Vol. 14, No. 1: 17–30

Implementing As-Rigid-As-Possible Shape Manipulation and Surface Flattening

Takeo Igarashi
The University of Tokyo, Japan / JST ERATO

Yuki Igarashi
The University of Tokyo, Japan

Abstract. This article provides a description of an "as-rigid-as-possible shape manipulation" implementation that is clearer and easier to understand than the original. While the original paper used triangle-based representation, we use edge-based representation to simplify the coding. We also extend the original algorithm to allow the user to place handles on arbitrary positions of the mesh. In addition, we show that the same algorithm can be used for surface flattening with quality and performance comparable to popular flattening methods.

1. Introduction

The goal of this article is to provide a clear and easy-to-understand description of the implementation of the "as-rigid-as-possible shape manipulation" algorithm [Igarashi et al. 05]. We have modified the original algorithm to improve the user experience and simplify coding. We also show that the same algorithm can be used for surface flattening with quality comparable to popular flattening methods.

The algorithm computes a natural-looking deformation of a two-dimensional shape according to the user's specification. The user places an arbitrary number of point handles on the input shape and manipulates those handles.

© A K Peters, Ltd.

1086-7651/09 $0.50 per page

The system then deforms the shape to follow the handles while keeping the local geometry as rigid as possible. Using this technique, the user can move, rotate, squash, stretch, and deform a model simply by grabbing and moving several handles.

This paper makes the following two small refinements to the original algorithm. First, while the original system allowed the user to place handles only on the mesh vertices, the algorithm described here allows the user to place handles at arbitrary locations inside the mesh triangles. This is a visible and significant improvement from the user's perspective. Second, while the original algorithm used triangle-based representation for computing this distortion, the algorithm described here uses edge-based representation. This second modification is not obvious to the end user but simplifies the coding significantly.

2. Problem Setup

The 2D shape is represented as a 2D triangle mesh. A handle is given as a specific vertex of the mesh (we first describe the method where only mesh vertices can be used as handles as in [Igarashi et al. 05] and then describe a method where the user can place handles at arbitrary locations). Given the handle vertices and target 2D position of these handles, the algorithm computes the 2D position of the mesh vertices (Figure 1).

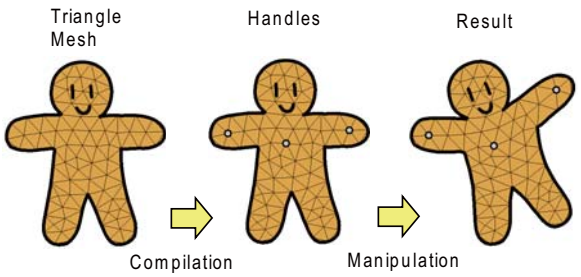


Figure 1. Problem setup: The system takes the triangle mesh and user-specified handles and returns new vertex positions when the handles are moved.

3. Basic Concept

Our algorithm obtains the coordinates of the deformed mesh by minimizing the distortion of each mesh triangle, that is, the difference between the original triangle and the resulting triangle. The question is how to define the

distortion of a triangle. If the triangle only moves and rotates, the distortion should be zero. If the triangle shears or scales, the distortion should represent the amount of shearing and scaling. Ideally, we wish to have a metric that represents such distortion as a linear function of vertex coordinates so that we can obtain the result as a closed-form solution. Unfortunately, no such linear presentation exists [Sorkine et al. 04, Weng et al. 06]. Therefore, we obtain an approximation by decomposing the non-linear optimization problem into a sequence of linear problems (Figure 2). The first step obtains the deformation result allowing free translation, rotation, and uniform scaling, while penalizing non-uniform scaling and shearing. In the resulting shape, the position and rotation are correct but the scale is wrong. The next step then takes the result of this first step (specifically, the orientation of each triangle) and obtains the final deformation result allowing free translation, while penalizing rotation, shearing, and scaling.

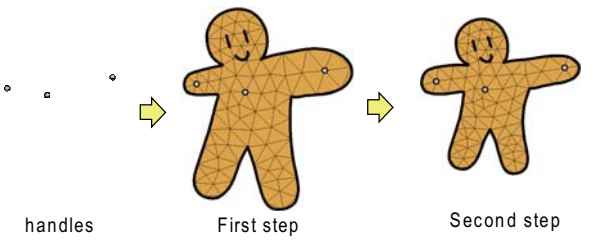


Figure 2. Basic concept: The system first applies an optimization that allows free rotation and scaling and then applies an optimization that allows only translation to adjust the scale.

In the actual coding described below, we represent the distortion for each triangle edge, not each face. This choice is, in a sense, arbitrary. We can use the vertex, the edge, or the face as a basis for computing local distortion and obtain similar results (Figure 3). Vertex-based representation ($v_i - \sum_{j \in N_i} v_j$, second-order differential or Laplacian) is a popular choice for three-dimensional meshes because it naturally represents local bumps and concavities [Sorkine et al. 04]. However, its meaning is less intuitive for 2D meshes, and it causes difficulty when assigning rigidity to the mesh. What does "this vertex is more rigid" actually mean? A triangle-based representation provides a much more intuitive definition. It is very natural to imagine rotating and scaling individual triangles and to see a mesh as an assembly of individual triangles. It is also natural to say "this triangle is more rigid." This is the reason why the original paper [Igarashi et al. 05] used triangle-based representation. The problem is that the actual coding becomes somewhat complicated because we need to examine three edges of a triangle to define its distortion. This leads to our choice of edge-based representation ($v_i - v_j$,

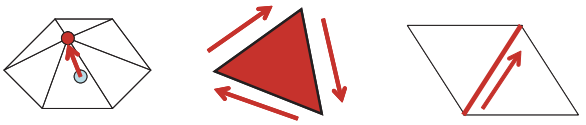


Figure 3. Different representations.

first-order differential). This is somewhat less intuitive for first-time readers, but the coding becomes much simpler and more straightforward.

4. Algorithm

4.1. Baseline Algorithm

We first describe the baseline algorithm to clarify the idea before describing the actual implementation . Our goal is to find vertex coordinates that minimize the distortion of edge vectors under the given handle constraints. To do so, we solve the following equation in a least-squares sense:

$$v'_j - v'_i = v_j - v_i \ (\{i,j\} \in E) \ \text{ subject to constraints } \ v'_i = C_i \quad (i \in C),$$

where v_i is the vertex coordinate of the original rest shape, v'_i is the vertex coordinate of the deformed shape, E is a set of edges (all edges are directed), C is a set of handles, and C_i is the handle coordinate. Combining these into a single cost function, we obtain the following least-squares minimization problem:

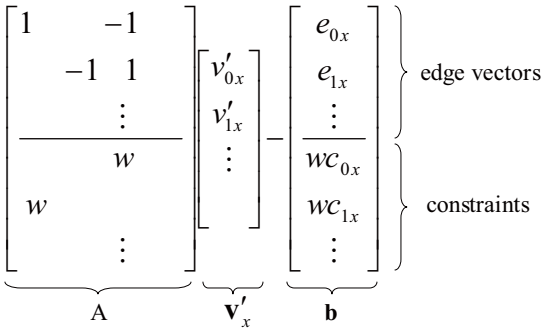
$$\arg\min_{v' \in V} \left\{ \sum_{\{i,j\} \in E} \left\| (v'_j - v'_i) - (v_j - v_i) \right\|^2 + w \sum_{i \in C} \left\| v'_i - C_i \right\|^2 \right\}, \qquad (1)$$

where w is a weight factor; we currently use a value of $w = 1000$.

This is a linear optimization problem for which we can obtain the result as a closed form solution by solving a linear matrix equation. The derivation is as follows. We first rewrite Equation (1) in matrix form:

$$\arg\min_{\mathbf{v}'} \left\| A\mathbf{v}' - \mathbf{b} \right\|^2.$$

We then solve this separately for the x- and y-components. The entries of the matrix equation for the x-component, where e is an edge vector, are as follows:



If e_0 starts at v_0 and ends at v_1, then $e_0 = v_1 - v_0$. Matrix A is identical for both x- and y-components. The result of this least-squares minimization is obtained by solving a normal equation:

$$A^{\mathrm{T}} A \mathbf{v}' = A^{\mathrm{T}} \mathbf{b}.$$

4.2. First Step: Similarity Transformation

The problem with this formalization is that it does not allow rotation [Sorkine et al. 04]. Ideally, the cost function should be zero if the shape simply rotates without any distortion (rotation invariance). However, the cost function to be minimized in Equation (1) becomes non-zero when the mesh (edge) rotates, because the rotated vector minus the original vector is non-zero. This is a fundamental limitation of the simple linear representation, and many attempts have been made to achieve rotation invariance. Most approaches explicitly compute rotations beforehand and use the rotated differentials on the right-hand side of Equation (1) [Zayer et al. 05]. That approach is not applicable to the circumstances of our problem because the user's handles do not have orientation information. Recent approaches have used non-linear solvers [Weng et al. 06], but they require iterative computation and are not suitable for sudden, large handle displacements.

Our approach is to use an implicit optimization method [Sorkine et al. 04] to rotate the original local differential (edge vector, right-hand side of Equation (1)) by a rotation matrix T_k that maps the nearby vertices v around the edge to the new locations v'. That is, we represent the (unknown) rotation T_k as a function of (unknown) positions of deformed vertices v', multiply them by the original edge vectors, and then compute these unknown vertex positions all together during optimization. As a result, the rotation (which is actually a similarity transform that allows free rotation and uniform scaling) in 2D can

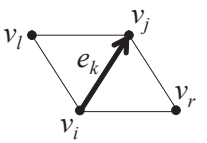


Figure 4. Edge neighbors.

be represented in a linear form,

$$T_k = \left\{ \begin{array}{cc} c_k & s_k \\ -s_k & c_k \end{array} \right\}, \tag{2}$$

and thus we can solve the system as a linear optimization problem[1]. The derivation is as follows.

The rotation matrix T_k is given as a transformation that maps the vertices around the edge to new positions as closely as possible in a least-squares sense. We sample four vertices around the edge as a context to derive the local transformation T_k. It is possible to sample an arbitrary number of vertices greater than three here, but four is the most straightforward, and we have found that it produces good results. An exception applies to edges on the boundary. In those cases, we only use three vertices to compute T_k:

$$T_k = \arg\min_{T_k} \sum_{v \in N(e_k)} \left\| T_k v - v' \right\|^2, \tag{3}$$

where (see Figure 4)

$$N(e_k) = \{v_i, v_j, v_l, v_r\}.$$

Using Equation (2), Equation (3) can be rewritten as

$$\begin{aligned} \{c_k, s_k\} &= \arg\min_{c_k, s_k} \sum_{v \in N(e_k)} \left\| \left[\begin{array}{cc} c_k & s_k \\ -s_k & c_k \end{array} \right] \left[\begin{array}{c} v_x \\ v_y \end{array} \right] - \left[\begin{array}{c} v'_x \\ v'_y \end{array} \right] \right\|^2 \\ &= \arg\min_{c_k, s_k} \sum_{v \in N(e_k)} \left\| \left[\begin{array}{cc} v_x & v_y \\ v_y & -v_x \end{array} \right] \left[\begin{array}{c} c_k \\ s_k \end{array} \right] - \left[\begin{array}{c} v'_x \\ v'_y \end{array} \right] \right\|^2 \end{aligned}$$

[1]In 3D, even similarity transformations do not have a linear parameterization, and it is necessary to find the best rotations iteratively [Sorkine and Alexa 07].

$$
\begin{aligned}
&= \arg\min_{c_k, s_k} \left\| \begin{bmatrix} v_{ix} & v_{iy} \\ v_{iy} & -v_{ix} \\ v_{jx} & v_{jy} \\ v_{jy} & -v_{jx} \\ v_{lx} & v_{ly} \\ v_{ly} & -v_{lx} \\ v_{rx} & v_{ry} \\ v_{ry} & -v_{rx} \end{bmatrix} \begin{bmatrix} c_k \\ s_k \end{bmatrix} - \begin{bmatrix} v'_{ix} \\ v'_{iy} \\ v'_{jx} \\ v'_{jy} \\ v'_{lx} \\ v'_{ly} \\ v'_{rx} \\ v'_{ry} \end{bmatrix} \right\|^2 \\
&= \arg\min_{c_k, s_k} \left\| G_k \begin{bmatrix} c_k \\ s_k \end{bmatrix} - \begin{bmatrix} v'_{ix} \\ v'_{iy} \\ \vdots \end{bmatrix} \right\|^2
\end{aligned}
$$

This is again a standard least-squares problem and the solution is given as

$$
\begin{bmatrix} c_k \\ s_k \end{bmatrix} = \left(G_k^t G_k\right)^{-1} G_k^t \begin{bmatrix} v'_{ix} \\ v'_{iy} \\ \vdots \end{bmatrix}.
$$

This shows that T_k is linear in v_i, v_j, v_l, and v_r. We now apply this (implicitly-defined) local transformation T_k to the original edge vector $(v_j - v_i)$ in Equation (1).

$$
\arg\min_{v'} \left\{ \sum_{\{i,j\} \in E} \left\| \left(v'_j - v'_i\right) - T_{ij} \left(v_j - v_i\right) \right\|^2 + w \sum_{i \in C} \left\| v'_i - C_i \right\|^2 \right\} \quad (4)
$$

The terms inside of the left-hand summation are transformed as follows:

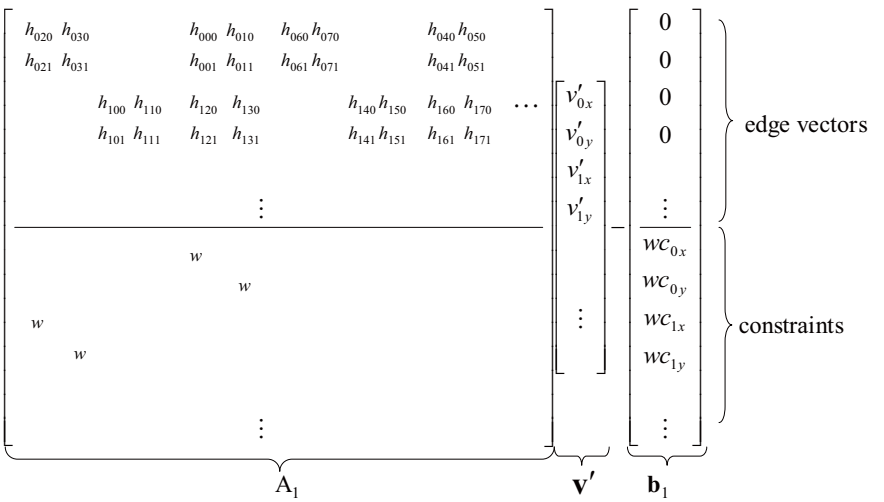
$$
\begin{aligned}
& (v'_j - v'_i) - T_{ij}(v_j - v_i) \\
= \; & (v'_j - v'_i) - \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix} e_k \\
= \; & (v'_j - v'_i) - \begin{bmatrix} e_{kx} & e_{ky} \\ e_{ky} & -e_{kx} \end{bmatrix} \begin{bmatrix} c_k \\ s_k \end{bmatrix}
\end{aligned}
$$

$$
\begin{aligned}
&= (v'_j - v'_i) - \begin{bmatrix} e_{kx} & e_{ky} \\ e_{ky} & -e_{kx} \end{bmatrix} \left(G_k^t G_k\right)^{-1} G_k \begin{bmatrix} v'_{ix} \\ v'_{iy} \\ \vdots \end{bmatrix} \\
&= \left(\begin{bmatrix} -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} e_{kx} & e_{ky} \\ e_{ky} & -e_{kx} \end{bmatrix} \left(G_k^t G_k\right)^{-1} G_k \right) \\
&\qquad \times \begin{bmatrix} v'_{ix} \\ v'_{iy} \\ \vdots \end{bmatrix} \\
&= \begin{bmatrix} h_{k00} & h_{k10} & h_{k20} & h_{k30} & h_{k40} & h_{k50} & h_{k60} & h_{k70} \\ h_{k01} & h_{k11} & h_{k21} & h_{k31} & h_{k41} & h_{k51} & h_{k61} & h_{k71} \end{bmatrix} \begin{bmatrix} v'_{ix} \\ v'_{iy} \\ v'_{jx} \\ v'_{jy} \\ v'_{lx} \\ v'_{ly} \\ v'_{rx} \\ v'_{ry} \end{bmatrix}
\end{aligned}
$$

Now Equation (4) can be written in the following matrix form:

$$\arg\min_{\mathbf{v}'} \left\| A_1 \mathbf{v}' - \mathbf{b}_1 \right\|^2.$$

The entries of the matrix equation look like



Here, we compute x- and y-components together and obtain the answer by solving a normal equation,

$$A_1^t A_1 \mathbf{v}^{'} = A_1^t \mathbf{b}_1. \tag{5}$$

This gives almost the right answer, but a problem remains because this solution allows free scaling. If the user moves the handles far apart, the resulting shape inflates; when the handles are moved close together, the shape deflates. See the middle image in Figure 2. Therefore, the next step is to adjust the scale.

4.3. Second Step: Scale Adjustment

The second step takes the rotation information from the result of the first step (i.e., computing the explicit values of T_k' and normalizing them to remove the scaling factor), rotates the original edge vectors e_k by the amount T_k', and then solves Equation (1) using the original rotated edge vectors. That is, we compute the rotation of each edge by using the result of the first step,

$$T_k' = \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix}, \qquad \begin{bmatrix} c_k \\ s_k \end{bmatrix} = \left(G_k^t G_k\right)^{-1} G_k^t \begin{bmatrix} v_i' \\ v_j' \\ v_l' \\ v_r' \end{bmatrix} \tag{6}$$

and then normalize it

$$T_k' = \frac{1}{c_k^2 + s_k^2} \begin{Bmatrix} c_k & s_k \\ -s_k & c_k \end{Bmatrix}.$$

We compute T_k' for each edge and then insert this transformation into Equation (1):

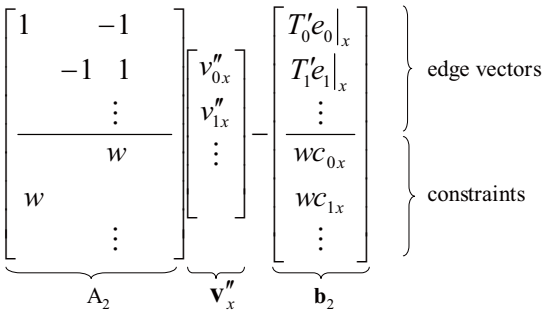
$$\arg\min_{v^{''} \in V} \left\{ \sum_{\{i,j\} \in E} \left\| (v_j^{''} - v_i^{''}) - T_{ij}'(v_j - v_i) \right\|^2 + w \sum_{i \in C} \left\| v_i^{''} - C_i \right\|^2 \right\}$$

($T_{ij}'$is constant in 2nd step).

Again, this is linear in $\mathbf{v}^{''}$, so we can rewrite this in matrix form as

$$\arg\min_{\mathbf{v}^{''}} \left\| A_2 \mathbf{v}^{''} - \mathbf{b}_2 \right\|^2$$

We solve this separately for the x- and y-components. The entries of the matrix equation for the x-component look like



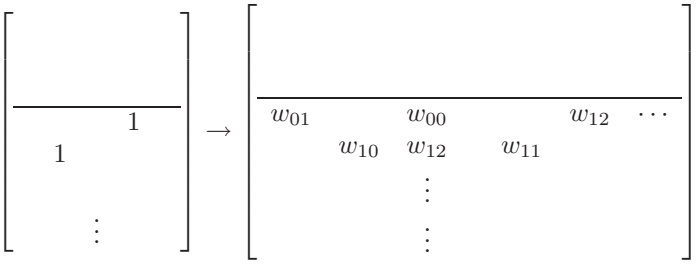
Matrix A_2 is identical for both x- and y-components. We obtain the answer by solving a normal equation,

$$A_2^t A_2 \mathbf{v}'' = A_2^t \mathbf{b}_2. \tag{7}$$

The final result appropriately fixes the unwanted scaling observed in the result of the first step (Figure 2 right).

5. Allowing Handles on Arbitrary Positions in the Mesh

The above algorithm allows the user to place handles only on vertex positions. The user cannot place a handle on an arbitrary position inside a triangle. This is because the algorithm represents the constraint as an association between a handle and a single vertex ($v_i = c_i$). This constaint can be easily fixed by representing the handle location by barycentric coordinates ($w_{i0}v_{i0} + w_{i1}v_{i1} + w_{i2}v_{i2} = c_i$). This leads to a very simple modification of the overall procedure: simply modify the bottom half of A_1 and A_2 as follows:



6. Implementation Notes

The system solves two sparse linear matrix equations, Equations (5) and (7), using a fast solver [Davis 03, Toledo et al. 03] each time the user moves the handles (interactive update). We accelerate this computation by applying pre-computations when the original rest shape is defined (registration) and when the handles are added or removed (compilation). In the registration step, we compute the top half of A_1 and A_2 (we call the results L_1 and L_2), as well as $L_1^t L_1$ and $L_2^t L_2$. We also compute $(G^t G)^{-1} G$ in Equation (6). In the compilation step, we first compute the bottom half of A_1 and A_2 (we call the results C_1 and C_2) as well as $C_1^t C_1$ and $C_2^t C_2$. We then factor $A_1^t A_1 = L_1^t L_1 + C_1^t C_1$ and $A_2^t A_2 = L_2^t L_2 + C_2^t C_2$. These matrices remain constant and only $\mathbf{b}_1$ and $\mathbf{b}_2$ change during the interactive update, so we simply apply back substitution to solve the matrix equations reusing the factorization results.

For the sake of simplicity, the energy function described here does not have weights for edges. This works well for evenly triangulated meshes but can cause a problem when the mesh is not even. In that case, it is necessary to give weights to edges in the energy function (multiply each row of A and $\mathbf{b}$ with the corresponding edge weight). We recommend using the cotangent weight formula $w_{ij} = \frac{1}{2}(\cot \angle ilj + \cot \angle irj)$ [Sorkine and Alexa 07] (see Figure 4).

We use a standard sparse linear-matrix solver and do not exploit the special structure of the matrix other than its sparseness. The problem is well-conditioned and no degeneracy occurs as long as more than two handles are provided and there are no co-incident handles or degenerated triangles in the mesh. The algorithm is always stable regardless of the position of the handles because of the nature of least-squares formulation. It works without failure even in cases of extreme deformations. However, extreme deformations can cause fold over of the mesh, reversing some of the triangles.

7. As-Rigid-as-Possible Surface Flattening

Our deformation algorithm can be used for surface flattening (unwrapping) with a slight modification. The basic concept behind the 2D deformation algorithm is to minimize the difference between a triangle in the original 2D mesh and one in the deformed 2D mesh (i.e., to compute the as-rigid-as-possible mapping of the 2D triangle). We can use the identical measurement to calculate the difference between the 3D triangle and the 2D triangle, which results in a simple flattening method (i.e., computing the as-rigid-as-possible mapping of the 3D triangle to the 2D triangle while preserving mesh connectivity).

The above edge-based algorithm can be used for surface flattening by making the following changes. For each edge e_i, we locally flatten the edge and two adjacent triangles, obtaining 2D coordinates of the edge (a, b) and

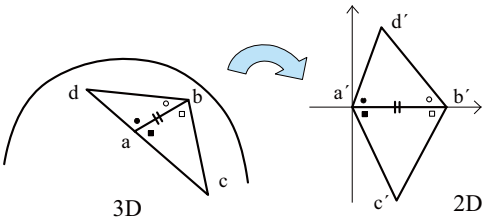
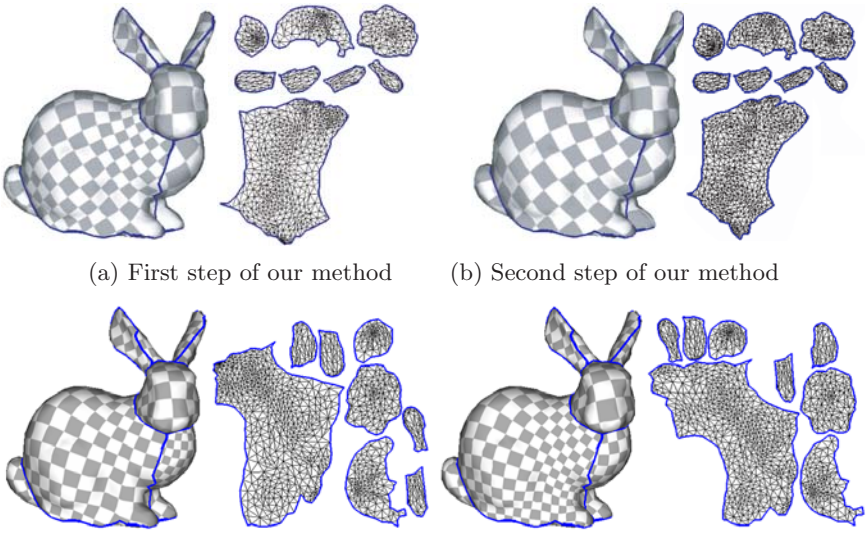


Figure 5. Local flattening of an edge and adjacent faces.

nearby vertices (c, d). Specifically, we define $a' = (0, 0)$, $b' = (|a - b|, 0)$, $c' = (|c - a| \cos \angle cab, |c - a| \sin \angle cab)$, $d' = (|d - a| \cos \angle dab, |d - a| \sin \angle dab)$ and use these values (a', b', c', d') in the optimization (Figure 5). It may seem somewhat counterintuitive, but the above algorithm does not use any mesh-connectivity information other than computing edge vectors and Tcs_k, making it possible to use this approach. It is necessary to constrain at least two vertices to solve the optimization problem. The choice of the two constraints is arbitrary, but we choose to use the end points of an edge at the center of the mesh.



(a) First step of our method (b) Second step of our method

(c) LSCM [Lévy et al. 02] (d) ABF++ [Sheffer et al 05]

Figure 6. Comparison of our method to existing flattening methods. Note that pattern size on the 3D surface is more uniform in our method.

It is difficult to compare the quality of flattening with other methods because the goals vary depending on the target application. Experience has shown that our algorithm generates results that are almost indistinguishable from those of popular flattening methods [Sheffer et al. 06] for almost developable meshes. When the mesh is far from developable, it respects scale consistency while sacrificing conformality (Figure 6). It is possible to improve the quality of our method by using other interactive refinement [Weng et al. 06, Liu et al. 08]. In terms of performance, our current implementation is comparable to state-of-the-art methods [Sheffer et al. 06] after introducing hierarchical methods and optimizing the computation for the particular matrix structure.

We do not claim that our algorithm is better than existing flattening algorithms, but we believe that our method can be a choice when scale consistency is more important. For example, our approach can be useful for designing cloth patterns by flattening a target 3D geometry [Julius et al. 05, Igarashi and Igarashi 08], because cloth should not stretch or compress too much. We also believe that our particular formulation (computing the as-rigid-as-possible mapping) could provide a basis for specific extensions, such as locally controlled rigidity, by changing edge weights.

References

[Davis 03] Timothy A. Davis. "Umfpack Version 4.1 User Guide." Technical report TR-03-008, University of Florida, 2003.

[Igarashi and Igarashi 08] Yuki Igarashi and Takeo Igarashi. "Pillow: Interactive Flattening of a 3D Model for Plush Ttoy Design." In *SmartGraphics*, edited by A. Butz, B. Fisher, A Krüger, P Olivier, and M. Christie, pp. 1–7, Lecture Notes in Computer Science 5166. Berlin: Springer-Verlag, 2008.

[Igarashi et al. 05] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. "As-Rigid-as-Possible Shape Manipulation." *Proc. SIGGRAPH '05, Transactions on Graphics* 24: 3 (2005), 1134–1141.

[Julius et al. 05] Dan Julius, Vladislav Kraevoy, and Alla Sheffer. "D-Charts: Quasi-Developable Mesh Segmentation." *Computer Graphics Forum (Proceedings of Eurographics 2005)* 24:3 (2005), 981-990.

[Lévy et al. 02] Bruno Lévy, Petitjean Sylvain, Ray Nicolas and Maillot Jerome. "Least Squares Conformal Maps for Automatic Texture Atlas Generation." *Proc. SIGGRAPH '02, Transactions on Graphics* 21:3 (2002), 362–371.

[Liu et al. 08] Ligang Liu, Lei Zhang, Yin Xu, Craig Gotsman, and Steven J. Gortler. "A Local/Global Approach to Mesh Parameterization." *Computer Graphics Forum, Symposium on Geometry Processing* 27:5 (2008), 1495–1504.

[Sheffer et al 05] Alla Sheffer, Bruno Lévy, Maxim Mogilnitsky, and Alexander Bogomyakov. "ABF++: Fast and Robust Angle Based Flattening." *ACM Transactions on Graphics* 24:2 (2005), 311–330.

[Sheffer et al. 06] Alla Sheffer, Emil Praun, and Kenneth Rose. *Mesh Parameterization Methods and Their Applications.* Hannover, MA: Now Publishers,Inc., 2006.

[Sorkine and Alexa 07] Olga Sorkine and Marc Alexa. "As-Rigid-as-Possible Surface Modeling." In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing 2007* pp.109–116. Aire-la-Ville, Switzerland: Eurographics Assoc., 2007.

[Sorkine et al. 04] Olga Sorkine, Yaron Lipman, Daniel Cohen-Or, Marc Alexa, Christian Rössl, and Hans-Peter Seidel. "Laplacian Surface Editing." In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pp.179–188. Aire-la-Ville, Switzerland: Eurographics Assoc., 2004.

[Toledo et al. 03] Sivan Toledo, Doron Chen and Vladimir Rotkin. "TAUCS. A Library of Sparse Linear Solvers." http://www.tau.ac.il/~stoledo/taucs/

[Weng et al. 06] Yanlin Weng, Weiwei Xu, Yanchen Wu, Kun Zhou, and Baining Guo. "2D Shape Deformation Using Nonlinear Least Squares Optimization." *The Visual Computer* 22:9-11 (2006), 653–660.

[Zayer et al. 05] Rhaleb Zayer, Christian Rössl, Zachi Karni, and Hans-Peter Seidel. "Harmonic Guidance for Surface Deformation." *Computer Graphics Forum (Proceedings of Eurographics 2005)* 24:3 (2005) 601–609.

Web Information:

Additional material can be found online at http://jgt.akpeters.com/papers/Igarashi Igarashi09/ and http://www-ui.is.s.u-tokyo.ac.jp/~takeo/.

Takeo Igarashi, The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, 113-0033, Tokyo, Japan (takeo@acm.org)

Yuki Igarashi, The University of Tokyo, 4-6-1 Komaba, Meguro-ku, 153-8904, Tokyo, Japan (yukim@acm.org)

Received December 8, 2008; accepted in revised form April 17, 2009.