Shape reconstruction with intrinsic priors

Yohai S. Devir Guy Rosman Alexander M. Bronstein
Michael M. Bronstein
Ron Kimmel
{yd|rosman|bron|mbron|ron}@cs.technion.ac.il
Department of Computer Science
Technion - Israel Institute of Technology

March 1, 2009

Abstract

Shape-from-X, a problem of shape reconstruction from some measurements, is a classical inverse problem in computer vision. In this paper, we propose a framework for intrinsic regularization of such problems. The assumption is that we have a shape intrinsically similar to (a bending of) the unknown shape we have to reconstruct. For that goal, we formulate a variation with respect to vertex coordinates of a triangulated-mesh approximating the continuous shape. The numerical core of the proposed method is based on differentiating the fast marching update step for geodesic distance computation.

Keywords: Fast marching, surfaces, geodesic distances, shape reconstruction, isometry

1 Introduction

In many vision tasks, both human and computer vision, one tries to deduce the shape of an object given an observation or a measurement thereof. For example, in shadow theater (Fig. 1) we recognize shapes of objects from silhouettes projected on a screen. Another reconstruction task is performed by our brain al-

Figure 1: Shadow theater example - a shadow of a bird and a camel being cast by two hands.

most unnoticeably when deducing the shape of a three-dimensional (3D) object from a pair of two-dimensional (2D) images of the object observed by our two eyes. In computer vision, these two tasks are known correspondingly as *shape from silhouette* and *shape from stereo*. Other computer vision tasks involving deducing the shape of a 3D object from other kinds of measurements include *shape from motion*, where the measurement is a sequence of still images; *shape from shading*, were the measurement is a single image of the object illuminated by a light source; and others. All these problems in which an object is reconstructed based on some measurement are known as *shape reconstruction problems*. They constitute a subset of what is called *inverse problems*.

A common aspect of such problems is that in many cases they are underdetermined, in the sense that measuring different objects may yield similar measurements. For example, the essence of the shadow theater is that it is hard to distinguish between shadows cast by an animal and shadows cast by hands. If a prior knowledge about the unknown object is given (e.g. the shadow is cast by an animal in conjunction with a set of possible animals) we have better chances reconstructing the object. Such additional information is known as a *prior*.

Of particular interest are reconstruction problems involving non-rigid shapes. The world surrounding us is full with objects such as live bodies, paper products, plants, clothes etc. Every such object may be deformed to an infinite number of different postures. While bending, though, objects tends to preserve their internal geometric structure. Two objects differing by a bending are said to be *intrinsically*

similar. In many cases, while we do not know the measured object, we have a prior on its intrinsic geometry. For example, in the shadow theater, though we do not know which exact posture of the hand casts the shadow, we know that the unknown object is a deformation of a hand. In other words, our prior is a shape of the hand, and the variability of the reconstructed object is the bending of the hand.

There are many approaches to specific reconstruction of shapes-from-X. Among those are the following. *Shape from shading (SFS)* [29] where a 3D shape is recovered from one or more 2D images, utilizing the variations of shading in the image. The reconstruction is usually done by minimizing some cost functional or iteratively reconstruct the object from known initialization points. Among common priors to SFS are smoothness of the surface, brightness, similarity between intensity gradients of the image extracted from the input image, symmetry or a combination of a few of them. *Bundle adjustment* [25] where a 3D shape is reconstructed given a set of 2D images of the shape from different viewpoints taken by possibly non-calibrated cameras. *pose transplantation (shape completion)* where a full shape is constructed given only the the spatial locations of a few of its points. This is done, for example, when creating an animated character mimicking the motion of a human actor by placing a few markers on his body, and moving the animated character accordingly. In [1], a full model of various bodies and postures is used for this task. Here we use a single static model of a specific actor. *Shape preserving denoising* where a partially smooth shape is recovered from a shape with high frequency geometric components. Algorithms that eliminate high frequency geometric noise by smoothing a noisy surface such as [10, 16] do not incorporate knowledge about the geometry of the surface thus may distort its shape. Maintaining the intrinsic similarity between the noisy surface and the smoothed one can limit the shape's distortion.

Due to the under-determined nature of shape-from-X applications, knowledge about the intrinsic structure of the shape may improve the result of existing methods. In particular, by using a functional measuring intrinsic similarity, the knowledge about the intrinsic structure of the shape may be incorporated in any of the many methods of solving shape from X problems that minimize some functional. Certain functionals can be minimized by solving the resulting Euler-Lagrange equations, possibly by a gradient descent process.

In order to utilize the intrinsic similarity of a shape as a prior we need to define a functional describing the distortion between two shapes. A few papers such as [14] and [19] computes the *local* distortion of the mesh edges and therefore unsuitable to compute a deformation that affects the *global* distortion of the mesh. The scale and location at which the geodesic distances are preserved can play

an important role in some applications, while restriction to preserving length of edges between neighboring vertices may integrate to undesired deformations at larger scales.

One of the first attempts to measure intrinsic similarity between arbitrary surfaces was made by Elad and Kimmel in [15] who mapped the metric structure of the shapes to a low dimensional Euclidean space using multidimensional scaling (MDS) and compare the resulting shapes using extrinsic methods. Yet, low dimensional Euclidean embedding allows representing the intrinsic geometry of the shapes only approximately. This problem can be solved by using the Gromov-Hausdorff (GH) distance [18] as first suggested by Mémoli and Sapiro [22]. In [6] Bronstein et al. proposed a method formulating the GH distance using numerical scheme similar to MDS (dubbed generalized MDS or GMDS) [7], in which one surface is embedded into another while reducing the intrinsic dissimilarity between the embedded surface and the embedding one. Since this method limits one surface to be embedded into another, it cannot be adopted for finding a shape that is embedded in $\mathbb{R}^3$.

In [5] Bronstein et al. show a method to avoid this limitation by deriving a combination of intrinsic and extrinsic similarities criteria. This problem can be solved as a particular case of the inverse problems considered here, where the measurement space is the set of extrinsic coordinates of the points in $\mathbb{R}^3$ and the distance on the measurement space is the Hausdorff distance between the coordinates of the shapes. Since their method is based on calculating the geodesic distances on a shape using Dijkstra's algorithm, it is sensitive to the shape triangulation and is affected by the metrication error as explained in Sec. 3.1.

In this paper we show a method less sensitive to triangulation changes and that converges to the exact measure of intrinsic dissimilarity. The numerical core of our method is based on an extension to the *fast marching algorithm* that enables us to calculate the derivatives of pairwise geodesic distances w.r.t. the mesh's vertices spatial locations.

A different approach to metric change as was in [2],[24],[28] in which a conformal equivalent metric satisfying a given conditions is calculated on a given mesh. Those methods can be used for measuring the intrinsic dissimilarity between shapes by cascading transformations from one shape to some embedding and from the embedding to the second shape, such as in [21]. In contrast to our method, the embedding of the mesh in those methods is fixed, while the limit of the generated metric for short paths is not the Euclidean embedding distance.

The rest of the paper is organized as follows: Section 2 introduces the mathematical background. Section 3 reviews the methods of calculating geodesic dis-

tances. Section 4 presents the proposed reconstruction framework. Section 5 demonstrates the proposed method applied to some examples. Finally, we provide concluding remarks in Section 6.

2 Mathematical background

2.1 Shape space

We model a shape X as a *metric space* - a pair (X, d_X) where X is a 2D smooth, compact, connected and complete Riemannian manifold embedded into the Euclidean space $\mathbb{R}^3$, and $d_X : X \times X \to \mathbb{R}^+ \cup \{0\}$ is the geodesic metric. We denote by $\mathbb{X}$ the shape space in which each element is a shape.

Given two shapes $W, X \in \mathbb{X}$, we define the *correspondence* between them as $C \subset X \times W$ such that $\forall x \in X \exists w \in W : (x, w) \in C$ and $\forall w \in W \exists x \in X : (x, w) \in C$. We say $x \in X$ and $w \in W$ are in correspondence if $(x, w) \in C$. The *distortion* of a correspondence is given by

$$\operatorname{dis} C = \max_{\substack{(x,w)\in C \\ (x',w')\in C}} |d_X(x, x') - d_W(w, w')|$$

which measures how different the corresponding metric structures in X and W are.

A subset $X' \subseteq X$ is said to be an *ε-net* in (or *ε-covering* of) X if

$$\max_{x\in X} \min_{x'\in X'} d_X(x, x') \leq \varepsilon.$$

If a correspondence C has $\operatorname{dis} C \leq \varepsilon$ and the set of all points in X having correspondence in W, $X' = \{x : \exists w, (x, w) \in C\}$ is an ε-net, then X and W are *ε-isometric*.

The similarity of two shapes in $\mathbb{X}$ is measured using a metric $d_{\mathbb{X}}$. Since the elements of $\mathbb{X}$ are metric spaces we can use the Gromov-Hausdorff [18] distance as $d_{\mathbb{X}}$,

$$d_{\mathrm{GH}} = \min_C \operatorname{dis} C \tag{1}$$

If $d_{\mathrm{GH}}(X, W) \leq \varepsilon$, then X and W are 2ε-isometric, and if X and W are ε-isometric, $d_{\mathrm{GH}}(X, W) < 2\varepsilon$. In particular, $d_{\mathrm{GH}}(X, W) = 0$ iff X and W are isometric.

When dealing with sampled shapes, it is important to bear in mind that most polyhedral shapes (including triangulated meshes) without boundaries are in fact rigid, i.e. do not have incongruent isometries. This was conjectured by Euler, proven for convex shapes by Cauchy [9], proven that almost all such shapes are rigid by Gluck [17], though disproved by Connelly [11]. As a result, most closed sampled shapes originated from computer models are rigid, so two deformations may be only approximate isometries. Moreover, objects that in the real world are nonrigid, such as palm's skin, are isometric due to small wrinkles near the joints. Triangulated meshes approximating two deformations of such object with limited resolution will fail to capture and represent those wrinkles. As a result, two meshes approximating two isometric deformations of a non-rigid shape may get further from being isometric.

2.2 Measurement space

Let $\mathbb{Y}$ be a space, and $d_{\mathbb{Y}} : \mathbb{Y} \times \mathbb{Y} \to \mathbb{R}^+ \cup \{0\}$ be a distance on $\mathbb{Y}$. Next, let P be an operator $P : \mathbb{X} \to \mathbb{Y}$. For example, in the SFS inverse problem, $\mathbb{Y}$ is the space of gray scale images, P is an operator assigning a patch on the shape to a pixel in the image whose intensity is determined by the average dot product between the surface normal at a point in this patch, and the illumination direction. Another example, of shape completion, $Y = P(X), X \in \mathbb{X}$ is a sparse set of points on X, $d_{\mathbb{Y}}$ may be the *Hausdorff distance*,

$$d_{\mathrm{H}}(Y, Z) = \max\{\max_{z \in Z} d_{\mathbb{R}^3}(z, Y), \max_{y \in Y} d_{\mathbb{R}^3}(y, Z)\}, \tag{2}$$

where $d_{\mathbb{R}^3}(x, Y) = \min_{y \in Y} \|x - y\|$ denotes the minimal Euclidean distance between a point $y \in Y$ and the point z.

2.3 Reconstruction problems

We formulate the generic problem of shape reconstruction as follows. Given $Y = P(\tilde{X})$ where $\tilde{X}$ is an unknown shape, recover a shape $\hat{X} \approx \tilde{X}$ that best fits the given data

$$\hat{X} = \operatorname*{argmin}_{X \in \mathbb{X}} d_{\mathbb{Y}}(P(X), Y). \tag{3}$$

In the general case where P is not injective and does not preserve distances, this problem is ill-posed, and it is possible to find many solutions for which

$d_{\mathbb{Y}}(P(\hat{X}), Y)$ is small, while $d_{\mathbb{X}}(\hat{X}, \tilde{X})$ is large. Moreover, in some cases, even if there is a single solution $\hat{X}$ such that $d_{\mathbb{Y}}(P(\hat{X}), Y)$ is small, it is not always known how to find it.

Therefore, it is common practice in inverse problems to solve a different problem in which aside from $Y = P(\tilde{X})$, a prior shape X_0 is also given, such that $d_{\mathbb{X}}(\tilde{X}, X_0)$ is small while $d_{\mathbb{Y}}(Y, P(X_0))$ may be large. Using the prior shape as a regularization in Eq. 3, we solve the following problem

$$\hat{X} = \underset{X \in \mathbb{X}}{\operatorname{argmin}}\, d_{\mathbb{X}}(X, X_0) + \lambda d_{\mathbb{Y}}(P(X), Y), \tag{4}$$

where $\lambda > 0$ is some parameter.

We look for a shape X which is intrinsically similar to the prior X_0 and whose measurement is similar to Y. Our way of solving Problem (4) consists of deforming an initial shape X and evaluating the result of this deformation on $d_{\mathbb{X}}(X, X_0$ and $d_{\mathbb{Y}}(P(X), Y)$.

3 Geodesic distance computation

Having the surface sampled at N points $\{x_i\}_{i=1}^N$, whose coordinate vectors are correspondingly denoted by $\{\overline{x}_i\}_{i=1}^N$, a *triangulated mesh* can be defined with the surface's samples as vertices.

Computation of geodesic distances is a key step in the calculation of intrinsic similarity. The straightforward method of approximating geodesic distances over a triangulated mesh is to consider the mesh as a weighted chordal graph (V, E) for which the weight of an edge is the Euclidean distance between its ends. Next, define the geodesic distance between two vertices as the shortest path connecting those vertices. This is commonly done using Dijkstra's algorithm [12].

Sethian [23] and Tsitsiklis [26] introduced methods for approximating weighted distance maps on regularly sampled domains with a weighted flat metric. Sethian coined the term *fast marching method* to his algorithm. By interpreting the geometry behind the numerical update step of these solvers, Kimmel and Sethian [20] extended the fast matching method to triangulated domains.

The *geodesic distance map* from a source point $x_s \in X$ is defined as $d_{x_s} : X \to \mathbb{R}^+$ such that for each $x \in X$ we have $d_{x_s}(x) = d_X(x_s, x)$. On a triangulated surface, approximating a continuous one, the geodesic distance map can be computed by solving an *eikonal equation* on the surface. The eikonal equation captures the fact that the distance map can be defined by its gradient whose magnitude equals one almost everywhere.

The fast marching algorithm estimates the geodesic distances from a given source vertex x_s by simulating a front starting at the source and propagating with a constant unit velocity. The front arrives to a vertex x at time $T(x)$ ($T(x_s) = 0$) indicating that the distance between x and the source vertex is $T(x) = d_X(x_s, x)$.

The FMM algorithm is similar to Dijkstra's algorithm, except for its *update* step. Unlike Dijkstra's algorithm in which a distance is assigned to a vertex based on a single neighboring vertex and the connecting edge, the FMM algorithm assigns a distance to a vertex based on the distance values assigned to two of its adjacent neighbors and the triangle containing the three vertices. The values assigned to the three corners define a function on the triangle whose gradient magnitude is equal to one. For a recent description and numerical analysis of the update step modified for a parallel processor we refer to [27].

Let $\{a, b, c\}$ be the edges of the *acute* triangle $\{x_1, x_2, x_0\}$. Given the distance values t_1 and t_2, the FMM *update* step estimates t_0 by

$$\begin{aligned} t_0 &= \mathcal{F}_{\text{FMM}}(t_1, t_2, x_0, x_1, x_2) \\ &= \begin{cases} t_1 + \frac{rv}{c} + \frac{s}{c}\sqrt{c^2 - v^2} & \text{some condition} \\ & \text{holds} \\ \min\{t_1 + b, t_2 + a\} & \text{otherwise} \end{cases} \end{aligned} \tag{5}$$

where $v = t_2 - t_1$, $r = \frac{b^2+c^2-a^2}{2c}$ and $s = \sqrt{b^2 - r^2}$. It is basically a solution to the quadratic equation that restricts the gradient to be one.

In the case of *obtuse* triangles, using Eq. 5 may lead to solutions which do not satisfy the consistency ($t_0 > \max\{t_1, t_2\}$) and monotonicity (increasing t_1 or t_2 increases t_0) conditions.

Kimmel and Sethian suggested in [20] to handle obtuse angles by splitting these angles in order to form two new acute ones (referred as *virtual triangles*). The splitting is done by unfolding the triangle opposing the obtuse angle to the $x_0x_1x_2$ plane. We denote this triangle by $x_1x_2x_3$ and the location of x_3 in the $x_0x_1x_2$ plane by $\hat{x}_3$. If the line between x_0 and $\hat{x}_3$ splits the obtuse angle to two acute ones, t_0 is calculated using Eq. 5 in the triangles $x_0x_1\hat{x}_3$ or $x_0x_2\hat{x}_3$. If the $x_0\hat{x}_3$ line splits the obtuse angle to an obtuse and an acute angle, this process is repeated recursively to a triangle adjacent to triangle $x_1x_2x_3$ until a vertex that splits the obtuse angle to two acute ones is found. For more details see Appendix B.

3.1 Dijkstra vs. FMM

Approximating geodesic distances using Dijkstra's algorithm has two drawbacks. First, given two ε-isometric shapes X and Y, the distances measured between corresponding locations on X and Y using FMM may defer by at most $\varepsilon + \delta$, where δ depends on the sampling and can be arbitrarily small as $N \to \infty$. Using Dijkstra, δ cannot be arbitrarily small since the graph induces a metric which is inconsistent with that of the underlying continuous surfaces. This phenomena is known as *metrication error*. The effect of this drawback on the similarity measure is demonstrated in Fig. 2.

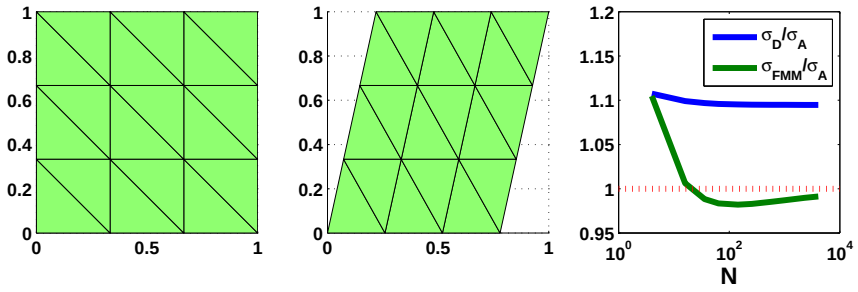

Figure 2: The accuracy of measuring the distortions of pairwise distances using Dijkstra's algorithm and FMM. A unit square (left) sampled uniformly in various resolutions (N is the number of samples) is sheered (center). The Frobenius norm of the difference between the matrix of pairwise distances in the original surface and the matrix of pairwise distances in the sheered square is denoted by σ. This norm was calculated using FMM (σ_{FMM}), Dijkstra (σ_D) and analytically (σ_A). Right: The ratios of σ_{FMM}/σ_A and σ_D/σ_A calculated in various resolutions.

In particular, Dijkstra's algorithm is sensitive to the triangulation. Distances measured on a shape with one triangulation may significantly differ from those measured on the same shape triangulated differently. Therefore, two shapes formed from the same set of samples and two different triangulations may seem dissimilar when calculating the stress using Dijkstra's algorithm and nearly isometric when calculated using FMM.

4 Implementation Considerations

Since ε-isometry is sensitive to outliers, we use the following notion of intrinsic similarity known as *stress*. Given two surfaces W and X, both with N samples, we define the stress $\sigma(W, X)$ as

$$\sigma\left(X, W\right) = \min_{C} \sum_{\substack{j,k \\ (x_j, w_j)\in C \\ (x_k, w_k)\in C}} \left(d_X\left(x_j, x_k\right) - d_W\left(w_j, w_k\right)\right)^2, \tag{6}$$

where C is a correspondence between X and W.

Using σ as a version of the intrinsic dissimilarity, we solve the following version of Eq. 4

$$\hat{X} = \underset{X\in\mathbb{X}}{\operatorname{argmin}}\, \sigma(X, X_0) + \lambda d_{\mathbb{Y}}(P(X), Y). \tag{7}$$

This problem can be solved as an optimization problem where the optimization variables are the coordinates of the vertices of X. The initialization is the prior X_0 unless a better guess is available. In order to find $\hat{X}$ of Eq. 7, we need to differentiate both $d_{\mathbb{Y}}(P(X), Y_0)$ and $\sigma(X, X_0)$ w.r.t. the coordinates of the samples of X. Calculating the gradient of $d_{\mathbb{Y}}(P(X), Y_0)$ for a few operators P is described in the Appendix C;

In [5], the gradient of $\sigma(X, X_0)$ is derived from the calculation of $\sigma(X, X_0)$ using Dijkstra's algorithm. Here, we calculate a less trivial yet more accurate gradient that is based on FMM. Vertex-wise, the optimization has the form of

$$\overline{x}_i \leftarrow \overline{x}_i + \overline{\alpha}_i, \tag{8}$$

where $\overline{\alpha}_i = (\alpha_i^x, \alpha_i^y, \alpha_i^z)$ is the distance x_i moves in each coordinate direction.

We denote by α the matrix in which $(\alpha)_{i,j} = \alpha_i^j$. Since the analysis is similar for all coordinates, we omit the superscript index for notation simplicity.

4.1 Gradient of the stress

Given two shapes X and X_0, both with N samples and a correspondence C, the stress between X and X_0 can be written as

$$\sigma\left(X, X_0\right) = \sum_{\substack{j,k \\ (x_j, x_{0j})\in C \\ (x_k, x_{0k})\in C}} \left(d_X\left(x_j, x_k\right) - d_{X_0}\left(x_{0j}, x_{0k}\right)\right)^2.$$

The pairwise distances $d_X(x_j, x_k)$ must be recomputed after every iteration of the optimization by running the FMM algorithm N times, one for each vertex as a source. The pairwise distances in X_0, $d_{X_0}(x_j, x_k)$ are pre-computed once calculated in a similar way only once and can be considered as an input to the optimization problem.

Differentiating the stress w.r.t. α_i, we obtain

$$\frac{\partial \sigma\left(X, X_0\right)}{\partial \alpha_i} = 2 \sum_{j,k} \left(d_X\left(x_j, x_k\right) - d_{X_0}\left(x_{0j}, x_{0k}\right)\right) \cdot \frac{\partial d_X\left(x_j, x_k\right)}{\partial \alpha_i} \tag{9}$$

Consider a vertex x_0, and a source vertex x_s. We denote the triangle which is used for the final calculation of the distance between x_s and x_0 as $x_0x_1x_2$. Given the spatial locations of the triangle vertices $\{\overline{x}_0, \overline{x}_1, \overline{x}_2\}$. Also given the estimated distances of x_1 and x_2 from the source $(d_X(x_1, x_s), d_X(x_2, x_s))$, and the derivatives of these estimated distances w.r.t. the movement of each vertex $\left(\frac{\partial d_X(x_1,x_s)}{\partial \alpha_i}, \frac{\partial d_X(x_2,x_s)}{\partial \alpha_i}\right)$. The derivatives of the distance between x_0 and x_s, denoted by $\frac{\partial d_X(x_0,x_s)}{\partial \alpha_i}$, can be calculated by differentiating Eq. 5 using the chain rule.

$$\begin{aligned} \nabla_\alpha d_X(x_0, x_s) = \\ D\mathcal{F}_{\text{FMM}}\left(\nabla_\alpha d_X(x_1, x_s), \nabla_\alpha d_X(x_2, x_s),\right. \\ \left. d_X(x_1, x_s), d_X(x_2, x_s), \overline{x}_0, \overline{x}_1, \overline{x}_2\right). \end{aligned} \tag{10}$$

Details of deriving the *update* step are described in Appendix A.

We note that the derivatives of the triangle edges' length w.r.t. $\{\overline{x}_i\}$ are calculated only once for all source vertices.

If the angle $\angle\overline{x}_1\overline{x}_0\overline{x}_2$ is obtuse, Eq. 10 can be adapted by splitting $\angle\overline{x}_1\overline{x}_0\overline{x}_2$ as explained in Sec. 3. The derivatives of the distances between $\hat{x}_3$ and $\{x_1, x_2, x_0\}$ w.r.t. α_i are calculated using the chain rule along the process of the unfolding. See Appendix B for details. Consequently, the derivatives of $d_X(x_0, x_s)$ w.r.t. α_i are calculated according to Eq. 10.

In order to calculate all derivatives of all pairwise distances, we propose the following modification of the FMM algorithm. At the initialization step, we also initialize $\frac{\partial d_X(x_s,x_s)}{\partial \alpha_l} = 0 \, \forall x_l \in X$. In every FMM iteration, we calculate the derivatives $\frac{\partial d_X(x_s,x_i)}{\partial \alpha_l}$ immediately after calculating $d_X(x_s, x_i)$ at the same triangle containing x_i used for calculating $d_X(x_s, x_i)$.

Although the space complexity of storing $\nabla_\alpha d_X$ is $\mathcal{O}\left(N^3\right)$, in uniformly sampled non-pathological surfaces there are $\mathcal{O}\left(N^{1/2}\right)$ vertices that affect $d_X(x_i, x_s)$

for a give vertex x_i. Therefore, the space complexity of storing *the non-zero entries* of $\nabla_\alpha d_X$ is $\mathcal{O}\left(N^{5/2}\right)$.

Moreover, by updating $\nabla_\alpha \sigma(X, X_0)$ on the fly, we need to store only the derivatives that may be used for future calculations. These are the derivatives of $d_X(x, x_s)$ for vertices x whose distance from x_s was already set (in some FMM related papers such vertices are denoted by *black* or *fixed* vertices) and $\exists x_j \in \mathcal{N}(x)$ whose distance from x_s was not set (using the same terminology, this vertex is denoted as a *red* vertex). Other derivatives of distances may be discarded.

In non-pathological surfaces the space complexity of storing the needed derivatives in any step of the algorithm is $\mathcal{O}(N))$. This, however, does not reduce the time complexity which is $\mathcal{O}\left(N^{5/2}\log(N)\right)$. For practical reasons the following adjustments are made.

4.2 Path fixing

The phenomena of vertex x_0 for which there are two different shortest paths connecting to the source is called *a shock*. The gradient of the distance function along a shock is not well defined. If $d(x_0, x_s)$ is calculated in triangle $x_0x_1x_2$, increasing t_1 or t_2 may not lead to an increase of t_0, since t_0 may be updated through another path and another triangle. This means that the stress between X and Y as a function of the coordinations of the vertices of X is not always a differentiable function. However, the stress calculated using a set of fixed paths connecting every two vertices is always a differentiable function. Therefore, we calculate a pairwise distance map as a reference and then keep the order of the updates fixed.

A shock which is formed inside a single triangle (x_0 is updated by $\min(t_1 + b, t_2 + a)$ is treated as explained in Section 4.4.

This update process has two desirable properties. Assuming that at a certain stage of an optimization process, vertex x is moved away from it's neighbor vertices such that its mean Euclidean distance to it's neighboring vertices is much bigger than the average distance between its neighboring vertices. We denote this detail of a shape as *a spike*. Let also assume that this spike did not exist in the reference surface used for the stress calculation. If the paths are not fixed, vertex x will affect only the distances from x to the rest of the mesh, thus the projection of the gradient of the stress over the vector describing a motion of this vertex alone will be relatively small, causing this vertex to remain a spike. At the other end, all pairwise distances that used to be affected by x are affected by x's neighboring vertices. Therefore, every movement of these vertices will cause a relatively large change in the stress. This will cause these vertices to be kept in place preserv-

ing the spike and forming a local minima of the cost function. Fixing the paths eliminates this phenomena.

Another effect of fixing the paths is a runtime reduction, since the order of updating the vertices is known and there is no need to find the unset vertex whose distance from the source is minimal.

4.3 Smoothing the transition between calculations

$d_X(x_0, x_s)$ may be calculated according to triangle $x_0x_1x_2$ in a couple of different ways. If $\angle x_1x_0x_2$ is acute, it will be updated using Eq. 5 in triangle $x_0x_1x_2$. If $\angle x_1x_0x_2$ is obtuse, it will be updated using Eq. 5 in the virtual triangle $x_0x_1\hat{x}_3$ or $x_0x_2\hat{x}_3$. Moreover, since $\angle x_1x_0\hat{x}_3$ and $\angle x_2x_0\hat{x}_3$ must be non-obtuse, $\hat{x}_3$ may not be the same vertex at different optimization steps.

As a consequence, a small perturbation in the mesh may cause a change in the calculation resulting a relatively large change in the distances calculations, and therefore a discontinuity of the stress. This happens in the following scenarios.

- **The updating triangle is changing from obtuse to acute or vice-versa.** Assuming a vertex x_0 is updated from triangle $x_0x_1x_2$. If $\angle x_1x_0x_2$ is about $90°$, a small perturbation of $\overline{x}_0, \overline{x}_1$ or $\overline{x}_2$ in the updated triangle could cause the update scheme to change from a calculation in the $x_0x_1x_2$ triangle to calculation in the $x_0x_1\hat{x}_3$ and $x_0x_2\hat{x}_3$ triangles. This scenario is demonstrated in Fig. 3 (Left).

- **Obtuse triangle splitting is about to change.** (Fig. 3, middle). Assuming a vertex x_0 is updated from an obtuse triangle $x_0x_1x_2$ ($\angle x_1x_0x_2$ is obtuse). The line $\hat{x}_3x_0$ should split $\angle x_1x_0x_2$ to two acute angles which are smaller than $\angle x_1x_0x_2$. If due to a small perturbation $\hat{x}_3x_0$ will not split $\angle x_1x_0x_2$ as required, $\angle x_1x_0x_2$ will be split by connecting it to a different vertex, $\hat{x}_4$. In Fig. 3 (middle), this happens when $\angle x_2x_0\hat{x}_3$ is almost right. It should be noted that if $\angle x_1x_0x_2 \approx \pi$, both $\angle x_2x_0\hat{x}_3$ and $\angle x_1x_0\hat{x}_3$ are almost right and therefore, a small perturbation may change the virtual connectivity between three scenarios.

- **A combination of more than one change of calculation.** A small perturbation of the location of $\overline{x}_i$ may change more than one calculation. For example, as seen in Fig. 3 (right), $\angle x_2x_0x_1$ is almost right, and a small movement of $\hat{x}_3$ in the $-y$ direction will cause that $\angle\hat{x}_3x_0x_1$ to be larger

than $\angle x_2 x_0 x_1$, therefore changing the virtual triangles such that x_0 will be connected to $\hat{x}_4$. Note that $\angle \hat{x}_3 x_0 x_1$ may be still acute in such a scenario.

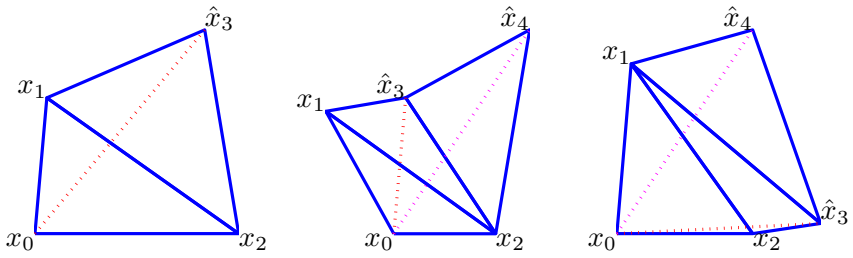

Figure 3: The cases in which the method of calculation is changing. In all images - updating x_0 comes from $x_0 x_1 x_2$ triangle. Left: The angle of the update triangle changes from acute to obtuse. Middle: A change of the virtual triangles. Right: A combination of the scenarios on the left and the middle.

We denote a mesh in which a small perturbation *may* change the way a distance is calculated as a *nonstable mesh*. We also denote the calculation that is performed assuming that some angle's type will be changed from an acute to obtuse or vice versa as a *speculative calculation*, and the mesh in which the said angle has changed is denoted as *speculative mesh*.

The discontinuities are treated by using a weighted average between the current scenario and the speculative scenario assuming an angle affecting the vertex's distance calculation is turned obtuse. We define an angle θ as *almost obtuse* if $\cos(\theta) < L_1$ for some small constant L_1 (we use $L_1 = 0.05$). We calculate a distance using speculative scenarios if at least on of the angles affecting the current calculation is nearly obtuse. The weight we between current calculation and the speculative ones, denoted by h, is calculated by

$$h(V, L_0, L_1) = \frac{maxH}{2}\left(1 + \sin\left(\frac{\pi}{2}\frac{2V - L_1 - L_0}{L_1 - L_0}\right)\right), \tag{11}$$

where $V = \cos(\theta)$, and L_0 is 0 for transitions in non-virtual triangles (the first scenario) and the maximum between 0 and cosine of the split triangle angle (that

may be almost obtuse) when changing from one splitting of the triangle to a different one (at the second scenario). $maxH$ is $h(\frac{\theta}{2}, L_0, L_1)$ if both $\angle\hat{x}_3x_0x_1$ and $\angle\hat{x}_3x_0x_2$ are almost obtuse and 1 otherwise.

In some cases the speculative calculation has other angle which is almost obtuse and therefore the speculative distance is recursively calculated as another weighted average. In the case of a nearly straight angle, both $\angle\hat{x}_3x_0x_1$ and $\angle\hat{x}_3x_0x_2$ may change to be obtuse. Therefore, the speculative calculation is a weighted average of the speculative calculation assuming $\angle\hat{x}_3x_0x_2$ is about to become obtuse and the speculative calculation assuming $\angle\hat{x}_3x_0x_1$ is about to become obtuse. The weight we use between this two speculative calculations is $h(\angle\hat{x}_3x_0x_1, 0, \cos\frac{\angle x_2x_0x_1}{2})$.

Theoretically, since a speculative calculation may be based on another speculative calculation, there can be a $\mathcal{O}(2^N)$ speculative calculations where N is the number of vertices in the mesh. However, for small values of L_1 in non-pathological meshes, there are almost always less than 2^5 speculative calculations.

Derivatives of h are calculated by the chain rule using the derivatives of the edges forming the triangles affecting the weighted sum.

4.4 Eliminating shocks within a triangle

In case that a vertex is updated using Dijkstra scheme (the second case in Eq. 5), a shock may be formed when $t_1 + b = t_2 + a$. This shock causes meshes for which the derivative of the distance function does not exist. We eliminate this problems by calculating the t_0 in those cases by

$$t_0 = \begin{cases} \min\{t_1 + b, t_2 + a\} & \|(t_1 + b) - (t_2 + a)\| > \varepsilon \\ h(t_2 + a) + (1 - h)(t_1 + b) & \text{otherwise} \end{cases} \tag{12}$$

where ε is some small constant and h is calculated using Eq. 11 using $V = (t_1 + b) - (t_2 + a)$, $L_0 = -\epsilon$ and $L_1 = \epsilon$.

4.5 Limiting the paths affecting the stress

In some applications, making local patches intrinsically similar to the corresponding local patches in a second shape is more important than keeping all pairwise distances similar. However, due to the definition of the stress a relatively small distortion in a single long pairwise distance has more effect on the final result than a few relatively large distortions in short pairwise distances. Therefore we give more weight to local pairwise distances.

Ignoring completely the relatively long pairwise distances also reduces the runtime complexity. This is done by calculating the stress as

$$\sigma\left(X, X_0\right) = \sum_{\substack{j,k \\ (x_j, x_{0j}) \in C \\ (x_k, x_{0k}) \in C \\ d_{X_0}(x_{0j}, x_{0k}) \leq K}} \left(d_X\left(x_j, x_k\right) - d_{X_0}\left(x_{0j}, x_{0k}\right)\right)^2. \tag{13}$$

for some constant K and the correspondence C.

Since the paths are fixed, K that is smaller than the length of the longest edge in X_0 may lead to infinite distances and should be avoided.

4.6 Convergence speedup

In general, we use gradient descent optimization with line search in order to solve Eq. 7. In order to speed up the convergence we use the following techniques.

Vector extrapolation After every twelfths iteration, we estimate the limit of the series formed by the last twelve shapes. We do this using the MPE algorithm [8]. The extrapolated limit of the series is then taken as a descent direction and a line search is done along this direction.

Alternating minimization Since the optimization is ill-conditioned, an alternating optimization was performed, alternating between a set of vertices whose gradient's norm is more than 3 standard deviations from the mean norm of all vertices, and the set of the rest of the vertices. The two sets were updated every six iterations.

multi-resolution optimization Unless stated otherwise, the result is calculated in a coarse to fine sequence, where a coarser model is a subsampling of a finer model. After solving the problem on a coarse model, a finer model is extrapolated from the coarse result and is used as an initialization for the optimization of a finer model.

5 Results

Most of the previous papers dealing with derivation of the metric of a surface have used the gradient of the metric for comparing different shapes. In this section we

show a few examples of using the gradient of the metric for synthesizing a shape from a measurement given a prior of a nonrigid deformation of that shape. The dogs and person's models were taken from TOSCA database [4].

5.1 Shape-preserving denoising

In this example, two corresponding models of a palm are used. One model (denoted by X_p) is used as a prior while a random noise of normally distributed magnitude and uniformly distributed directions was added to every vertex of the second model (denoted by X_n). The cost function used in this example is

$$\text{cost}(X) = \sigma(X, X_p) + \lambda_1 d_{\text{ext}}(X, X_n) + \lambda_2 D_{\text{vol}}(X, X_p, r_v)$$

using $r_v = 0.05 \cdot \text{vol}(X_p)$, $\lambda_1 = 1.1 \cdot 10^{-4}$ and $\lambda_2 = 1.1 \cdot 10^{-1}$. The stress was calculated using only pairs of points whose geodesic distance measured on the prior is at most K, where K was chosen arbitrarily to 10 (for comparison, the shape's diameter is 58).

The results are shown in Figure. 4. The optimization was done in four resolutions of 331, 826, 1652 and 3304 vertices, with 200, 200, 200 and 500 iterations at each resolution correspondingly. The convergence is shown in Fig. 5.

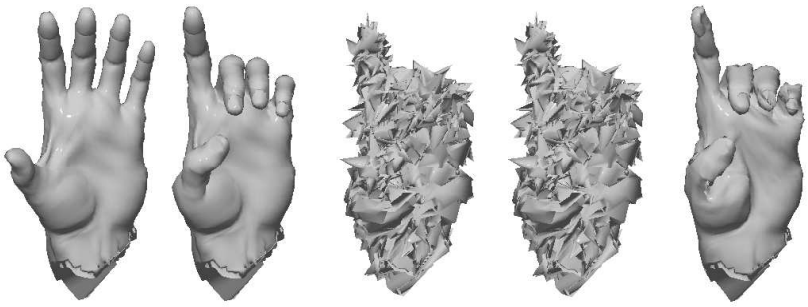

Figure 4: Shape denoising. From left to right: the given prior, the unknown shape, the measurement - a noisy shape, denoising using the prior's volume only, denoising using the prior's intrinsic geometry and volume.

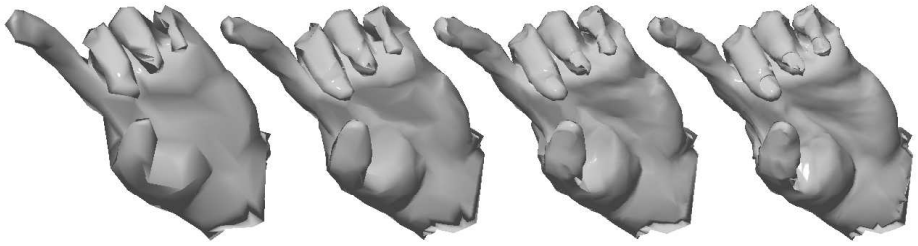

Figure 5: Convergence of the optimization process. From left to right: After 200 iterations in coarsest resolution, after 200 more iterations in 2^{nd} resolution, after 200 more iterations in 3^{rd} resolution, after 500 more iterations in the finest resolution.

5.2 Sparse model fitting

In this example, we show how the stress can be reduced as a postprocessing of an arbitrary pair of nearly isometric shapes. Here we demonstrate it as a postprocessing for a shape completion. The inputs we use in this experiment are an initial shape X_0 (Fig. 6 left) and a set of 12 markers marked both on X_0 and on a target model (Fig. 6 2^{nd} from left). The deformation was made using the algorithm of Ben-Chen and Weber [3] (Fig. 6 3^{rd} from left), we denote its result as X_d.

We minimize the following cost function

$$\text{cost}(X) = \sigma(X, X_0) + \lambda_1 \sum_{x_i \in X} d_{\mathbb{R}^3}(x_i, X_d)^2 + \lambda_2 D_{\text{vol}}(X, X_0, r_v).$$

where $d_{\mathbb{R}^3}(x_i, X_d)$ is the minimal Euclidean distance from x_i to a point on the faces of X_d. We use $r_v = 0.05 \cdot \text{vol}(X_0)$, $\lambda_1 = 1$ and $\lambda_2 = 10^{-5}$.

The stress between the prior and the unknown shape is 1.11×10^4. The stress between the prior and the deformation's result is 1.08×10^4. After 800 iterations, the stress between the prior and the optimization's result was 2.4×10^3.

5.3 Bundle adjustment

The bundle adjustment problem is defined as follows. Let K denote the a number of cameras looking at N points on the surface (we ignore the subject of occlusions

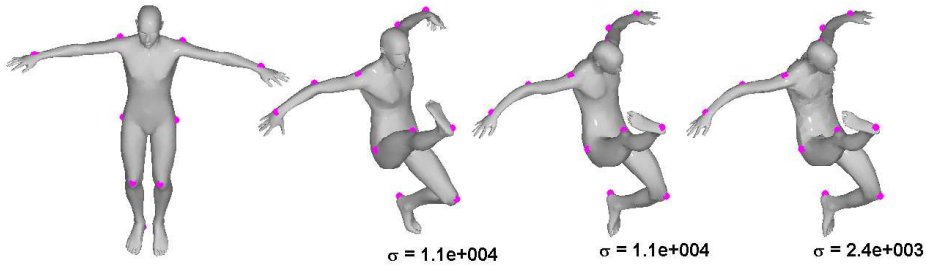

Figure 6: Stress reduction after sparse model fitting. From left to right: the prior shape, the unknown shape, the result of the deformation to fit the marker points, the result after reducing the stress.

and mismatches). Let $\{s_i^{(k)}\}_{i=1}^N \subseteq \mathbb{R}^2, k \in 1,\ldots,K$ denote a set of projected coordinates obtained by projecting the points $(x_i)_{i=1}^N \subseteq \mathbb{R}^3$ into a set of cameras with projection matrices $P_k, k = 1,..,K$, $P_k(\tilde{x}_i) = \tilde{s}_i^{(k)}$, where $\tilde{x}_i$ and $\tilde{s}_i^{(k)}$ are homogenous coordinates representation of x_i and $s_i^{(k)}$ respectively.

Next, let $\{r_i^{(k)}\}_{i=1}^N \subseteq \mathbb{R}^2, k \in 1,\ldots,K$ denote a set of noisy projected coordinates, $r_i^{(k)} = s_i^{(k)} + n$, where $n \sim N(0,\sigma)$ is Gaussian noise.

In this example, we try to reconstruct the unknown object X given an intrinsically similar prior shape X_p, and correspondences functions $g^{(k)} : r_i^{(k)} \rightarrow X_p$.

The cost function we minimize is

$$\text{cost}(X) = \sigma(X, X_p) + \lambda_1 \sum_{k=1}^{K} \sum_{i=1}^{N} \|r_i^{(k)} - P_k(g^{(k)}(r_i^{(k)}))\|_2^2 + \lambda_2 D_{\text{vol}}(X, X_p, r_v). \tag{14}$$

Deriving of the reconstruction error (the second term in Eq. 14) is explained in [25].

In this example we use $\sigma = 0.005$, the camera focal length is 10, $r_v = 0.05 \cdot \text{vol}(X_p)$, $\lambda_1 = 1 \cdot 10^5$ and $\lambda_2 = 1 \cdot 10^1$. Optimization was stopped when the cost decrease between iterations was less than 10^{-8}. Optimization was done on a single scale.

The camera arrangement and 2D images that are used as the reconstruction's input are shown in Fig. 7. Results are shown in Fig. 5.3.

The stress between the unknown surface and the prior was about 888, while the stress between the result surface and the prior was about 297.

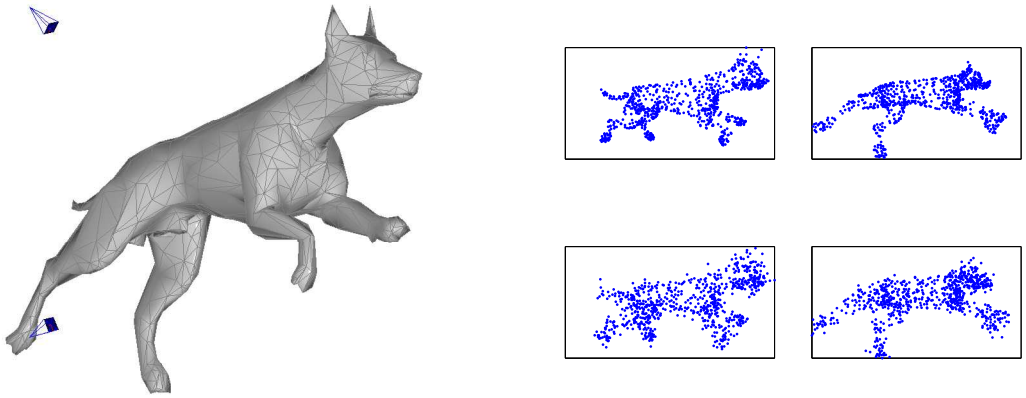

Figure 7: Left: The unknown shape, cameras location is indicated by the blue pyramids. Right: The shape vertices as viewed by the two cameras (top row) and after the noise is added (bottom row).

6 Conclusions

We propose a framework to for solving inverse problems involving non rigid shapes. Our approach assume knowledge of a prior shape intrinsically similar to the unknown shape to be reconstructed. The numerical core of our method is calculating the gradient of the geodesic distances matrix w.r.t. the coordinates of the shapes' vertices. This computation is based on differentiating the update step of the fast marching method. It has been demonstrated on various applications.

Further extensions of the proposed method include reducing the complexity by restricting the distortion to a meaningful subset of all pairwise geodesic distances, compensating for the limited resolution of the model by giving smaller weight to distances that are changed in deformations (i.e. near model joints), compensating for inaccurate correspondence between the given surfaces, and compensation for nonuniform sampling.

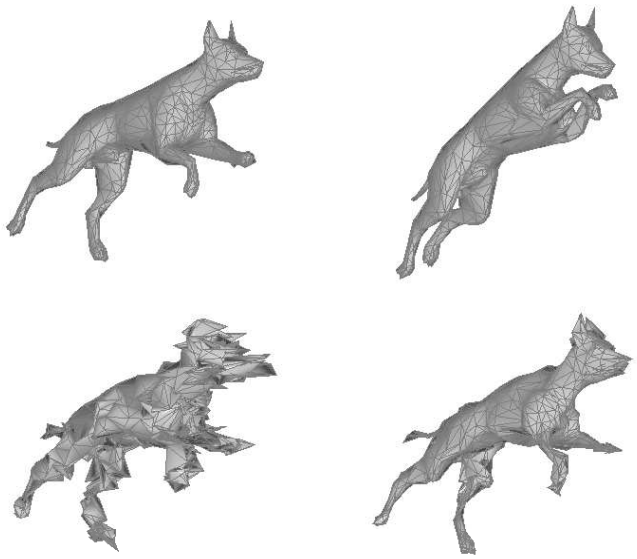

Figure 8: Unknown shape (top left), an intrinsically similar shape used as a prior (top right), Reconstruction result from noisy data in Fig. 7 with the intrinsic prior (bottom left - converged after 1501 iterations), and without the intrinsic prior (bottom right) - converged after 4265 iterations.

Acknowledgment

The authors are grateful to Yuval Rabani for his fruitful comments, and to Ofir Weber and Mirela Ben-Chen for their assistance with the 3D models. The model of the arm is courtesy of Autodesk. This research was supported by ONR grant No. N00014-06-1-0978, by The Israel Science Foundation grant No. 623/08 and by New York Metropolitan Research Fund.

References

[1] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis. SCAPE: Shape completion and animation of people. *ACM Trans. Graph*, 24(3):408–416, 2005. 3

[2] M. Ben-Chen, C. Gotsman, and G. Bunin. Conformal flattening by curvature prescription and metric scaling. *Comput. Graph. Forum*, 27(2):449–458, 2008. 4

[3] M. Ben-Chen, O. Weber, and C. Gotsman. Variational harmonic maps for space deformation. 18

[4] A. M. Bronstein and M. M. Bronstein. Nonrigid world 3d database v 1.0 - TOSCA (Toolbox for Surface Comparison and Analysis), 2007. `http://tosca.cs.technion.ac.il`. 17

[5] A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Topology-invariant similarity of nonrigid shapes. *International J. of Computer Vision*. to appear. 4, 10

[6] A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Efficient computation of isometry-invariant distances between surfaces. *SIAM J. Scientific Computing*, 28(5):1812–1836, 2006. 4

[7] A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Generalized multidimensional scaling: A framework for isometry-invariant partial surface matching. *PNAS*, 103:1168–1172, Jan. 2006. 4

[8] S. Cabay and L. W. Jackson. A polynomial extrapolation method for finding limits and antilimits of vector sequences. *SIAM J. on Numer. Anal.*, 13(5):734–752, Oct. 1976. 16

[9] A. L. Cauchy. Deuxième mémoire sur les polygones et polyèdres (second memoire on polygons and polyhedra. *J. de l'École Polytechnique*, (9):9–87, 1813. 6

[10] U. Clarenz, U. Diewald, and M. Rumpf. Anisotropic geometric diffusion in surface processing. In *IEEE Visualization*, pages 397–405, 2000. 3

[11] R. Connelly. The rigidity of polyhedral surfaces. *Math. Mag.*, 52(5):275–283, 1979. 6

[12] Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. 7

[13] D. Eberly. Distance between point and triangle in 3d, March 2008. `http://-www.geometrictools.com/-Documentation/-ex̂tttDistancePoint3Triangle3.pdf`. 28

[14] I. Eckstein, J.-P. Pons, Y. Tong, C. C. J. Kuo, and M. Desbrun. Generalized surface flows for mesh processing. In *Fifth Eurographics Symposium on Geometry Processing*, pages 183–192, 2007. 3

[15] A. Elad (Elbaz) and R. Kimmel. On bending invariant signatures for surfaces. *IEEE Trans. Pattern Anal. Mach. Intell*, 25(10):1285–1295, 2003. 4

[16] S. Fleishman, I. Drori, and D. Cohen-Or. Bilateral mesh denoising. *ACM Trans. Graph.*, 22(3):950–953, 2003. 3

[17] H. Gluck. Almost all simply connected closed surfaces are rigid. In *Geometric Topology (Proc. Conf., Park City, Utah, 1974)*, volume 438 of *Lecture Notes in Mathematics*, pages 225–239. Springer-Verlag, Berlin, 1975. 6

[18] M. Gromov. Structures métriques pour les variétés riemanniennes. 1981. 4, 5

[19] M. Kilian, N. J. Mitra, and H. Pottmann. Geometric modeling in shape space. *ACM Trans. Graph.*, 26(3):64, 2007. 3

[20] R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds. *PNAS*, 95(15):8431–8435, 1998. 7, 8

[21] N. Litke, M. Droske, M. Rumpf, and P. Schröder. An image processing approach to surface matching. In *Third Eurographics Symposium on Geometry Processing*, pages 207–216, July 2005. 4

[22] F. Mémoli and G. Sapiro. A theoretical and computational framework for isometry invariant recognition of point cloud data. *Found. Comput. Math.*, 5(3):313–347, 2005. 4

[23] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *PNAS*, 93(4):1591–1595, 1996. 7

[24] B. Springborn, P. Schröder, and U. Pinkall. Conformal equivalence of triangle meshes. *ACM Trans. Graph.*, 27(3):77:1–77:??, Aug. 2008. 4

[25] B. Triggs, P. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. Bundle adjustment: A modern synthesis. In *Vision Algorithms Workshop: Theory and Practice*, pages 298–372, 1999. 3, 19

[26] J. N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Trans. on Automatic Control*, 40(9):1528–1538, 1995. 7

[27] O. Weber, Y. S. Devir, A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Parallel algorithms for approximation of distance maps on parametric surfaces. *ACM Trans. Graph*, 27(4), 2008. 8

[28] G. Wolansky. Incompressible, quasi-rigid deformations of 2-dimensional domains, July 01 2007. 4

[29] R. Zhang, P.-S. Tsai, J. E. Cryer, and M. Shah. Shape from shading: A survey. *IEEE Trans. Pattern Anal. Mach. Intell*, 21(8):690–706, 1999. 3

A Derivation of the update step

Given a triangle formed by three vertices x_A, x_B and x_C (A, B and C are some indices), with corresponding edges a, b and c, and corresponding coordinations column vectors of the vertices $\{\overline{x}_A, \overline{x}_B, \overline{x}_C\}$.

Also given a source vertex x_s, the geodesic distances between x_s and x_A, x_B, (t_A, t_B) and the derivatives of those distances $\left(\frac{\partial t_A}{\partial \alpha_i}, \frac{\partial t_B}{\partial \alpha_i}\right)^{\mathrm{T}}$.

Next, we assume w.l.o.g. that t_C is updated in the $x_A x_B x_C$ triangle.

If the triangle is not a virtual triangle, the derivatives of the edges' length are calculated as follows.

$$\begin{aligned}\frac{\partial}{\partial \alpha_i^j}\begin{pmatrix} a & b & c \end{pmatrix} &= \frac{\partial}{\partial \alpha_i^j}\begin{pmatrix} \|\overline{x}_B - \overline{x}_C\|_2 \\ \|\overline{x}_A - \overline{x}_C\|_2 \\ \|\overline{x}_A - \overline{x}_B\|_2 \end{pmatrix}^{\mathrm{T}} \\ &= \begin{pmatrix} (\overline{x}_B - \overline{x}_C)\cdot \hat{e}_j \left(\delta\left(B-i\right) - \delta\left(C-i\right)\right) \\ (\overline{x}_A - \overline{x}_C)\cdot \hat{e}_j \left(\delta\left(A-i\right) - \delta\left(C-i\right)\right) \\ (\overline{x}_A - \overline{x}_B)\cdot \hat{e}_j \left(\delta\left(A-i\right) - \delta\left(B-i\right)\right) \end{pmatrix}^{\mathrm{T}} \end{aligned} \tag{15}$$

where $\delta(k)$ is the Kronecker delta, and $\hat{e}_j$ is a unit vector parallel to the j^{th} axis. The derivation of the length of edges in virtual triangles is described in Appendix B.

If t_C is updated using Dijkstra's update scheme, that is $t_C = \min\{t_A + b, t_B + a\}$,

$$\frac{\partial t_C}{\partial \alpha_i^j} = \begin{cases} \frac{\partial t_A}{\partial \alpha_i^j} + \frac{\partial b}{\partial \alpha_i^j} & t_A + b \le t_B + a \\ \frac{\partial t_B}{\partial \alpha_i^j} + \frac{\partial a}{\partial \alpha_i^j} & t_A + b > t_B + a \end{cases} \tag{16}$$

From now and on we assume that t_C is updated using Eq. **??**. In order to derive Eq. **??**, we first calculate the derivatives of r and s w.r.t. the lengthes of the edges and of t_C w.r.t t_A and t_B.

$$\frac{\partial r}{\partial \begin{pmatrix} a, & b, & c \end{pmatrix}} = \begin{pmatrix} \frac{-a}{c}, & \frac{b}{c}, & -\frac{b^2-c^2-a^2}{2c^2} \end{pmatrix}, \tag{17}$$

$$\frac{\partial s}{\partial \begin{pmatrix} a, & b, & c \end{pmatrix}} = \begin{pmatrix} \frac{ra}{sc}, & \frac{bc-rb}{sc}, & \frac{r}{s}\frac{b^2-c^2-a^2}{2c^2} \end{pmatrix}, \tag{18}$$

$$\frac{\partial t_C}{\partial \begin{pmatrix} t_A, & t_B \end{pmatrix}} = \begin{pmatrix} 1 - \frac{r}{c} + \frac{sv}{c\sqrt{c^2-v^2}}, & \frac{r}{c} - \frac{sv}{c\sqrt{c^2-v^2}} \end{pmatrix}. \tag{19}$$

Next, we derive Eq. **??** w.r.t. the edge length, using the chain rule and Eqs. 17

and 18, yielding

$$\frac{\partial t_C}{\partial \left(\begin{array}{cccc} a, & b, & c \end{array} \right)} = \left(\begin{array}{l} \frac{v}{c}\frac{\partial r}{\partial a} + \frac{\sqrt{c^2-v^2})}{c}\frac{\partial s}{\partial a} \\ \frac{v}{c}\frac{\partial r}{\partial b} + \frac{\sqrt{c^2-v^2}}{c}\frac{\partial s}{\partial b} \\ \frac{v}{c}\left(\frac{-r}{c} + \frac{\partial r}{\partial c}\right) + \frac{\sqrt{c^2-v^2}}{c}\frac{\partial s}{\partial c} + s\left(\frac{1}{\sqrt{c^2-v^2}} - \frac{c^2-v^2}{c^2}\right) \end{array} \right)^{\mathrm{T}}. \tag{20}$$

Combining Eqs. 20, 19 and 15, we obtain

$$\frac{\partial t_C}{\partial \alpha_i^j} = \frac{\partial t_C}{\partial \left(\begin{array}{cc} t_A, & t_B \end{array} \right)} \left(\begin{array}{c} \frac{\partial t_A}{\partial \alpha_i} \\ \frac{\partial t_B}{\partial \alpha_i} \end{array} \right) + \frac{\partial t_C}{\partial \left(\begin{array}{ccc} a, & b, & c \end{array} \right)} \frac{\partial}{\partial \alpha_i^j} \left(\begin{array}{ccc} a & b & c \end{array} \right)^{\mathrm{T}}. \tag{21}$$

B Derivation of triangle unfolding

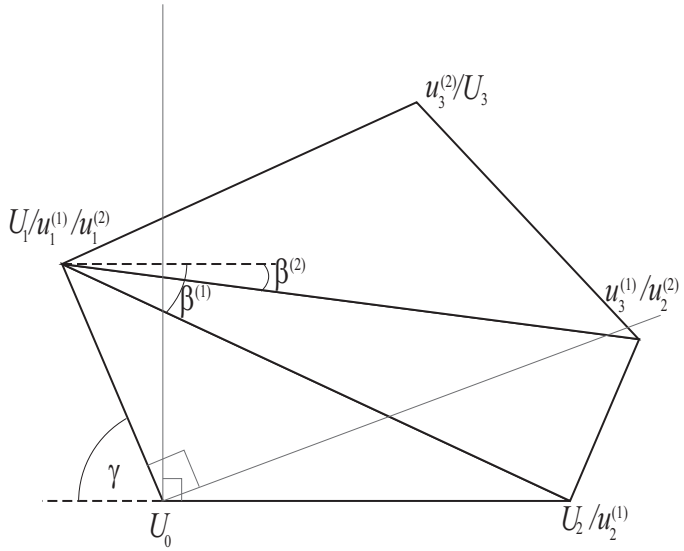

Figure 9: An example of unfolding triangles in order to split an obtuse triangle to two virtual triangles with acute angles. The superscripts denote different stages, the different names of a vertex denotes the variables pointing to this vertex in different iterations. u_3 of the first iteration crates an obtuse angle with U_0 and u_1 of the first iteration and therefore a second iteration is needed.

First we review the triangle splitting process, according to the example described in Fig. 9. For convenient, we use U_0, U_1, U_2 and U_3 to denote the vertices of the original triangle and the vertex that creates the two virtual triangles. u_1, u_2 and u_3 denotes the vertices of the last triangle to be unfolded. The unfolding is done according to Alg. 1. The algorithm is based on defining a 2D coordinate system in which U_0 is at the origin, U_2 is located on the $\hat{x}$ axis and the $\hat{y}$ coordinate od U_1 ($(U_1)_y$) is non negative. Iteratively, triangles are unfolded to this plane until a vertex that forms two acute triangles with U_0, U_1 and U_2 is found. The algorithm uses a function named *getNextTrig* that given a triangle and an edge returns a the third vertex of the other triangle containing the edge. If no such vertex exists (that is, the edge is on the boundary of the mesh) it returns an empty set. We denote the $\hat{x}$ and $\hat{y}$ coordinates of a vertex u_i in the above coordinates system by u_i^x and u_i^y respectively (or $\tilde{u}_i$ as a 2×1 column vector), and the location of vertex u_i in the *3D* mesh's space by $\overline{u}_i$.

The derivation of the virtual triangle edges' lengths is done in parallel to the unfolding of the triangles. This is done as follows. Derivatives the edges lengths (step 1 of Alg. 1) w.r.t. all vertices locations is done as described in Eq. 15.

Next, deriving the initial values of u_1^x, u_1^y, u_2^x and u_2^y (steps 4 and 5 of Alg. 1) is as follows.

$$\frac{\partial u_2^x}{\partial \alpha_i^j} = \frac{\partial E_1}{\partial \alpha_i^j}, \qquad \frac{\partial u_2^y}{\partial \alpha_i^j} = 0. \tag{22}$$

Using

$$\frac{\partial u_1^x}{\partial \begin{pmatrix} E_0, & E_1, & E_2 \end{pmatrix}} = \begin{pmatrix} -\frac{E_0}{E_1}, & \frac{E_1^2 - E_2^2 + E_0^2}{2E_1^2}, & -\frac{E_2}{E_1} \end{pmatrix}, \tag{23}$$

and the chain rule we obtain

$$\frac{\partial u_1^x}{\partial \alpha_i^j} = \sum_{k=0}^{2} \frac{\partial u_1^x}{\partial E_k} \frac{\partial E_k}{\partial \alpha_i^j}, \qquad \frac{\partial u_1^y}{\partial \alpha_i^j} = \frac{1}{u_1^y}(E_2 \frac{\partial E_2}{\partial \alpha_i^j} - u_1^x \frac{\partial u_1^x}{\partial \alpha_i^j}). \tag{24}$$

We define $\nabla_\alpha U_1^x$, $\nabla_\alpha U_1^y$, $\nabla_\alpha U_2^x$ and $\nabla_\alpha U_2^y$ as the derivatives of u_1^x, u_1^y, u_2^x and u_2^y at this point.

Next, deriving β (step 12),

$$\begin{aligned} \frac{\partial \beta}{\partial \alpha_i^j} &= \frac{\partial}{\partial \alpha_i^j} \frac{u_2^x - u_1^x}{\|\overline{u}_1 - \overline{u}_2\|_2} \\ &= \frac{\left(\|\overline{u}_2\|_2^2 \left(\frac{\partial u_x^2}{\partial \alpha_i^j} - \frac{\partial u_x^1}{\partial \alpha_i^j} \right) + (u_2^x - u_1^x)(u_2^y - u_1^y) \left(\frac{\partial u_y^1}{\partial \alpha_i^j} - \frac{\partial u_y^2}{\partial \alpha_i^j} \right) \right)}{\|\overline{u}_1 - \overline{u}_2\|_2} \end{aligned} \tag{25}$$

We define $e_1 = \|\overline{u}_2 - \overline{u}_3\|$ and $e_3 = \|\overline{u}_1 - \overline{u}_2\|$. Deriving e_1, e_2 and e_3 w.r.t. α_i^j is done as described in Eq. 15.

Deriving x_t (steps 14) w.r.t. e_1, e_2 and e_3,

$$\frac{\partial x_t}{\partial \begin{pmatrix} e_1, & e_2, & e_3 \end{pmatrix}} = \begin{pmatrix} -\frac{e_1}{e_3}, & \frac{e_2}{e_3}, & \frac{e_1^2 - e_2^2 + e_3^2}{2e_3^2} \end{pmatrix}. \tag{26}$$

Using Eq. 26 and the chain rule we obtain

$$\frac{\partial x_t}{\partial \alpha_i^j} = \sum_{e \in \{e_1, e_2, e_3\}} \frac{\partial x_t}{\partial e} \frac{\partial e}{\partial \alpha_i^j} \tag{27}$$

Using $\frac{\partial y_t}{\partial x_t} = -\frac{x_t}{y_t}$, $\frac{\partial y_t}{\partial e_2} = \frac{e_2}{y_t}$, Eq. 27 and the chain rule we obtain

$$\frac{\partial y_t}{\partial \alpha_i^j} = \frac{\partial y_t}{\partial e_2} \frac{\partial e_2}{\partial \alpha_i^j} + \frac{\partial y_t}{\partial x_t} \frac{\partial x_t}{\partial \alpha_i^j}. \tag{28}$$

Finally, deriving $\overline{x}_3$ (step 16) is done as follows

$$\begin{pmatrix} \frac{\partial u_3^x}{\partial \alpha_i^j} \\ \frac{\partial u_3^y}{\partial \alpha_i^j} \end{pmatrix} = \begin{pmatrix} 1 & \cot(\beta) \\ -\cot(\beta) & 1 \end{pmatrix} \begin{pmatrix} x_t \\ y_t \end{pmatrix} + \begin{pmatrix} \cos(\beta) & -\sin(\beta) \\ \sin(\beta) & \cos(\beta) \end{pmatrix} \begin{pmatrix} \frac{\partial x_t}{\partial \alpha_i^j} \\ \frac{\partial y_t}{\partial \alpha_i^j} \end{pmatrix} + \begin{pmatrix} \frac{\partial u_1^x}{\partial \alpha_i^j} \\ \frac{\partial u_1^y}{\partial \alpha_i^j} \end{pmatrix} \tag{29}$$

if the iteration step is repeated, we update the derivatives of $\tilde{u}_2$ or $\tilde{u}_1$ using $\nabla_\alpha \tilde{u}_1 = \nabla_\alpha \tilde{u}_2$ if the condition in step 21 is true and $\nabla_\alpha \tilde{u}_2 = \nabla_\alpha \tilde{u}_2$ otherwise.

When reaching step 18, calculating the derivatives of the distance between U_3 and U_2 (denoted by U_3U_2)w.r.t. α_i^j is done as follows.

$$\frac{\partial U_3U_2}{\partial \alpha_i^j} = \frac{U_2^x - U_3^x}{U_3U_2} \left(\frac{\partial U_2^x}{\partial \alpha_i^j} - \frac{\partial U_3^x}{\partial \alpha_i^j} \right) + \frac{U_3^y}{U_3U_2} \frac{\partial U_2^y}{\partial \alpha_i^j}. \tag{30}$$

The derivatives of the distances between U_3 to U_0 and U_3 to U_1 w.r.t. α_i^j are done in a similar manner.

C Derivation of other properties

C.1 The Hausdorff distance gradient

In this work we derived a non symmetric L_2 variant of the Hausdorff distance (Eq. 2),

$$d_{\mathrm{H}}(Z,Y) = \sum_{i=1}^{N} \left(d_{\mathbb{R}^3}(z_i, Y)\right)^2 . \tag{31}$$

where $d_{\mathbb{R}^3}(z_i, Y)$ is the minimal Euclidean distance between z_i and a point *on a face* of Y, denoted by $\hat{y}$.

$\hat{y}$ is found by checking the minimal distance to each of Y's triangles relatively close to z_i. This set of triangles is found using an approximate nearest neighbor algorithm over z_i and Y's vertices.

Assuming that $\hat{y}$ is contained in a single triangle consisted from the vertices y_1, y_2, y_3, $d_{\mathbb{R}^3}(z_i, \hat{y})$ is a quadratic equation depending on the coordinates of y_1, y_2, y_3 and z_i, and therefore it's derivation according to the coordinates of z_i is straightforward. Refer to [13] for details of this quadratic equation.

In the case that $\hat{y}$ is contained in more than one triangle of Y, we average the gradient of $d_{\mathbb{R}^3}(z_i, \hat{y})$ over all the triangles containing $\hat{y}$.

C.2 The area similarity gradient

Consider a surface X. The surface area of X is defined by

$$\mathrm{area}\,(X) = \sum_{x_1,x_2,x_3 \in \mathrm{trigs}(X)} \frac{1}{2} \left\| (\overline{x}_2 - \overline{x}_1) \times (\overline{x}_3 - \overline{x}_1) \right\|_2, \tag{32}$$

where $\mathrm{trigs}\,(X)$ is the triangulation of X.

The derivation of Eq. 32 is cumbersome but simple and therefore omitted.

Given a second surface Y, we define the difference between the areas of X and Y, by $D_{\mathrm{area}}(X,Y) = (\mathrm{area}(X) - \mathrm{area}(Y))$. The derivatives of the *squared* area difference is

$$\nabla_\alpha D_{\mathrm{area}}(X,Y)^2 = D_{\mathrm{area}}(X,Y) \nabla_\alpha \mathrm{area}(X) \tag{33}$$

We also define an area difference with tolerance r as

$$D_{\mathrm{area}}(X,Y,r) = \begin{cases} \left(|\mathrm{area}(X) - \mathrm{area}(Y)| - r\right)^2 & |\mathrm{area}(X) - \mathrm{area}(A)| > r \\ 0 & \text{otherwise} \end{cases} ,$$

for which the deriving is also straightforward.

C.3 The volume similarity gradient

Consider a surface X. The volume of X is calculated using Green's theorem

$$\begin{aligned}\text{vol}\left(X\right) &= \iiint_X dv = \iint_{S(X)} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}^{\mathrm{T}} N(s)ds \\ &= \sum_{t \in \text{trigs}(X)} \text{area}(t)(N(t))_x, \end{aligned} \tag{34}$$

where $S(X)$ is the surface of X, $N(s)$ is the normal vector at point s, $(N(t))_x$ is the first entry of the normal to triangle t, area(t) is the area of triangle t, and trigs (X) is the triangulation of X.

Since, area$(t) = \frac{1}{2}\left\|(\overline{x}_2 - \overline{x}_1) \times (\overline{x}_3 - \overline{x}_1)\right\|_2$ and $(N(t))_x = \left((\overline{x}_2 - \overline{x}_1) \times (\overline{x}_3 - \overline{x}_1)\right)_x$ the derivation of area$(t)(N(t))_x$ is cumbersome and simple and therefore omitted. The derivation of Eq. 34 is as follows.

$$\frac{\partial \text{vol}\left(X\right)}{\partial \alpha_i^j} = \qquad\qquad = \sum_{\substack{t \in \text{trigs}(X) \\ x_i \in t}} \frac{\partial}{\partial \alpha_i^j} \text{vol}(t)(N(t))_x. \tag{35}$$

Given a second surface Y, we define the difference between the volumes of X and Y, by $D_{\text{vol}}(X,Y) = (\text{vol}(X) - \text{vol}(Y))$. The derivatives of the *squared* volume difference is

$$\nabla_\alpha D_{\text{vol}}(X,Y)^2 = D_{\text{vol}}(X,Y)\nabla_\alpha \text{vol}(X) \tag{36}$$

We also define a volume difference with tolerance r as

$$D_{\text{vol}}(X,Y,r) = \begin{cases} \left(|\text{vol}(X) - \text{vol}(Y)| - r\right)^2 & |\text{vol}(X) - \text{vol}(Y)| > r \\ 0 & \text{otherwise} \end{cases}$$

for which the deriving is straightforward.

Algorithm 1: Splitting an obtuse angled triangle to virtual acute triangles.

Input: Numerical grid **U** and an acute triangle $U_1U_0U_2$

Output: A splitting vertex U_3 and the lengths of U_0U_3, U_1U_3 and U_2U_3 in the $U_1U_0U_2$ plane

Initialization

$E_0 \longleftarrow \|\overline{U}_1 - \overline{U}_2\|; \quad E_1 \longleftarrow \|\overline{U}_0 - \overline{U}_2\|; \quad E_2 \longleftarrow \|\overline{U}_0 - \overline{U}_1\|$

$u_1 \longleftarrow U_1; \quad u_2 \longleftarrow U_2$

$\tilde{u}_0 \longleftarrow (0,0)^{\mathrm{T}}$

$\tilde{u}_1 \longleftarrow (E_2\cos(\angle U_2U_0U_1, E_2\sin(\angle U_2U_0U_1))^{\mathrm{T}}$

$\tilde{u}_2 \longleftarrow (E_1,0)^{\mathrm{T}}$

$trig_{curr} \longleftarrow U_1U_0U_2$

$\alpha \longleftarrow \frac{-u_1^x}{u_1^y}$

Iteration

while *true* **do**

  $u_3 \longleftarrow getNextTrig(trig_{curr}, u_1u_2)$

  if $u_3 == \emptyset$ **then**

    RETURN $\emptyset$

  $\beta \longleftarrow \angle u_2u_1(u_2^x, u_1^y)$

  $e_2 \longleftarrow \|\overline{u}_3 - \overline{u}_1\|$

  $x_t \longleftarrow e_2\cos(\angle u_3u_1u_2)$

  $y_t \longleftarrow e_2\sin(\angle u_3u_1u_2)$

  $\tilde{u}_3 \longleftarrow \tilde{u}_1 + \begin{pmatrix} \cos(\beta) & -\sin(\beta) \\ \sin(\beta) & \cos(\beta) \end{pmatrix} \begin{pmatrix} x_t \\ y_t \end{pmatrix}$

  if $u_3^x > 0$ *AND* $\frac{u_3^y}{u_3^x} > \tan(\gamma)$ **then**

    $U_3 \longleftarrow u_3$

    $\begin{pmatrix} U_3U_2 & U_3U_0 & U_3U_1 \end{pmatrix} \longleftarrow$
    $\begin{pmatrix} \sqrt{(u_3^x - u_2^x)^2 + u_3^{y2}} & \sqrt{u_3^{x2} + u_3^{y2}} & \sqrt{(u_3^x - u_1^x)^2 + (u_3^y - u_1^y)^2} \end{pmatrix}$

    RETURN

  if $u_3^x < 0$ **then**

    $u_1 \longleftarrow u_3$

  else

    $u_2 \longleftarrow u_3$

  $trig_{curr} \longleftarrow u_1u_2u_3$