Texture Mapping using Surface Flattening via Multi-Dimensional Scaling

Gil Zigelman, Ron Kimmel and Nahum Kiryati

Abstract— **We present a novel technique for texture mapping on arbitrary surfaces with minimal distortions, by preserving the local and global structure of the texture. The recent introduction of the fast marching method on triangulated surfaces, made it possible to compute a geodesic distance map from a given surface point in $O(n \lg n)$ operations, where n is the number of triangles that represent the surface. We use this method to design a surface flattening approach based on multi-dimensional scaling (MDS). MDS is a family of methods that map a set of points into a finite dimensional flat (Euclidean) domain, where the only given data is the corresponding distances between every pair of points. The MDS mapping yields minimal changes of the distances between the corresponding points. We then solve an 'inverse' problem and map a flat texture patch onto the curved surface while preserving the structure of the texture.**

Keywords— **Texture mapping, multi-dimensional scaling, fast marching method, Geodesic distance, Euclidean distance.**

I. Introduction

The texture mapping problem is closely related to the inverse problem of flattening a curved surface into a plane. In the context of mapping the surface of the earth this is known as the 'map maker problem'. It has been shown by Gauss in 1828 that an isometric mapping between two surfaces of different intrinsic curvature is not possible. In other words, it is impossible to map a convoluted surface onto a plane or a sphere without introducing metric distortions because both surfaces differ with respect to their Gaussian curvature. Thus, only approximate solutions are possible. Therefore, flattening algorithms can only aim for minimal geometric distortions but cannot prevent distortions altogether.

The computer graphics community has made many attempts to solve the problem of mapping flat texture images onto curved surfaces. The main problems with most of the existing methods are that they

- introduce large deformations and distortions to the original texture, and
- involve high computational complexity.

Environment mapping [5], [14], is one technique that creates the effect of environment reflections on surfaces. It maps the original 2D texture to a sphere or a cube surrounding the surface. Then, the surface normal at each point is used to find the intersection of the reflected viewing vector with the surrounding simple object, and assigns the texture at that point to the corresponding surface point. These methods do not preserve the local area of the texture and introduce local deformations. Moreover, mapping the 2D texture onto a sphere causes distortions to begin with. In order to minimize these artifacts, one has to distort the original flat texture image before mapping onto a sphere.

Bier and Sloan [4] extended the environment mapping idea and proposed a two step procedure. First, the texture is mapped onto a simple object (preferably preserving the area) and then it is mapped from the simple object to the given surface, using, for example, the surface normal's intersection with the simple object. This method also introduces visible deformations, however, it can decrease the distortions which exist in the previous methods.

Kurzion, Möller and Yagel [18] try to preserve area. They use a cube as a simple surrounding object. For each point they find two curvature values in specially selected directions, and then change the density of the surrounding image respectively. This method is area preserving, however, it creates shear effects. It is also limited to smooth surfaces with C^2 continuity.

Arad and Elber [1] preserve the local texture area by finding, for a specific viewing direction, the four intersection curves (in the parametric space) between a swept rectangle in the viewing direction and the surface. Then, they warp the square texture image to fit the four intersection curves. The texture image is warped before mapping. This method is useful in cases where one wants to map a texture on a small portion of a surface.

Bennis, Vézien, and Iglésias [3] first piecewise flatten the surface and then map the texture onto each flattened part. The flattening of a region grows around an isoparametric curve selected manually. They use a distortion metric as a control and stop the growth when the accumulated distortion exceeds a given threshold. They permit discontinuities on the mapped texture in order to minimize distortions.

Floater [11], [12], presented a 'shape preserving' texture mapping. He first limits the triangulated surface boundary vertices to specific coordinates of a convex polygon in R^2. Then he solves a set of linear equations that force all other interior vertices to be a convex combination of their neighbors. He proves that this way there will be no self

G. Zigelman is with the Department of Computer Science, Technion, Haifa 32000, Israel, E-mail: zgil@cs.technion.ac.il

R. Kimmel is with the Department of Computer Science, Technion, Haifa 32000, Israel, E-mail: ron@cs.technion.ac.il

N. Kiryati is with the Department of Electrical Engineering–Systems, Tel Aviv University, Tel Aviv 69978, Israel, E-mail: nk@eng.tau.ac.il

intersections of triangles on the flat texture plane. Since the boundary texture coordinates are fixed, deformations obviously occur, especially global ones.

Lévy and Mallet [19] extended Floater's ideas by adding flexibility to the scheme. Instead of solving a set of linear equations, they minimize a global criterion in a least square sense. This allows adding new measures in addition to Floater's linear equations. These measures try to preserve the perpendicularity and constant spacing of isoparametric curves traced on the surface. The constraints involve local orthogonality of the chart, and local homogeneity in the sense that isoparametric curves are restricted to be straight between adjacent triangle. The constraints add freedom to Floater model, yet require many iterations to converge, with about 100 iterations for 3000 triangles taking about a minute on SGI R4000 processor. The constraints also enable them to make the texture continuous through cuts on the surface. In general, these local constraints can not be satisfied, so the problem is redefined as a least square problem with control over the orthogonality and straightness of the isoparametric curves of the chart. The constraints have local influence, and the whole scheme depends mainly on boundary conditions (otherwise the problem is ill-posed), though boundary conditions can be given inside the domain and in a sense work like an extrapolation.

In [2], Azariadis and Aspragathos proposed to minimize a functional that combines a dissimilarity measure for neighboring vertices and an area measure for the flattened triangles. They also restrict two curves in their mapping to have identical lengths as two selected curves on the surface. Roughly speaking, this constraint can be considered as a boundary condition for their problem. Their non-linear optimization scheme handles in a few seconds about thousand vertices. Actually, had they extended their length term in their energy definition to non-neighboring vertices, they would have been able to ignore both the boundary constraint and the area terms, since these measures are implicitly included within multi-dimensional scaling functionals.

Neyret and Cani [21] dealt with any surface topology by tiling together small textured patches with matching boundaries. Their method is limited to textures with relatively small details, as the tiles should be relatively small. A solution to a similar problem was introduced by Praun, Finkelstein and Hoppe in [22]. They detect features in a small texture patch, and repeatedly paste them onto any given surface until it is completely covered. These methods are not suitable for mapping an image onto a curved domain.

The work of Wolfson and Schwartz [28], and Schwartz, Shaw and Wolfson [25], introduced a clever flattening method, the ingredients of which are revisited in this paper. They first use a computationally intensive way for finding the geodesic distance between pairs of points on the surface. Then, they use a specific MDS (Multi-Dimensional Scaling) approach to flatten the surface using these geodesic distances, and by minimizing the functional presented by Sammon in [24], which resembles the Stress-1 functional [6]. Their method involves high computational complexity and therefore is not practical.

Motivated by [28] we introduce a new efficient mapping method that preserves both the local and the global structure of the texture, with minimal shearing effects. It enables realistic texture mapping on any given surface which is homeomorphic to a plane. Our method avoids the need for an intermediate surface or boundary conditions and does not require any smoothness condition on the surface.

Most of the previous flattening methods require boundary conditions mainly due to the fact that they try to integrate local measures. In the proposed method boundary conditions are not required for a valid solution. Moreover, the structure preserved by the proposed scheme is determined both by distances between close points on the surface and between distant points, therefore, both the 'local' as well as the 'global' structure of the texture are preserved. The method is based on two numerical tools that replace the numerical machinery used in [28], namely, the fast marching method on triangulated domains [15], and classical multi-dimensional scaling [6], [7]. Section II briefly reviews the fast marching method on triangulated domains, which is used in order to calculate the geodesic distances between points on the surface. In Section III we discuss some matrix operations that are useful for the MDS method presented in Section IV. Section V explains how to perform the texture mapping. Section VI provides experimental results and Section VII gives concluding remarks.

II. Finding Geodesic Distances

The first step in our flattening procedure is finding the geodesic distances between pairs of points on the surface. For this task we use the fast marching method on triangulated domains, introduced by Kimmel and Sethian in [15]. This method is an extension to curved domains of Sethian's fast marching method [26], and Tsitsiklis Eikonal solvers on flat domains [27]. An alternative method for finding the geodesic distances, based on graph search and length estimators, was presented by Kiryati and Székely [16].

Sethian's idea of using the fast marching approach for distance computation is to efficiently solve an Eikonal equation $|\nabla T| = 1$, anchored at the source point p, namely $T(p) = 0$. The solution T is a distance function, and its numerical solution is computed by a monotone update scheme that is proven to converge to the 'viscosity' smooth solution. The idea is to iteratively construct the distance function by patching together small planes supported by neighboring grid points with gradient 1, starting from the source point and propagating outwards.

As we go to triangulated domains, we need to carefully deal with the update step of one vertex in the triangle, while the T values at the two other vertices are given. Roughly speaking, all we need to do is to solve a quadratic equation for the new vertex, and select the larger solution. This simple update approach would work after preprocessing obtuse triangles, as explained in [15]. The vertices are updated in an increasing T order, similar to the classical Dijkstra graph search method, the only difference being the update step which now incorporates the underlying geometry of the problem.

The fast marching method on triangulated domains enables us to compute geodesic distances in $O(n \lg n)$ where n is the number of triangle vertices that represent the curved surface. This means that we can calculate all the necessary geodesic distances in $O(n^2 \lg n)$. We show in Section IV that we do not have to calculate the geodesic distances for all pairs of vertices on the surface. We select a subset of the vertices and thereby speed up the algorithm.

Let us give a simple explanation for the algorithm of fast marching on triangulated domains. Given a point on a triangulated surface, consider the problem of finding the geodesic distance (the distance on the surface) from the given point to the rest of the points on the surface. We would like to be able to efficiently compute the values of this function at each vertex on the triangulated surface.

Consider first the well known Dijkstra graph search algorithm that can be used to find a rough approximate solution to the problem. The idea is to describe the surface as a graph in which the edges of the triangles are non-directed weighted edges in the graph connecting all the vertices, where the weight w equals to the Euclidean distance,

$$w(e_{v_i v_j}) = d(v_i, v_j).$$

Here $d(v_i, v_j)$ is the Euclidean distance between the two vertices v_i and v_j, and $e_{v_i v_j}$ is the edge connecting them. The vertices can be dually represented as points in 3D.

The Dijkstra algorithm consists of the following steps,
Init : Let v_0 be the source vertex. Set $T(v_0) = 0$, and set the rest of the vertices to $T(v) = \infty$.
Step 1: Update all vertices v_i which are connected through one edge to v_0 to

$$T(v_i) = \min(T(v_i), T(v_0) + w(e_{v_0 v_i})).$$

Step 2: Insert the new updated vertices to a min-heap data structure. If the vertex is already in the heap, then just update its value and location.
Step 3: If the heap is empty return, else remove the vertex at the top of the min-heap structure, and name it v_0.
Step 4: Go to Step 1.

A simple computational analysis shows that for graphs with a small degree at each vertex, the computational complexity for computing the shortest graph distance from one selected vertex to the rest of the vertices in the graph takes $O(n \lg n)$, where n is the total number of vertices in the graph. The reason is that each vertex is selected once, and each update of the heap takes at most $\lg n$.

This could have been a perfect solution to our problem. Unfortunately, any graph search based algorithm imposes an artificial non-geometric metric while computing the distance. A simple 2D example is the L_1 or Manhattan distance, also known as chess-board distance, in which the path is restricted to vertical (south to north) and horizontal (east to west) edges, like the streets in Manhattan. It does not really matter how narrow and dense these streets are, getting from the south-west corner to the north-east corner of town will always have a distance that equals to traveling from the south-west corner to the south-east corner, and then from the south east corner to the north east corner, see Figure 1. This property is also known as metrication error or numerical inconsistency. It means that a graph search based method is not guaranteed to converge to the continuous solution no matter how much one refines the underlying graph.

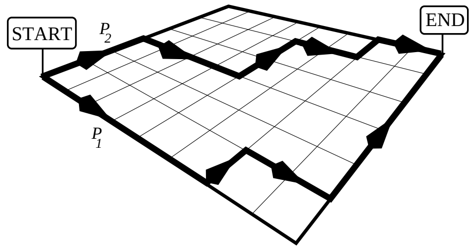

Fig. 1. The length of the path P_1 is equivalent to the length of the path P_2. As long as the path is restricted to horizontal and vertical directions, the Euclidean shortest path can not be extracted.

We now look at a different alternative, one that incorporates the geometry of the problem into the solution. A close observation of a distance function from a given point in the plane shows that its slope equals one almost everywhere. We can construct the distance function, as a monotonically increasing function defined over the domain with a unit slope. The trick is to replace the update step (Step 2) in the Dijkstra algorithm such that instead of sensing the distance through one edge $T(v_0) + w(e_{v_0 v_i})$, the new update now attempts to sense the value of the distance by considering two neighboring vertices of the same triangle, and update the value of the vertex such that the slope of the plane defined by the function over the triangle equals one.

For example, assume v_1, v_2, and v_3 are the vertices of one triangle, and without loss of generality assume the three are defined on the xy plane and therefore given by two coordinates. Given $T(v_1)$ and $T(v_2)$, the question is how to update $T(v_3)$.

Since we want to compute a distance map we would like a gradient that equals to 1. In general we have two possible solutions for $T(v_3)$, one with $T(v_3)$ smaller than $T(v_2)$ and/or smaller than $T(v_1)$, and another with $T(v_3)$ larger than $T(v_1)$ and $T(v_2)$. We would obviously like to select the second solution. Formally, the solution is a result of a quadratic equation that defines the angle between the normal of the plane $\{(v_1, T(v_1)), (v_2, T(v_2)), (v_3, T(v_3))\}$ and the normal to the plane $\{v_1, v_2, v_3\}$, to be $\pi/4$. So, $T(v_3)$ is the larger solution to the quadratic equation that defines the plane through the points $\{(v_1, T(v_1)), (v_2, T(v_2)), (v_3, T(v_3))\}$ that has a unit gradient magnitude with respect to the coordinate plane defined by the triangle $\{v_1, v_2, v_3\}$, see Figure 2.

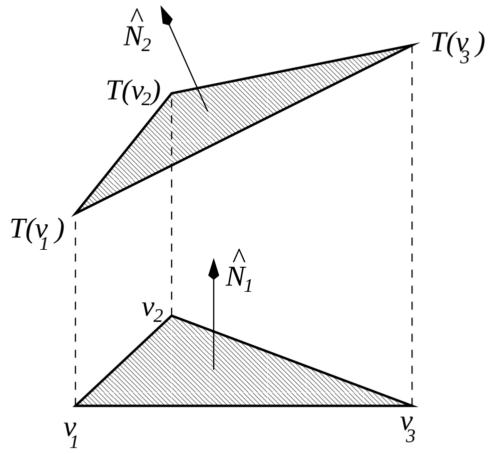

Fig. 2. The value of $T(v_3)$ is set so that $T(v_3) > T(v_1)$, $T(v_3) > T(v_2)$, and the gradient magnitude of the plane $\{(v_1, T(v_1)), (v_2, T(v_2)), (v_3, T(v_3))\}$ is 1, that is, $\hat{N}_1 \cdot \hat{N}_2 = \cos(\pi/4)$.

III. Matrix Properties

In this section we review some definitions and matrix properties [17] that will help us explain the 'classical scaling' method. We define the matrix $\mathbf{X}_{n\times m}$ to represent the coordinates of n points in an m dimensional Euclidean space R^m. The square Euclidean distance between point i and point j is defined as

$$d_{ij}^2 = \sum_{a=1}^{m} (x_{ia} - x_{ja})^2 .$$

Let the matrix $\mathbf{E}$ denote the square Euclidean distances between each pair of points in $\mathbf{X}$, that is $E_{ij} = d_{ij}^2$. Then, $\mathbf{E}$ can be compactly written as

$$\mathbf{E} = \mathbf{c}\mathbf{1}' + \mathbf{1}\mathbf{c}' - 2\mathbf{X}\mathbf{X}', \tag{1}$$

where $\mathbf{1}$ is a vector of ones of length n, $\mathbf{c}$ is a vector of length n in which $c_i = \sum_{a=1}^{m} x_{ia}^2$, and $'$ denotes the transpose operator.

Consider the translation of the coordinate origin of the points defined by $\mathbf{X}$ to a new location. Define the location to be an affine combination of the points themselves, i.e., $\mathbf{s}' = \mathbf{w}'\mathbf{X}$ where $\sum_{i=1}^{n} w_i = 1$, and for every i, $w_i \geq 0$. Then, the coordinates with respect to the new origin, denoted by the matrix $\mathbf{X}_s$, are given by

$$\begin{aligned} \mathbf{X}_s &= \mathbf{X} - \mathbf{1}\mathbf{s}' \\ &= (\mathbf{I} - \mathbf{1}\mathbf{w}')\mathbf{X} \\ &= \mathbf{P}_w\mathbf{X}. \end{aligned} \tag{2}$$

Multiplying $\mathbf{P}_w$ by a vector of ones on either side yields a vector of zeros,

$$\mathbf{P}_w\mathbf{1} = (\mathbf{1}'\mathbf{P}_w)' = \mathbf{0}.$$

When choosing $\mathbf{s}$ to be the center of mass of the points, i.e. by using

$$\mathbf{w} = \frac{1}{n}\mathbf{1},$$

the corresponding $\mathbf{P}_w$, denoted by $\mathbf{J}$, and defined by

$$\mathbf{J} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}', \tag{3}$$

is called a centering matrix, since it sets the origin to be the center of mass. If $\mathbf{X}$ is already column centered, i.e., the center of mass of the points defined by $\mathbf{X}$ is the origin of their coordinates, then

$$\mathbf{J}\mathbf{X} = \mathbf{X}. \tag{4}$$

IV. The Flattening Approach

MDS (Multi-Dimensional Scaling) is a set of mathematical techniques used to uncover the "geometric structure" of datasets, see e.g. [17]. For example, given a set of objects with proximity values amongst themselves, we can use MDS to create a 2D map of these objects, that is easier to comprehend or analyze. Rubner and Tomasi [23] use MDS for texture classification. They define metric perceptual similarities between textures, and use MDS in order to visualize the metrics.

Here we use the MDS in a similar way. As proximity values we use the geodesic distances measured between every two points on the surface, and the resulting map represents the flattening of the curved surface.

The input to the MDS is an $n \times n$ symmetric matrix $\mathbf{M}$. The M_{ij} element in the matrix $\mathbf{M}$ is the squared geodesic distance between point i and point j, where n is the number of points on the surface. We calculate the geodesic distances efficiently as mentioned in Section II.

Most MDS methods are based on finding the coordinates x^k, $k \in [1, ..., m]$ where m is the dimension we are interested in, from the given distances. Our goal is to reconstruct the $n \times 2$ matrix $\mathbf{X}$, containing 2D coordinates corresponding to the surface points. Naturally, for surfaces with effective Gaussian curvature it is impossible for the Euclidean distance between every pair of points on the flat domain ($\mathbf{X}$) to be identical to the geodesic distance between their corresponding pair of points on the surface, see for example [9]. In the flattening problem, we try to map the surface points to a plane such that the error between the corresponding distances is as small as possible under some criterion.

Let $\mathbf{M}$ be an $n \times n$ matrix where each entry is defined by $M_{ij} = T_{v_i}^2(v_j)$, i.e. the square geodesic distances between the surface points defined by the vertices v_i and v_j. One direct and simple multi-dimensional scaling approach is known as 'Classical Scaling', see [6], [7]. The idea is to approximate the matrix $\mathbf{M}$ of the square geodesic distances between surface points by a matrix $\mathbf{E}$ that defines square Euclidean distances between corresponding points given by their coordinates $\mathbf{X}$ on the plane. Notice that rank($\mathbf{E}$)=2. 'Classical Scaling' is closely related to the singular value decomposition (SVD) method, and involves the following steps:

- Compute the $n \times n$ symmetric matrix $\mathbf{M}$ of the square geodesic distances between surface points.
- Apply double centering and normalization to $\mathbf{M}$: $\mathbf{B} = -\frac{1}{2}\mathbf{J}\mathbf{M}\mathbf{J}$, where $\mathbf{J}$ is an $n \times n$ centering matrix defined in Equation (3). We want to approximate square

geodesic distances by square Euclidean distances. Therefore by rewriting $\mathbf{M}$, as shown in Equation (1), assuming that $\mathbf{X}$ is column centered, we have

$$\begin{aligned}\mathbf{B} &= -\frac{1}{2}\mathbf{J}(\mathbf{c}\mathbf{1}' + \mathbf{1}\mathbf{c}' - 2\mathbf{X}\mathbf{X}')\mathbf{J} \\ &= -\frac{1}{2}\mathbf{J}\mathbf{c}\mathbf{0}' - \frac{1}{2}\mathbf{0}\mathbf{c}'\mathbf{J} + \mathbf{J}\mathbf{X}\mathbf{X}'\mathbf{J} \\ &= \mathbf{X}\mathbf{X}'. \end{aligned} \tag{5}$$

• Compute the spectral decomposition of $\mathbf{B}$, $\mathbf{B} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}'$. We want to approximate $\mathbf{B}$ as well as possible (in the least squares sense) by a matrix of rank two. From spectral decomposition properties, we know that this can be accomplished by taking the two largest eigenvalues and their corresponding eigenvectors. The 'power method' [17] is an efficient numerical tool for finding these items. A rough analysis of the power method computational complexity is $O(n^2 d)$ where d is the number of dimensions (number of eigenvectors) one needs to extract, see [17]. In our 2D case, we can consider this step as $O(n^2)$, where n is the number of vertices selected as sparse key points that we use for the flattening. A similar method with a somewhat improved computational complexity is the Lanczos algorithm with the Rayleigh-Ritz procedure, for further details see [8], [13].

• Denote the 2×2 diagonal matrix of the first two largest positive eigenvalues as $\mathbf{\Lambda}_+$, and denote $\mathbf{Q}_+$ as the $n \times 2$ matrix of their corresponding eigenvectors. The $n \times 2$ coordinate matrix of classical scaling is given by

$$\hat{\mathbf{X}} = \mathbf{Q}_+ \mathbf{\Lambda}_+^{\frac{1}{2}}, \tag{6}$$

in which row i contains the flattened coordinates of the original surface point i. We thereby obtained a distance preserving flattening of the surface.

In order to get a more intuitive understanding of the classical scaling procedure, let us assume that the surface is developable. That is, the surface can be perfectly flattened. This means that the square geodesic distances matrix $\mathbf{M}$ is identical to the square Euclidean distances matrix $\mathbf{E}$ given in Equation (1). Moreover, the resulting multiplication $\mathbf{X}\mathbf{X}'$ in Equation (5) is equal to $\hat{\mathbf{X}}\hat{\mathbf{X}}'$. Therefore, $\mathbf{B}$ has only two eigenvalues, and $\hat{\mathbf{X}}$ is a perfect flattening. General surfaces with effective Gaussian curvature are not developable, see for example [9]. Therefore, the resulting matrix $\mathbf{B}$ will have more than two non-zero eigenvalues. By selecting the two largest eigenvalues and their corresponding eigenvectors, we can approximate the matrix $\mathbf{B}$. The approximation error is determined by the rest of the eigenvalues that we ignore. Classical scaling approximates the matrix $\mathbf{B}$ by a matrix of lower rank. $\hat{\mathbf{X}}$ can be computed from the lower rank approximation of $\mathbf{B}$. $\hat{\mathbf{X}}$ minimizes the Strain loss function defined as

$$\begin{aligned} L(\hat{\mathbf{X}}) &= \left\| -\frac{1}{2}\mathbf{J}\left(\mathbf{E} - \mathbf{M}\right)\mathbf{J} \right\| \\ &= \left\| \hat{\mathbf{X}}\hat{\mathbf{X}}' - \mathbf{B} \right\|. \end{aligned} \tag{7}$$

We note that the general theory of multi-dimensional scaling encompasses alternative loss functions and related flattening procedures. Classical scaling uses a simple loss function that yields an efficient minimization procedure.

The classical scaling procedure is very simple to program. For example, given the square geodesic distances matrix $\mathbf{M}$ of dimensions $n \times n$, a short and simple Matlab implementation of the steps described above is given by

$$\begin{array}{rcl} \mathbf{J} &=& \mathrm{eye}(n) - \mathrm{ones}(n)./n; \\ \mathbf{B} &=& -0.5 * \mathbf{J} * \mathbf{M} * \mathbf{J}; \\ && \%\ \text{Find largest eigenvalues+their eigenvectors:} \\ [\mathbf{Q}, \mathbf{L}] &=& \mathrm{eigs}(\mathbf{B}, 2, \text{'LM'}); \\ && \%\ \text{Extract the coordinates:} \\ \mathrm{newy} &=& \mathrm{sqrt}(\mathbf{L}(1,1)). * \mathbf{Q}(:,1); \\ \mathrm{newx} &=& \mathrm{sqrt}(\mathbf{L}(2,2)). * \mathbf{Q}(:,2); \end{array}$$

The vectors newx and newy hold the flattened coordinates.

Computing the geodesic distances between every pair of vertices in a complex triangulated surface, and the spectral decomposition of the corresponding distance matrix is computationally expensive. In practice we select a subset of the vertices and apply the flattening procedure on this subset. The geodesic distance between each pair of points in this set is calculated using the complete surface model. Thus, after proper flattening of the subset of anchor vertices, we need to correctly interpolate the local coordinates in order to find the local map of the rest of the vertices.

As an example, in Figure 3b we show the flattening of a 3D object shown in Figure 3a. Figure 4 presents another example, of flattening a cylinder, which is a developable surface. The flattened texture preserves both local and global features of the surface texture. The surfaces in Figures 3 and 4 include approximately 40,000 vertices. 25×25 points were sampled uniformly on a regular grid in the range-image domain, and the results were obtained in about 30 seconds.

(a) (b)

Fig. 3. An example of a face flattening. (a) A 3D reconstruction of a face. (b) The flattened texture image of the face.

Figure 5 compares the geodesic distance versus the Euclidean distance of the flattened surface shown in Figure 3. The result approximates the diagonal line, which would have been the (geometrically impossible) perfect flattening outcome.

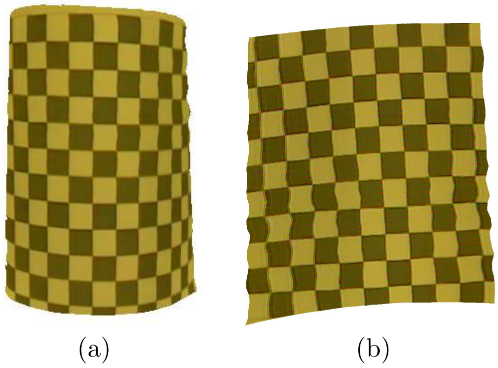

Fig. 4. An example of a cylinder flattening. (a) A 3D reconstruction of a cylinder. (b) The flattened texture image of the cylinder. Both the local and global features of the surface texture are preserved.

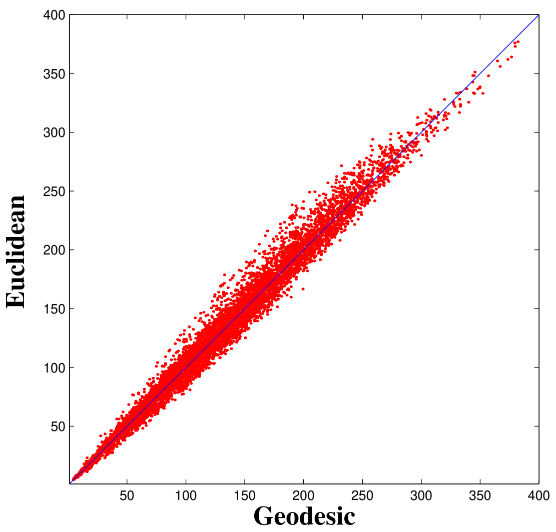

Fig. 5. The geodesic distance on the surface versus the Euclidean distance after flattening. The data corresponds to the face surface shown in Figure 3. The result approximates the diagonal line, which would have been the geometrically impossible perfect flattening outcome.

V. Using MDS for Texture Mapping

Low distortion mapping of a curved surface onto the plane is useful for mapping planar texture onto the surface. After applying classical scaling to the measured geodesic distances between the surface points, we get a 2D flattened version of the surface. We now have the mapping of every point on the 3D surface to its corresponding 2D flattened point. Given a flat texture image we can easily map each point from the 2D flattened map to a point on the texture plane.

Next, we would like to map the texture back to the surface. The technique is straightforward. For each vertex P on the surface

- find the corresponding 2D point in the flattened map,
- translate the 2D coordinates to the texture image coordinates,
- use this point's color as texture.

If the triangulation of the surface is not dense enough, we might encounter aliasing effects. These can be solved by selectively subdividing large triangles into small ones as was implemented in our experiments. Determining the texture in the newly created vertices is done by applying the same subdivision on the 2D corresponding triangle in the texture image plane, and taking the proper interpolated colors from the corresponding image points.

In order to map the texture onto the surface with minimal distortions we take the following steps. First, we flatten the surface by classical scaling applied to the geodesic distances between the selected sub-grid vertices. The flattening procedure gives us a simple mapping between the plane and the surface. Since we consider only a subset of the vertices we need to locally interpolate the map for the rest of the vertices.

VI. Experimental Results

We tested our technique on surfaces obtained using a 3D laser scanner developed in our laboratories. The scanner creates a textured range image on a rectangular grid that is considered as a parameterization plane. A set of vertices in this range image is chosen as an input to the MDS algorithm. After flattening using the selected vertices, as shown in Figure 6, the planar coordinates for the vertices that were not selected, are linearly interpolated using their relative location in the initial parameterization plane.

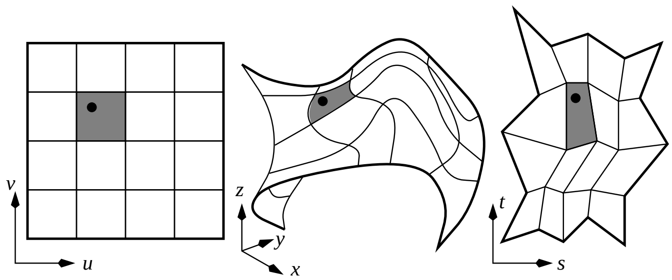

Fig. 6. Left: Given (u, v) parameterization plane. Middle: The surface embedded in R^3. Right: The flattening result as an (s, t) parameterization. The locations of the decimated vertices within each patch in the new (s, t) parameterization are bi-linearly interpolated according to their locations in the given (u, v) parameterization.

For example, in our scanned objects, we associate each bilinear patch defined between neighboring selected vertices on the initial parametric plane (in the range image of our experiments, P_a^R, P_b^R, P_c^R, and P_d^R in Figure 7) to a bilinear patch defined by the corresponding vertices of the flattened surface (P_a^{MDS}, P_b^{MDS}, P_c^{MDS}, and P_d^{MDS} in Figure 7). We next apply a scan conversion procedure to map the points within each MDS planar patch in the (s, t) plane to a corresponding range image bilinear patch in the (u, v) plane. For example, the point P^{MDS} in Figure 7 is mapped to the point P^R. Then, P^R is a mapped from the image plane to the surface point P^S. Using this procedure we can create a flat surface image, like those shown in Figures 3b and 4b. In order to map texture from the plane to specific surface points, a reverse operation is performed.

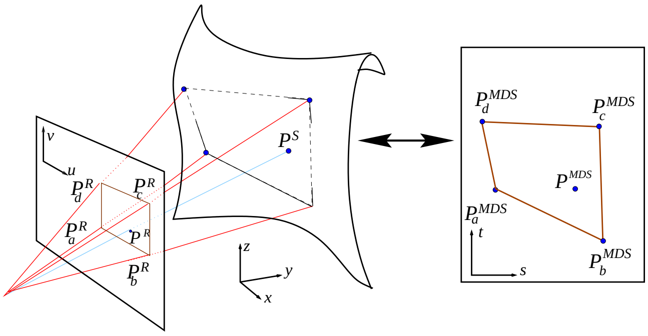

Fig. 7. A method for mapping between the surface and the flat plane. In our experiments the surface is given as a range image, i.e. a specific (u, v) parametric form. Left: The sub-grid rectangle's vertices selected for the MDS procedure. Each vertex corresponds to a surface point in 3D. Right: After flattening, we find four corresponding points which are overlaid on the texture image.

For more complex surfaces, a general geometry sensitive decimation-interpolation method is presented in [10]. It is based on a simple iterative decimation technique, like the sequential polygon reduction algorithm in [20]. First, we flatten the subset vertices that survive the sequential decimation procedure [20], using the unsampled surface for the geodesic distance computation. Next, we plug back the decimated vertices one by one, in a reverse order to their removal sequence, while restricting them to the plane. Each decimated vertex holds the relative distance to its neighbors on the surface. That is, we keep the relationship information while decimating the mesh, and then plug back vertex by vertex to the plane and get the desired interpolation result for non-regular triangulations.

Figure 8 demonstrates a chess-board texture mapped to the face surface with minimal distortions. Figures 9 and 10 present additional examples.

Figure 11 shows the result of a chess-board texture mapped onto a synthetic $(\sin x \sin y)$ graph surface. This example shows the behavior near surface points with positive and negative Mean/Gaussian curvatures. Figure 12 demonstrates the ability of the method to handle (synthetic) objects that are not restricted to be range images. Again, the proposed geodesic distance preserving mapping maintains the general structure of the texture.

Finally, in Figure 13 we compare our MDS texture mapping results with the shape preserving algorithm presented by Floater [11], [12]. As can be seen, the proposed geodesic distance preserving mapping reduces the deformations and better preserves the local and global structure of the texture.

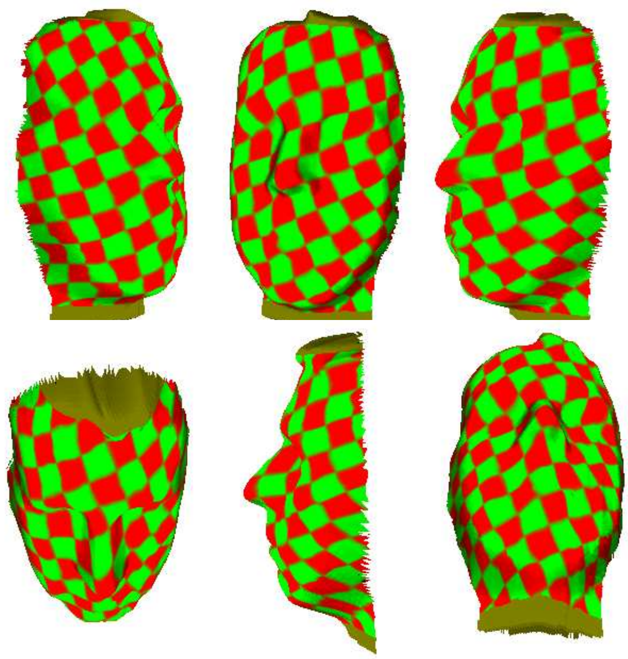

Fig. 8. Chess board texture mapped onto the head object.

(a) (b)

Fig. 9. Texture mapped onto the head object via our global and local structure preserving procedure.

VII. Conclusions

We presented a simple and general structure preserving texture mapping approach with minimal distortions. Using the fast marching method on triangulated domains we efficiently calculate geodesic distances between pairs of surface points. It enables us to achieve accurate measurements that characterize the geometry of the surface, with a reasonable computational complexity. Next, we used the simplest MDS method, known as 'classical scaling', to flatten the surface, and used the flattened surface to back project a flat texture image onto the curved surface. The method is computationally efficient, the surface does not have to be smooth, and boundary conditions are not necessary. It is unnecessary to apply any pre-warping or deformations to the original texture image.

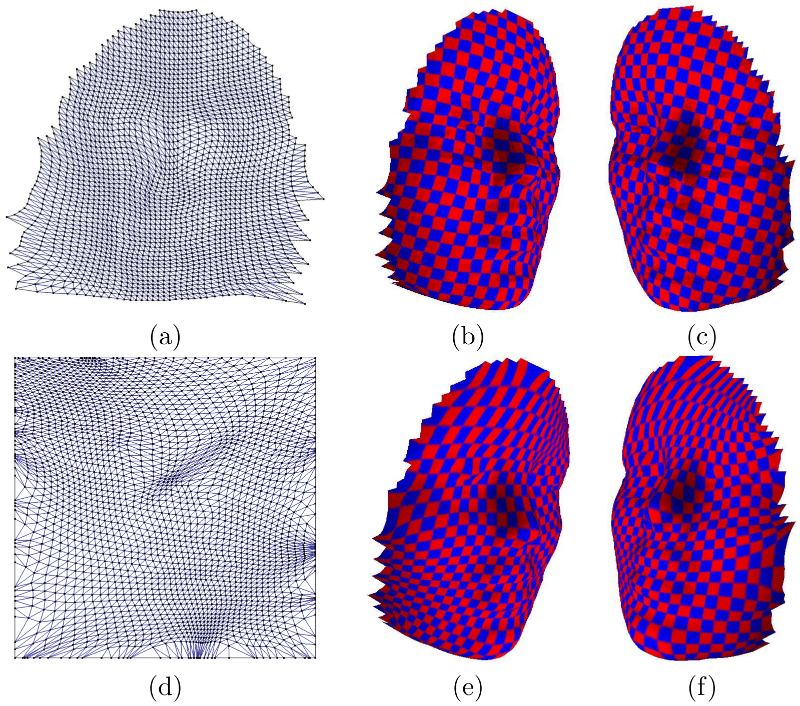

Fig. 13. A comparison between Floater's shape preserving mapping and the MDS mapping. Parametric coordinates and the corresponding texture mapping for a face model created by the proposed method (a, b, c), and Floater's shape preserving procedure (d, e, f).

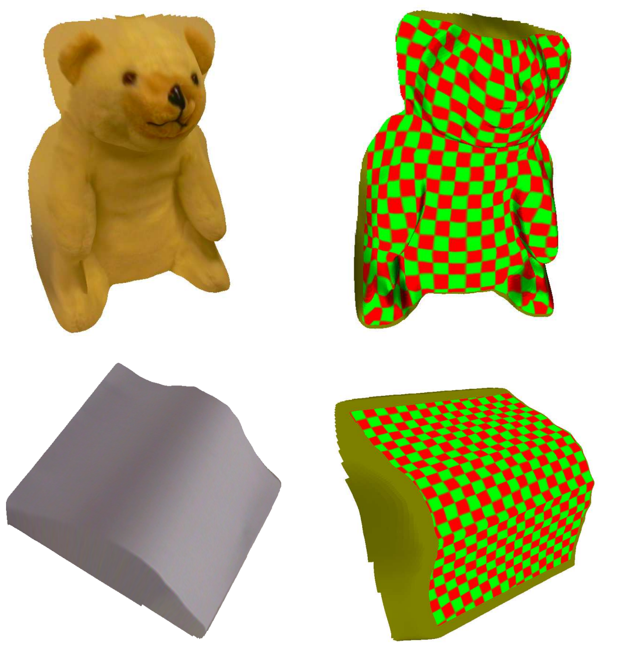

Fig. 10. Texture mapping results: Left: The original objects. Right: The objects textured using the algorithm proposed in this paper.

VIII. Acknowledgments

We thank Haim Schweitzer from the University of Texas at Dallas for raising our interest in MDS, and Gershon Elber and Craig Gotsman from the Technion, Israel, for their valuable help in pointing us to related work. We also thank Ilya Ekshteyn and Vitaly Surazhsky from the

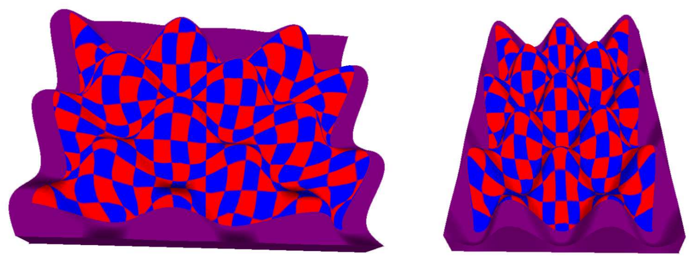

Fig. 11. Feature preserving texture mapping for a synthetic surface with saddle, maximum, and minimum points.

Technion, Israel, for assisting in comparing with Floater's shape preserving method. At the Technion and TAU this research was supported in part by the German Israeli Foundation for Scientific Research and Development (GIF). At TAU this research was also supported in part by the Adams Super-Center for Brain Studies.

References

[1] N. Arad and G. Elber. Isometric texture mapping for free-form surfaces. *Computer Graphics forum*, 16(5):247–256, December 1997.

[2] P. N. Azariadis and N. A. Aspragathos. On using planar developments to perform texture mapping on arbitrarily curved surfaces. *Computers and Graphics*, 24(4):539–554, August 2000.

[3] C. Bennis, J. M. Vézien, and G. Iglésias. Piecewise surface flattening for non-distorted texture mapping. *ACM Computer Graphics*, 25(4):237–246, July 1991.

[4] E. A. Bier and K. S. Sloan. Two-part texture mapping. *IEEE Computer Graphics and Applications*, pages 40–53, September 1986.

[5] J. F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Comm. of the ACM*, 19(10):542–547, October 1976.

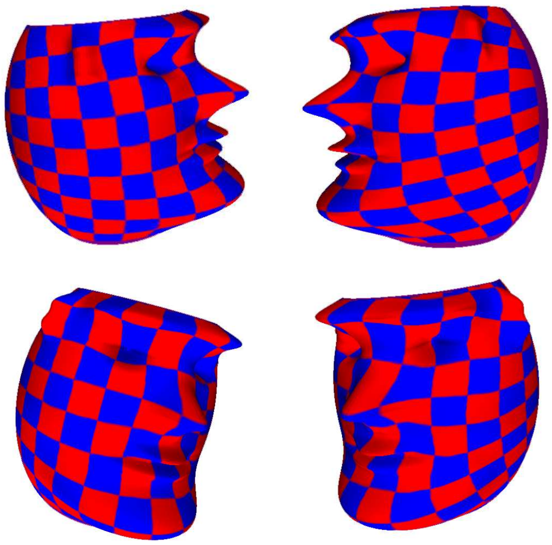

Fig. 12. An example of feature preserving texture mapping on a complex synthetic object.

[6] I. Borg and P. Groenen. *Modern Multidimensional Scaling, Theory and Applications*. Springer Series in Statistics, 1997.

[7] T. F. Cox and M. A. Cox. *Multidimensional Scaling*. CRC Press, 1994.

[8] J.W. Demmel. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, 1997.

[9] M P Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall Inc., New Jersey, 1976.

[10] A. Elbaz (Elad) and R. Kimmel. Canonical representation of 3D objects via multi-dimensional scaling. Center of intelligent systems technical report, Technion, Israel Institute of Technology, 2001.

[11] M. S. Floater. Parametrization and smooth approximation of surface triangulations. *Comp. Aided Geom. Design*, 14:231–250, 1997.

[12] M. S. Floater. Parametric tilings and scattered data approximation. *International Journal of Shape Modeling*, 4:165–182, 1998.

[13] G.H. Golub and C.F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 1989.

[14] N. Greene. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, pages 21–29, November 1986.

[15] R. Kimmel and J. A. Sethian. Computing geodesics on manifolds. *Proc. of National Academy of Sciences*, 95:8431–8435, August 1998.

[16] N. Kiryati and G. Székely. Estimating shortest paths and minimal distances on digitized three-dimensional surfaces. *Pattern Recognition*, 26(11):1623–1637, 1993.

[17] J. B. Kruskal and M. Wish. *Multidimensional Scaling*. Sage Publications, Beverly Hills, California, 1978.

[18] Y. Kurzion, T. Möller, and R. Yagel. Size preserving pattern mapping. In *Proc. of IEEE Visualization*, pages 367–373, Research Triangle Park, NC, October 1998.

[19] B. Lévy and J. L. Mallet. Non-distorted texture mapping for sheared triangulated meshes. In *Proc. of Compuer Graphics, Annual Conference Series*, pages 343–352, 1998.

[20] S. Melax. A simple, fast, and effective polygon reduction algorithm. *Game Developer*, November 1998. http://www.cs.ualberta.ca/ melax/polychop/.

[21] F. Neyret and M. P. Cani. Pattern-based texturing revisited. In *SIGGRAPH 99 Conference Proceedings*, pages 235–242. ACM SIGGRAPH, Addison Wesley, Aug. 1999.

[22] E. Praun, A. Finkelstein, and H. Hoppe. Lapped textures. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 465–470. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[23] Y. Rubner and C. Tomasi. Texture metrics. In *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics*, volume 5, pages 4601–4607, October 1998.

[24] J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, 18(5):401–409, May 1969.

[25] E. L. Schwartz, A. Shaw, and E. Wolfson. A numerical solution to the generalized mapmaker's problem: Flattening nonconvex polyhedral surfaces. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 11(9):1005–1008, September 1989.

[26] J. A. Sethian. *A Review of the Theory, algorithms, and applications of level set methods for propagating interfaces.* Acta Numerica. Cambridge University Press, 1996.

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

[28] E. Wolfson and E. L. Schwartz. Computing minimal distances on polyhedral surfaces. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 11(9):1001–1005, September 1989.

Gil Zigelman was born in Nahariya, Israel, in 1974. Received his B.A. (with honors) in computer science in 1996, and his M.Sc. in 2000, from the Technion – Israel Institute of Technology. His research interests are in image analysis and synthesis, real time systems, computer vision and computer graphics. He was awarded the Gutwirth fellowship. He was invited to present his research results in two Siggraph chapter meetings in Israel. He worked at IBM, Israel, during the years 1996-1998, and has been working at 3DV-Systems since 2001.

Ron Kimmel received his B.Sc. (with honors) in computer engineering in 1986, the M.S. degree in 1993 in electrical engineering, and the D.Sc. degree in 1995 from the Technion – Israel Institute of Technology. During the years 1986-1991 he served as an R&D officer in the Israeli Air Force. During the years 1995-1998 he has been a postdoctoral fellow at Lawrence Berkeley National Laboratory, and the Mathematics Department, University of California, Berkeley. Since 1998, he has been a faculty member of the Computer Science Department at the Technion, Israel. His research interests are in computational methods and their applications including topics in differential geometry, numerical analysis, image processing, computer vision, computer aided design, robotic navigation, and computer graphics. Dr. Kimmel was awarded the Hershel Rich Technion innovation award, the Henry Taub Prize for excellence in research, Alon Fellowship, the HTI Postdoctoral Fellowship, and the Wolf, Gutwirth, Ollendorff, and Jury fellowships. He has been a consultant of HP research Lab in image processing and analysis during the years 1998-2000, and to Net2Wireless/Jigami research group 2000-2001.

Nahum Kiryati was born in Haifa, Israel, in 1958. He received the B.Sc. degree in Electrical Engineering and the Post-B.A. degree in the Humanities from Tel Aviv University, Israel, in 1980 and 1986 respectively. He received the M.Sc. degree in Electrical Engineering in 1988 and the D.Sc. degree in 1991, both from the Technion, Israel Institute of Technology, Haifa, Israel. He was with the Image Science Laboratory, Institute for Communication Technology, ETH-Zurich, Switzerland, and with the Department of Electrical Engineering, Technion, Haifa, Israel. He is now with the Department of Electrical Engineering–Systems, Tel Aviv University. Dr. Kiryati served as a Governing Board member of the International Association for Pattern Recognition (IAPR), and is currently an associate editor of the journal "Pattern Recognition". His research interests are in image analysis and computer vision.