Approximate 2D-3D Shape Matching for Interactive Applications

Christoph Petzsch[1] Paul Roetzer[1] Zorah Lähner[1,2] Florian Bernard[1]
[1]University of Bonn [2]Lamarr Institute

Abstract

Matching a 2D contour to a non-rigidly deformed 3D mesh is a challenging problem due to ambiguities arising from dimensionality differences. In the past, product graph based methods were only able to either produce fast but noisy solutions, or smooth but slow solutions (the latter enabled by higher-order costs computed in the conjugate product graph). In this work, we propose an approximation of these higher-order costs so that they can be computed in the ordinary product graph. This leads to an efficient algorithm for high-quality 2D-3D shape matching and enables novel applications, like an interactive user interface which allows to refine the solution gradually. We show theoretically that our method is efficient, and we experimentally validate that the accuracy gap of our approximation to the optimum is small in practice. Our code is available.[1]

1. Introduction

The problem of shape matching, i.e. determining vertex correspondences between two shapes, has received increased attention in previous years [8, 20, 31]. If the input shapes are non-rigidly transformed, the problem becomes complicated due to many degrees of freedom and it can be shown that it is NP-hard (even for shapes with geodesic isometries) when formulated as a QAP [23]. Most research in this field has been devoted to 2D-2D [38] and 3D-3D [31] matching, while 2D-3D matching has been mostly neglected. This is despite a diverse range of potential applications, including transferring deformations from 2D contours to 3D meshes, retrieval via 2D drawings, and artistic arrangement of curves. Moreover, interaction with 2D shapes is often easier for non-expert users. With that, transferring functions or properties (imposed in 2D) from 2D to 3D would benefit interactive applications. A promising approach to tackle 2D-3D shape matching are path-based methods [26, 34] as they ensure continuity of the solution (neighbouring edges on 2D shape are matched to neighbouring edges on 3D shape) and furthermore such approaches are solvable in polynomial time. Lähner *et al*. [26] can compute correspondences efficiently, yet, they rely purely on point-wise feature. Consequently, they do not consider deformation costs, which in turn often causes unrealistic twisting in the solution, see Fig. 1. In contrast, Roetzer *et al*. [34] introduce the *conjugate product graph*, which enables the incorporation of higher-order costs and thus allows to explicitly consider deformation costs. While this leads to high-quality correspondences, it significantly increases computation time due to the much larger conjugate product graph. However, computation time is critical for interactive applications. Thus, there is a need for an efficient algorithm for high-quality 2D-3D correspondences.

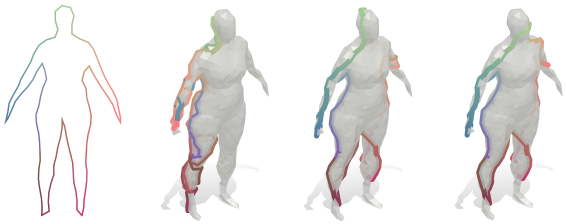

Figure 1. Visualisation of **matching a 2D contour to a 3D shape** with various methods. Colours indicate corresponding points. Previous methods were either inaccurate (Lähner *et al*.) or slow (Roetzer *et al*.), while our method produces high-quality results in reasonable runtime.

In this work we fill this gap and propose an approximation for the higher-order cost functions of [34]. This leads to results of similar quality, which, however, can be computed efficiently without the need for the larger conjugate product graph. We summarise our main contributions as follows:

- We propose the first method for 2D-3D shape matching that attains two desirable properties that have not been achieved at the same time: regularisation with higher-order costs (leading to high-quality correspondences) and an efficient algorithm that can be used in interactive applications.
- We experimentally demonstrate that in all cases our approximation leads to objective values at most 10% higher than the ones computed by Roetzer *et al*. [34].
- In particular, this enables applications that involve high-

[1] https://github.com/christophpetzsch/sm-2D3D-approx

quality landmark-based 2D-3D shape matching in interactive time, specifically with 2D shapes with up to 300 vertices and 3D shapes with up to 2100 vertices.

2. Related Work

Shape Matching Non-rigid shape matching is a widely studied problem, both in 2D [29] and 3D [11, 36], with methods being based on axiomatic [15, 17, 21, 31] and learning-based paradigms [8, 9, 16, 22, 28]. While learning-based methods are capable of predicting accurate results very fast during inference, they typically require a lot of training data and usually cannot guarantee desirable properties of the solution. In contrast, axiomatic methods can provide guarantees, e.g. per-instance optimality gaps [5, 18, 27], or geometric consistency [13, 14, 33, 35, 43]. Yet, axiomatic methods typically have high computational complexity in the 3D setting. In contrast, for lower-dimensional matching problems, there exist efficient (polynomial-time) and globally optimal algorithms, such as for matching sequences (or 1D signals) using dynamic time warping [19, 37, 44], and matching 2D contours using shortest paths [38]. These methods particularly benefit from the fact that the solution has a one-dimensional structure, so that it can be expressed as a *path* in a certain graph. Recently, a path-based approach has also been proposed for 3D-3D shape matching [32]. The setting of non-rigidly matching closed (i.e. without boundary) 2D contours to 3D shapes has also been addressed using path-based formalisms. Lähner *et al*. [26] cast 2D-3D shape matching as finding a shortest path in a product graph, which can be solved to optimality efficiently. However their approach heavily relies on spectral features, which may locally be inaccurate. The resulting matchings are often noisy (see Fig. 1). Roetzer *et al*. [34] overcome these limitations by introducing the conjugate product graph. This allows to regularise the solution space with a rigidity prior which leads to more natural and smooth matchings. Yet, it increases computational complexity significantly due to the (approx. $10\times$) larger conjugate product graph. In this work, we approximate the solution of [34], which drastically reduces computation times while still using a rigidity prior leading to high-quality matching results.

Shortest Path Approximation Dijkstra's algorithm [12] for shortest path computation in graphs is very efficient and yields globally optimal solutions in polynomial time (assuming non-negative edge-costs). However, it requires knowledge of the full graph and static edge-wise costs, which is too restrictive for some problem settings. General shortest path computations on grids [39] or meshes [30] are less efficient and many approximations have been proposed over the years [10, 24, 40, 42]. Other applications might include a dynamic graph computation with edge insertions and deletions [1], or the computation of higher-order costs that cannot be captured in fixed edge weights. In [26, 34], authors dynamically compute the graph to find shortest paths with a version of Dijkstra's algorithm. Furthermore, [34] lifts the product graph to the conjugate product graph which allows to use a rigidity prior leading to high-quality 2D-3D correspondence. Yet, this also leads to a larger graph, and thus to increased computation times. Here, the usage of the conjugate product graph is essential since the higher-order costs cannot be computed in the ordinary (i.e. not a conjugate) product graph used in [26]. To overcome this, we propose an approximation of the cost function of [34] using merely the ordinary product graph. The key idea is to consider a greedy approach that fixes predecessors when visiting vertices.

3. Background on 2D-3D Shape Matching

Below, we summarise the main ideas of 2D-3D shape matching. For an in-depth discussion see [26, 34].

Shapes as Graphs A directed graph $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$ consists of a set of vertices $\mathcal{V}_\mathcal{G}$ and a set of oriented edges $\mathcal{E}_\mathcal{G} \subseteq \{(v_1, v_2) | v_1, v_2 \in \mathcal{V}_\mathcal{G}, v_1 \neq v_2\}$ (an *oriented* edge means that $(v_1, v_2) \in \mathcal{E}_\mathcal{G}$ does not imply $(v_2, v_1) \in \mathcal{E}_\mathcal{G}$). A 2D contour $\mathcal{M}$ with m vertices can be discretised as a directed loop, i.e. a graph $\mathcal{M} = (\mathcal{V}_\mathcal{M}, \mathcal{E}_\mathcal{M})$ satisfying

$$\mathcal{V}_\mathcal{M} = \{0, \dots, m-1\}, \tag{1}$$

$$\mathcal{E}_\mathcal{M} = \big\{(k, k+1) | k \in \{0, ..., m-2\}\big\} \cup \big\{(m-1, 0)\big\}. \tag{2}$$

Similarly, a 3D triangle mesh $\mathcal{N}$ can be formalised as a directed graph $\mathcal{N} = (\mathcal{V}_\mathcal{N}, \mathcal{E}_\mathcal{N})$. To this end, we convert each undirected edge (i.e. the edges of the triangles) to two opposite-oriented edges. Consequently, for every directed edge $(v_1, v_2) \in \mathcal{E}_\mathcal{N}$, its opposite-oriented edge $(v_2, v_1) \in \mathcal{E}_\mathcal{N}$ is also in $\mathcal{E}_\mathcal{N}$.

Product and Conjugate Graphs We denote by $\mathcal{E}^+ = \mathcal{E} \cup \{(v, v) | v \in \mathcal{V}\}$ the *extended* edge set, in which self-edges (v, v) (i.e. edges from and to the same vertex) are called *degenerate*. For a contour $(\mathcal{V}_\mathcal{M}, \mathcal{E}_\mathcal{M})$ and a 3D mesh $(\mathcal{V}_\mathcal{N}, \mathcal{E}_\mathcal{N})$, we define the product graph $\mathcal{P} = (\mathcal{V}_\mathcal{P}, \mathcal{E}_\mathcal{P})$ via

$$\mathcal{V}_\mathcal{P} = \mathcal{V}_\mathcal{M} \times \mathcal{V}_\mathcal{N}, \tag{3}$$

$$\mathcal{E}_\mathcal{P} = \{(e_1^\mathcal{M}, e_2^\mathcal{N}) \mid e_1^\mathcal{M} \in \mathcal{E}_\mathcal{M}^+,\ e_2^\mathcal{N} \in \mathcal{E}_\mathcal{N}^+, \\ e_1^\mathcal{M} \text{ or } e_2^\mathcal{N} \text{ non-degenerate}\}. \tag{4}$$

We use the term *layer* to refer to the set of vertices $\mathcal{V}_k := k \times \mathcal{V}_\mathcal{N}$ (i.e. all product vertices referring to the same contour vertex k).

Finally, for a directed graph $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$, we define the *conjugate* graph (also called the line graph) $\mathcal{G}^*$ by

$$\mathcal{V}^*_\mathcal{G} = \mathcal{E}_\mathcal{G}, \tag{5}$$
$$\mathcal{E}^*_\mathcal{G} = \{((v_1, v_2), (v_2, v_3)) \mid (v_1, v_2), (v_2, v_3) \in \mathcal{V}^*_\mathcal{G}\}. \tag{6}$$

This turns edges into vertices, connecting them if they share a vertex between them. We refer the reader to [34] for visualisations of these concepts.

2D-3D Shape Matching as Shortest Cyclic Paths 2D-3D shape matching can be cast as finding a cyclic shortest path in the product graph $\mathcal{P}$. To this end, we make $\mathcal{P}$ acyclic by duplicating $\mathcal{V}_0$ (so that we obtain layer $\mathcal{V}_m$ and we connect layer $\mathcal{V}_{m-1}$ to $\mathcal{V}_m$ rather than $\mathcal{V}_0$) and search for a shortest path from layer $\mathcal{V}_0$ to layer $\mathcal{V}_m$ which starts and ends on the same 3D-vertex, see also [26, 34]. Formalising 2D-3D shape matchings this way ensures continuity, i.e. neighbouring 2D vertices get matched to neighbouring 3D vertices. The so-called degenerated edges allow for 'vertex-to-edge' and 'edge-to-vertex' matchings, which are necessary to account for potential differences in discretisation and for non-isometric deformations.

The cost of a matching (i.e. corresponding to a given cyclic path in the product graph) can be measured by summing up the costs of the edges along the path, i.e. by using a function $c : \mathcal{E}_\mathcal{G} \to [0, \infty)$. The shortest (cyclic) path within the product graph can be computed by using variants of Dijkstra's algorithm [12]. In particular, this can be efficiently done, as the structure of the product graph allows to use dynamic programming: every layer $\mathcal{V}_k$ only has outgoing edges into layers $\mathcal{V}_k$ and $\mathcal{V}_{k+1}$. Consequently, we can iterate through the layers from first to last and we only need to maintain priority heaps for each individual layer (for general graphs, we would need to maintain priority heaps for the whole graph to ensure optimality of the path). To enforce consistency between the first and last path point in $\mathcal{V}_0$ and $\mathcal{V}_m$, we apply the branch-and-bound method of [2] as proposed in [26].

Cost Function The simplest choice of cost functions compares point-wise features with each other. However, due to the difference in dimension between 2D and 3D shapes most common features for 2D-2D or 3D-3D shape matching are not applicable. Lähner *et al*. [26] use adapted spectral features, specifically Heat Kernel Signatures [41] and Wave Kernel Signatures [3], but results show that these are locally noisy. While this allows to compute matchings efficiently, the result often twists on the 3D shape (see e.g. Fig. 1), mainly because higher-order directional information cannot be taken into account. These issues have been addressed by Roetzer *et al*. [34] with the following cost terms:

- A data term based on the difference of local thickness of 2D and 3D shape respectively, i.e. for a product vertex $v = (i, j)$ they define
$$d_{\text{data}}((i, j)) := \psi_1(|l_i^{2D} - l_j^{3D}|) \tag{7}$$
where l_i^{2D} is the thickness of the 2D mesh at vertex i, and similarly for l_j^{3D}. Here ψ_1 is the robust loss function from Barron [4].
- A regularisation term based on the differences between deformations of subsequent edges on 2D shape and on 3D shape respectively. Specifically, for product edges (v_1, v_2) and (v_2, v_3), they compute rotations $R_{(v_1,v_2)}$ and $R_{(v_2,v_3)}$ that transform the coordinate frame of the 2D edge into the coordinate frame of the 3D edge. Using the geodesic distance of the rotations, represented using unit quaternions $q_{(v_1,v_2)}$ and $q_{(v_2,v_3)}$, this yields the regularisation term
$$d_{\text{reg}}(v_1, v_2, v_3) := \psi_2(\arccos(\langle q_{(v_1,v_2)}, q_{(v_2,v_3)} \rangle)), \tag{8}$$
where again ψ_2 is the loss function from Barron [4] but with different parameters.

Using Dijkstra's algorithm, we cannot use these cost functions to compute globally optimal paths (in the ordinary product graph), as we need cost functions depending only on edges, i.e. pairs of vertices, as opposed to using pairs of edges as in (8). Thus, in [34], Dijkstra is instead run on the conjugate product graph with costs assigned to conjugate edges, which have a scope of *two product edges*. This yields significant improvements in matching quality, at the cost of an increased computation time due to the (approx. $10\times$) larger conjugate product graph. In the end, the cost for a conjugate product vertex (v_j, v_i) can be written as

$$E[(v_j, v_i)] = \min_{k^*} E[(v_{k^*}, v_j)] + d_{\text{data}}(v_i) + d_{\text{reg}}(v_{k^*}, v_j, v_i), \tag{9}$$

where $E[(v_j, v_i)]$ denotes the total cost of the path going from the bottom layer $(\mathcal{V}_0, \mathcal{V}_1)$ to (v_j, v_i), which can be optimised using Dijkstra and branch-and-bound in the conjugate product graph.

4. Fast Approximate with Rigidity Prior

A major disadvantage of running Dijkstra on the ordinary product graph is that we can take no higher-order information into account. In contrast, in the conjugate product graph, we can take higher-order information into account. Yet, this comes at the cost of having a much larger graph which in turn leads to higher computation times. We propose to use a *blended graph*, a mix of the ordinary and conjugate product graph. This allows to approximate higher-order costs without significantly increasing the product graph size and consequently not significantly increasing computation times.

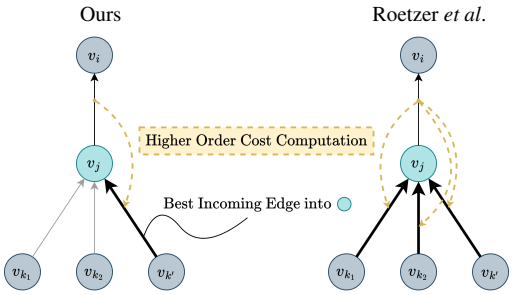

Figure 2. Illustration of **higher-order cost computation** as done by us (left) and done by Roetzer *et al.* [34] (right). Dashed yellow lines indicate combinations that are considered in respective methods. Edges that are ignored as part of our approximation are greyed out. In short, our approximation does not consider all possible combinations which might lead to an overall higher energy but rarely has a big impact on matching quality in practice.

Higher-Order Approximation Our cost function is an approximation of the higher-order cost function of Roetzer *et al.* [34] but can be computed on the ordinary product graph. The higher-order costs depend on two subsequent edges. Thus, we remember the edge used to enter a vertex and use this edge in the computation of the cost of any succeeding vertices. To this end, we store the predecessor $v_{k'} = Pr(v_j)$ of a vertex v_j before computing the cost of vertices that come after v_j. Then, the cost of v_i, when entered from v_j, can be computed as follows

$$E[v_i] = E[v_j] + d_{\text{data}}(v_i) + d_{\text{reg}}(v_{k'}, v_j, v_i). \tag{10}$$

Here the predecessor $v_{k'}$ of v_j satisfies

$$k' := \arg\min_k E[v_k] + d_{\text{data}}(v_j) + d_{\text{reg}}(v_{Pr(v_k)}, v_k, v_j) \tag{11}$$

where v_k are the potential predecessors of v_j. In other words, we choose $v_{k'}$ as the locally optimal predecessor of v_j by considering the previous path. Yet, we do not consider the effect the choice of $v_{k'}$ has on the *subsequent* path (since we cannot know the subsequent path at this point, cf. also to Eq. (9) where v_i is considered when determining k^*). This leads to the approximation and loss of guarantee of global optimality of the computed path w.r.t. the rigidity regularisation. We illustrate the cost computation in Fig. 2.

Blended graph Computing the (approximated) cost of vertices requires knowledge about which vertices are used to enter their predecessors. Thus, computing the cost of vertices on layer $\mathcal{V}_1$ requires knowing the predecessors of the vertices on layer $\mathcal{V}_0$ (which are on $\mathcal{V}_{m-1}$). Yet, these are only known at the end of each Dijkstra call. This means our approximation is not applicable here. Instead, we use conjugate vertices for the first layers. Thus, the first two layers of our graph are $(\mathcal{V}_0, \mathcal{V}_1)$ and $(\mathcal{V}_1, \mathcal{V}_2)$. Conjugate edges are then used to enter $(\mathcal{V}_1, \mathcal{V}_2)$ from $(\mathcal{V}_0, \mathcal{V}_1)$. We use dummy edges to enter $\mathcal{V}_2$ from $(\mathcal{V}_1, \mathcal{V}_2)$ (i.e. we connect any conjugate product vertex (v_i, v_j) on layer $(\mathcal{V}_1, \mathcal{V}_2)$ with a (ordinary) product vertex v_k on layer $\mathcal{V}_2$ if $v_j = v_k$). For dummy edges, we do not compute any costs but instead copy respective costs from $(\mathcal{V}_1, \mathcal{V}_2)$. The final ordinary product graph layer is $\mathcal{V}_m$ (the copy of $\mathcal{V}_0$), and we use the edges entering $\mathcal{V}_m$ to compute the costs of $(\mathcal{V}_m, \mathcal{V}_{m+1})$ ($\mathcal{V}_{m+1}$ being the copy of $\mathcal{V}_1$). In summary, the resulting *blended* graph consists of two conjugate product graph layers at the bottom, one conjugate product graph layer at the top, and in-between we use ordinary product graph layers, see also Fig. 3. Consequently, our branch-and-bound method runs on the conjugate layer $(\mathcal{V}_0, \mathcal{V}_1)$ (unlike Lähner *et al.* [26] which runs on the ordinary product graph layer $\mathcal{V}_0$). We discuss differences in size of respective graphs in Sec. 5.

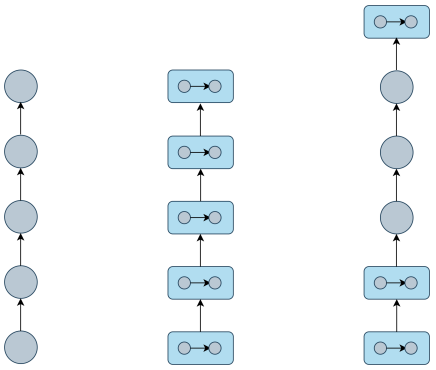

Figure 3. From left to right: **schematic illustration** of the graphs used by Lähner *et al.* [26], by Roetzer *et al.* [34] and us. Gray vertices indicate ordinary product layers and blue nodes indicate conjugate product layers. Our blended graph only uses conjugate layers for the first layers (where our approximation is not possible) and for the last layers (to enforce consistency).

5. Theoretical Analysis

In this section we theoretically analyse the computational complexity of the algorithms of Lähner *et al.* [26], Roetzer *et al.* [34] and ours based on the number of vertices and edges of each layer.

For any graph $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$, the runtime of Dijkstra is dominated by the delete- and update-operations of the minheap (we use a binary heap in our implementation). The former happens at most once per vertex and the latter at most once per edge. Both operations require $O(\ln(|\mathcal{V}_\mathcal{G}|))$ pairwise comparisons of costs of minheap elements. We denote with α and β the smallest integers such that deletions require at most $\alpha \ln(|\mathcal{V}_\mathcal{G}|)$ and updates at most $\beta \ln(|\mathcal{V}_\mathcal{G}|)$ pairwise comparisons. Thus, the total number of pairwise comparisons for Dijkstra is

$$(\alpha|\mathcal{V}_\mathcal{G}| + \beta|\mathcal{E}_\mathcal{G}|) \ln(|\mathcal{V}_\mathcal{G}|). \tag{12}$$

	[26]	[34]	Ours
Number of vertices on first layer	n	$13n$	$13n$
Number of vertices on middle layers	n	$13n$	n
Number of edges on middle layers	$13n$	$143n$	$13n$
Number of layers	m	m	m
Worst-case complexity of single Dijkstra call	$(\alpha + 13\beta)$ $mn\ln(n)$	$13(\alpha + 11\beta)$ $mn\ln(n)$	$(\alpha + 13\beta)$ $mn\ln(n)$
Worst-case number of Dijkstra calls	n	$13n$	$13n$
Worst-case total complexity	$(\alpha + 13\beta)$ $mn^2\ln(n)$	$169(\alpha + 11\beta)$ $mn^2\ln(n)$	$13(\alpha + 13\beta)$ $mn^2\ln(n)$

Table 1. Worst-case **computational complexity** of different shape matching methods. Here, $n := |\mathcal{V}_\mathcal{N}|$ is the number of 3D vertices, $m := |\mathcal{V}_\mathcal{M}|$ is the number of 2D vertices and α, β denote constants which dependent on the implementation of the minheap (i.e. the complexity of deleting the smallest element and updating the cost of an element). For easier exposure, we approximate the worst-case complexity of a single Dijkstra call of ours with $(\alpha+13\beta)mn\ln(n)$ (instead of $(\alpha+13\beta)(m-2)n\ln(n)+13(\alpha+11\beta)3n\ln(n)$).

Since we have a minheap for each layer, we need to determine the number of vertices and edges per layer. We note that a layer can be a ordinary product graph layer or a conjugated product graph layer. Following the argumentation in [34], we assume that each vertex of a 3D shape $\mathcal{N}$ has 6 neighbours on average. From that, we conclude that $|\mathcal{E}_\mathcal{N}| \approx 6 \cdot |\mathcal{V}_\mathcal{N}|$ (we note that converting undirected edges into directed edges doubles the number of edges). Since the product graph contains two product edges per 3D edge (one degenerate, one non-degenerate) as well as one product edge per 3D vertex (resembling a self-edge of said 3D vertex), we have 13 times more product edges than product vertices. On the conjugate layers, every vertex is connected to 6 degenerate and 5 non-degenerate (conjugate) vertices. Thus, we have 11 times more edges than vertices.

Combining all this information, we report the total computational complexity in Tab. 1. We note that this also includes the worst-case number of iterations of the branch-and-bound method. Above all, branch-and-bound requires at most as many iterations as there are vertices on the first layer. Furthermore, for easier exposure, we neglect the 3 conjugate layers in our *blended graph* for worst-case computational complexity analysis of a single Dijkstra call. In fact, for large m (i.e. the number of contour vertices), the computational complexity of a single Dijkstra call of ours is dominated by the $m - 2$ ordinary product graph layers.

6. Experiments

In the following, we evaluate our method's performance w.r.t. matching quality and runtime in comparison to previous works (Sec. 6.2, Sec. 6.3). We also test the robustness of our method (Sec. 6.4). Finally, we show that runtime improvements enable applications like interactive 2D-3D shape matching and 3D shape retrieval (Sec. 6.5).

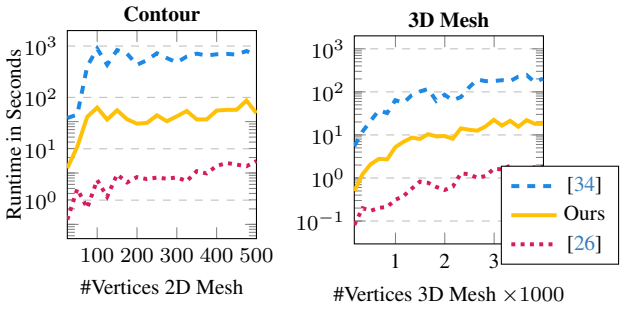

Figure 4. We show the **mean runtime** in seconds using a log scale for the approaches by Lähner *et al*. [26], Roetzer *et al*. [34] and our approach. On the left, we fix 3D mesh resolution and alter 2D mesh resolution. On the right,we fix 2D mesh resolution and alter 3D mesh resolution. Lähner *et al*. [26] solves a smaller problem and thus is the fastest (while it cannot incorporate higher-order costs). In comparison to the other higher-order-cost-approach by Roetzer *et al*. [34], ours is significantly faster.

6.1. Experimental Setup

All experiments are run on an Intel Core i9-12900K with 64GB of DDR5 RAM.

Datasets We use the 2D-3D matching datasets proposed in [26] based on FAUST [6] and TOSCA [7] 3D shape matching datasets. FAUST consists of 10 classes of humans with different body shapes with 6890 vertices each. All classes have their shapes in 10 different poses, resulting in 100 deformed 3D shapes. Furthermore, for each class the dataset provides a 2D contour of the null pose with roughly 900 vertices with point-wise ground-truth correspondence to respective 3D shapes. In addition, 2D and 3D shapes have consistent part-segmentation.

TOSCA contains 80 human and animal shapes with up to $12k$ vertices in 9 different classes. Each class consists of 3D shapes in different poses and comes with at least one 2D contour with roughly 900 vertices. TOSCA does not contain point-wise ground-truth correspondences from 2D to 3D shapes but contains consistent part-segmentation.

Competing Methods We compare against Roetzer *et al*. [34] and Lähner *et al*. [26], which are the only two methods tackling the same non-rigid path-based 2D-3D setting. Roetzer *et al*. uses the conjugate product graph which gives qualitatively good but slow results. Lähner *et al*. operates on the ordinary product graph which is efficient but leads

to discretisation noise and does not allow to employ higher-order costs. Similarly as in [34], we run Lähner *et al*. without segmentation features as this would lead to an unfair advantage (segmentation features are essentially a coarse pre-matching). See Sec. 3 for an explanation of the differences between methods.

Metrics We use the following metrics to evaluate the performance of the different methods. We note that in contrast to 3D-3D shape matching, the different dimensionality and challenging non-bijectivity leads to ambiguity in the solutions and therefore there does not exists rigorous ground-truth correspondence. While we evaluate the commonly used matching errors, we additionally use segmentation errors and encourage the reader to judge matching performance based upon qualitative examples.

Segmentation errors are defined as the normalised geodesic distance from the correct segment, i.e.

$$\varepsilon_{\text{seg}}(x,y) = \min_{y' \in \mathcal{N}} \frac{\text{dist}_{\mathcal{N}}(y,y')}{\text{diam}(\mathcal{N})} \text{ s.t. } \sigma_{\mathcal{N}}(y') = \sigma_{\mathcal{M}}(x). \tag{13}$$

where $\text{dist}_{\mathcal{N}}$ denotes the geodesic distance on $\mathcal{N}$ and $\sigma_\bullet$ denotes the segmentation labels.

The normalised **matching errors** are defined as

$$\varepsilon_{\text{geo}}(x,y) = \frac{\text{dist}_{\mathcal{N}}(y,\hat{y})}{\text{diam}(\mathcal{N})}. \tag{14}$$

Here, $\hat{y}$ is the ground-truth corresponding point. For both metric, $\text{diam}(\cdot)$ describes the diameter of a shape.

For both error metrics, we visualise cumulative error-curves which plot the percentage of points that have an error lower than a certain threshold over respective error thresholds. The **area under the curve (AUC)** is the integral of said cumulative segmentation or matching error curves (larger AUC values are better).

6.2. Runtime

We evaluate the runtime of the different approaches for 2D-3D shape matching on FAUST for varying 2D and 3D mesh resolutions, see Fig. 4. Our method is significantly faster than [34] while producing similar qualitative result, see Fig. 8. We also show the ratio of the runtime and approximation factor between Roetzer *et al*. and our approach, see Fig. 5 for different 2D and 3D mesh sizes. While we decrease the runtime with up to a factor of 20, the approximation stays within 10% and is often lower. The high variation in the runtime stems from the varying branches of the branch-and-bound approach.

6.3. Shape Matching Performance

We quantitatively compare our results using the ground-truth correspondences and segmentation correspondences

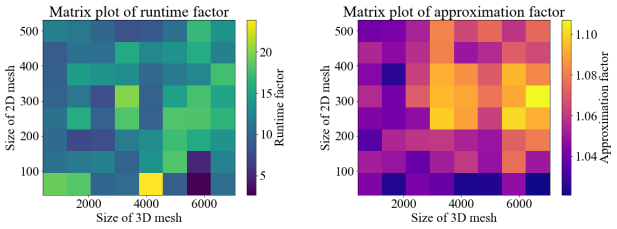

Figure 5. **(Left)** The mean ratio of the runtime of Roetzer *et al*. [34] relative to the runtime of our algorithm. As can be seen, our speedup is roughly an order of magnitude in most cases. **(Right)** The mean ratio of the cost of the matching computed by our algorithm relative to the cost of the matching computed by Roetzer *et al*. Despite no known theoretically derived maximum approximation factor, the ratio rarely exceeds 110%.

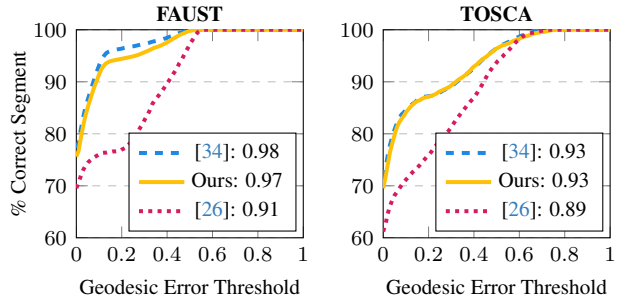

Figure 6. Cumulative **segmentation error plots** on FAUST (left) and TOSCA (right). Numbers in legends are AUCs. Our method performs similar to [34] and is significantly better than [26].

given for FAUST and TOSCA. The segmentation errors on FAUST and TOSCA are reported in Fig. 6 and show that our results are on-par with the method [34]. The approximation factor of our method versus [34] is plotted in Fig. 5. While our computed matchings costs are up to 10% higher, this does not affect matching quality as badly which can be seen from roughly equal segmentation errors.

In Fig. 7, we show matching errors on FAUST for all methods. We emphasise that possible ground-truth correspondences are given for this dataset, but there are lots of ambiguities in the 2D-3D shape matching setting such that there is no single correct solution. The results show that again our methods performs on-par with [34] and considerably better than [26]. Also, when left-right swapped solutions are removed from the error-curves, our methods still performs similarly well compared to [34], see, Fig. 7 (right).

Finally, we show qualitative results of all methods in Fig. 8. This shows that our method also visually produces matchings of similar quality compared to [34].

6.4. Robustness Evaluation

We test the robustness of our method w.r.t. Gaussian noise and w.r.t. varying discretisations.

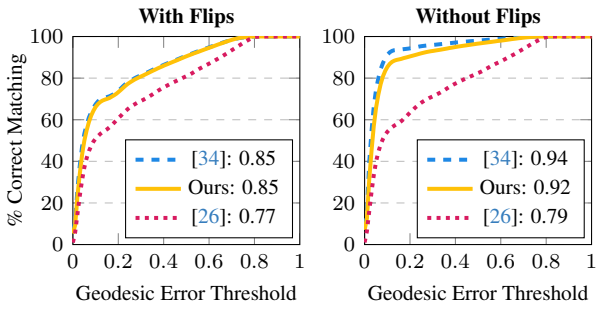

Figure 7. Cumulative **matching error plots** on FAUST including left-right-flips (left) and when removing left-right-flips (right).

Gaussian Noise We displace the vertices of the 3D shapes by three-dimensional Gaussian random vectors. We show the effect for different variances in Fig. 9 for our method and Roetzer *et al*. [34], since we are especially interested how our approximation of higher-order costs is affected. For low levels of noise, the methods perform similarly, but unlike [34] our method is not robust against large amounts of noise very likely due to the approximate nature of ours.

Varying Discretization Furthermore, we test the robustness of our method against varying discretisation of the contour. Results can be seen in Fig. 9 and show that the effect of non-uniform discretisation on our method is minimal emphasised by equal AUCs (for uniform sampling it is 0.97 and for non-uniform sampling it is 0.95).

6.5. Applications

We show that our significant runtime improvements enable (previously impossible) applications of 2D-3D shape matching.

Interactive Contour Matching Even though our method returns good correspondences in many cases without landmarks, the 2D-3D settings includes many ambiguities which might lead to undesirable solutions. We implemented a GUI in which the solution will be improved on-the-fly after the user dynamically adds landmark correspondences. By fixing one or multiple landmarks, the layers in the product graph containing these landmark correspondence can be reduced to a single vertex. This allows to completely remove the branch-and-bound strategy and with that further reduces computation times significantly. In turn, this enables working on high-resolution meshes at interactive speeds (e.g. a contour with 300 vertices and a 3D mesh with 2100 vertices can be matched in around $0.9s$). The GUI with selected landmark vertices can be seen in Fig. 10. We show further results in the supplementary, see Sec. 6.5.

	Lähner *et al*. [26]	Ours
Cat	0.3925	**0.6700**
Centaur	0.6438	**1.0000**
Human	0.5497	**0.7851**
Dog	**0.8084**	0.5587
Horse	0.9415	**1.0000**
Wolf	0.2436	**1.0000**

Table 2. Average Precision of our approach and the approach of Lähner *et al*. [26] for **shape retrieval**. We outperform Lähner *et al*. in all categories except for category dog.

Non-Rigid 3D Shape Retrieval For this task, we query a set of 3D shapes with a 2D contour. After computing the matching between the 2D contour and *all* 3D shapes we return the best-matching 3D shapes according to the matching cost. This is especially useful since drawing the rough shape of the 3D target in 2D is much easier for users than modelling a rough 3D shape. Furthermore, this allows users to explore a 3D shape collection without any semantic labels easily. We note that we only compare our method to Lähner *et al*. and not to Roetzer *et al*. since computation times for Roetzer *et al*. are not feasible for this task (a single query would already take days). The retrieval ranking is computed by comparing the matching energies (the lower the energy the better the ranking score). To obtain comparable energies for different classes and resolutions, we normalise the computed energy by using the length of the computed path in the five-dimensional product graph. This yields a useful measure for shape similarity. The results of our approach and that of [26] are shown in Tab. 2. We outperform [26] for all classes except for class dog. We note that retrieval results for [26] are worse than reported in [26] because the segmentation feature is not used and the shapes were downsampled. We provide further retrieval examples in the supplementary, see Fig. A.2.

7. Discussion and future work

Learned Features The thickness- and rigidity-based features of Roetzer *et al*. [34] and our algorithm yield high-quality matchings. Yet, we believe that learned features could allow for significant improvements, especially because local rigidity requires both shapes to be scaled consistently (which cannot be assumed in general).

Approximate Branch-and-Bound In this paper, we decreased the computation time of a single Dijkstra call. Yet, through the branch-and-bound strategy, multiple Dijkstra calls are necessary to determine a cyclic solution. To further improve computation times, one could adapt the approximate branch-and-bound method introduced in [25] to decrease the number of Dijkstra calls and thus further decrease computation times.

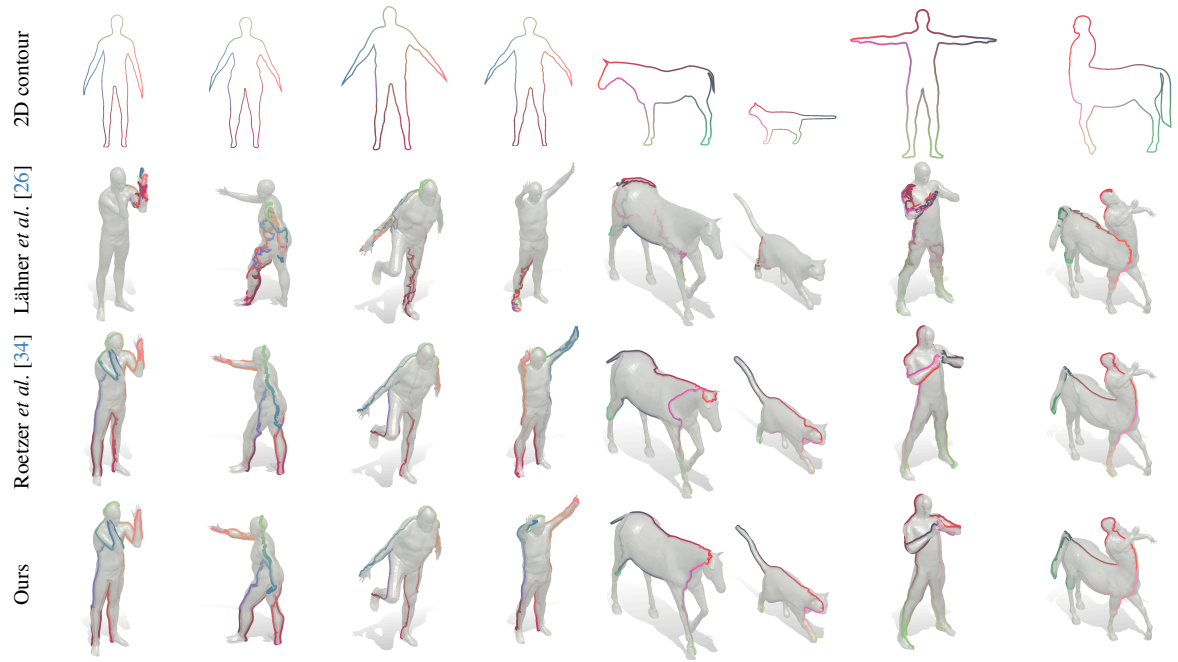

Figure 8. **Qualitative results** of Lähner *et al*. [26], Roetzer *et al*. [34] and our method on FAUST (first four columns) and TOSCA (last four columns). Our results are on-par with [34]. Yet, we can compute results much faster compared to [34].

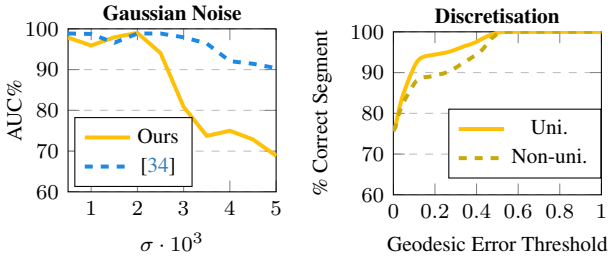

Figure 9. **(Left)** The effects of Gaussian noise on the AUC by our algorithm and by Roetzer *et al*. [34]. The fact that we only compute an approximation means that we lose a lot of the robustness to noise compared to [34]. **(Right)** The effect of a uniform vs. non-uniform discretisation. We can see that our algorithm performs better with uniform discretisation but is robust overall.

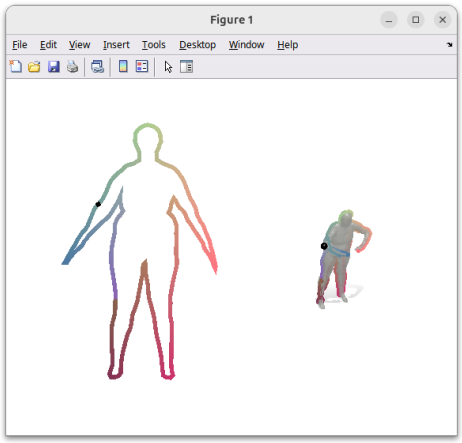

Figure 10. Our Matlab-based GUI for **interactive landmark matching**. The user can click on vertices on the contour (left) and on the 3D mesh (right) to match the former to the latter. Landmark vertices are used to improve the matching (within under a second).

8. Conclusion

We have introduced an approximation method for higher-order cost computation which allows to efficiently compute high-quality 2D-3D shape correspondences. This higher-order cost computation was previously only possible when using the larger conjugate product graph and thus was inefficient. In contrast, algorithms running on the smaller, ordinary product graph would lead to worse but more efficient results. Our algorithm combines efficiency and high-quality by approximating higher-order regularisation costs. We achieve this, by fixing edge-predecessors preliminary. While this can lead to sub-optimal results, we empirically show that sacrificing global optimality has nearly no effect on matching quality in practice. In addition, this leads to a significant decreases in computation time which enables applications like interactive landmark matching and shape retrieval.

Acknowledgments We thank Tobias Weißberg for valuable feedback on the manuscript.

References

[1] J. an den Brand and D. Nanongkai. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. *Annual Symposium on Foundations of Computer Science*, 2019. 2

[2] Ben Appleton and Changming Sun. Circular shortest paths by branch and bound. *Pattern Recognition*, 36(11):2513–2520, 2003. 3

[3] Mathieu Aubry, Ulrich Schlickewei, and Daniel Cremers. Pose-Consistent 3D Shape Segmentation Based on a Quantum Mechanical Feature Descriptor. In *Pattern Recognition*. Springer Berlin Heidelberg, 2011. 3

[4] Jonathan T. Barron. A general and adaptive robust loss function. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 3

[5] Florian Bernard, Zeeshan Khan Suri, and Christian Theobalt. Mina: Convex mixed-integer programming for non-rigid shape alignment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13826–13835, 2020. 2

[6] Federica Bogo, Javier Romero, Matthew Loper, and Michael J. Black. Faust: Dataset and evaluation for 3d mesh registration. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3794–3801, 2014. 5

[7] Alexander Bronstein, Michael Bronstein, and Ron Kimmel. *Numerical Geometry of Non-Rigid Shapes*. Springer Publishing Company, Incorporated, 1 edition, 2008. 5

[8] Dongliang Cao, Paul Roetzer, and Florian Bernard. Unsupervised learning of robust spectral shape matching. *ACM Transactions on Graphics (TOG)*, 2023. 1, 2

[9] Dongliang Cao, Marvin Eisenberger, Nafie El Amrani, Daniel Cremers, and Florian Bernard. Spectral meets spatial: Harmonising 3d shape matching and interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3658–3668, 2024. 2

[10] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. The heat method for distance computation. *Commun. ACM*, 60(11):90–99, 2017. 2

[11] Bailin Deng, Yuxin Yao, Roberto M Dyke, and Juyong Zhang. A survey of non-rigid 3d registration. In *Computer Graphics Forum*, pages 559–589. Wiley Online Library, 2022. 2

[12] E.W. Dijkstra. A note on two problems in connexion with graphs. In *Numerische Mathematik*, 1959. 2, 3

[13] Viktoria Ehm, Paul Roetzer, Marvin Eisenberger, Maolin Gao, Florian Bernard, and Daniel Cremers. Geometrically consistent partial shape matching. In *3DV*, 2024. 2

[14] Viktoria Ehm, Paul Roetzer, Marvin Eisenberger, Maolin Gao, Florian Bernard, and Daniel Cremers. Geometrically consistent partial shape matching. In *2024 International Conference on 3D Vision (3DV)*, pages 914–922. IEEE Computer Society, 2024. 2

[15] Marvin Eisenberger, Zorah Lahner, and Daniel Cremers. Smooth Shells: Multi-Scale Shape Registration With Functional Maps. In *CVPR*, 2020. 2

[16] Marvin Eisenberger, Aysim Toker, Laura Leal-Taixé, and Daniel Cremers. Deep shells: Unsupervised shape correspondence with optimal transport. *Advances in Neural information processing systems*, 33:10491–10502, 2020. 2

[17] Danielle Ezuz, Behrend Heeren, Omri Azencot, Martin Rumpf, and Mirela Ben-Chen. Elastic correspondence between triangle meshes. In *Computer Graphics Forum*, pages 121–134. Wiley Online Library, 2019. 2

[18] Maolin Gao, Paul Roetzer, Marvin Eisenberger, Zorah Lähner, Michael Moeller, Daniel Cremers, and Florian Bernard. SIGMA: Quantum scale-invariant global sparse shape matching. In *International Conference on Computer Vision (ICCV)*, 2023. 2

[19] Omer Gold and Micha Sharir. Dynamic time warping and geometric edit distance: Breaking the quadratic barrier. *ACM Trans. Algorithms*, 14(4), 2018. 2

[20] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. 3d-coded : 3d correspondences by deep deformation. In *ECCV*, 2018. 1

[21] Florine Hartwig, Josua Sassen, Omri Azencot, Martin Rumpf, and Mirela Ben-Chen. An elastic basis for spectral shape correspondence. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–11, 2023. 2

[22] Puhua Jiang, Mingze Sun, and Ruqi Huang. Non-rigid shape registration via deep functional maps prior. In *Advances in Neural Information Processing Systems*, pages 58409–58427. Curran Associates, Inc., 2023. 2

[23] Itay Kezurer, Shahar Z Kovalsky, Ronen Basri, and Yaron Lipman. Tight relaxation of quadratic matching. In *Computer Graphics Forum (CGF)*, 2015. 1

[24] R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds. *Proceedings of the National Academy of Sciences*, 95:8431–8435, 1998. 2

[25] Zorah Lähner. *Continuous Correspondence of Non-Rigid 3D Shapes*. PhD thesis, Technical University of Munich (TUM), 2021. 7

[26] Zorah Lahner, Emanuele Rodola, Frank R Schmidt, Michael M Bronstein, and Daniel Cremers. Efficient globally optimal 2d-to-3d deformable shape matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2185–2193, 2016. 1, 2, 3, 4, 5, 6, 7, 8, 11, 12

[27] Haggai Maron, Nadav Dym, Itay Kezurer, Shahar Kovalsky, and Yaron Lipman. Point registration via efficient convex relaxation. *ACM Transactions on Graphics (TOG)*, 35(4): 73, 2016. 2

[28] Simone Melzi, Jing Ren, Emanuele Rodolà, Abhishek Sharma, Peter Wonka, and Maks Ovsjanikov. Zoomout: spectral upsampling for efficient shape correspondence. *ACM Transactions on Graphics (TOG)*, 38(6):1–14, 2019. 2

[29] Damien Michel, Iasonas Oikonomidis, and Antonis Argyros. Scale invariant and deformation tolerant partial shape matching. *Image and Vision Computing*, 29(7):459–469, 2011. 2

[30] Joseph S. B. Mitchell, David M. Mount, and Christos H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16:647–668, 1987. 2

[31] Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. Functional maps: a flexible

representation of maps between shapes. *ACM Transactions on Graphics (ToG)*, 31(4):1–11, 2012. 1, 2

[32] Paul Roetzer and Florian Bernard. Spidermatch: 3d shape matching with global optimality and geometric consistency. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. 2

[33] Paul Roetzer, Paul Swoboda, Daniel Cremers, and Florian Bernard. A scalable combinatorial solver for elastic geometrically consistent 3d shape matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 428–438, 2022. 2

[34] Paul Roetzer, Zorah Lähner, and Florian Bernard. Conjugate product graphs for globally optimal 2d-3d shape matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21866–21875, 2023. 1, 2, 3, 4, 5, 6, 7, 8, 11, 12

[35] Paul Roetzer, Ahmed Abbas, Dongliang Cao, Florian Bernard, and Paul Swoboda. Discomatch: Fast discrete optimisation for geometrically consistent 3d shape matching. In *In Proceedings of the European conference on computer vision (ECCV)*, 2024. 2

[36] Yusuf Sahillioğlu. Recent advances in shape correspondence. *Vis. Comput.*, 36(8):1705–1721, 2020. 2

[37] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 1978. 2

[38] Frank R. Schmidt, Eno Toeppe, and Daniel Cremers. Efficient planar graph cuts with applications in computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Miami, Florida, 2009. 1, 2

[39] J A Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996. 2

[40] Nicholas Sharp and Keenan Crane. You can find geodesic paths in triangle meshes by just flipping edges. *ACM Trans. Graph.*, 39(6), 2020. 2

[41] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. A concise and provably informative multi-scale signature based on heat diffusion. In *SGP*, 2009. 3

[42] Vitaly Surazhsky, Tatiana Surazhsky, Danil Kirsanov, Steven J. Gortler, and Hugues Hoppe. Fast exact and approximate geodesics on meshes. *ACM Trans. Graph.*, 24(3): 553–560, 2005. 2

[43] Thomas Windheuser, Ulrich Schlickewei, Frank R Schmidt, and Daniel Cremers. Geometrically consistent elastic matching of 3d shapes: A linear programming solution. In *2011 International Conference on Computer Vision*, pages 2134–2141. IEEE, 2011. 2

[44] Jiaping Zhao and Laurent Itti. shapedtw: Shape dynamic time warping. *Pattern Recognition*, 74:171–184, 2018. 2