), $\Lambda(\mathbb{R}^{p,q}) = \mathrm{Cl}(p, q)$ its geometric algebra, and $\{b_J\}_{J\subset[n]}$ a basis of blades induced by a basis $b_1, \dots, b_n$ of V. Given the multivector inner product $\langle\cdot,\cdot\rangle$, we can compute the reciprocal basis blades by first computing the gram matrix $\mathbf{G}_{JK} = \langle b_J, b_K\rangle$, and then setting $b^J = \sum_K \mathbf{G}^{-1}_{JK} b_K$. Note that that is how we implement the musical isomorphisms $\sharp, \flat$ that we use to define our metric aware `vjp`. As a final remark, we mention that even for an orthonormal basis in a Euclidean GA $\Lambda(\mathbb{R}^n)$, while the Gram matrix on vectors on vectors might be the identity, the full gram matrix $\mathbf{G}$ will not: consider, for example, the basis vector $e_{12} = e_1 \wedge e_2$. Then, $e^{12} = -e_{12}$. Nondegeneracy of $\langle\cdot,\cdot\rangle$ ensures that $\mathbf{G}$ is invertible.

Another frequent fact that we will use throughout is that we can write a multivector as coordinates in some basis. For any multivector A, we have:

$$A \;=\; \textstyle\sum_J \langle b^J, A\rangle\, b_J \;=\; \sum_J \langle A, b_J\rangle\, b^J \qquad \text{for all } A \in \Lambda(V). \tag{12}$$

In general, this is how we can express a multivector A in coordinates relative to a basis; we will use this expansion throughout.

A.1 BASIS INDEPENDENCE OF $\partial_{\mathcal{I}}$

Proposition 5 *Let $\mathcal{I} \subset 2^{[n]}$ and take $W := \mathrm{span}\{b_J\}_{J\in\mathcal{I}} = \mathrm{span}\{b^J\}_{J\in\mathcal{I}}$. The operator $\partial_{\mathcal{I}} F = \sum_{J\in\mathcal{I}} b^J\, \partial F/\partial X_J$ depends only on the subspace W and the pairing, not on the basis $\{b_J\}_{J\in\mathcal{I}}$ chosen for W.*

Proof. Suppose that we had a different basis $\{c_J\}_{J\in\mathcal{I}}$ of W. By virtue of $\{b_J\}$ being a basis for W, every c_J must be expressible as $c_J = \sum_K \mathbf{A}_{JK} b_K$ for some matrix $\mathbf{A}$. The reciprocals $\{b^J\}$ also form a basis, and thus should satisfy $c^J = \sum_K \mathbf{D}_{JK} b^K$ for some matrix $\mathbf{D}$. We can then note:

$$\delta_{JL} = \langle c^J, c_L\rangle = \sum_{K,M} \mathbf{D}_{JK}\mathbf{A}_{LM}\langle b^k, b_M\rangle \tag{13}$$

$$= \sum_K \mathbf{D}_{JK}\mathbf{A}_{LK} = (\mathbf{D}\mathbf{A}^\top)_{JL} \tag{14}$$

Therefore, $\mathbf{D} = (\mathbf{A}^\top)^{-1}$.

Next, note that for an arbitrary multivector $X \in W$, we have:

$$X = \sum_J X'_J c_J \tag{15}$$

$$= \sum_J X'_J \left(\sum_K A_{JK} b_K \right) \tag{16}$$

$$= \sum_K \left(\sum_J A_{JK} X'_J \right) b_K \tag{17}$$

$$= \sum_K X_K b_K \tag{18}$$

This relates the coordinates X_K and X'_J, and immediately implies:

$$\frac{\partial X_K}{\partial X'_J} = A_{JK} \tag{19}$$

With this established, we can easily simplify the following multivector derivative expressed in the $\{c_J\}$ basis as one expressed in the $\{b_J\}$ basis:

$$\partial F = \sum_J c^J \frac{\partial F}{\partial X'_J} \tag{20}$$

$$= \sum_J \left[\sum_K b^K (A^\top)^{-1}_{JK} \right] \left[\sum_L \frac{\partial F}{\partial X_L} \frac{\partial X_L}{\partial X'_J} \right] \tag{21}$$

$$= \sum_{K,L} b^K \left[\sum_J (A^\top)^{-1}_{JK} A_{JL} \right] \frac{\partial F}{\partial X_L} \tag{22}$$

$$= \sum_{K,L} b^K \delta_{KL} \frac{\partial F}{\partial X_L} \tag{23}$$

$$= \sum_K b^K \frac{\partial F}{\partial X_K} \tag{24}$$

□

Remark 6. The condition that the $\{b^J\}$ span the same subspace as $\{b_J\}$ can be relaxed, but convolutes the proof slightly. As an example of such an instance, consider the conformal euclidean algebra $\Lambda(\mathbb{R}^{n+1,1})$ with basis $e_1, \ldots, e_n, e_+, e_-$, and then consider the change of basis $e_\infty = e_+ + e_-$ and $e_o = \frac{1}{2}(e_- - e_+)$. It is easy to verify that $\langle e_\infty, e_o \rangle = -1$, meaning that the reciprocal of e_∞ is $-e_o$. We can thus see that any space W containing e_∞ components without the e_o components will have a reciprocal space that is not contained within it.

A.2 Proof of Lemma 2

Forward-mode proof. Take $X = \sum_{J \in \mathcal{I}} X_J b_J \in W$. Then by definition,

$$\frac{\partial F}{\partial X_J} = \frac{d}{dt} F(X + t b_J)\big|_{t=0} = dF_X[b_J] = \texttt{jvp}(F, X, b_J) \tag{25}$$

so $\partial_{\mathcal{I}} F(X) = \sum_{J \in \mathcal{I}} b^J \texttt{jvp}(F, X, b_J)$. □

Reverse-mode proof. Letting $\mathcal{O}$ index the output blades, so that $F(X) \in \text{span}\{b_K : K \in \mathcal{O}\}$ for all X. Take $\mathcal{I}$ to be the corresponding input index set over which X varies. Note that the adjoint is defined implicitly by the property $\langle C, dF_X[A] \rangle = \langle dF^*_X[C], A \rangle$ for the multivector inner product

$\langle \cdot, \cdot \rangle$:

$$\partial_{\mathcal{I}} F = \sum_{J \in \mathcal{I}} b^J dF_X[b_J] \tag{26}$$

$$= \sum_{J \in \mathcal{I}} b^J \left[\sum_{K \in \mathcal{O}} \langle b^K, dF_X[b_J] \rangle b_K \right] \tag{27}$$

$$= \sum_{J \in \mathcal{I}} b^J \left[\sum_{K \in \mathcal{O}} \langle dF_X^*[b^K], b_J \rangle b_K \right] \tag{28}$$

$$= \sum_{K \in \mathcal{O}} \left[\sum_{J \in \mathcal{I}} \langle dF_X^*[b^K], b_J \rangle b^J \right] b_K \tag{29}$$

$$= \sum_{K \in \mathcal{O}} dF_X^*[b^K] b_K \tag{30}$$

(31)

and we are done as our metric aware $\texttt{vjp}_{\langle \cdot, \cdot \rangle}$ computes dF^* with respect to the multivector inner product instead of the Euclidean one. □

Remark 7. If we assume that F varies over the basis index set $\mathcal{O}$, and takes inputs X which vary over the basis index set $\mathcal{I}$, then computing the forward mode $\partial_{\mathcal{I}}$ requires $|\mathcal{I}|$ `jvp` calls and geometric products. Computing the reverse mode $\partial_{\mathcal{O}}$ requires $|\mathcal{O}|$ `vjp` calls and geometric products. This implies that for computing multivector derivatives of scalar functions is most efficient in reverse mode, while most vector derivatives are more efficient to compute with the forward mode form since they only require n `jvp`'s.

B Software API and Implementation

Our implementation is written in JAX (Bradbury et al., 2018) to be natively differentiable, and begins with the basic means of constructing Geometric Algebras and the operators on them. This base functionality is similar to that of other libraries such as `jaxga`, `clifford`, `numga`, and `galgebra`. However, once we get into the more calculus heavy operations, the autodiff capability of JAX makes it possible to go further than what these libraries provide.

B.1 Algebras and Multivectors

In this work, we only consider geometric algebras over real vector spaces (though in principle we could support complex vector spaces as this is natively supported in JAX). Such an algebra is specified by nonnegative integers (p, q, r), with p denoting the number of positive basis vectors ($e^2 = 1$), q denoting the number of negative basis vectors ($e^2 = -1$), and r denoting the number of zero basis vectors ($e^2 = 0$). We write this geometric algebra as $\Lambda(\mathbb{R}^{p,q,r}) = \mathrm{Cl}(p, q, r)$. Because many of the autodiff primitives require a nondegenerate metric, we will oftentimes assume that $r = 0$ to satisfy this condition. Below, we give an example where we first create the spacetime algebra $\Lambda(\mathbb{R}^{1,3})$, and then create some multivectors and perform some basic operations on them:

```
from ga_core.ga import Algebra
st = Algebra(1, 3, 0, ['e0', 'e1', 'e2', 'e3'])
# Cl(1,3): e0^2 = +1, e_i^2 = -1
a = -st.e0 + st.e1 + 2*st.e2
b = st.e2 - st.e3
a*b             # -2 - e02 + e03 + e12 - e13 - 2 e23
(geometric product)
(a*b).grade(0) # -2
(<ab>_0, the inner product)
a ^ b           # -e02 + e03 + e12 - e13 - 2 e23
(outer product)
(a ^ b) * st.I # 2 e01 - e02 - e03 + e12 + e13
(dual w.r.t. I = e0123)
~(a ^ b)        # e02 - e03 - e12 + e13 + 2 e23          (reverse)
```

The `Algebra` class stores all the operators on $\Lambda(V)$ as sparse tensors. Thus, the above operations are all implemented as sparse tensor contractions, and thus run efficiently on GPU and are natively differentiable.

Just as JAX provides mapping utilities to make extending a function to a batch of inputs easy, we write our `MVArray` as a JAX pytree so that it natively pairs with vmap:

```
def f(x):
    return x*x

xs = st.array([
    1 + st.e01,          # scalar + 2-vector
    st.e0 + st.e12,      # 1-vector + 2-vector
    st.e01 + st.e23,     # non-simple 2-vector (not a blade)
])
xs.shape                 # (3,)
f(xs[0])                 # 2 + 2 e01
jax.vmap(f)(xs)          # [2 + 2 e01, 2 e012, 2 e0123]
```

We also provide general helper functions for the constructions used in the experiments. Given `Algebra`(p,q,0), we can conformalize it to `Algebra`(p+1,q+1,0). For example, for the experiments in Section 4.4, we use conformalized $\Lambda(\mathbb{R}^n)$, $\Lambda(\mathbb{R}^{n+1,1})$. This class provides utilities like conformal embedding for points, and ways of creating various translation and rotation rotors. (Note that translations cannot be expressed as rotors in the base algebra $\Lambda(\mathbb{R}^n)$, only rotations can). Other geometric primitives are also easily accessible in this class: for example spheres in the conformal euclidean case and lightcones in conformal spacetime. For further exposition, see Doran & Lasenby (2003); Dorst et al. (2007).

For all `Algebra` classes, we also provide utilities for easy manipulation of rotors, general exponential and log maps (not closed form), Cayley transforms, along with the trivialization toolbox to easily map (co)tangents between tangent spaces.

B.2 The Multivector Derivative as a Function Transformation

Lemma 2 gives us a recipe to implement the multivector derivative in a few lines of code within our library. For a function which takes in a single multivector and outputs a single multivector, the general multivector derivative can be implemented as:

```
    def mv_deriv_fwd(f, input_blade_idxs=None):
    """ partial_I f(X) = sum_{J in I} b^J * jvp(f, X, b_J) """
    def df(X):
        alg = X.algebra
        idxs = input_blade_idxs if input_blade_idxs is
                    not None else arange(2 ** alg.n)
        b_lo = alg.basis_blades[idxs]          # MVArray [b_J]
        b_up = alg.recip_basis_blades[idxes]   # MVArray [b^J]
        jvp  = lambda A: jax.jvp(f, (X,), (A,))[1]    # dF_X[A]
        # Note below that '*' is a broadcasted geometric product!
        return (b_up * jax.vmap(jvp)(b_lo)).sum()
    return df

def mv_deriv_rev(f, output_blade_idxs=None):
    """ partial_O f(X) = sum_{K in O} vjp_<,>(f, X, b^K) * b_K """
    def df(X):
        alg  = X.alg
        idxes = input_blade_idxs if input_blade_idxs is
                    not None else arange(2 ** alg.n)
        b_lo = alg.basis_blades[idxs]          # MVArray [b_J]
        b_up = alg.recip_basis_blades[idxes]   # MVArray [b^J]
        _, f_vjp = jax.vjp(f, X)
        # metric-aware vjp: vjp(f, X, C^flat)^sharp = dF_X^*[C]
        adj = lambda C: alg.sharp(f_vjp(alg.flat(C))[0])
        # Note below that '*' is a broadcasted geometric product!
        return (jax.vmap(adj)(b_up) * b_lo).sum()
    return df
```

We provide two methods, `mv_deriv_fwd` and `mv_deriv_rev` which generalize this to functions which may have arbitrary pytrees of multivectors as inputs and ouptuts. Both methods are extensionally equal in the value they compute, but do so via forward and reverse mode AD, respectively. As mentioned in the text, you can choose which one to use depending on the context of the derivatives you are computing.

B.3 Vector Manifolds and their Transformations

Much of this work revolves around doing calculations on `VectorManifold` objects. This enables us to easily compute differential operators on the manifold, and with some mild additional assumptions, a way to also perform Monte Carlo integration on these manifolds. Below, we give the API for this class to demonstrate the capabilities of this class, where we have simplified what is written down from the actual code for the purposes of pedagogy. We emphasize the relative simplicity of the library juxtaposed with the rich, varied set of applications enabled by this framework. The derivative functions simply call `mv_deriv_fwd` with different arguments, and everything else are mere compositions of the projection operator `proj` and these derivatives.

We stress that geometric quantities like curvature information about the manifold come directly downstream from the shape operator $S(A) = \dot{\partial}_M \dot{P}(A)$. The $\cdot$'s indicate what is being differentiated (in our library we achieve this by applying a `stop_grad` to A). The curvature tensors are built from its grade-raising part, the *curl tensor* $S_a := \dot{\partial}_M \wedge \dot{P}(a)$. For tangent vectors a and b, the Riemann curvature is a bivector-valued function of the tangent bivector $a \wedge b$, from which one can compute the Ricci tensor and the scalar curvature:

$$(32)\quad R(a\wedge b) = \sigma\, P\big(S_a \times S_b\big), \qquad \mathrm{Ric}(a) = \sigma\, \partial_M \rfloor S_a, \qquad R = \partial_a \cdot \mathrm{Ric}(a) = \textstyle\sum_i \langle e^i, \mathrm{Ric}(e_i)\rangle,$$

where $A \times B = \frac{1}{2}(AB - BA)$ is the commutator, $\{e_i\}$ is any tangent frame, and $\sigma = +1$ for a definite ambient metric and -1 for an indefinite one (Hestenes & Sobczyk, 1984, Eqs. 5-1.5a, 5-1.19, 5-1.21). The sectional curvature of the tangent plane $a \wedge b$ is

$$K(a \wedge b) = -\frac{\langle R(a \wedge b)\, \widetilde{a \wedge b} \rangle_0}{\langle (a \wedge b)\, \widetilde{a \wedge b} \rangle_0}, \tag{33}$$

where the minus sign accounts for a Euclidean 2-blade squaring to -1; for a surface, $K = R/2$ is the Gaussian curvature. With these conventions the unit sphere has $K = 1$ and $R = 2$, and S^n, H^n, dS^n and AdS^n all come out correctly signed. None of these require a chart or Christoffel symbols, and are coherent in any dimension and signature.

```
class VectorManifold:

    def __init__(...):
        # There are various ways to construct a Vector Manifold,
        # but every vector manifold must provide:
        self.alg = ... # The ambient Algebra object
        self.I_fn = ... # Function M -> unit k blade of self.alg

        # If we can sample points on M (with a density),
        # then we provide a Monte Carlo, multivector valued
        # integration method
        self.sample_pts = ...
        self.dX_fn = ... # (scalar valued density)

        # Finally, we might have a bounding VectorManifold
        # to compute, e.g., normal derivatives
        self.bdry = ...

    def factorize(self, A):
        """ Given some $k$-field which is everywhere a blade,
        return a list of vector factors [a1, ..., ak]: """


    def proj(self, F):
        """ The projection operator
        Given F : M -> Cl, returns P(F) = (F | I.inverse()) | I"""

    def derivative(self, F, cov=True, deriv_type='full'):
        """ The manifold derivative operator
        Given F : M -> Cl, returns the following operators D@F, where:
        - D in [partial_M, nabla_M] # Intrinsic or covariant
        - @ in [*, ^, |] # 'full', 'outer', 'inner'"""

    def laplacian(self, F, cov = True):
        """ Second order manifold derivatives
        Given F : M -> Cl, returns one of:
        -nabla^2 F (Hodge Laplacian, grade preserving)
        -partial^2 F"""

    def shape_operator(self, F):
        """ Hestenes's shape operator
        Given F: M -> Cl, S(F) = partial_M P(stop_grad(F))"""

    def riemann_curvature(self, a, b):
        """ Riemann curvature bivector R(a ^ b) = sigma * P(S_a x S_b),
        with which Ric(a) and scalar curvature can be defined."""

    def integrate(self, F, G=None):
        """ Directed Monte Carlo integral Given F, G : M -> Cl,
        estimates int_M G dX F,  dX = I |dX|"""

    def transform(self, f, f_inv):
        """ Manifold transformation operator
        Given a right invertible f and a section f_inv,
        we can create a new VectorManifold M'"""
```

As an example of how we used this library for the experiments, see these examples which build on the API we provide:

```
# (1) Divergence-free flow on any 2-manifold
v = M.derivative(lambda x: phi_theta(x) * M.I_fn(x), deriv_type='inner')
# nabla_M | (phi I)
M.derivative(v, deriv_type='inner')
# = 0 identically

# (2) Hyperbolic plane from a flat disk in Cl(2,1)
# Geodesic polar coordinates on the hyperboloid H^2
# in Cl(2,1): apex e_n, e_n^2 = -1
def polar_to_hyperboloid(u):
    r = norm(u)
    return cosh(r) * e_n + sinhc(r) * u

def hyperboloid_to_polar(x):
    x_s = x.grade(1) - x_n(x) * e_n
    s = norm(x_s)
    return (arcsinh(s) / s) * x_s

H2 = disk.transform(polar_to_hyperboloid, hyperboloid_to_polar)

# (3) Differentiable Euler characteristic of a learned surface
M1  = square.transform(flow_theta, flow_theta_inv)
chi = M1.integrate(lambda x: M1.I_fn(x).inverse(),
                   G=lambda x: M1.scalar_curvature(x) / (4*pi))
jax.grad(loss)(theta)
# loss built from chi, differentiated end to end
```

With the directed measure $dX = I(x)|dX|$ and dS the corresponding measure on ∂M (if it exists), the fundamental theorem of multivector calculus becomes:

$$\int_M \dot{G}\, dX\, \dot{\partial}_M F \;=\; \int_{\partial M} G\, dS\, F, \tag{34}$$

for fields F, G (Doran & Lasenby, 2003; Hestenes & Sobczyk, 1984). Its grade components recover Stokes', Green's, and the divergence theorem in any dimension. Our toolkit provides a computational way to approximate either side of this equation. In addition to giving you the ability to solve PDEs, this theorem also gives rise to constructive ways to approximate both inverse functions via the inverse function theorem or implicit functions via the implicit function theorem.

C Experiment Details

C.1 Shared Setup

All experiments are implemented in JAX with 64-bit floats. Except for the implicit-surface networks of Appendix C.3.A, every experiment runs on CPU; wall-clock times quoted below are from an Apple M3 laptop unless stated otherwise. The implicit-surface networks were trained on a cluster with one Nvidia L40 GPU for speed (but the training also could be done on the laptop).

C.2 Explicit Manifold Learning

C.2.A Non-Euclidean Tilings

Symmetries This experiment requires 3 inputs: a manifold (as a `VectorManifold` object), a symmetry group acting on that manifold, and a target shape. On S^2, the group is the icosahedral rotation group $I \cong A_5$ of order 60, generated by rotations of order 5, 3 and 2 about a vertex, a face centre and an edge midpoint of the icosahedron. The initial tiling, the icosidodecahedron (12 pentagons and 20 triangles), is the orbit of one pentagonal and one triangular seed tile under this

group. On H^2 the group is the $(2,3,7)$ Coxeter group, generated by the reflections in the sides of a hyperbolic triangle with angles $\pi/2$, $\pi/3$ and $\pi/7$. Each reflection is a single vector versor $X \mapsto -rXr^{-1}$ in the conformal algebra $\mathrm{Cl}(3,1)$ (in fact in its $\mathrm{Cl}(2,1)$ subalgebra spanned by e_1, e_2, e_n) (Doran & Lasenby, 2003; Dorst et al., 2007). The rectified $\{3,7\}$ tiling is the orbit of a seed tile under this group. This construction easily generalizes to any hyperbolic $\{p,q\}$ tiling, i.e. any p, q with $\frac{1}{p} + \frac{1}{q} < \frac{1}{2}$. Because this group is infinite, for the figures we enumerate it breadth-first up to hyperbolic radius $\rho \leq 3.5$, for a total of 304 tiles (208 triangles and 96 heptagons); see Fig. 11.

Model and Loss Recall that we desire a divergence free 1-field on our 2 dimensional manifold M, and achieve this via $v_\theta = \nabla_M \rfloor (\phi_\theta I_M)$ for a scalar valued neural network ϕ_θ, which guarantees v_θ is divergence free by construction. This network is a random-Fourier-feature network (64 frequencies giving 128 features, with scale 4.0 on S^2 and 2.5 on H^2) followed by an MLP of width 64, depth 2 and GELU activations. It depends only on position and not on time, so v_θ is an autonomous vector field whose time-1 flow is a diffeomorphism. To make the flow equivariant under the tiling's symmetry, we symmetrize the potential,

$$\Phi_\theta(x) = \frac{1}{|G|} \sum_{g \in G} \mathrm{sgn}(g)\, \phi_\theta(g \cdot x), \tag{35}$$

where $\mathrm{sgn}(g) = -1$ for orientation-reversing g: a reflection flips I_M, so Φ_θ must be odd under reflections for v_θ to be equivariant. On S^2, G is the full rotation group. On H^2, G is a finite patch of 96 group elements (those mapping the origin into the 8 nearest tiles), so equivariance there is only approximate.

On H^2 we work on the hyperboloid in $\mathbb{R}^{2,1}$, parameterized by the geodesic polar coordinates $u \mapsto \cosh|u|\, e_n + \frac{\sinh|u|}{|u|} u$ of Appendix B.3, and display the result on the Poincaré disk by stereographic projection. To realize our tiling, we take the boundary of a single seed tile and integrate v_θ between $t = 0$ and $t = 1$ acting on these points using a fixed step size RK4 integrator (15 steps, $h = 1/15$). On S^2 the points are reprojected onto the sphere after every step; on H^2 we integrate in the parameter u, pushing v_θ forward by the differential of the inverse parameterization (a single `jvp`), so no reprojection is needed.

Let $x_1, \ldots, x_T$ be the points on the boundary of the seed tile (7 per edge, so $T = 35$ on S^2 and $T = 49$ on H^2), and let Φ^θ denote the RK4 flow map. Each x_i is matched to the target point y_i at the same polar angle ϑ_i about the tile centre. The targets are given by the radius profiles $r(\vartheta) = R_0\big(1 + 0.3\cos(5\vartheta + 0.6)\big)$ on S^2 (a chiral, $\mathbb{Z}_5$-symmetric pinwheel) and $r(\vartheta) = R_0\big(1 + 0.18\cos 7\vartheta + 0.05\cos 14\vartheta\big)$ on H^2 (a D_7-symmetric gear), each mapped onto the manifold by the exponential map at the tile centre, with R_0 the mean radius of the seed pentagon on S^2 and the circumradius of the seed heptagon on H^2. We minimize

$$\mathcal{L}(\theta) = \frac{1}{T} \sum_{i=1}^{T} \big\| \Phi^\theta(x_i) - y_i \big\|^2, \tag{36}$$

where $\|\cdot\|$ is the Euclidean norm in $\mathbb{R}^3$ on S^2 and in Poincaré-disk coordinates on H^2.

Training & Metrics We optimize the parameters of the neural network with the Adam optimizer with a learning rate of $1.5 \cdot 10^{-3}$ and gradient clipping at global norm 2, for 80 full-batch steps (seed 42). We achieve final losses of $1.25 \cdot 10^{-3}$ on the sphere and $9.49 \cdot 10^{-4}$ on the disk. We show how the tiling looks during the integration, along with the training loss curves, in Fig. 12. Training for each took < 1 hour on the laptop.

C.2.B Backpropagation Through the Euler Characteristic

Flow Parameterization We parameterize a velocity field $v_\theta(t, x)$ on $\mathbb{R}^3$ as a ConcatSquash MLP (Grathwohl et al., 2019) with three layers $3 \to 96 \to 96 \to 3$ and tanh activations, where each layer computes $W_1 x \odot \sigma(W_2 t) + W_3 t$, so that time enters through gating and bias. Integrating v_θ from $t = 0$ to $t = 1$ with fixed-step RK4 (4 steps) gives the flow map φ_θ, a diffeomorphism of $\mathbb{R}^3$ (Chen et al., 2018). As a corollary, any manifold $M \subset \mathbb{R}^3$ flowing under φ_θ will be diffeomorphically mapped to a new manifold M'; our AD toolbox immediately gives us access to all the differential

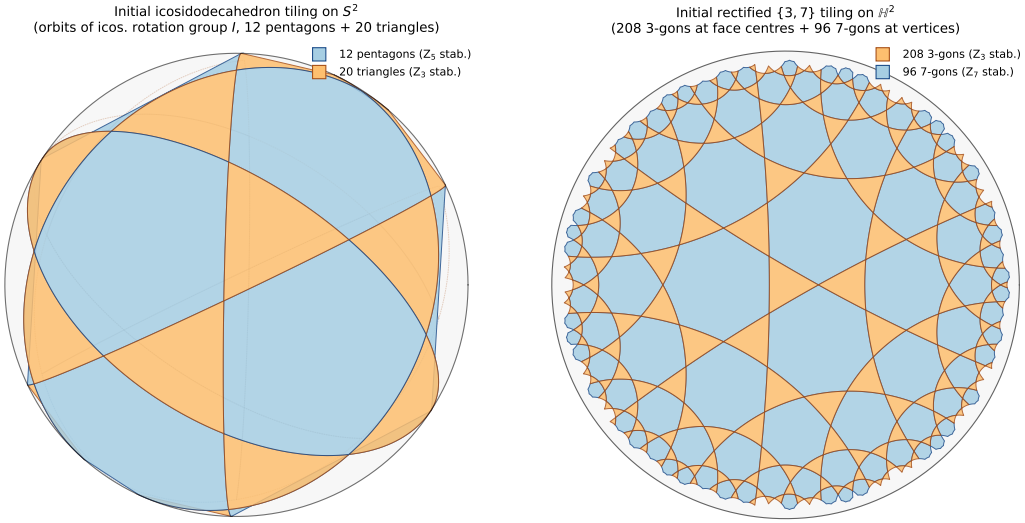


Figure 11: Initial tilings. *Left:* the icosidodecahedron on S^2. *Right:* the rectified $\{3, 7\}$ tiling of the Poincaré disk up to hyperbolic radius 3.5.

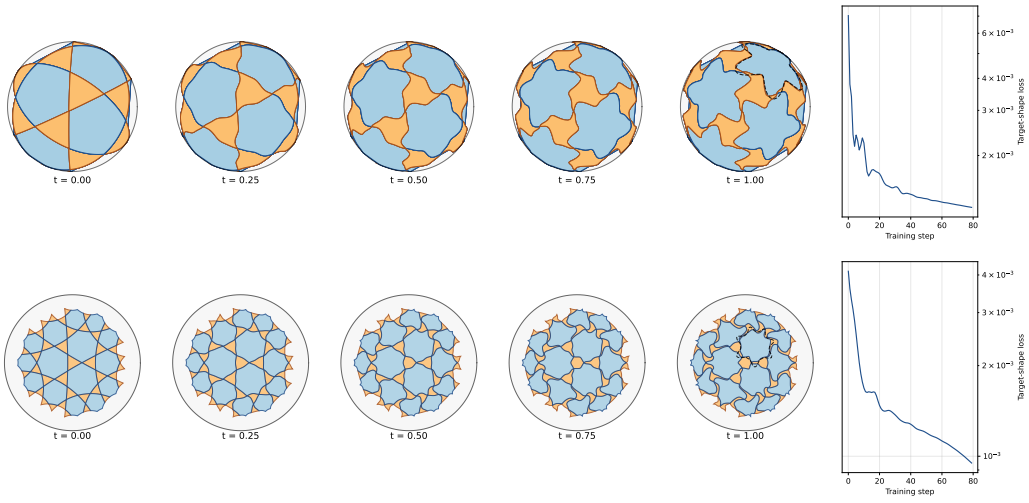


Figure 12: The tilings under the learned divergence-free flow at $t = 0, 0.25, 0.5, 0.75, 1$. *Top:* pentagons (blue) deformed into a chiral $\mathbb{Z}_5$ pinwheel on S^2, under the icosahedral rotation group. *Bottom:* heptagons (blue) deformed into a D_7 gear on H^2, under the $(2, 3, 7)$ Coxeter group. In both, the triangles (orange) flow under the same trained velocity field, and the dashed curve at $t = 1$ is the target. The rightmost panels show the training loss (log scale) against the training step.

operators on M'. Additionally, if M is compact, then M' will also be and so we can integrate on this manifold. This is what allows us to estimate quantities such as the Euler characteristic of the manifold. For the particular experiment in the main text, we took M to be the square $[-1,1]^2 \times \{0\} \subset \mathbb{R}^3$, parameterized by $u \in \mathcal{D} = [-1,1]^2$.

Loss For the experiment in the paper, we compute the scalar curvature R (Appendix B.3) at each sampled point of the surface, using the pushed-forward coordinate vectors $r_u = \partial_u \varphi_\theta$, $r_v = \partial_v \varphi_\theta$ as the tangent frame:

$$R = \partial_a \cdot \mathrm{Ric}(a) = \textstyle\sum_i \langle e^i, \mathrm{Ric}(e_i)\rangle, \qquad \{e_1, e_2\} = \{r_u, r_v\}. \tag{37}$$

On a 2-manifold the scalar curvature is simply a rescaling of the Gaussian curvature, $R = 2K$; Gaussian curvature, however, is only defined for 2-manifolds, whereas R is defined in any dimension and signature and our library computes it with no dimension-specific code. We estimate the Euler characteristic from B uniform samples $u_i \in \mathcal{D}$ via Eq. 8:

$$\hat{\chi}_\theta = \frac{|\mathcal{D}|}{4\pi B} \sum_{i=1}^{B} R_i \, |\mathrm{d}X|_i, \qquad |\mathrm{d}X|_i = |r_u \wedge r_v|(u_i). \tag{38}$$

So, the loss function we minimize, matching the unit sphere's scalar curvature $R_* = 2$, is:

$$\mathcal{L}(\theta) = \frac{\sum_i |\mathrm{d}X|_i \left(\frac{1}{2}(R_i - R_*)\right)^2}{\sum_i |\mathrm{d}X|_i} + \lambda_{\mathrm{GB}} \left(\hat{\chi}_\theta/2 - 1\right)^2, \qquad \lambda_{\mathrm{GB}} = 1.5. \tag{39}$$

As stated in the text, we optimize this function with and without the Gauss–Bonnet term ($\lambda_{\mathrm{GB}} = 0$), and see the difference in the learned manifold M'. For these experiments, we train with $B = 96$ fresh samples per step and evaluate with $B = 4096$.

It may seem paradoxical that $\hat{\chi}_\theta$ changes during training at all, since χ should be invariant with respect to any diffeomorphism. The resolution is that Gauss–Bonnet for a surface with boundary reads

$$\int_M K \, |dX| + \int_{\partial M} k_g \, dS = 2\pi\chi(M) \tag{40}$$

with k_g the geodesic curvature of the boundary. Since the square has $\chi = 1$, our estimator satisfies $\hat{\chi}_\theta = 1 - \frac{1}{2\pi}\int_{\partial M} k_g \, dS$. Driving $\hat{\chi}_\theta \to 2$ therefore forces $\int_{\partial M} k_g \, dS \to -2\pi$, i.e. the boundary must shrink to a small loop that pinches the surface closed, which is exactly the behavior in Fig. 3.

Training & Metrics We train the parameters θ via the Adam optimizer with a learning rate of $2 \cdot 10^{-3}$ and gradient clipping at global norm 5 (seed 0), for 4000 iterations with the Gauss–Bonnet term and 150 iterations without it. We show loss curves, along with the intermediate Euler characteristic estimates, in Figs. 13 and 14. The final Euler characteristic estimate (on $B = 4096$ evaluation samples) comes out to $\hat{\chi} \approx 1.97$, with surface area 12.29 (98% of 4π) and area-weighted mean Gaussian curvature 1.006, indicating that the learned M' is pretty close to a fully closed unit sphere, whereas without the Gauss–Bonnet term the square collapses to a small spherical cap with $\hat{\chi} \approx 0.05$, area 0.34 and mean curvature 0.98. On an Apple M3 laptop CPU, training takes 444 s with the Gauss–Bonnet term (4000 iterations) and 78 s without it (150 iterations). We also show more intermediate steps of the final learned flows being applied in the top rows of the same figures.

C.3 Implicit Manifold Learning

C.3.A Minimal Surfaces

Surface Representation As stated in the text, the idea of this experiment is to parameterize an $(n-k)$-dimensional surface with boundary Γ via the k-form $\omega_\theta = \omega_\Gamma + \partial_{\mathcal{I}_1} \wedge \eta_\theta$ where ω_Γ is the Biot-Savart field of Eq. 9. Here $G(x) = |x|^{2-n}/((n-2)S_{n-1})$ is the Green's function of the Laplacian in $\mathbb{R}^n$, with S_{n-1} the area of the unit $(n-1)$-sphere, so $\partial_{\mathcal{I}_1} G(x) = -x/(S_{n-1}|x|^n)$. Computationally, we evaluate the integral on a fixed closed polyline discretization of Γ, summing $\partial_{\mathcal{I}_1} G(x - m_i) \wedge \Delta\Gamma_i$ over segment midpoints m_i and segment vectors $\Delta\Gamma_i$ and dualizing. We use 120 segments in codimension one; in codimension two we use 480 segments for the torus knot and

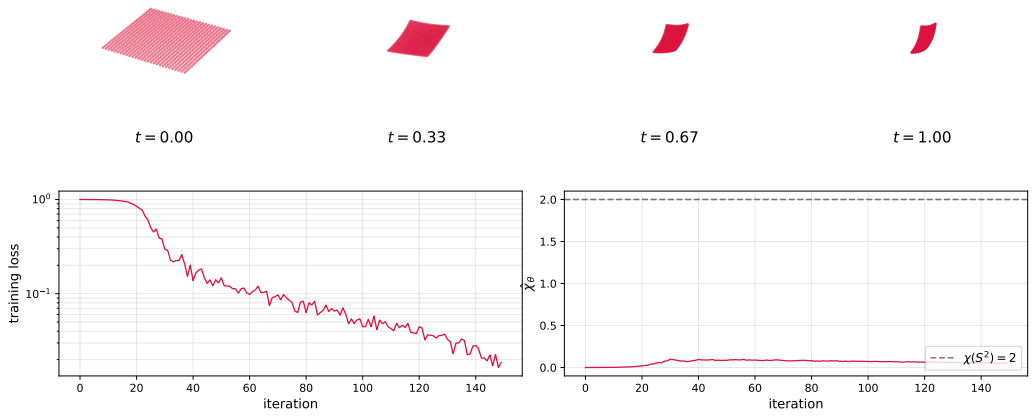


Figure 13: The square under the curvature loss alone ($\lambda_{\mathrm{GB}} = 0$). *Top:* the learned flow at $t = 0, \frac{1}{3}, \frac{2}{3}, 1$, which collapses the square into a small spherical cap. *Bottom:* training loss (left) and the Monte Carlo Euler characteristic estimate $\hat{\chi}_\theta$ (right) over training; without the topological term, $\hat{\chi}_\theta$ stays far from the sphere's value of 2.

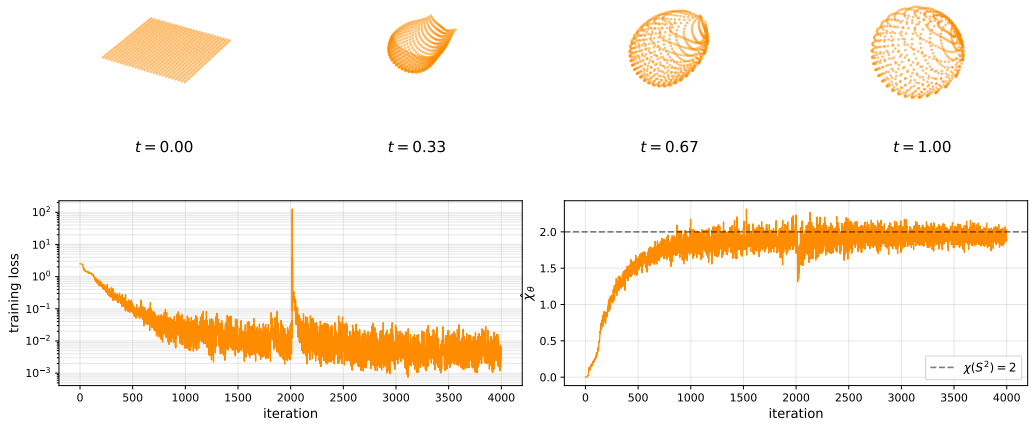


Figure 14: The square under the curvature loss with the Gauss–Bonnet term ($\lambda_{\mathrm{GB}} = 1.5$). *Top:* the learned flow at $t = 0, \frac{1}{3}, \frac{2}{3}, 1$, which closes the square into a sphere. *Bottom:* training loss (left) and the Monte Carlo Euler characteristic estimate $\hat{\chi}_\theta$ (right) over training, which approaches 2.

320 for the polynomial graph, and regularize the kernel by replacing $|x|^2$ with $|x|^2 + \epsilon^2$, $\epsilon = 0.06$. η_θ is a random-Fourier-feature network (Gaussian frequencies, sin and cos features) followed by a softplus MLP producing a $(k-1)$-form: in codimension one, a scalar potential with 1024 frequencies at scale 2.0 and an MLP of width 256 and depth 4; in codimension two, a 1-form with 96 frequencies at scale 0.5 and an MLP of width 128 and depth 3 whose last layer is initialized to zero, so that training starts from $\omega_\theta = \omega_\Gamma$. Again using $(\partial_{\mathcal{I}_1}\wedge)^2 = 0$, this parameterization ensures that $\partial_{\mathcal{I}_1} \wedge \omega_\theta = \delta_\Gamma$, which is equivalent to the condition that the represented surface has boundary Γ.

The Mass Norm As stated in the text, the objective that we minimize is the *mass norm* of ω_θ. While we gave an informal definition for the mass norm, the formal definition is:

$$\|\omega\| := \inf\left\{ \textstyle\sum_j |\beta_j| \;:\; \omega = \textstyle\sum_j \beta_j B_j,\ B_j \text{ unit } k\text{-blades} \right\}. \tag{41}$$

There is no closed form for this infimum for $3 \le k \le n-3$, so this is why we focus our experiments on the $(n,k) = (3,1)$ and $(4,2)$. For $k = 1$ the mass norm is the Euclidean norm. For the $n = 4, k = 2$ case, there is a closed form (Federer, 1969) that is easily differentiable almost everywhere: splitting ω into its self-dual and anti-self-dual parts $\omega_\pm = \frac{1}{2}(\omega \pm \star\omega)$,

$$\|\omega\| = \sqrt{2}\,\max\big(|\omega_+|, |\omega_-|\big). \tag{42}$$

Minimizing this loss ends up turning ω_θ into an approximately simple bivector (one that is a blade). In practice we smooth the pointwise norm as $m_\delta(\omega) = \sqrt{\|\omega\|^2 + \delta^2}$, and minimize a Monte Carlo estimate of its integral over a box $\Omega \supset \Gamma$,

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \frac{m_\delta(\omega_\theta(x_i))}{q(x_i)}, \qquad x_i \sim q, \tag{43}$$

with $N = 4096$. For the first 25% of training q is uniform on Ω; afterwards it is a mixture of 30% uniform samples and 70% samples from a lattice with cell probabilities proportional to $|\omega_\theta|$, refreshed every 250 steps. The box is $[-1.5, 1.5]^3$ in codimension one and a cube of half-width $\max|\Gamma| + 0.5$ in codimension two. In codimension one $\delta = 10^{-6}$ is fixed; in codimension two δ is annealed log-linearly from $5 \cdot 10^{-2}$ to $5 \cdot 10^{-6}$ over training.

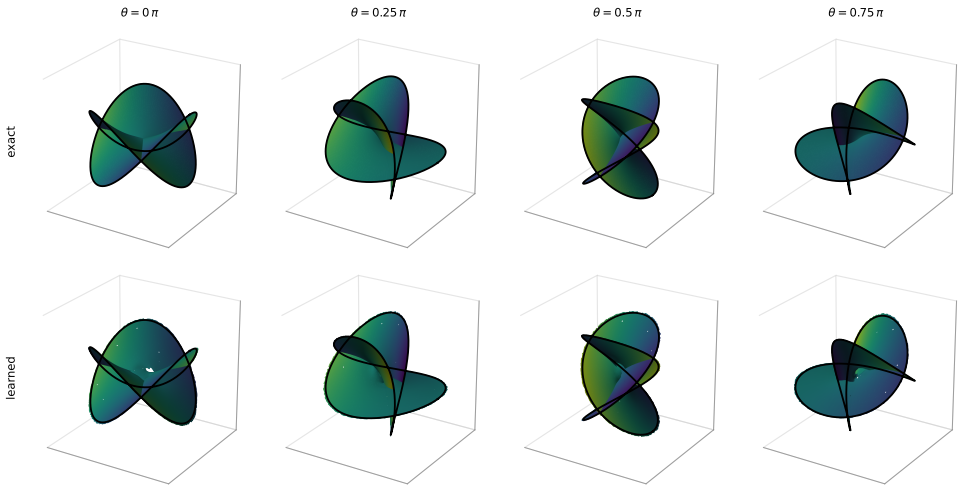
Extraction The learned surface is read off from the ω_θ alone; the extractor never sees the closed-form surface. In codimension one, $\omega_\theta = \omega_\Gamma + \nabla f$ with f the learned scalar potential, and the surface is the level set of f at its mean value on Γ, computed by marching cubes on a 192^3 grid. We keep only the faces where the Biot–Savart field ω_Γ and ∇f point the same way and the total field is large, i.e. where the current actually concentrates, then clean the mesh (largest component, hole filling), smooth it, and snap its rim onto Γ. In codimension two the surface is the ridge of $|\omega_\theta|$. We seed points on a grid around Γ (keeping the top 15% by mass) and on Γ itself, and move each by a few Newton steps that maximize $\log|\omega_\theta|$ within the normal plane of ω_θ (a simple ω_θ determines its own tangent and normal planes). We keep points at a local maximum inside a neighbourhood of the convex hull of Γ, densify the result by advancing along the tangent planes, and discard points that are not a maximum relative to a ring of radius 2.5ϵ in the normal plane. For rendering only, the points are meshed by local Delaunay triangulation in their tangent planes.

Training & Metrics We use Adam with gradient clipping at global norm 1 and a staircase exponential learning-rate decay. In codimension one we train for 10^5 steps with learning rate $5 \cdot 10^{-4}$, decayed by 0.6 every 10^4 steps, which takes about 40 minutes per boundary on one GPU. In codimension two we train for 27,000 steps with learning rate $5 \cdot 10^{-3}$, decayed by 0.7 every 2,700 steps (seed 0). The codimension-two ground truths are the graph of $w = z^3 - \lambda z$ over $|z| \le \rho$ ($\rho = 0.8$, $\lambda = 0.6$; exact area $\pi\rho^2(1+\lambda^2) + 3\pi\rho^6 = 5.205$) and the branched disk $u \mapsto (ru^2, su^3)$, $|u| \le 1$ ($r = 0.7$, $s = 0.5$; exact area $\pi(2r^2 + 3s^2) = 5.435$), whose boundary is the $(2,3)$ torus knot. Both are holomorphic curves and hence mass-minimizing (Federer, 1969). Table 1 compares the extracted point clouds against dense samples of the exact surfaces.

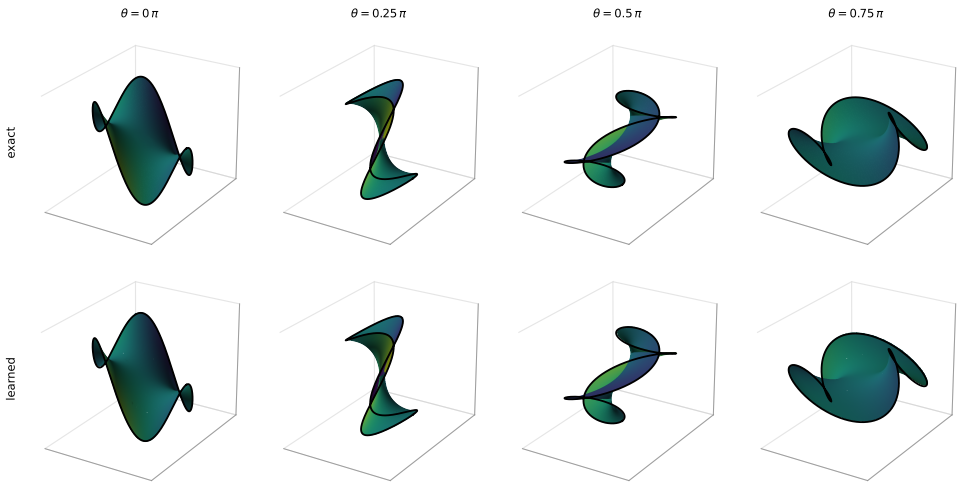
In addition to the two surfaces shown in the main text, we tested three further boundaries, with results in Table 1 and Fig. 16: the holomorphic graph $w = z^2$ over $|z| \le 0.8$ (exact area 4.584); two circles $|z| = r$, $|z| = R$ on the curve $zw = c$ ($r = 0.55$, $R = 1$, $c = 0.35$), which with opposite orientations bound the holomorphic annulus $\{zw = c\}$ (area 3.079) and with equal orientations bound the pair of anti-holomorphic disks $w = (c/\rho^2)\bar{z}$ (area 5.749). In every case the extraction matches the exact surface closely. Fig. 15 shows the two surfaces of the main text at more projection angles.

Table 1: Extracted codimension-two surfaces against the exact minimal surfaces. Chamfer is the symmetric mean nearest-neighbour distance; Hausdorff distances are one-sided (extraction to reference); coverage is the fraction of the reference within 0.03 of the extraction, and spurious the fraction of extracted points farther than 0.03 from the reference. Area is that of the render mesh.

boundary	Chamfer	Hausdorff (95%)	Hausdorff (max)	coverage	spurious	area (exact)
torus knot	0.0099	0.014	0.050	94.8%	0.13%	5.53 (5.44)
polynomial graph	0.0150	0.028	0.046	99.96%	1.51%	5.12 (5.21)
graph $w = z^2$	0.0123	0.023	0.041	100.0%	0.42%	4.69 (4.58)
annulus $zw = c$	0.0083	0.016	0.034	100.0%	0.08%	3.24 (3.08)
two disks $w = (c/\rho^2)\bar{z}$	0.0126	0.021	0.048	97.4%	1.04%	5.80 (5.75)

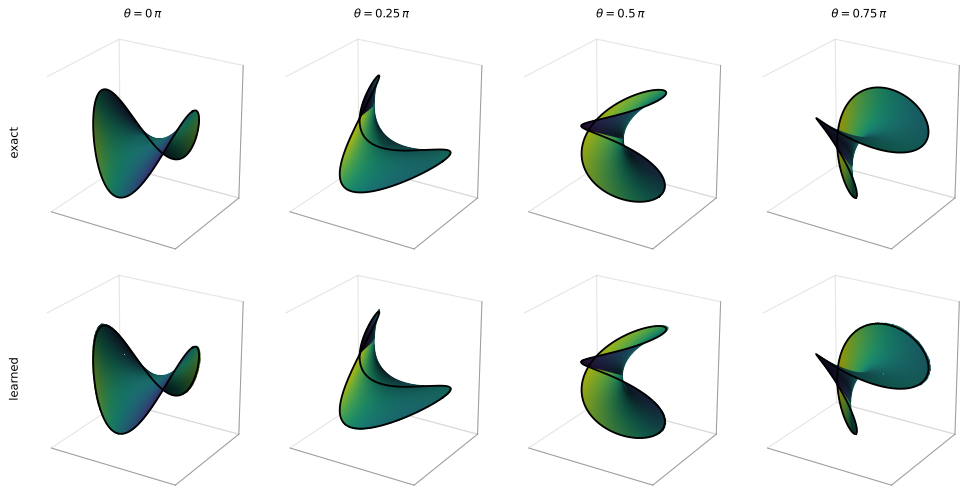


branched disk bounding the $(2, 3)$ torus knot

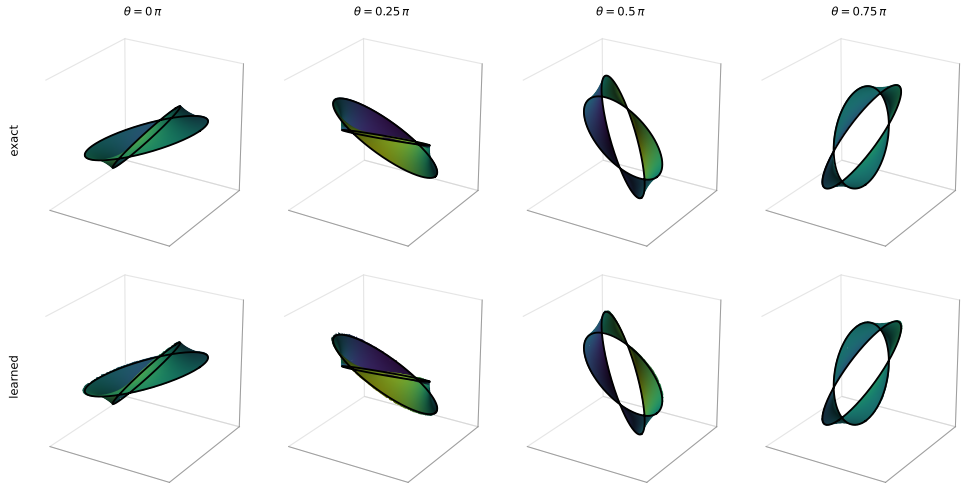


graph $w = z^3 - \lambda z$

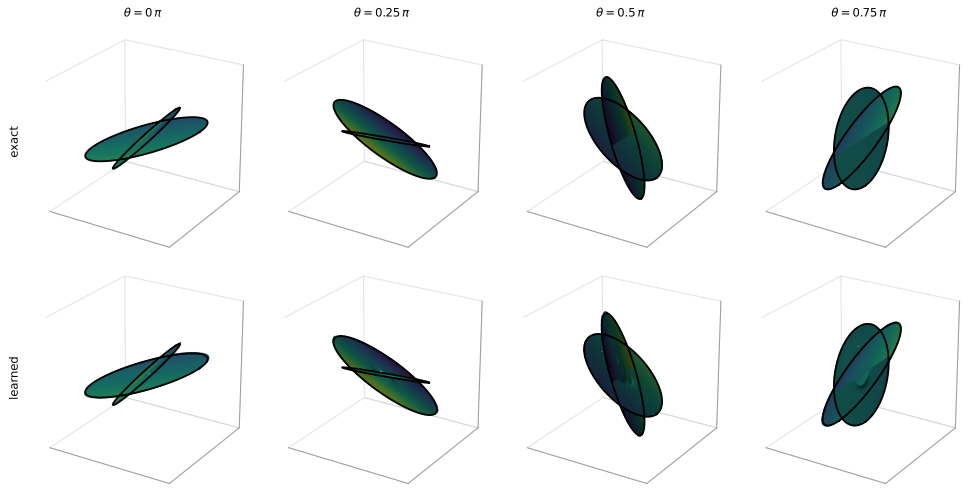
Figure 15: The two codimension-two surfaces of Figs. 5 and 6 at four projection angles $\theta = 0, \pi/4, \pi/2, 3\pi/4$. Each block shows the exact minimal surface (top row) and the surface extracted from the learned field (bottom row).



graph $w = z^2$



annulus $zw = c$



two disks $w = (c/\rho^2)\bar{z}$

Figure 16: The three further codimension-two boundaries of Table 1 at four projection angles $\theta = 0, \pi/4, \pi/2, 3\pi/4$. Each block shows the exact minimal surface (top row) and the surface extracted from the learned field (bottom row), rendered as in Fig. 15.

C.4 Inverse Design in Spacetime

Forward Simulator We work in the spacetime algebra $\mathrm{Cl}(1,3)$ with basis e_0, e_1, e_2, e_3 with $e_0^2 = 1$, $e_1^2 = e_2^2 = e_3^2 = -1$. We parameterize the particle's spatial velocity $v_\theta(t)$ in lab time t via an MLP (width 64, depth 3, GELU activations) of the Fourier features $(t, \sin f_k t, \cos f_k t)$ with 8 frequencies f_k evenly spaced in $[0.5, 6]$, and constrain the raw output w to $v = w/\sqrt{1+|w|^2}$, which enforces $|v| < 1$. The reason for this is that we want the resulting worldline $x_\theta(t) = t\, e_0 + r_\theta(t)$ to be *timelike* (slower than speed of light), which any particle with mass should obey. The spatial position $r_\theta(t)$ is the trapezoidal integral of v_θ on 341 knots over $t \in [-1, 7]$, with cubic Hermite interpolation between knots so that $\dot{r}_\theta = v_\theta$ exactly, and the particle's 4-velocity is proportional to $e_0 + v_\theta$. Then, suppose we observe something at spacetime location y. We desire to find the *retarded time* τ_r, the time at which the particle emitted the signal that the observer at y receives, i.e. the root of $(y - x_\theta(\tau_r))^2 = 0$ on the past light cone of y. This boils down to a 1 dimensional root finding problem, which we solve with a differentiable bisection search on $[-50, t_y - 10^{-3}]$ to a tolerance of 10^{-8} using Optimistix (Rader et al., 2024). The root is differentiated with respect to the trajectory via the implicit function theorem (Blondel et al., 2022; Griewank & Walther, 2008). This allows us to compute the Liénard–Wiechert potential

$$A(y) = \frac{q}{4\pi} \frac{u(\tau_r)}{|\langle y - x_\theta(\tau_r), u(\tau_r)\rangle|}, \qquad u = e_0 + v_\theta, \tag{44}$$

with $q = 1$; the normalization of u cancels between numerator and denominator. From this, the Faraday bivector is the single call $F = \partial_{\mathcal{I}_1} \wedge A$. Using the decomposition $F = E + IB$ with pseudoscalar $I = e_{0123}$, where $E = \frac{1}{2}(F - e_0 F e_0)$ and $B = \frac{1}{2} I^{-1}(F + e_0 F e_0)$, we get a recipe to read off the E and B fields in the $\sigma_k = e_k e_0$ frame, which is how we generate the streamline plots in Fig. 8. The final loss is a relative mean squared error between the predicted and observed F over the N observation events,

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \frac{\|F_\theta(y_i) - F_*(y_i)\|^2}{\|F_*(y_i)\|^2}, \tag{45}$$

where $\|\cdot\|$ is the Euclidean norm on the bivector coefficients. We normalize per event because $\|F\|^2 \sim |y - x|^{-4}$ spans more than four orders of magnitude across the observation events (a factor of about 200 between the 5th and 95th percentiles).

Training & Metrics The target trajectory worldline is the helix $r_*(t) = (0.4\cos 2t, 0.4 \sin 2t, 0.25t)$ in lab time t, and we reconstruct it over the window $t \in [1, 4]$. To provide data for the model, we evaluate the exact field F_* on a spatial grid of $5 \times 5 \times 4$ points covering $[-1.8, 1.8] \times [-1.8, 1.8] \times [-0.4, 1.6]$ at 21 evenly spaced times in $[1, 6.5]$, excluding events within a radius 1 of the true worldline where the potential is singular; this leaves 1944 noise-free training events. The 20 midpoint time slices (1848 events) are held out for model selection. The observation times extend past the reconstruction window because an observation at time t and distance $d \geq 1$ from the particle constrains the worldline only at the retarded time $t - d$. We optimize the trajectory with Adam for 7000 full-batch steps, with gradient clipping at global norm 1 and a learning rate warmed up over 100 steps to $3 \cdot 10^{-3}$ and then cosine-decayed to $2 \cdot 10^{-5}$ (seed 1), keeping the checkpoint with the lowest held-out loss.

The recovered worldline has a mean position error of $6 \cdot 10^{-4}$ and a maximum of $1.2 \cdot 10^{-3}$ over $t \in [1, 4]$, i.e. 0.2% and 0.3% of the orbit radius. Training takes 6 minutes on an Apple M3 laptop CPU. As a check of the forward model, the vacuum Maxwell residual $\partial_{\mathcal{I}_1} \cdot F$ at 16 random events away from the worldline has maximum $1.8 \cdot 10^{-7}$ and mean $1.6 \cdot 10^{-8}$, against field magnitudes of order one, consistent with the root-finder tolerance.

Causal coverage matters: training the same recipe on observation slices confined to the reconstruction window $[1, 4]$ (12 slices) recovers the early worldline (mean error $1.7 \cdot 10^{-3}$ on $[1, 2]$) but not the tail, which no observation can see: on $[3, 4]$ the mean error is $4.6 \cdot 10^{-2}$ (maximum 0.25), against $5 \cdot 10^{-4}$ (maximum $8 \cdot 10^{-4}$) with the extended observation window.

C.5 Optimization on Rotor Groups

Multiplicative Descent Let $G \subset \mathrm{Spin}(p,q)^+$ be a rotor group whose Lie algebra $\mathfrak{g}$ is spanned by bivectors $\{B_k\}_{k=1}^{\dim \mathfrak{g}}$, with reciprocal basis $\{B^k\}$ in $\mathfrak{g}$. The left-trivialized differential of f at g is the covector $df_e(g)[B] = df_g[gB]$ on $\mathfrak{g}$. Writing $\partial f(g) := \partial_{\mathcal{I}_{\mathrm{even}}} f(g)$ and using that the scalar part is cyclic, $df_g[gB] = \langle \partial f(g), gB\rangle = \langle \partial f(g)\, g, B\rangle$, so its coordinates in the basis $\{B^k\}$ are

$$c_k(g) = df_e(g)[B_k] = \big\langle \partial f(g)\, g,\ B_k \big\rangle, \qquad df_e(g) = \textstyle\sum_k c_k(g)\, B^k = P_{\mathfrak{g}^*}\big(\partial f(g)\, g\big). \tag{46}$$

This is Eq. 10 of the main text, where the restriction to $\mathfrak{g}$ is made explicit by $P_{\mathfrak{g}^*}$: $\partial f(g)\, g$ is a general even multivector, and only its pairing with the generators enters the update. For a scalar loss, $\partial f(g)$ is a single `vjp` (Lemma 2). In practice we feed the coordinates c_k to Adam, whose step $u_t \in \mathfrak{g}$ is exponentiated, $g_{t+1} = g_t \exp(u_t)$. This ensures every iterate is a valid rotor. The exponential is the algebra's general multivector exponential; in $\mathrm{Cl}(1,3)$ we evaluate it as the matrix exponential of the 16×16 left-multiplication operator $X \mapsto BX$, which JAX differentiates exactly. A subgroup of G spanned by a subset of the generators is selected by a $0/1$ mask on the coordinates c_k, so several subgroups of the same algebra can be optimized in a single vectorized call.

Choosing a Positive Definite Metric As discussed in the main text, $\sharp : \mathfrak{g}^* \to \mathfrak{g}$ must come from a positive-definite inner product on $\mathfrak{g}$ for $-\eta\, df_e^\sharp$ to be a descent direction, since then $df_e[-\eta\, df_e^\sharp] = -\eta \|df_e\|^2 < 0$. The multivector inner product $\langle A, B\rangle = \langle AB\rangle_0$ fails this in every case we consider. The metric on $\mathfrak{spin}(n)$ it is negative definite, since Euclidean bivector blades square to -1, so the "natural" step would be an ascent step. On $\mathfrak{spin}(1,3)$ it is indefinite (boosts $e_0 e_i$ square to $+1$, rotations $e_i e_j$ to -1), and on the translation subalgebra of $\mathrm{CGA}(3)$ it vanishes identically, since the generators $e_i \wedge e_\infty$ are null and mutually orthogonal. Nor is there a canonical alternative: non-compact groups admit no bi-invariant metric, so some non-invariant choice is unavoidable. We declare the chosen generators orthonormal,

$$\langle B_j, B_k\rangle_\sharp := \delta_{jk}, \qquad df_e(g) = \textstyle\sum_k c_k B^k \ \longmapsto\ df_e(g)^\sharp = \sum_k c_k B_k, \tag{47}$$

which is the identity on coordinates. This coincides with the choice a coordinate-array AD system makes implicitly, but here it is an explicit, replaceable ingredient.

Chart-based baselines We compare against two updates that keep chart coordinates $b \in \mathbb{R}^{\dim \mathfrak{g}}$, $g = \exp(\sum_k b_k B_k)$, as the optimizer state. The *static chart* (static trivialization, the recipe of Lezcano-Casado (2019)) runs Adam on b directly. Its gradient is not df_e: with $T_B := \exp(-B)\, d\exp_B = \varphi(-\mathrm{ad}_B)$, $\varphi(x) = (e^x - 1)/x$ and $\mathrm{ad}_B C = BC - CB$, the chart gradient is $T_B^*\, df_e(\exp B)$, which agrees with df_e only at $B = 0$. The *corrected chart* undoes this factor, applying $(T_B^{-1})^*$ to the gradient and T_B^{-1} to the Adam step; it agrees with the multiplicative update to first order and differs only in the retraction. We evaluate $\varphi(-\mathrm{ad}_B)$ by a block matrix exponential. Both chart updates break down where T_B is singular.

Subgroups of the conformal group The ten bivector generators of $\mathrm{CGA}(3) = \mathrm{Cl}(4,1)$, written in the null basis $e_o = \frac{1}{2}(e_+ + e_-)$, $e_\infty = e_- - e_+$, split into three rotations e_{ij}, three translations $e_i \wedge e_\infty$, three transversions $e_i \wedge e_o$, and the dilation $e_+ \wedge e_-$. The four groups of Fig. 9 are coordinate subspaces of this basis: translations $T(3)$ (dimension 3), rigid motions $SE(3)$ (rotations and translations, dimension 6), similarities $\mathrm{Sim}(3)$ (rigid motions and dilation, dimension 7), and the full conformal group (dimension 10). The source cloud P has 29 points along three perpendicular arms of lengths 1.5, 1.0 and 0.5, so that chirality and per-axis scale are visible. A single teacher motor $M^\star = \exp(B^\star)$ is drawn from the full conformal algebra (seed 13; coefficient scales 0.5 for translations, 0.3 for rotations, 0.25 for the dilation and 0.12 for transversions), and the target is $Y = \mathrm{down}\big(M^\star\, \mathrm{up}(P)\, \widetilde{M^\star}\big) + \sigma\epsilon$, where $\mathrm{up}(x) = e_o + x + \frac{1}{2}|x|^2 e_\infty$. The loss is the sum of squared residuals over the $29 \times 3 = 87$ coordinates. All four groups start from the same masked initialization and are optimized in one vectorized run of 500 multiplicative Adam steps at learning rate 0.1, which takes about 4 s on an Apple M3 laptop CPU. As references we run BFGS on each group's active coordinates and, for the three restricted groups, the closed-form optima (centroid alignment, Procrustes, and Umeyama).

Table 2 lists the final losses. The three restricted groups reach their closed-form optima, and the conformal group recovers the teacher: noise-free, the final motor matches $M^\star$ to a relative error of

$6 \cdot 10^{-12}$, while the static chart stalls at a loss of $8.1 \cdot 10^{-4}$ and recovers the teacher coefficients only to $1.6 \cdot 10^{-3}$. With noise $\sigma = 0.05$, all methods reach the same losses, and the conformal loss sits near the expected floor of $(87 - 10)\sigma^2 \approx 0.19$. The difference between the updates is starker away from the identity: started from a rotation of 4 rad, the descent path crosses $g = -1$, a conjugate point of exp at which every chart is singular. The multiplicative update passes through and converges (to $-M^\star$, the same transformation) with loss $9 \cdot 10^{-21}$, whereas the static chart stalls at 0.41 and the corrected chart diverges (Fig. 17). Finally, the renormalization trick $g \leftarrow g(g\tilde{g})^{-1/2}$, which works in $\mathrm{Cl}(1,3)$ (see below), is not a retraction in $\mathrm{Cl}(4,1)$: there $g\tilde{g}$ has a grade-4 part, the iterate leaves the group ($|g\tilde{g} - 1| \sim 10^{14}$), and the loss stalls near 10^2.

Table 2: Final losses for the four subgroups of the conformal group of Fig. 9, fitted to one shared conformal target, noise-free (top) and with noise $\sigma = 0.05$ (bottom).

	group	dim	multiplicative (ours)	static chart	BFGS	closed form
$\sigma = 0$	$T(3)$	3	2.7797	2.7797	2.7797	2.7797
	$SE(3)$	6	1.5897	1.5897	1.5897	1.5897
	$\mathrm{Sim}(3)$	7	0.39614	0.39614	0.39614	0.39614
	conformal	10	$1.7 \cdot 10^{-21}$	$8.1 \cdot 10^{-4}$	$2.5 \cdot 10^{-21}$	–
$\sigma = 0.05$	$T(3)$	3	2.9138	2.9138	2.9138	2.9138
	$SE(3)$	6	1.6620	1.6620	1.6620	1.6620
	$\mathrm{Sim}(3)$	7	0.56218	0.56218	0.56218	0.56218
	conformal	10	0.21071	0.21268	0.21071	–

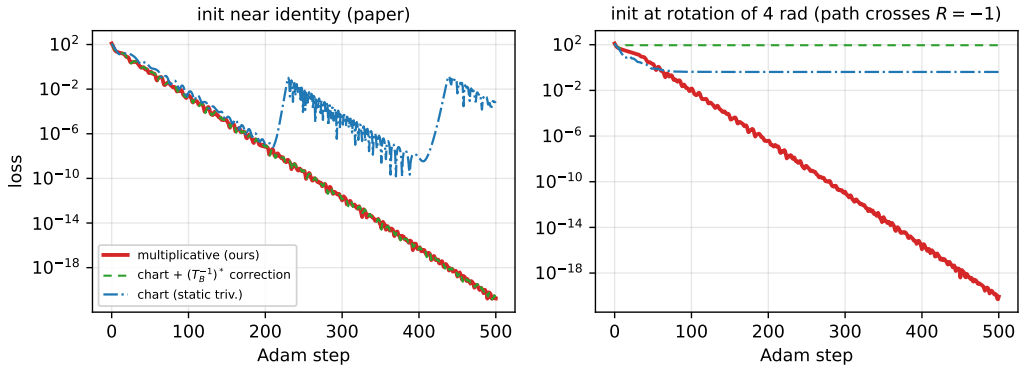


Figure 17: Training loss on the full conformal group under the multiplicative update (ours), the static chart, and the corrected chart. *Left:* from the default initialization near the identity. *Right:* from an initialization at a rotation of 4 rad, where the descent path crosses the conjugate point $g = -1$ of exp.

Lorentz frame recovery We work in $\mathrm{Cl}(1,3)$ with e_0 timelike, as in Appendix C.4, and $\mathfrak{g} = \mathfrak{spin}(1,3)$ spanned by the three boosts $e_0 \wedge e_i$ and three rotations $e_i \wedge e_j$. A static unit charge at the origin produces the Coulomb field F_k at $K = 48$ events on the shell $0.8 \le |r| \le 2$. The unknown ship frame is the rotor $R^\star$ combining a boost of generator rapidity 0.8 along $(1, 0.5, 0.2)$ with a rotation of 0.7 rad about $(0.3, -1, 0.6)$, and the ship measures $\tilde{F}_k = R^\star F_k \widetilde{R^\star} + \epsilon_k$, with Gaussian noise at 5% of the RMS signal ($\sigma = 1.83 \cdot 10^{-2}$ per bivector coefficient). We minimize $\mathcal{L}(R) = \frac{1}{6K}\sum_k \|\tilde{F}_k - RF_k\tilde{R}\|^2$, whose noise floor is σ^2. We compare the multiplicative update against the static and corrected charts, BFGS on the chart coordinates, and three parameterizations one would write without a Lorentz-group library: (i) the eight even-grade coefficients of R, renormalized after every step by $R \leftarrow R(R\tilde{R})^{-1/2}$ (in $\mathrm{Cl}(1,3)$, $R\tilde{R} = s + pI$ with $I^2 = -1$, so this is a complex inverse square root that must be re-derived for each algebra); (ii) a 4×4 matrix Λ with the penalty $\lambda\|\Lambda^\top\eta\Lambda - \eta\|_F^2$, with η the Minkowski metric and $\lambda \in \{0.1, 1, 10\}$; and (iii) the same matrix projected onto its polar factor after every step, the standard retraction of compact-manifold libraries, which projects onto $O(4)$. Every method runs 400 Adam steps from the same initialization, with the learning rate chosen from $\{0.3, 0.1, 0.03, 0.01\}$ as the smallest one within 5% of the best final

loss. We read the observer's rapidity off the recovered rotor from the image of the time axis; the ground truth is 0.7957 (speed $0.662c$), which differs from $2|B^{\star}_{\mathrm{boost}}| = 0.8$ by the Baker–Campbell–Hausdorff cross terms between the non-parallel boost and rotation. The whole comparison, including the learning-rate sweeps, takes 20 s on an Apple M3 laptop CPU.

Table 3 reports the results. Every on-group method reaches the noise floor, and on this task the three trivialized updates agree to all printed digits. The penalized matrix also reaches the floor but leaves the group (constraint violation $1.9 \cdot 10^{-3}$). The polar projection stalls at $48\sigma^2$ with violation 1.7, because $O(4)$ contains no boosts.

Table 3: Lorentz frame recovery from 48 noisy Faraday bivectors in $\mathrm{Cl}(1,3)$ (400 Adam steps; noise floor $\sigma^2 = 3.36 \cdot 10^{-4}$). Violation is $\|\Lambda^{\top}\eta\Lambda - \eta\|_F$ for the vector representation Λ of the recovered transformation; Λ error is $\min_{\pm}\|\Lambda \mp \Lambda^{\star}\|_F / \|\Lambda^{\star}\|_F$; rapidity error is against the ground truth 0.7957.

method	loss$/\sigma^2$	violation	Λ error	rapidity error
multiplicative (ours)	1.03	$7.3 \cdot 10^{-16}$	$3.9 \cdot 10^{-3}$	$1.4 \cdot 10^{-3}$
static chart	1.03	$1.0 \cdot 10^{-15}$	$3.9 \cdot 10^{-3}$	$1.4 \cdot 10^{-3}$
corrected chart	1.03	$6.1 \cdot 10^{-16}$	$3.9 \cdot 10^{-3}$	$1.4 \cdot 10^{-3}$
chart, BFGS (40 steps)	1.03	–	$3.9 \cdot 10^{-3}$	$1.4 \cdot 10^{-3}$
even multivector + renormalize	1.06	$2.0 \cdot 10^{-15}$	$1.0 \cdot 10^{-2}$	$1.1 \cdot 10^{-2}$
matrix + penalty ($\lambda = 0.1$)	1.03	$1.9 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	$1.7 \cdot 10^{-3}$
matrix + $O(4)$ polar projection	47.8	1.7	0.64	0.23