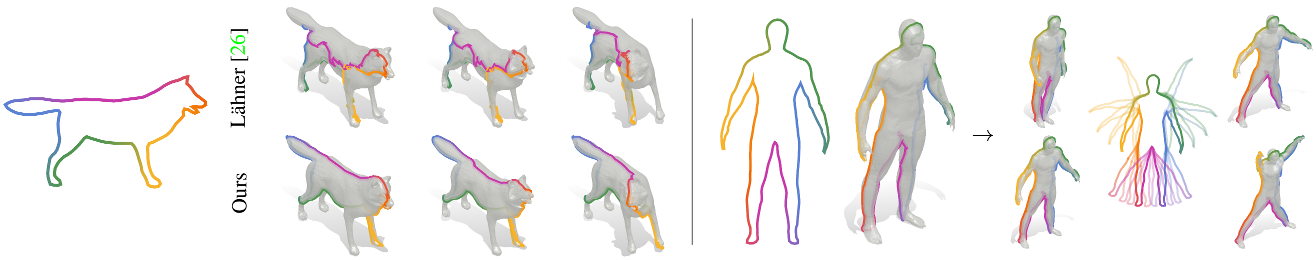
# Conjugate Product Graphs for Globally Optimal 2D-3D Shape Matching

Paul Roetzer[1] Zorah Lähner[2] Florian Bernard[1]
University of Bonn[1] University of Siegen[2]




Results of Lähner *et al.* [26] (top) and ours (bottom) on the TOSCA dataset. (i) Matching with our approach (ii) 2D to 3D deformation transfer

Figure 1. We propose a novel formalism for **globally optimal 2D contour to 3D shape matching** based on shortest paths in the *conjugate product graph*. For the first time we make it possible to incorporate higher-order costs within a shortest path-based matching formalism, which in turn enables to integrate powerful priors, *e.g.* favouring locally rigid deformations. **Left:** Our method produces compelling 2D-3D matchings that significantly outperform the previous state of the art [26]. **Right:** Sketch-based 2D to 3D deformation transfer by (i) computing a 2D-3D matching using our approach, (ii) manipulating the 2D sketch, and then transferring 2D deformations to the 3D shape.

## Abstract

*We consider the problem of finding a continuous and non-rigid matching between a 2D contour and a 3D mesh. While such problems can be solved to global optimality by finding a shortest path in the product graph between both shapes, existing solutions heavily rely on unrealistic prior assumptions to avoid degenerate solutions (e.g. knowledge to which region of the 3D shape each point of the 2D contour is matched). To address this, we propose a novel 2D-3D shape matching formalism based on the conjugate product graph between the 2D contour and the 3D shape. Doing so allows us for the first time to consider higher-order costs, i.e. defined for edge chains, as opposed to costs defined for single edges. This offers substantially more flexibility, which we utilise to incorporate a local rigidity prior. By doing so, we effectively circumvent degenerate solutions and thereby obtain smoother and more realistic matchings, even when using only a one-dimensional feature descriptor. Overall, our method finds globally optimal and continuous 2D-3D matchings, has the same asymptotic complexity as previous solutions, produces state-of-the-art results for shape matching and is even capable of matching partial shapes.*

## 1. Introduction

In recent years the computer vision community has put great effort into the matching of either two 2D or two 3D shapes. However, the task of matching a 2D shape to a 3D shape is a problem that has received less attention, even though it has a high practical relevance due to its wide variety of applications. For example, 2D-3D shape matching has the potential to bridge the gap between the 2D and 3D domain by making the interaction with 3D objects more accessible to non-experts, who typically find the manipulation of 2D shapes more intuitive. In addition to the modelling and manipulation of 3D shapes using 2D sketches (see Fig. 1), 2D-3D shape matching is relevant for 3D shape retrieval from 2D queries, for augmented reality interactions, for 3D image analysis based on 2D images (*e.g.* matching 2D X-ray image segmentations to 3D CT image segmentations), or for multimodal 2D-3D shape analysis.

2D-3D shape matching can be framed as finding a continuous mapping of a 2D contour (*e.g.* a sketch of an animal outline) to a 3D shape (*e.g.* a 3D model of this animal), see Fig. 1 (left). Here, the matched 2D contour that is deformed to the 3D shape should resemble the original 2D shape as much as possible, *i.e.* spatial shape deformations should be small. While humans have an intuitive and implicit understanding of *good* 2D-3D matchings, unfortunately, it is non-trivial to transfer this understanding into a rigorous mathematical framework: left-right flips are not distinguishable; the 2D shape does not contain all parts of the 3D shape (*e.g.* 2D shape of the wolf in Fig. 1 (left) contains only two legs); usually there is more than one *good* solution; and even slight deviations from a *good* matching can either be another good matching or can be a bad matching (*e.g.* zig-zagging on the 3D shape). In addition, phrasing 2D-3D shape matching as an optimisation problem requires to compute features on both shapes that allow to distinguish corresponding points from non-corresponding points – this is particularly difficult as many of the widely-used features for 2D or 3D shapes do not have a natural counterpart in the other domain, and are thus not directly comparable.

Nevertheless, previous work shows that matching a 2D contour to a 3D shape can be efficiently and globally optimal solved based on shortest paths in product graphs [26]. However, existing solutions require strong, unnatural assumptions (*e.g.* a coarse pre-matching, see Sec. 3.2) in order to resolve (some of) the above-mentioned difficulties. In this paper we present a novel graph-based formalism that relaxes previous unnatural assumptions, which in turn allows to solve a substantially more difficult setting of 2D-3D shape matching. Our main contributions are:

- We present a novel matching formalism based on conjugate product graphs that allow to encode more expressive higher-order information.
- For the first time this makes it possible to impose a local rigidity prior that penalises deformations, which in turn leads to previously unseen matching quality.
- Opposed to involved high-dimensional feature descriptors that were previously used (*e.g.* spectral features), our method requires only a simple one-dimensional feature that encodes the local object thickness – a feature that can naturally be defined for 2D and 3D shapes.
- Overall, our technical contributions allow us to solve 2D-3D matching for the first time without the requirement of a coarse pre-matching.

## 2. Related Work

In the following we summarise existing works that we consider most relevant in the context of this paper.

**Geometric Feature Descriptors.** Most matching approaches rely on point-wise features to decide what are good potential matches. For 2D contours, popular features are cumulative angles [49], curvature [25, 48] and various distance metrics [27, 31, 41, 48, 49]. One (for our work) notable example from the class of distance-based metrics is to consider the distance from each point to other parts of the contour along several fixed rays [31].

On 3D shapes, other feature types are predominant because the geometry is more complicated and 2D features often do not have a direct equivalent in 3D. While curvature does exist in 2D and in 3D, in 3D there are multiple notions of curvature, like mean, Gaussian and directional curvature. More popular are higher-dimensional features like the heat kernel signature [45] or wave kernel signature [1], which are based on spectral properties of the 3D surface, or the SHOT descriptor [47] based on the distribution of normals in the neighbourhood of a vertex. Recent approaches aim to learn suitable features for a specific matching pipeline [28, 29]. Overall, there is a discrepancy between 2D and 3D features, and even for features that can conceptually be calculated for both domains, they are typically not directly comparable. While [26] successfully addresses 2D-3D *shape retrieval* based on spectral 2D and 3D features, our experiments confirm that these features are insufficient to achieve precise *correspondences*. Similarly, many approaches learn multi-modal or multi-dimensional descriptors for entire shapes [16, 36, 50], but these are only useful in retrieval settings and not capable of point-to-point comparisons needed for finding reliable correspondences. For the 2D-3D shape matching problem, the use of learning-based methods is highly challenging, mainly due to the lack of suitable training data, which is non-trivial to produce, as well as the lack of similarity measures that do not require ground truth correspondences. Thus, in this paper we instead shift our focus on incorporating a powerful deformation prior, so that in turn substantially simpler feature descriptors are sufficient. We demonstrate that this allows to consider simple distance-based features which can be consistently computed both in 2D and 3D.

**2D-3D Matching.** Matching pairs of 2D objects is well-researched and it is widely known that respective solutions can be represented as paths in a graph. With that, shortest path algorithms can be used to efficiently find globally optimal solutions. This has for example been done for open contours, known as dynamic time warping [40], and closed contours [41], including invariance to scale and partiality [31]. Similarly, it was shown that matching 2D contours to 2D images (e.g. for template-based image segmentation) can be addressed using a similar framework [42]. Matching two 3D shapes is considerably harder as the solution is not a shortest path anymore but rather a minimal surface embedded in four-dimensional space. Thus, imposing constraints on the continuity of the solution is not possible in an efficient way [37, 53, 54]. While from an algorithmic perspective finding a 2D-3D matching is easier than the 3D-3D case (as the former also amounts to a (cyclic) shortest path problem [26]), quantifying matching costs is significantly more difficult for the 2D-3D case (cf. previous paragraph on feature descriptors). In this work we build upon the path-based 2D-3D matching formalism of [26] and propose a novel formalism that enables the use of higher-order costs (defined for chains of edges, opposed to costs of single edges). In turn, our formalism allows for the first time the incorporation of a spatial deformation prior, so that our framework requires substantially less descriptive features – in fact, we demonstrate that a one-dimensional distance-based feature descriptor (which is consistent for 2D and 3D shapes) is sufficient to successfully solve 2D-3D shape matching.

**Extensions of Graph Representations.** Graphs are not only relevant for diverse subfields of visual computing, such as e.g. image analysis [5, 7, 13, 22, 23, 34, 38, 39, 43], recognition [9, 12], tracking [18, 56], or mesh processing [4, 21, 26, 32], but also for a wide variety of other application domains, for example in DNA research [11], language processing [52], or social sciences [20]. In graph theory, many graph extensions have been proposed, including multilayer networks [24], dual graphs [15], hypergraphs [3], and product graphs [14], to name just a few. The product graph extends the concept of the Cartesian product (and other types of products) to graphs by additionally encoding neighbourhood information. This has been used in the context of different matching problems, including 2D-2D [41], 3D-3D [53], and 2D-3D [26] settings. Another graph extension relevant for this paper is the *conjugate graph* (also known as *line graph*), which encodes connectivity information into the vertices instead of edges [15]. This has for example been used for route planning [55] or graph link prediction [8]. In this work, we propose to combine product graphs with conjugate graphs (in fact we consider the conjugate graph of a product graph between two shapes) and showcase that this substantially increases modelling expressiveness and flexibility, and therefore allows for globally optimal 2D-3D shape matching.

## 3. Background & Notation

In this section we introduce our notation (also see Tab. 1), conjugate graphs, and the formalism for the matching of shapes as shortest path problem on a product graph.

**Definition 1** (Directed Graph) A directed graph $G$ is defined as a tuple $(V_G, E_G)$ of vertices $V_G$ and oriented edges $E_G \subseteq V_G \times V_G$. Oriented edges means $(v_1, v_2) \in E_G$ does not imply $(v_2, v_1) \in E_G$.

We directly work with discrete graph-based representations for shapes, *i.e.* contours sampled at $m$ many points represent 2D shapes, and (manifold) triangular surface meshes represent 3D shapes:

**Definition 2** (2D Shape) We define a 2D shape (or contour) $M$ as a tuple $(V_M, E_M)$ of $m$ vertices $V_M$ and $m$ oriented edges $E_M \subset V_M \times V_M$ s.t. $M$ is a directed cycle graph.

**Definition 3** (3D Shape) We define a 3D shape $N$ as a tuple $(V_N, E_N)$ of vertices $V_N$ and oriented edges $E_N \subset V_N \times V_N$ such that $N$ forms a 2D manifold in 3D space (triangular surface mesh, possibly with multiple boundaries).

We also consider an extended edge set, which contains all vertices represented as self-edges:

**Definition 4** (Extended Set of Edges) For a (2D or 3D) shape $X = (V_X, E_X)$, we define the extended set of edges $E_X^+ = E_X \cup \{(a, a) \mid a \in V_X\} \subset V_X \times V_X$. We call the additional edges *degenerate edges*.

| Symbol | Description |
|---|---|
| $M = (V_M, E_M)$ | 2D shape (closed contour) |
| $N = (V_N, E_N)$ | 3D shape (manifold triangular surface mesh) |
| $e^M$, $e^N$ | edge $e^M$ of contour, edge $e^N$ of mesh |
| $E_X^+$ | extended set of edges of shape $X \in \{M, N\}$ |
| $P = (V, E)$ | product graph of $M \times N$ with product vertices $V$ and product edges $E$ |
| $e$, $v$ | edge $e$, vertex $v$ of $P$ |
| $P = (V, E)$ | conjugate product graph of $P$ with vertices $V$ and edges $E$ |
| $e$, $v$ | edge $e$, vertex $v$ of $P$ |

Table 1. Summary of the notation used in this paper.

### 3.1. Conjugate Graphs

**Definition 5** (Conjugate Graph [15]) The conjugate graph $G$ of a directed graph $G$ is defined as a tuple $(V_G, E_G)$ with

$$V_G = E_G, \quad E_G = \{ ((v_1, v_2), (v_2, v_3)) \in V_G \times V_G \}.$$

Intuitively, the edges of $G$ become the vertices of $G$ and the conjugate edges connect pairs of adjacent edges of $G$ from Fig. 2 illustrates the construction of the conjugate graph.

Figure 2. Illustration of the *conjugate graph*. (a) Input graph $G$. (b) Each edge of $G$ becomes a conjugate vertex ( ). (c) The conjugate graph $G$ is now formed by connecting the newly introduced vertices ( ) by edges ( ) according to Definition 5 (*e.g.* conjugate vertex $(1, 3)$ and $(3, 4)$ are connected by a directed edge since they are both adjacent to vertex 3 in $G$).

### 3.2. Matching Formalism

Next, we introduce the product graph $G$ between a 2D contour and a 3D shape, and we summarise Lähner et al.'s [26] representation of a 2D-3D shape matching as shortest (cyclic) path in the product graph.

**Definition 6** (Product Graph) The product graph $\mathcal{P}$ of the 2D contour $\mathcal{M}$ and the 3D shape $\mathcal{N}$ is a tuple $(V_{\mathcal{P}}, E_{\mathcal{P}})$ of product vertices $V_{\mathcal{P}}$ and product edges $E_{\mathcal{P}}$, where

$$V_{\mathcal{P}} = V_{\mathcal{M}} \times V_{\mathcal{N}},$$
$$E_{\mathcal{P}} = \{ (e_1^{\mathcal{M}}, e_2^{\mathcal{N}}) \in V_{\mathcal{P}} \times V_{\mathcal{P}} \mid e_1^{\mathcal{M}} \in E_{\mathcal{M}}^{+}, e_2^{\mathcal{N}} \in E_{\mathcal{N}}^{+}, e_1^{\mathcal{M}} \text{ or } e_2^{\mathcal{N}} \text{ non-deg.} \}.$$

The product graph $\mathcal{P}$ is visualised in Fig. 3 (left). To simplify the notation, we will refer to the vertices and edges of the product graph only as $V$ and $E$ for the remainder of the paper. A matching between the 2D contour and the 3D shape can be represented as the subset $C \subset E_{\mathcal{M}} \times E_{\mathcal{N}}$ where tuples in $C$ indicate which edges of the 2D contour and 3D shape are in correspondence. Desirable properties of such matchings are that a) each edge on $\mathcal{M}$ is matched to at least one edge on $\mathcal{N}$, and b) the matching is continuous, *i.e.*, if two edges on $\mathcal{M}$ are adjacent, their matches on $\mathcal{N}$ should also be adjacent. These properties can be guaranteed if the solution is a (cyclic) path that goes through all layers of the product graph (cf. Fig. 3). A path that minimises costs defined on the (product graph) edges can efficiently be computed based on Dijkstra's algorithm [10]. To ensure the path is cyclic, Dijkstra's algorithm needs to be run multiple times (once for each vertex of the 3D mesh); however, the number of Dijkstra runs can be drastically reduced based on a simple branch and bound strategy, see [26].

Despite the theoretical elegance of Lähner *et al.*'s formalism, a major limitation is that shortest paths only take into account costs of individual edges. With that, the approach is not capable of penalising local deformations induced by a matching, this is only possible with *pairs* of product graph edges. Instead, the authors use high-dimensional features in combination with knowledge about pre-matched segmentations between 2D and 3D shapes. While such a pre-matching drastically reduces the search space and avoids many degenerate solutions, the knowledge of such a pre-matching is typically not available in practice.

## 4. Our 2D-3D Shape Matching Approach

In the following, we present our solution for 2D-3D shape matching which allows the incorporation of higher-order edge costs. With this, spatial deformations can be naturally penalised within a shortest path matching framework while still finding a globally optimal matching in polynomial time (with the same asymptotic complexity as [26]), see Section 3.2. We emphasise that we address a significantly more difficult problem setting than [26] since we do not rely on the unrealistic assumption that a coarse 2D-3D pre-matching is available. A summary of our notation can be found in Tab. 1.

### 4.1. Conjugate Product Graphs

Our formalism builds upon conjugate product graphs, i.e. the conjugate graph (Definition 5) of a product graph (Definition 6). We refer to the conjugate product graph as $\mathcal{P}^{\vee} = (V^{\vee}, E^{\vee})$. Here, edges in the product graph become vertices in the conjugate product graph and are connected based on the adjacency of vertices in the product graph, see Definition 5. Thus, an edge $e^{\vee} \in E^{\vee}$ in $\mathcal{P}^{\vee}$ has the scope of two edges in the product graph $\mathcal{P}$, i.e. $e^{\vee} = (e_1, e_2)$, $e_1, e_2 \in E$, see Fig. 4. In turn, this enables the definition of cost functions that consider two product graph edges simultaneously. We note that higher-order costs can also be defined by repeating the conjugation process, *e.g.* an edge in the conjugate of the conjugate product graph is formed by three edges of the product graph so that costs can be defined for triplets of product graph edges (and so on). For brevity and a simpler exposure, in the following we restrict ourselves w.l.o.g. to second-order costs.

Figure 3. Illustration of the product graph $\mathcal{P}$ (left) and the conjugate product graph $\mathcal{P}^{\vee}$ (right) for a water drop shape. **Left:** the product graph $\mathcal{P}$ is structured into layers (, , ), where each layer represents a single vertex on $\mathcal{M}$ and the entire 3D shape $\mathcal{N}$. **Right:** we illustrate (part of) the conjugate product graph $\mathcal{P}^{\vee}$ for the three-edge path in $\mathcal{P}$ (highlighted in pink), which becomes in $\mathcal{P}^{\vee}$. Conjugate product vertices are shown in orange and conjugate product edges are shown in pink.

### 4.2. Cost Function

We define our cost function $d : E^{\vee} \to \mathbb{R}$ for every edge $e^{\vee} = (v_1^{\vee}, v_2^{\vee}) = (e_1, e_2) \in E^{\vee}$ in the conjugate product graph as

$$d(e^{\vee}) = d_{data}(e^{\vee}) + d_{reg}(e^{\vee}). \tag{1}$$

$d_{data}$ is the data term which measures the similarity between the product edges based on feature descriptors. $d_{reg}$ is a local rigidity regulariser which ensures that adjacent edges on the 2D contour are deformed similarly to adjacent

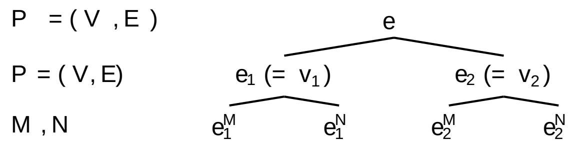



Figure 4. Hierachical relationship between edges of the involved graphs. An edge $e \in E$ of the conjugate product graph $P$ is formed by two edges $e_1, e_2 \in E$ of the product graph $P$ (which correspond to vertices $v_1, v_2 \in V$ in the conjugate product graph $P$, respectively). Each edge of $P$ is formed by one edge $e^M \in E_M^+$ and one edge $e^N \in E_N^+$ of the shapes $M$ and $N$, respectively.

(a) (b)

Figure 5. (a) The local thickness $\ell_p$ for the point $p$ is found by intersecting the ray from $p$ in the opposite normal direction (light red) with the shape. (b) Illustration of finding the rotation $R_e$ that aligns the 3D coordinate frame defined for a 2D contour edge $e^M$ and the 3D coordinate frame defined for a 3D shape edge $e^N$. The black vector shows the shape edge, the red vector the normal and the green vector their cross product.

elements on the 3D shape. We first describe the data term followed by the local rigidity regulariser.

**Data Term.** A major difficulty when comparing 2D and 3D shapes is that many of the existing geometric feature descriptors cannot be consistently defined for 2D and 3D shapes (*e.g.* although the notion of curvature exists for both shapes, in 3D the curvature is direction-dependent, which makes it difficult to compare 2D and 3D curvature). We build a simple one-dimensional descriptor based on the observation that corresponding points $i \in M$ and $j \in N$ of the same shape class should have a similar distance to the other side going through the interior of the respective shape, see Fig. 5a. As such, we consider *local thickness* as feature descriptor. It is computed as follows:

- For a vertex $i \in M$, the 2D local thickness $\ell_i^{2D}$ can be found by inverting its vertex-normal and finding the (first) intersection with the contour [31].
- For a vertex $j \in N$, the 3D local thickness $\ell_j^{3D}$ can be computed by finding the (first) intersection of ray from $j$ in the opposite vertex-normal direction with $N$. We employ a triangle-ray-intersection algorithm for this [33].

With that, we can define the local thickness difference for each conjugate product edge $e = (e_1, e_2) \in E$, so that our data term $d_{data}(e)$ reads

$$d_{data}(e) = \rho_1(|\ell_i^{3D} - \ell_j^{3D}|), \tag{2}$$

where $\ell_i^{3D}$ and $\ell_j^{3D}$ are local thickness values at vertices $i \in V_M$ and $j \in V_N$ on the 2D and 3D shape, respectively. Since we want to avoid taking into account the same local thickness value multiple times, we compute the local thickness difference $d_{data}(e)$ at a conjugate product vertex $e$ solely with the local thickness at vertex $i$ shared by $e_1^M$ and $e_2^M$, and respectively the local thickness at vertex $j$ shared by $e_1^N$ and $e_2^N$. Since potentially there may be large outliers in the local thickness near areas of large deformations, we additionally apply the function $\rho_1(\cdot)$ to the absolute value of the local thickness difference, which can for example be chosen to be a robust loss function. We have found that despite its simplicity, local thickness is an effective one-dimensional feature descriptor that, in combination with our local rigidity regulariser, enables faithful 2D-3D shape matchings, see Sec. 5.

**Regularisation.** Inspired by [4, 44], we employ a regularisation term which favours deformations that are locally rigid. To compute the regularisation of the conjugate product edge $e = (e_1, e_2) = ((e_1^M, e_1^N), (e_2^M, e_2^N))$ (cf. Fig. 4), we define a local 3D coordinate frame for each of the four (shape) edges $e_1^M, e_1^N, e_2^M, e_2^N$. To this end, we embed the 2D contour into 3D space by adding a third constant coordinate. With that, for both 2D contour and 3D shape, we can define a local 3D coordinate frame based on the normalised edge direction, outward-pointing unit normal, and their cross product. Subsequently, we solve the orthogonal Procrustes problem [46] in order to compute the rotation $R_{e_1}$ that aligns the 3D coordinate frame of $e_1^M$ to the 3D coordinate frame of $e_1^N$, and the rotation $R_{e_2}$ that aligns the 3D coordinate frame of $e_2^M$ to the coordinate of $e_2^N$, see Fig. 5b. In presence of degenerate edges we simply use the previous edge, see also Sec. 4.3.

By computing the geodesic distance between $R_{e_1}$ and $R_{e_2}$ on the Lie group $SO(3)$, we can quantify the amount of non-rigidity of the matching that is induced by $e$. For computational efficiency, we consider unit quaternion representations $q_e$ of $R_e$, so that our local rigidity regularisation term $d_{reg}$ for the conjugate product edge $e$ reads

$$d_{reg}(e) = \rho_2\left(\arccos(\langle q_{e_1}, q_{e_2}\rangle)\right), \tag{3}$$

where $\langle\cdot,\cdot\rangle$ is the inner product for quaternions [17]. $\rho_2(\cdot)$ is again a robust loss function, see Sec. 4.3.

### 4.3. Theoretical Analysis and Implementation

In the following we provide a theoretical analysis and additional implementation details.

**Cyclic Shortest Paths.** To find a cyclic shortest path, we can run an ordinary (non-cyclic) shortest path algorithm (*e.g.* Dijkstra's algorithm [10]) $|V_N|$ many times. To this end, we duplicate the last layers in the conjugate product graph (see Fig. 3), disconnect the duplicate layers from each other, and for each vertex from the 'upper duplicate' find the shortest path to the corresponding vertex in the 'lower duplicate'. The globally optimal shortest cyclic path is now formed by the minimum among the $|V_N|$ individual paths. To reduce the number of shortest paths that need to be computed, we can instead resort to a more efficient branch-and-bound strategy, we refer to the Appendix for details.

Degenerate Cases. Conjugate product vertices containing degenerate edges of the 3D shape do not contain directional information on the 3D shape which we need to compute $d_{reg}$. We inject the relevant directional information into the conjugate product graph by introducing new conjugate product vertices that reflect (non-degenerate) edges on the 3D shape adjacent to respective degenerate 3D edge.

Pruning. To decrease the size of the conjugate product graph $P$, we apply a pruning strategy. To this end, we prune conjugate product edges that reflect local turning points on the 3D shape since such paths represent undesirable matchings. In addition, we prune edges that first represent a degenerate edge of $M$, followed by a degenerate edge of $N$ (or vice-versa). Such combinations are equivalent to a matching with two non-degenerate edges. Overall, our pruning reduces the graph size (and thus runtime) and excludes obvious non-desirable solutions.

Runtime Analysis. The runtime of our algorithm depends on the size of the conjugate product graph and the number of shortest path runs. The number of vertices in $P$ corresponds to the number of edges in $P$. The number of edges in $P$ can be approximated by $c|V_M| \; |E_N| + |V_N|$ where $c$ is a constant that is related to the maximum number of neighbours of the vertices in $M$. Tab. 2 sums up the sizes of the product graph $P$ and the conjugate product graph $P$.

| | # vertices | # edges |
|---|---|---|
| $P$ | $\|V_M\| \; \|V_N\|$ | $\|V_M\| \; 2\|E_N\| + \|V_N\|$ |
| $P$ | $\|V_M\| \; 2\|E_N\| + \|V_N\|$ | $c \; \|V_M\| \; 2\|E_N\| + \|V_N\|$ |

Table 2. Comparison of sizes of the product graph $P$ and the conjugate product graph $P$.

Using $|E_N| \; 3|V_N|$ [6] shows that the conjugate product graph $P$ has 7 times more vertices and $c$ 11 (see Appendix) times more edges than the product graph $P$ which shows that asymptotically both graphs have the same size. In the worst case $O(|V_N|)$ shortest path problems – one for each vertex in $N$ – have to be solved. Together with the complexity of each Dijkstra run the final runtime can be estimated as $O \; |V_M| \; |V_N|^2 \; \log(|V_N|)$, which is the same as in [26]. We provide more details in the Appendix.

Implementation Details. We implement the shortest path algorithm in C++ wrapped in a MATLAB [30] mex-function. Computation of quantities on meshes, mesh simplification as well as local thickness computation are done using [19, 30]. For all experiments we choose $_1(x)$ to be the robust loss function of [2], for which we choose $_1 = \; 2$ and $c_1 = 0.15$. For $_2(x)$ we also choose the same loss function with $_2 = 0.7$ and $c_2 = 0.6$, but with a cubic bowl instead of a quadratic bowl as we want to ensure that small errors due to discretisation artefacts are not penalised. The choice of different $_1(x)$ and $_2(x)$ is required since $d_{data}$ and $d_{reg}$ have different ranges.

# 5. Experiments

In this section we compare our method on two datasets, conduct an ablation study, and showcase results on partial shapes and for sketch-based shape manipulation. We emphasise that the matching of contours to 3D meshes is ill-posed: the same contour can arise from different configurations, *i.e.* the ground-truth is not necessarily unique, the space of solutions that seem natural is even bigger, and evaluation criteria that capture this non-uniqueness do not exist.

Datasets. We evaluate on the following two datasets:

- TOSCA 2D-3D [26]: 80 shapes of 9 different classes (humans, animals, etc.) in different poses. For each class exists at least one 2D shape.
- FAUST 2D-3D [26]: 100 human shapes in different poses subdivided into 10 classes. Each class has one 2D shape. Ground-truth correspondences between 2D and 3D are available for all instances.

Both datasets contain segmentation information across all shapes which form consistent 2D part to 3D part mappings.

Competing Approach. The only other method able to produce continuous matchings between 2D contours and 3D shapes is [26]. Due to their weaker model expressiveness that prevents the incorporation of a deformation prior, they use global spectral features and a pre-matched segmentation as additional feature in order to prevent degenerate solutions (*e.g.* collapsing). To enable a fair comparison, for both methods we provide results with and without this pre-matching. However, we consider the pre-matching as unrealistic prior knowledge, and therefore regard the cases without pre-matching as main results. As we show in Fig. 1, our results are superior without the segmentation term even in comparison to [26] using the segmentation term.

## 5.1. Matching

Next we evaluate our approach on the task of 2D-3D shape matching. First, we introduce a new error metric designed for the ambiguous setting of matching a contour onto a mesh. Subsequently, we compare quantitatively and qualitatively to the approach by Lähner *et al.* [26].

Error Metric. We use two different error metrics: a) geodesic error and b) segmentation error. We only evaluate the geodesic error on FAUST due to the lack of 2D-3D ground truth correspondences in the TOSCA dataset. Additionally, there exist many valid matchings that may not cor-

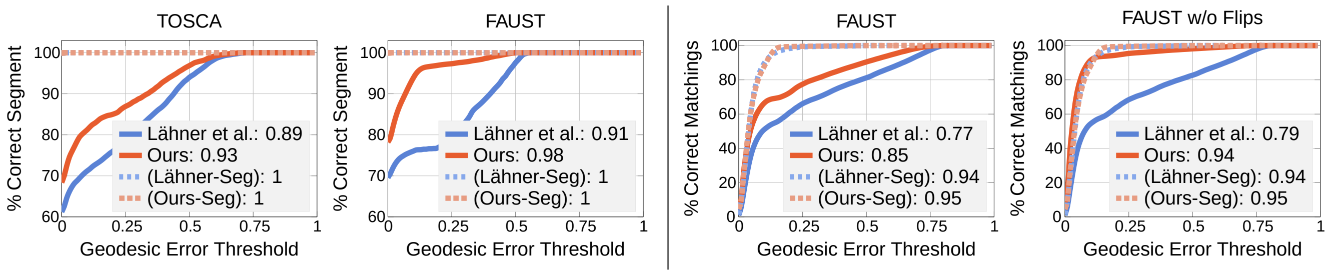



Figure 6. Quantitative comparison for the FAUST and TOSCA datasets. *Left:* Cumulative segmentation errors. The y-axis shows the percentage of points in the correct segment, and the x-axis the geodesic error threshold. Vacuously, when integrating the segmentation information into the optimisation (methods with suffix '-Seg', dashed lines), the results are perfect for both methods. *Right:* Cumulative geodesic errors on FAUST with and without left-right flips (manually removed for all approaches), which confirms that in many cases our method finds plausible solutions but does not resolve the intrinsic symmetry ambiguity. The y-axis shows the percentage of points below the x-axis threshold. We can see that our method consistently outperforms Lähner et al. [26]. Scores shown in the legends are respective areas under the curves.

respond to the ground truth because the problem is ill-posed as explained above. Hence, we aim to derive a more robust quantitative evaluation for 2D-3D matchings. For that, we utilise part-based shape segmentations, which are available for all classes in the FAUST and TOSCA datasets and are generally consistent between 2D and 3D shapes. We argue that a good solution must have the same segmentation in the target domain, i.e. on the 3D shape, as in the source domain, i.e. on the 2D shape. For both we plot the cumulative curves measuring for each geodesic error value the percentage of matches with an error lower than this.

**Geodesic Error.** Let $(x, y) \in C \subset V_M \times V_N$ be a computed match and $y^*$ be the ground-truth match of $x$. The normalised geodesic error of this matching is defined as

$$\epsilon_{geo}(x, y) = \frac{\mathrm{dist}_N(y, y^*)}{\mathrm{diam}(N)}. \quad (4)$$

Here $\mathrm{dist}_N : N \times N \to \mathbb{R}^+_0$ is the geodesic distance on $N$ and $\mathrm{diam}(N) = \max_{x,y \in N} \mathrm{dist}_N(x, y)$.

**Segmentation Error.** Let $\sigma_M(x)$ be the source segment of its matched point $y \in N$ and let $\sigma_N(y)$ be its target segment. We define the segmentation error as

$$\epsilon_{seg}(x, y) = \min_{\substack{y' \in N,\\ \sigma_N(y') = \sigma_M(x)}} \frac{\mathrm{dist}_N(y, y')}{\mathrm{diam}(N)}. \quad (5)$$

For shapes with symmetries or other ambiguities, we choose the best of all plausible segmentation combinations.

**Quantitative Matching Results.** In Fig. 6 (left) we show that our method outperforms the competing method by Lähner et al. [26] by a great margin in terms of the segmentation error, both on FAUST on TOSCA. Since for FAUST ground truth is available, in Fig. 6 (right) we show the percentage of correct matchings, there our method is superior. In addition, when left-right flips (which form plausible solutions that stem from shape symmmetries) are removed, our method (without pre-matching) is on par with the approach by Lähner et al. that uses pre-matching.

Figure 7. Qualitative results of our method on FAUST. We can see the occurrence of left-right flips (indicated by ) which nevertheless can be considered as plausible matchings.

**Qualitative Matching Results.** We also compare our method qualitatively to Lähner et al. [26]. Even though our method is not using segmentation information, matchings computed with our approach are consistently of better quality and reflect a more plausible path on the 3D shape, i.e. are locally straight, see Fig. 1, Fig. 7 and Fig. 8.

## 5.2. Ablation Study

We evaluate the performance of different parts of our cost function in Tab. 3 as well as the performance of local rigidity when using multidimensional spectral features.

## 5.3. Partial Shapes

We show experiments on partial shapes, for which we removed parts of either the 2D or 3D shape in FAUST, see Fig. 9. Our approach is substantially more robust in the partial setting compared to Lähner et al. [26], likely due to

2D Shape

Lähner et al.

Ours

Figure 8. Qualitative comparison of the method by Lähner et al. [26] (second row) and our approach (third row) on TOSCA. Our approach results in more plausible matchings despite that Lähner et al. use a coarse segmentation-based pre-matching. Our local rigidity regulariser, which is enabled by our novel conjugate product graph formalism, ensures that resulting paths on 3D target shapes are much smoother.

| Method | AUC " |
| --- | --- |
| Local Rigidity & Spectral | 0.95 |
| Local Rigidity | 0.76 |
| Local Thickness | 0.92 |
| Local Rigid. & Local Thick., ( $_1$(x) = $_2$(x) = jxj) | 0.89 |
| Ours | 0.98 |

Table 3. Ablation study on FAUST. The score is the area under the curve (AUC) of the cumulative segmentation errors. All introduced components increase performance. Our one-dimensional local thickness outperforms the multi-dimensional spectral features due to different intrinsic properties of 2D and 3D shapes.

2D Shape

Lähner et al.

Ours

Figure 9. Qualitative comparison of Lähner et al. [26] and ours on partial FAUST shapes. The global features of [26] result in poor matchings in scenarios without full shape, whereas we use local features and thus obtain valid partial matchings.

the locality of our features and strong spatial regularisation, in contrast to the global spectral features of [26].

### 5.4. Sketch-Based 3D Shape Manipulation

We show the high quality of our matchings by performimg 2D sketch-based 3D shape manipulation. After deforming the contour, the 3D shape is brought into a corresponding pose through as-rigid-as-possible shape deformation [44], see Fig. 1. Details can be found in the Appendix.

## 6. Discussion & Limitations

Our experimental results con rm that conjugate product graphs enable 2D-3D shape matching without the need of a coarse pre-matching. Even though we compute results to global optimality, scenarios like symmetries (e.g. for human shapes) lead to ambiguities that are challenging to re ect in the cost function, which may result in matchings that collapse to one side of the 3D shape, see Fig. 7 (bottom-right). Although our method has the same asymptotic complexity as [26], in practice the computation is slower due to the conjugate product graph being larger (by a constant factor) than the product graph (cf. Tab. 2, also see Appendix).

## 7. Conclusion

We presented conjugate product graphs for 2D-3D shape matching, which for the  rst time allows for the incorporation of higher-order costs within path-based matching formalisms. Our novel concept signi cantly increases model expressiveness and  exibility, allowing to inject desirable properties, like local rigidity regularisation, into respective optimisation problems. Our results show signi cant improvements in challenging matching settings, even allowing for 2D sketch-based 3D shape manipulation. Since our powerful higher-order regularisation allows to get rid of the need for global features, our method is the  rst that solves partial 2D-3D shape matching. We believe that our work is of high relevance to the  eld of shape analysis, and hope to inspire more work on inter-dimensional applications.

## Acknowledgements

PR is funded by the TRA Modellling (University of Bonn) as part of the Excellence Strategy of the federal and state governments. ZL is funded by a KI-Starter grant of the Ministry of Culture and Science of the State of North Rhine-Westphalia.

## References

[1] 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. 2

[2] Jonathan T Barron. A general and adaptive robust loss function. In *CVPR*, 2019. 6

[3] Claude Berge. *Hypergraphs: combinatorics of finite sets*. Elsevier, 1984. 3

[4] Florian Bernard, Frank R Schmidt, Johan Thunberg, and Daniel Cremers. A combinatorial solution to non-rigid 3d shape-to-image matching. In *CVPR*, 2017. 3, 5

[5] Florian Bernard, Christian Theobalt, and Michael Moeller. Ds*: Tighter lifting-free convex relaxations for quadratic matching problems. In *CVPR*, 2018. 3

[6] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon Mesh Processing*. CRC press, 2010. 6, 12

[7] Yuri Boykov and Gareth Funka-Lea. Graph cuts and efficient nd image segmentation. *International journal of computer vision*, 70(2):109–131, 2006. 3

[8] Lei Cai, Jundong Li, Jie Wang, and Shuiwang Ji. Line graph neural networks for link prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. 3

[9] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, 2004. 3

[10] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, 1959. 4, 5

[11] Hossein Eghdami and Majid Darehmiraki. Application of dna computing in graph theory. *Artificial Intelligence Review*, 2012. 3

[12] Pedro F Felzenszwalb and Daniel P Huttenlocher. Pictorial structures for object recognition. *International journal of computer vision*, 2005. 3

[13] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, 1984. 3

[14] Richard H Hammack, Wilfried Imrich, Sandi Klavžar, Wilfried Imrich, and Sandi Klavžar. *Handbook of product graphs*. CRC press Boca Raton, 2011. 3

[15] Frank Harary. *Graph Theory*. Addison-Wesley, 1969. 3

[16] Robert Herzog, Daniel Mewes, Michael Wand, Leonidas Guibas, and Hans-Peter Seidel. Lesss: Learned shared semantic spaces for relating multi-modal representations of 3d shapes. *Computer Graphics Forum*, 2015. 2

[17] Du Q Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 2009. 5

[18] Umar Iqbal, Anton Milan, and Juergen Gall. Posetrack: Joint multi-person pose estimation and tracking. In *CVPR*, 2017. 3

[19] Alec Jacobson et al. gptoolbox: Geometry processing toolbox, 2021. http://github.com/alecjacobson/gptoolbox. 6, 13

[20] Long Jin, Yang Chen, Tianyi Wang, Pan Hui, and Athanasios V Vasilakos. Understanding user behavior in online social networks: A survey. *IEEE communications magazine*, 2013. 3

[21] Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. Learning 3d mesh segmentation and labeling. In *ACM SIGGRAPH*, 2010. 3

[22] Jörg Hendrik Kappes, Markus Speth, Bjoern Andres, Gerhard Reinelt, and Christoph Schnörr. Globally optimal image partitioning by multicuts. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*. Springer, 2011. 3

[23] Margret Keuper, Evgeny Levinkov, Nicolas Bonneel, Guillaume Lavoue, Thomas Brox, and Bjoern Andres. Efficient decomposition of image and mesh graphs by lifted multicuts. In *CVPR*, 2015. 3

[24] Mikko Kivelä, Alex Arenas, Marc Barthelemy, James P Gleeson, Yamir Moreno, and Mason A Porter. Multilayer networks. *Journal of complex networks*, 2014. 3

[25] Zhang Kun, Ma Xiao, and Li Xinguo. Shape matching based on multi-scale invariant features. *IEEE Access*, 2019. 2

[26] Zorah Lähner, Emanuele Rodolà, Frank R Schmidt, Michael M Bronstein, and Daniel Cremers. Efficient globally optimal 2d-to-3d deformable shape matching. In *CVPR*, 2016. 1, 2, 3, 4, 6, 7, 8, 11, 12, 13

[27] Haibin Ling and David W. Jacobs. Shape classification using the inner-distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007. 2

[28] O. Litany, T. Remez, E. Rodolà, A. M. Bronstein, and M. M. Bronstein. Deep functional maps: Structured prediction for dense shape correspondence. In *ICCV*, 2017. 2

[29] Riccardo Marin, Marie-Julie Rakotosaona, Simone Melzi, and Maks Ovsjanikov. Correspondence learning via linearly-invariant embedding. In *NeurIPS*, 2020. 2

[30] The Mathworks, Inc., Natick, Massachusetts. *MATLAB version 9.13.0.2049777 (R2022b)*, 2022. 6

[31] Damien Michel, Iasonas Oikonomidis, and Antonis Argyros. Scale invariant and deformation tolerant partial shape matching. *Image and Vision Computing*, 2011. 2, 5

[32] Joseph SB Mitchell, David M Mount, and Christos H Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 1987. 3

[33] Tomas Möller and Ben Trumbore. Fast, minimum storage ray/triangle intersection. In *ACM SIGGRAPH 2005 Courses*. 2005. 5

[34] Eric N Mortensen and William A Barrett. Intelligent scissors for image composition. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, 1995. 3

[35] Per-Olof Persson and Gilbert Strang. A simple mesh generator in matlab. SIAM review, 2004. 13

[36] Jie Qin, Shuaihang Yuan, Jiaxin Chen, Boulbaba Ben Amor, Yi Fang, Nhat Hoang-Xuan, Chi-Bien Chu, Khoi-Nguyen Nguyen-Ngoc, Thien-Tri Cao, Nhat-Khang Ngo, Tuan-Luc Huynh, Hai-Dang Nguyen, Minh-Triet Tran, Haoyang Luo, Jianning Wang, Zheng Zhang, Zihao Xin, Yang Wang, Feng Wang, Ying Tang, Haiqin Chen, Yan Wang, Qunying Zhou, Ji Zhang, and Hongyuan Wang. Shrec'22 track: Sketch-based 3d shape retrieval in the wild. Computers & Graphics, 2022. 2

[37] Paul Roetzer, Paul Swoboda, Daniel Cremers, and Florian Bernard. A scalable combinatorial solver for elastic geometrically consistent 3d shape matching. In CVPR, 2022. 2

[38] Stefan Roth and Michael J Black. Fields of experts: A framework for learning image priors. In CVPR. IEEE, 2005. 3

[39] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "grabcut" interactive foreground extraction using iterated graph cuts. ACM transactions on graphics (TOG), 2004. 3

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

[41] Frank R Schmidt, Dirk Farin, and Daniel Cremers. Fast matching of planar shapes in sub-cubic runtime. In IEEE International Conference on Computer Vision, 2007. 2, 3

[42] Thomas Schoenemann, Frank R. Schmidt, and Daniel Cremers. Image segmentation with elastic shape priors via global geodesics in product spaces. In BMVC, 2008. 2

[43] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. IEEE Transactions on pattern analysis and machine intelligence, 2000. 3

[44] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In SGP, 2007. 5, 8, 13

[45] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. A concise and provably informative multi-scale signature based on heat diffusion. In SGP, 2009. 2

[46] Jos MF Ten Berge. Orthogonal procrustes rotation for two or more matrices. Psychometrika, 1977. 5, 13

[47] Federico Tombari, Samuele Salti, and Luigi Di Stefano. Unique signatures of histograms for local surface description. In ECCV, 2010. 2

[48] Remco C Veltkamp. Shape matching: Similarity measures and algorithms. In Proceedings International Conference on Shape Modeling and Applications, 2001. 2

[49] Remco C Veltkamp and Michiel Hagedoorn. State of the art in shape matching. Principles of visual information retrieval, 2001. 2

[50] Fang Wang, Le Kang, and Yi Li. Sketch-based 3d shape retrieval using convolutional neural networks. In CVPR, 2015. 2

[51] Yu Wang, Alec Jacobson, Jernej Barbič, and Ladislav Kavan. Linear subspace design for real-time shape deformation. TOG, 2015. 13

[52] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In HLT and EMNLP, 2005. 3

[53] Thomas Windheuser, Ulrich Schlickewei, Frank R. Schmidt, and Daniel Cremers. Geometrically consistent elastic matching of 3D shapes: A linear programming solution. In ICCV, 2011. 2, 3

[54] Thomas Windheuser, Ulrich Schlickwei, Frank R. Schimdt, and Daniel Cremers. Large-Scale Integer Linear Programming for Orientation Preserving 3D Shape Matching. Computer Graphics Forum, 2011. 2

[55] Stephan Winter. Modeling costs of turns in route planning. GeoInformatica, 2002. 3

[56] Li Zhang, Yuan Li, and Ramakant Nevatia. Global data association for multi-object tracking using network flows. In CVPR, 2008. 3

# Appendix

## A. Segmentation Pre-Matching

In Fig. 10, we visualise the pre-matching which is used by the approach in [26]. It is obvious that the injection of such information into the objective function makes finding valid solutions substantially easier.

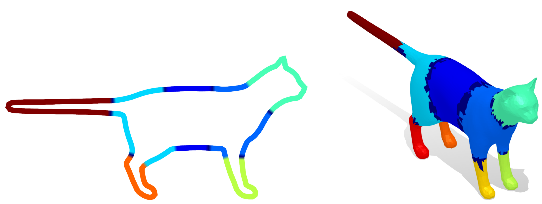


Figure 10. Visualisation of pre-matched segmentation information on cat from the TOSCA dataset. Different colours encode different segments, while darkest blue encodes the transition between different segments.

For a fair comparison, in addition to our method that does *not* use this information, we also evaluate "Ours-Seg.", in which we incorporate the above segmentation information as an additional feature descriptor, see [26] for details.

## B. Branch and Bound Algorithm

Algorithm 1 describes our optimisation strategy. We adapt the branch and bound algorithm introduced in [26] to conjugate product graphs and implement runtime improvements by increasing chances of finding tighter upper bounds earlier.

The final goal of the optimisation is to find a *cyclic* path with minimal cost. However, Dijkstra's algorithm only finds shortest (but not necessarily cyclic) paths. To that end, we represent the (conjugate) product graph as sequential graph, in which the first and last layers are duplicates, such that a path going from the same vertex in the first and last layer corresponds to a cyclic path.

Thus, the cyclic path with minimal cost can be found by computing the shortest path for every vertex on the first layer to every respective vertex on the last layer, and subsequently choosing among the computed paths the one with minimal cost. In general, this requires to solve a total of $2|E_N| + |V_N|$ (ordinary) shortest path problems, and is computationally more expensive than the branch-and-bound strategy that we pursue.

The main idea of branch-and-bound is to iteratively subdivide the search space, while tightening upper and lower bounds using the results of previous iterations. In that sense, instead of searching for shortest paths from each vertex on the first layer to each respective vertex on the last layer, we search for the shortest path from a set of vertices $B \subseteq V$ on the first layer to the respective set of vertices $B$ on the last layer, see Fig. 11 (left). There is no guarantee that the path $C = (v_1, \ldots, v_{|C|})$ from $B$ (first layer) to its duplicate $B$ (last layer) with minimal energy is indeed cyclic, *i.e.* that the final vertex $v_{|C|}$ in the last layer is indeed the same as the starting vertex $v_1$ in the first layer. If $C$ is not cyclic, we partition $B$ into smaller, disjunct subsets $B_1$ and $B_2$ (with $B_1 \cup B_2 = B$ and $B_1 \cap B_2 = \emptyset$) until a cyclic path is found (this is the *branching strategy*, see Fig. 11). The partitioning is done by calculating Voronoi cells around edges $e_1^N$ and $e_{|C|}^N$ on 3D shape assuming $e_1^N$ and $e_{|C|}^N$ are not identical (where the conjugate product vertex $v_1 = (e_1^M, e_1^N)$ contains edge $e_1^N$ on 3D shape and conjugate product vertex $v_{|C|} = (e_{|C|}^M, e_{|C|}^N)$ contains edge $e_{|C|}^N$ on 3D shape). If $e_1^N$ and $e_{|C|}^N$ are identical we partition according to $B_1 = B \setminus \{v_{|C|}\}$ and $B_2 = \{v_{|C|}\}$.

The path cost $d_C$ of non-cyclic paths (*i.e.* $v_1 \neq v_{|C|}$) is a lower bound $b(\ )$ on the path cost of the globally optimal cyclic path. Whenever $v_1$ and $v_{|C|}$ are equal (meaning that

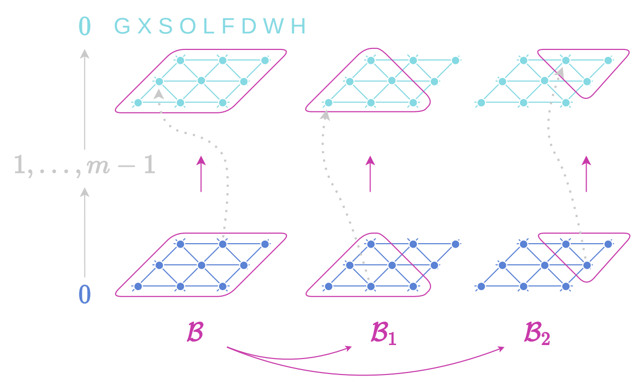



Figure 11. Illustration of the **branching strategy** in Algorithm 1. First, the shortest path from all vertices in $B$ on the first layer to the same vertices on the duplicate first layer (which amounts to the last layer) is computed. The resulting shortest path from $B$ on the first layer to $B$ on the last layer might not start and end on the same vertex (since we are searching for a shortest path from a set of vertices to a set of vertices). Whenever this is the case, $B$ is partitioned into two sets $B_1$ and $B_2$, for which in subsequent iterations shortest paths are computed analogously as for $B$.

$C$ is a cyclic path), an upper bound $b_{\text{upper}}$ is found, which might already be the globally optimal path, but can only be identified as such if all other branches do not yield cyclic paths with lower costs. Hence, the algorithm has to explore all other branches, which in the worst case are as many as there are vertices on one layer (*i.e.* $2|E_N| + |V_N|$ many).

While searching for the optimal path, the algorithm only explores paths with cost $d_C < b_{\text{upper}}$ and thus performance can be improved if tighter upper bounds $b_{\text{upper}}$ are found as early as possible. We improve the branch-and-bound algorithm of [26] by computing all paths $C_{\text{all}}$ of a branch, and then search within these for cyclic paths to find lower values of the upper bound $b_{\text{upper}}$ earlier. We want to point out that no additional computational effort is required to compute $C_{\text{all}}$ using the implementation of [26], since all paths are already available (see Fig. 13 for runtime comparisons).

## C. Number of Conjugate Product Edges

As mentioned in the main paper, the conjugate product graph $P$ has 7 times more vertices than $P$ and $c \approx 11$ times more edges. In the following we derive $c$. To this end, we count outgoing edges of each conjugate product vertex (which is sufficient since $P$ is cyclic). Further, we assume that on average each vertex $j$ of the 3D shape $N$ is connected to 6 edges [6]. Thus, each (directed) edge on 3D shape is connected to 5 other directed edges via their shared vertex, see Fig. 12.

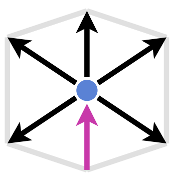


Figure 12. Subset of a triangle mesh. Directed pink edge is connected to all directed black edges via blue vertex.

In conclusion, each conjugate product vertex is connected to 5 conjugate product vertices on the same layer (reflecting *degenerate 2D* conjugate product vertices) and 6 conjugate product vertices on the next layer (reflecting 5 *non-degenerate* conjugate product vertices and 1 *degenerate 3D* conjugate product vertex). In total, each conjugate product vertex is connected to $c \approx 11$ other conjugate product vertices. In combination with the number of vertices of the conjugate product graph $|V| = |V_M| \cdot 2|E_N| + |V_N|$ we obtain the number of edges of $P$.

## D. Runtime

### D.1. Runtime Analysis

In the following we estimate runtime complexity of our branch-and-bound algorithm for conjugate product graphs. To this end, we use $|E_N| \approx 3|V_N|$ [6] to obtain $|V| \approx 7 \cdot |V_M||V_N|$ and $|E| \approx c \cdot 7 \cdot |V_M||V_N|$.

The runtime of Dijkstra on an arbitrary graph $G = (V_G, E_G)$ is $O\big((|E_G| + |V_G|) \log(|V_G|)\big)$ where $(|E_G| + |V_G|)$ indicates the number of update steps to be made, and $\log(|V_G|)$ indicates the complexity to access the priority heap that is used to keep track of the next nodes to be updated.

In our case, the number of update steps is $(|E| + |V|) \approx c \cdot 14 \cdot |V_M||V_N|$ (with $c \approx 11$). We make use of the strictly

**Input :** 2D shape $M = (V_M, E_M)$,
3D shape $N = (V_N, E_N)$
**Output:** Optimal Path $C_{\text{opt}} \subset V$

// First branch is complete first layer
$B_0 \leftarrow \{v = (e^M, e^N) \mid i_0 = 0, e^M = (i_0, i_1)\}$;
// Initialise bounds and branches
$b(B_0) \leftarrow 0$;
$b_{\text{upper}} \leftarrow \infty$;
$B_{\text{Branches}} \leftarrow B_0$;
// Run until no branches with a gap between lower and upper bound exist
**while** $\min_{B \in B_{Branches}} b(B) < b_{upper}$ **do**
  $B \leftarrow \operatorname{argmin}_{B \in B_{\text{Branches}}} b(B)$;
  $B_{\text{Branches}} \leftarrow B_{\text{Branches}} \setminus B$;
  Compute all paths $C_{\text{all}} = \{C_1, C_2, \ldots\}$ with path cost $d_{C_i} < b_{\text{upper}}$ starting and ending in $B$;
  **if** $C_{all} = \emptyset$ **then**
    // No path which meets $d_C < b_{\text{upper}}$
    continue;
  $C \leftarrow \operatorname{argmin}_{C \in C_{\text{all}}} d_C$;
  // Check if current path is cyclic
  **if** $v_1 = v_{|C|}$ **then**
    **if** $d_C < b_{upper}$ **then**
      $b_{\text{upper}} \leftarrow d_C$;
      $C_{\text{opt}} \leftarrow C$;
  **else**
    // Cut current branch into two parts
    **if** $e^N_1 = e^N_{|C|}$ **then**
      $B_1 \leftarrow B \setminus \{v_{|C|}\}$;
      $B_2 \leftarrow \{v_{|C|}\}$;
    **else**
      Compute $B_1, B_2$ as Voronoi cells around $e^N_1$ and $e^N_{|C|}$ respectively;
    // Add new branches
    $B_{\text{Branches}} \leftarrow B_{\text{Branches}} \cup \{B_1, B_2\}$;
    // Update lower bounds
    $b(B_1) = b(B_2) = d_C$;
    // Try to tighten upper bound
    **for** $C \in C_{all}$ **do**
      **if** $v_1 = v_{|C|}$ **then**
        **if** $d_C < b_{upper}$ **then**
          $b_{\text{upper}} \leftarrow d_C$;
          $C_{\text{opt}} \leftarrow C$;

**Algorithm 1:** Branch and Bound for *Cyclic* Shortest Path on (Conjugate) Product Graph

directed order of the $|V_M|$ layers of $P$, which allows to use a heap that scales with the number of vertices of one layer $O(|V_N|)$ (also see [26]). In summary, the runtime complexity of a single Dijkstra run in our conjugate product graph $\mathcal{P}^*$ is $O(|V_M||V_N|\log(|V_N|))$.

To find the optimal cyclic path among all possible cyclic paths, we run Dijkstra not just once but at most $O(|V_N|)$ times (without any branch-and-bound optimisation), which leads to a final runtime complexity of $O(|V_M||V_N|^2\log(|V_N|))$.

### D.2. Runtime Comparison

In Fig. 13, we show the median runtime for the approach by Lähner *et al*. [26] and our approach. The plot shows that both approaches have the same asymptotic behaviour. Due to the use of the *larger* conjugate product graph $\mathcal{P}^*$ in comparison to product graph $\mathcal{P}$ (see also C), our approach takes by a constant factor more time to compute results. For a fair comparison with equal graph sizes, we additionally include computation times of our approach on the product graph $\mathcal{P}$ which shows the improved performance when using Algorithm 1. Nevertheless, we emphasise that our approach (on $\mathcal{P}^*$) still requires polynomial time while being the only one that is able to compute 2D-3D matchings without the need for pre-matching.

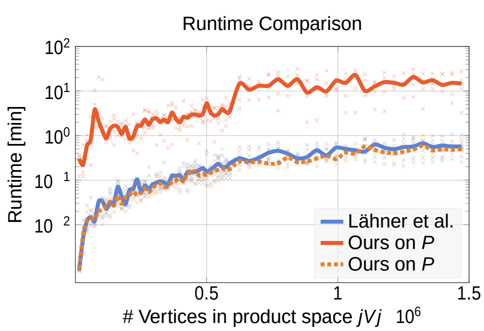



Figure 13. **Runtime comparison** of the approach by Lähner *et al*. [26] and ours. We fix the size of various 3D shapes, gradually increase the number of vertices of respective 2D shapes (by subsampling) and measure times to compute matching results both with our approach and the approach of Lähner *et al*. Results of our approach on the conjugate product graph $\mathcal{P}^*$ are not directly comparable since $\mathcal{P}^*$ contains more vertices and edges than $\mathcal{P}$. For a fair comparison, we include runtimes of our approach on the product graph $\mathcal{P}$ that show the improved performance which Algorithm 1 offers. The x-axis shows the number of vertices $|V|$ of the *product graph* $\mathcal{P}$ (to which the number of vertices $|V^*|$ of the *conjugate product graph* is related to via $|V^*| \approx 7|V|$). The y-axis shows the runtime in minutes. Points (light colours) are individual experiments, while thick lines are median runtimes. Spikes in computation time stem from a varying number of branches needed to compute the optimal path.

## E. 2D to 3D Deformation Transfer

We compute 2D to 3D deformation transfer by applying the following steps:

**2D-3D Matching** We find a matching between 2D and 3D shape using our approach.

**2D Deformation** We deform the 2D shape by using a skeleton which allows for different articulation of arms, legs and head. In combination with biharmonic weights [19,51], we obtain a smooth deformation of the 2D shape (we tessellate the interior of the contour for biharmonic weight computation [35]).

**2D-3D Alignment** We find the optimal alignment $T_{2D}^{3D}$ of 2D shape and matched vertices on 3D shape by introducing a third, constant coordinate for 2D vertices and solving the (orthogonal) Procrustes problem [46].

**3D Deformation** We apply the deformation to the 3D shape by transforming the deformation on the 2D shape using $T_{2D}^{3D}$, applying the transformed deformation to a small subset of 3D vertices (chosen by furthest distance) and using their new positions as a constraint when deforming all other vertices of the 3D shape with the as-rigid-as-possible method of [44].

| Mean Edge Length 2D Shape | AUC ↑ |
|---|---|
| $0.5\,\bar{e}$ | 0.96 |
| $0.75\,\bar{e}$ | 0.97 |
| $1\,\bar{e}$ | **0.98** |
| $1.25\,\bar{e}$ | 0.97 |
| $1.5\,\bar{e}$ | 0.95 |

Table 4. Ablation study on the sensitivity of our approach to **different discretisations**. The score is the area under the curve (AUC) of the cumulative segmentation errors. We fix the discretisation of 3D shape and vary edge lengths of 2D shape. $\bar{e}$ depicts the mean edge length of 3D shape.

## F. Ablation: Discretisation

In Tab. 4 we evaluate the robustness of our method w.r.t. to different discretisations. For all our experiments in the main paper we reduce influence of discretisation by decimating 3D shapes to half of their original resolution, which results in more uniform edge lengths [19]. Additionally, we re-sample 2D shapes with edge lengths according to the mean edge length of the decimated 3D shape.

## G. Qualitative Results on FAUST

In Fig. 14 we show additional qualititve results.