# A Constrained Least Squares Approach to Interactive Mesh Deformation

Yasuhiro Yoshioka
The University of Tokyo
yoshioka@nakl.t.u-tokyo.ac.jp

Hiroshi Masuda
The University of Tokyo
masuda@nakl.t.u-tokyo.ac.jp

Yoshiyuki Furukawa
Institute of Advanced Industrial Science and Technology
y-furukawa@aist.go.jp

## Abstract

*In this paper, we propose a constrained least squares approach for stably computing Laplacian deformation with strict positional constraints. In the existing work on Laplacian deformation, strict positional constraints are described using large values of least squares weights, which often cause numerical problems when Laplacians are described using cotangent weights. In our method, we describe strict positional constraints as hard constraints. We solve the combination of hard and soft constraints by constructing a typical least squares matrix form using QR decomposition. In addition, our method can manage shape deformation under over-constraints, such as redundant and conflicting constraints. Our framework achieves excellent performance for interactive deformation of mesh models.*

## 1 Introduction

Mesh deformation is useful in a variety of applications in computer graphics and computer-aided design. In recent years many discrete deformation techniques based on Laplacian mesh representation have been published [20]. They can support interactive work by encoding differential properties and positional constraints in a linear system.

In most existing work [17, 21], Laplacian operators are described by the uniform weighting method. This approach is numerically stable, but may produce distorted shapes when triangles in the mesh are not uniformly constructed. Meyer et al. [16] showed that the mean curvature can be approximated using Laplacians with cotangent weights. Their work illustrates that the uniform weights preserve the mean curvature at each vertex only when the mesh consists of conformal and uniform triangles. Obviously the cotangent weights are better than the uniform weights. However, the cotangent weights may cause numerical problems because typical linear solvers fail to solve linear systems for badly shaped meshes. To solve Laplacians with cotangent weights, Botsch and Kobbelt [5] remeshed the mesh models so that the Voronoi area was nearly equal at each vertex. This approach is useful, but it is difficult to faithfully preserve the details of shapes by automatic remeshing.

We propose a novel constrained least-squares approach in this paper. This method allows us to stably solve a linear system of Laplacians defined by cotangent weights without remeshing. In our method, positional constraints are described as hard constraints. Then a combination of hard and soft constraints is converted to a typical least squares system using QR decomposition. This method computes vertex positions that preserve differential properties stably and efficiently.

A well-known problem of hard constraints is over-constraints. If over-constraints are involved, the solver halts the computation. We show how our framework resolves over-constrained situations, which include redundant and conflicting constraints.

In addition to stability, our method satisfies positional constraints precisely. In the existing methods, positional constraints are approximately solved by placing large weights in the least squares system. As the values of weights become increasingly large, the solver satisfies positional constraints more strictly, but it becomes numerically more unstable. satisfied.

In the following section, we review the related work on 3D shape deformation. In Section 3, we describe our mesh deformation framework with hard constraints. In Section 4, we evaluate the stability and performance of the proposed method through experiments. We conclude the paper in Section 5.

## 2 Related Work

Interactive mesh editing techniques have been intensively studied [20]. Such research aims to develop modeling tools for intuitively modifying free-form surfaces while preserving the geometric details.

In typical interactive mesh editing, the user first selects the region to be fixed, and then the vertices to be used as the manipulation handle. When the user drags the positions of handle vertices on the screen, the surface is deformed according to the handle manipulation.

There are several types of approach for mesh editing: free-from deformation (FFD), multiresolution mesh editing, and partial differential equation (PDE)-based mesh editing.

FFD methods are very popular approaches. They modify shapes implicitly by deforming 3D space in which objects are located [6, 15, 19]. However, it is difficult to manage geometric constraints defined on vertices, edges and faces, because FFD does not directly work on geometric shapes.

Multiresolution approaches [8, 10, 11, 12, 27] decompose a surface into a base mesh and several levels of details. Mesh editing can be performed at various resolutions. Botsch et al. [4, 5] applied this technique to interactive mesh editing. A mesh model is decomposed into two-level resolutions and the smooth base is interactively deformed using energy minimization techniques. Geometric details are then recovered on the modified smooth shape.

PDE-based approaches directly deform the original mesh so that the differential properties are preserved. The positions of the handle and fixed vertices and the differential properties of the surface are treated as boundary conditions during the editing processes. Laplacians are most commonly used to represent differential properties.

PDE-based approaches are categorized as non-linear and linear methods. Non-linear methods solve Laplacian or Poisson equations using non-linear iterative solvers [2, 7, 18, 22, 24]. These methods produce fair surfaces, but they are time-consuming and it is difficult to deform shapes interactively.

Linear PDE-based approaches encode Laplacians and positional constraints in a linear system and obtain the deformed shapes by solving the linear system [4, 21]. A discrete Laplacian is defined at each vertex by the weighted sum of difference vectors between the vertex and its adjacent neighbors. When the weights in the sum are represented using the Voronoi area and cotangents [16], the Laplacian vector approximates the mean curvature at the vertex.

Yu et al. [25] introduced a similar technique called Poisson editing, which manipulates the gradients of the coordinate functions $(x, y, z)$ of the mesh. The vertex positions are calculated by solving discrete Poisson equations. Zhou et al. [26] proposed volumetric Laplacians to preserve the volumetric properties for large deformations. Nealen et al. [17] introduced a sketch-based interface on the Laplacian framework.

Since Laplacian vectors are defined in the local coordinate systems [1, 21], one or more vertices must be specified in the global coordinate system to determine all vertex positions. Therefore, the total number of Laplacian vectors and positional constraints is greater than that of the unknowns. Sorkine et al. [21] solved this over-constrained problem approximately as a least squares system.

Several authors have discussed methods for rotating Laplacian vectors according to the deformation of surfaces. Lipman et al. [13] estimated the local rotations on the underlying smooth surface. Sorkine et al. [21] approximated rotations as linear forms and solves them using the least squares method. Lipman et al. [14] also proposed a rotation-invariant method. They encoded rotations and positions in two separate linear systems, in which the local frame on each vertex was represented as the differences between the local frame and adjacent ones, and the vertex positions as relative coordinates on the local frames.

# 3 Framework with Hard Constraints

## 3.1 Preliminaries

Let $M = (V, E, F)$ be a given triangular mesh with $n$ vertices. $V$, $E$ and $F$ are the set of vertices, edges and faces, respectively. Each vertex has a three-dimensional coordinate $\mathbf{p}_i = (\mathbf{p}_{\mathbf{x}i}, \mathbf{p}_{\mathbf{y}i}, \mathbf{p}_{\mathbf{z}i})$. A discrete Laplacian $\delta_i = (\delta_{\mathbf{x}i}, \delta_{\mathbf{y}i}, \delta_{\mathbf{z}i})$ is defined as:

$$\delta_i = \sum_{j \in N(i)} w_{ij}(\mathbf{p}_i - \mathbf{p}_j), \tag{1}$$

where $N(i) = \{j | (i, j) \in E\}$ is the set of immediate neighbors of vertex $i$.

We can represent (1) as a matrix equation:

$$L\mathbf{p} = \delta. \tag{2}$$

We can describe positional constraints as linear equations. Constraints on the position of a vertex, a point on an edge and a point on a face can be shown in (3), (4) and (5), respectively:

$$\mathbf{p}_i = \mathbf{u} \tag{3}$$

$$t\mathbf{p}_i + (1-t)\mathbf{p}_j = \mathbf{u}' \tag{4}$$

$$s\mathbf{p}_i + t\mathbf{p}_j + (1-s-t)\mathbf{p}_k = \mathbf{u}'', \tag{5}$$

where $u, u', u'' \in \mathbb{R}^3$ are certain constant positions.

In addition, we introduce vectorial constraints as follows:

$$\mathbf{p}_i - \mathbf{p}_j = \mathbf{v}, \tag{6}$$

where $v \in \mathbb{R}^3$ is a certain constant vector. This type of equation constrains the relative positions of vertices. For example, when the relative positions are defined on edges

of a triangle, the mesh is deformed so that the triangle has the same shape.

When all of these constraints are defined as soft constraints, they are solved by a least squares method, such as:

$$A^T A\mathbf{x} = A^T\mathbf{c}, \tag{7}$$

where $\mathbf{x} = \mathbf{p}$, $A = \begin{bmatrix} L \\ \vdots \end{bmatrix}$ and $\mathbf{c} = \begin{bmatrix} \delta \\ \vdots \end{bmatrix}$. The positions of the handle vertices are contained in the vector $c$.

Since $A^T A$ is a sparse symmetrical positive definite matrix, (7) can be very efficiently solved using Cholesky factorization [3, 20, 23]. After the matrix is factorized once, $\mathbf{x}$ is repeatedly calculated according to the positions of the handle vertices.

The method for determining $w_{ij}$ is important for numerical stability and quality. A simple method is the following uniform weights approach:

$$w_{ij} = \frac{1}{|N(i)|}. \tag{8}$$

When badly shaped long triangles do not exist and the sizes of triangles are nearly equal, the uniform weights work well. However, this weighting method does not produce good results in general cases.

The following cotangent weights approach is a better approximation, because $\delta_i$ approximates the local normal direction and the local mean curvature [22]:

$$w_{ij} = \frac{1}{2Area(i)}(\cot\alpha_{ij} + \cot\beta_{ij}), \tag{9}$$

where $Area(i)$ is the Voronoi area of a vertex $i \in V$, and $\alpha_{ij}$ and $\beta_{ij}$ are the angles opposite to edge $(i, j)$[16].

However, when the mesh contains triangles that are too long or too small, Laplacians constructed by cotangent weights may cause numerical problems. Figure 2 shows such an example. Since this mesh has many badly shaped triangles, the linear solver fails to calculate deformed shapes that preserve Laplacians with cotangent weights. We need to remove this side effect of cotangent weights. One solution is to remesh models so that each triangle has the same Voronoi area [5]. However, remeshing is not always useful, because it may change the details of shapes.

Therefore, we introduce hard constraints in the deformation framework and solve them using the equality-constrained least squares method. We observe that the reason why this instability occurs is that relatively large weights need to be specified on positional constraints. By treating them as hard constraints, the numerical calculation is stabilized. In addition, since hard constraints are precisely satisfied, our framework is suitable for applications in which positional constraints should be precisely satisfied.

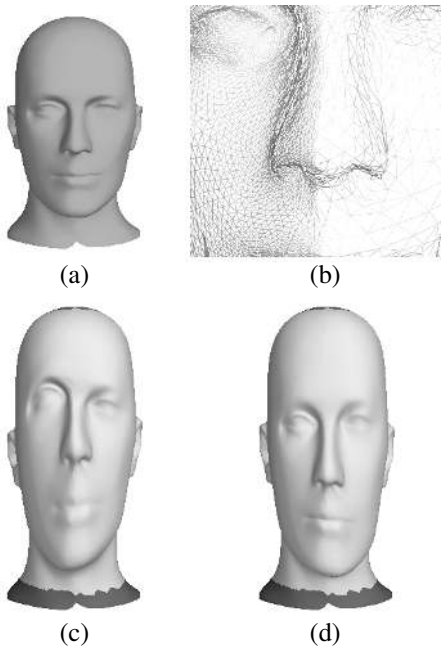



**Figure 1. Comparison of weighting. (a,b) The original model subdivided unsymmetrically. (c) An example of uniform weighting. (d) An example of cotangent weighting.**

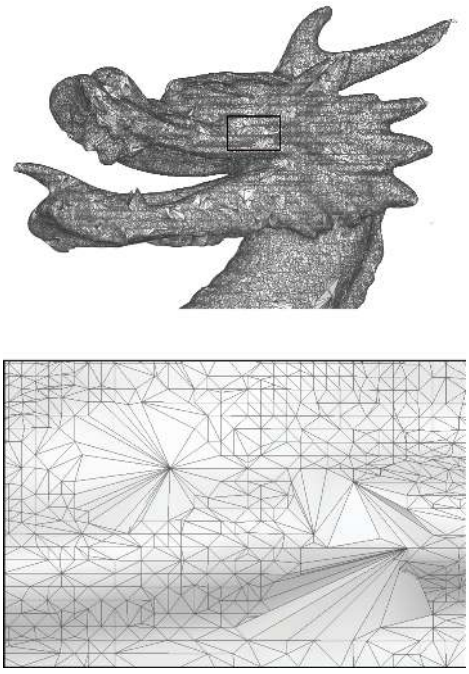


**Figure 2. An example of a mesh that causes numerical problems.**

### 3.2 Constrained Least Squares

We define hard constraints as follows:

$$B\mathbf{x} = \mathbf{d}. \tag{10}$$

Our mesh deformation framework with hard constraints can be formalized as (11) and can be solved using the equality-constrained least squares method [9].

$$\min_{B\mathbf{x}=\mathbf{d}} \|A\mathbf{x} - \mathbf{c}\|_2, \tag{11}$$

where $A \in \mathbb{R}^{l\times n}$, $B \in \mathbb{R}^{m\times n}$, $\mathbf{c} \in \mathbb{R}^l$, $\mathbf{d} \in \mathbb{R}^m$, $m \leq n \leq l$.

For clarity, we assume that both $A$ and $B$ have full rank. Then $B^T$ is decomposed by QR factorization [9] as follows:

$$B^T = QR, \tag{12}$$

where $Q \in \mathbb{R}^{n\times n}$ is an orthogonal matrix, and $R \in \mathbb{R}^{n\times m}$ is an upper triangular matrix.

We rewrite these as follows:

$$Q = \begin{matrix}\begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \\ m \quad n-m\end{matrix} \tag{13}$$

$$R = \begin{matrix}\begin{bmatrix} R_1 \\ 0 \end{bmatrix} \\ m \\ n-m\end{matrix} \tag{14}$$

$$Q^T\mathbf{x} = \begin{matrix}\begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix} \\ m \\ n-m \\ .\end{matrix} \tag{15}$$

Thus, (11) is expressed in the following form:

$$\min_{R_1^T\mathbf{y}=\mathbf{d}} \|AQ_1\mathbf{y} + AQ_2\mathbf{z} - \mathbf{c}\|_2. \tag{16}$$

Since $\mathbf{y}$ is determined by the constraint equation $R_1^T\mathbf{y} = \mathbf{d}$, $\mathbf{z}$ is obtained by solving the following unconstrained least-squares problem:

$$\min_{\mathbf{z}} \|AQ_2\mathbf{z} - (\mathbf{c} - AQ_1\mathbf{y})\|_2. \tag{17}$$

By solving this least squares system, we obtain:

$$(AQ_2)^T AQ_2\mathbf{z} = (AQ_2)^T(\mathbf{c} - AQ_1\mathbf{y}). \tag{18}$$

Finally, we can obtain $\mathbf{x}$ as follows:

$$\mathbf{x} = Q \begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix}. \tag{19}$$

We can solve (18) very efficiently using Cholesky factorization, because $(AQ_2)^T(AQ_2)$ is a symmetrical positive definite matrix and its size is smaller than $A$.

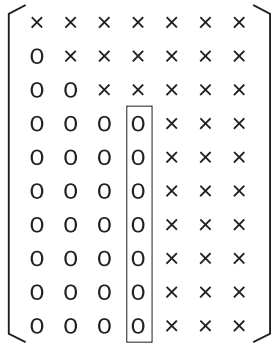


**Figure 3. An example showing the detection of rank deficiency in the matrix. After the 3rd column is processed, the diagonal and lower elements in the 4th column become zero, and then all the elements in the 4th Householder vector become zero, which implies rank deficiency.**

### 3.3 Conflictions of Constraints

In our framework, hard constraints are strictly satisfied in the solutions. Therefore, if conflicting or redundant constraints are involved, they lead to the rank deficiency in $B$, and the solver halts the computation.

We can resolve both redundant and conflicting constraints by detecting and removing the deficiency of the rank during the processes of QR decomposition.

We compute QR decomposition $B^T = QR$ by Householder factorization [9]. Each column of $B^T$ represents a hard constraint. In the process of Householder factorization, each column is processed sequentially. Vertices have $n$ degrees of freedom (DoF) at the beginning of decomposition, and they have $n - i$ DoF when the column $i$ has been processed. If $B^T$ has no redundant constraint, vertices have $n - m$ DoF after QR decomposition.

If column $j$ in $B^T$ is a redundant constraint, we cannot decrease the degree of freedom by processing column $j$, because the diagonal and lower elements of column $j$ are equal to zero after the previous $j-1$ columns are processed, as shown in Figure 3. Therefore, we can detect the redundant constraints.

After the column of a redundant constraint is detected, it is skipped and then the next column is processed. By removing all the redundant constraints, a unique solution is determined using the unconstrained least squares method.

Then (13), (14) and (15) are substituted with (20), (21)

and (22):

$$Q = \begin{matrix}[Q_1 & Q_2] \\ r & n-r\end{matrix} \tag{20}$$

$$R = \begin{matrix}\begin{bmatrix}R_1 \\ O\end{bmatrix} & \begin{matrix}r \\ n-r\end{matrix}\end{matrix} \tag{21}$$

$$Q^T\mathbf{x} = \begin{matrix}\begin{bmatrix}\mathbf{y} \\ \mathbf{z}\end{bmatrix} & \begin{matrix}r \\ n-r\end{matrix}\end{matrix}, \tag{22}$$

where $r$ is the rank of the hard constraint matrix $B$. The number of skipped constraints is shown as $n - r$.

After detecting all redundant constraints, we examine whether each constraint satisfies the solution. If a redundant constraint is consistent with the solution, it is ignored. If it conflicts with the solution, we can send out a warning message to correct the specification.

## 4 Experimental Results

In this section, we show some results of deformation based on the equality-constrained least squares method.

Figure 4 shows examples of deformed shapes. This mesh has badly shaped triangles, as shown in Figure 2. Our method can produce good results using cotangent weights, because we can represent Laplacians using cotangent weights, even when the mesh consists of non-uniform triangles.

Figure 5 shows sample models that are commonly used for evaluation in computer graphics. We solve the linear systems constructed by these models using TAUCS [23], which is a well-known solver of Cholesky factorization.

Table 1 shows the numerical stability. When all constrains are treated as soft constraints and their matrix is constructed using cotangent weights, the solver fails to calculate the Bunny and Dragon models. Our method can successfully calculate both cases.

Figure 6 shows the mesh model with conflicting constraints. In Figure 6b, all constraints are described as soft constraints and a compromised shape is generated. In this case, it is difficult to detect which constraints are conflicting. Figure 6c shows that our method resolves the conflicting constraints by detecting and removing them.

Our method consists of three phases: set-up, precomputation and solution. In the set-up phase, we construct the matrices for hard and soft constraints. In the precomputation phase, we obtain the least squares systems by QR decomposition and factorize them. In the solution phase, we compute the positions of vertices. After the matrix is factorized once in the set-up and precomputation phases, the vertex positions are repeatedly calculated in the time spent on the solution phase. A longer computation time is required

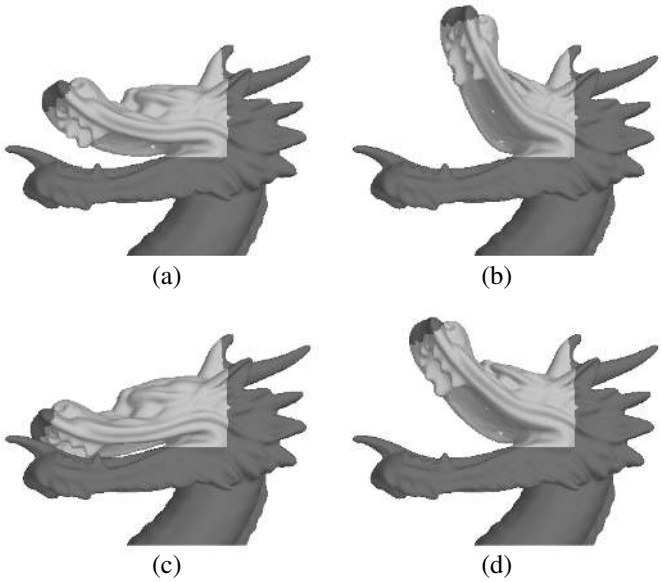



**Figure 4. Examples of the opening and closing of the mouth of a dragon. The red areas consist of constrained vertices: (a) is the original shape and (b), (c) and (d) are variations of deformation.**

in the set-up and solution phases than in the solution phase. Figure 7 compares the total time for the first two phases.

The calculation time was measured on a laptop computer with 2.0-GHz Pentium M CPU, with 1.0 GB of RAM and Windows OS. The result shows that our method with hard and soft constraints is as fast as the method with only soft constraints.

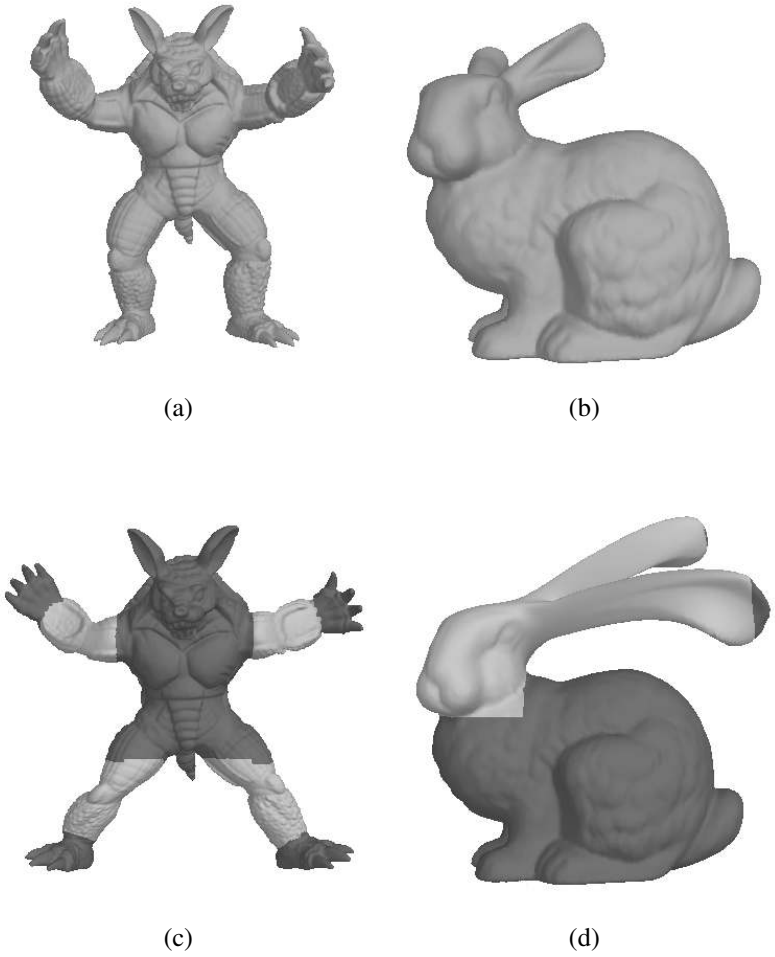
It is possible to add various types of constraints, as well as vertex positions. In Figure 10 and Figure 11, we add vectorial constraints that fix the relative positions of vertices. These constraints are strictly satisfied in our framework. Figure 9 shows the computation time for these cases. When we constrain many relative positions, the computation time for calculating the least squares matrix $(AQ_2)^T(AQ_2)$ increases, because the matrix $AQ_2$ becomes less sparse. In the current implementation, we deal with $AQ_2$ on the assumption that it is sparse matrix, which causes inefficiency. Thus, there is room for improving the computation times.

## 5 Conclusion

In this paper, we show a constrained least squares approach for stably computing Laplacian deformation with strict positional constraints. Our experimental results for mesh models, which are commonly used for evaluation in the computer graphics community, show that our method is

**Table 1. Comparison of stability against numerical problems with some samples. ($\times$ indicates that numerical problems arise.)**

| | Only soft constraints | | Soft & hard constraints using constrained least squares | |
|---|---|---|---|---|
| Weighting | Uniform | Cotangent | Uniform | Cotangent |
| (a)Armadillo (Figure 5c) | ◯ | ◯ | ◯ | ◯ |
| (b)Bunny (Figure 5d) | ◯ | $\times$ | ◯ | ◯ |
| (c)Dragon (Figure 4) | ◯ | $\times$ | ◯ | ◯ |



(a) (b)

(c) (d)

**Figure 5. Examples of deformed models. (a) and (b) are deformed to (c) and (d), respectively.**

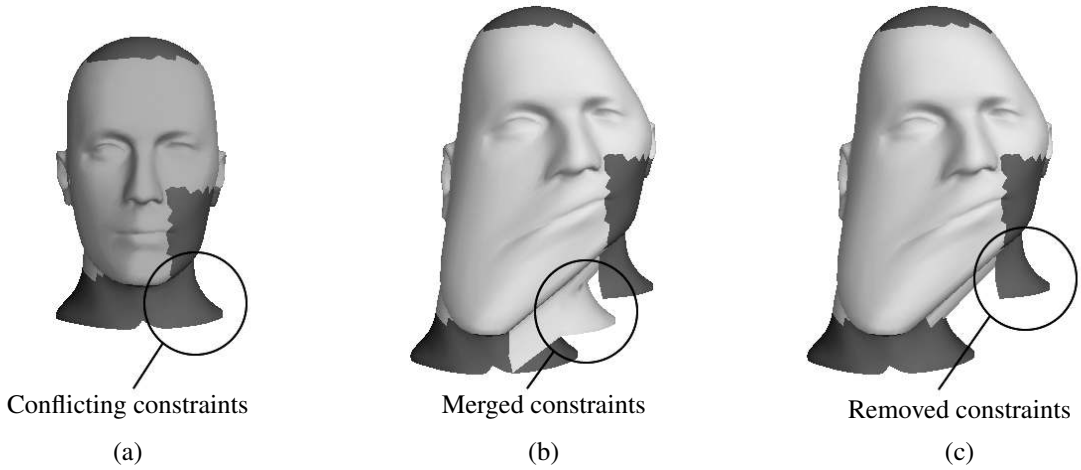



**Figure 6. Management of conflicting constraints. The red areas consist of constrained vertices. (a) The original model and constraints including conflicts. (b) A result for the case in which all constraints are soft. (c) A result for the case in which conflicting constraints are managed.**

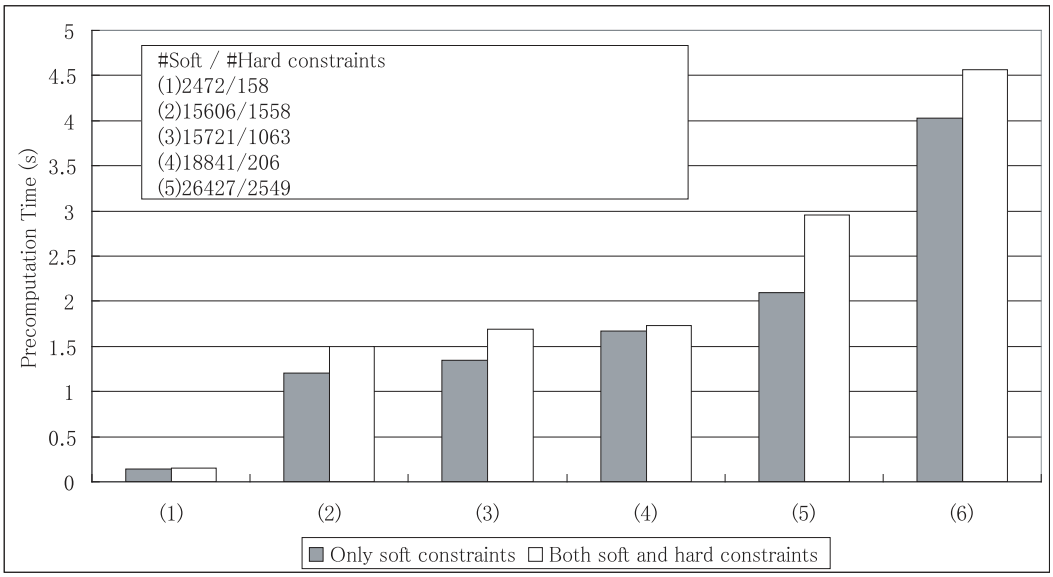



**Figure 7. Comparison of precomputation time for different methods. In the case of only soft constraints, we convert hard constraints to soft constraints. All the soft constraints are Laplacian coordinates, and all the hard constraints are to fix vertex positions.**

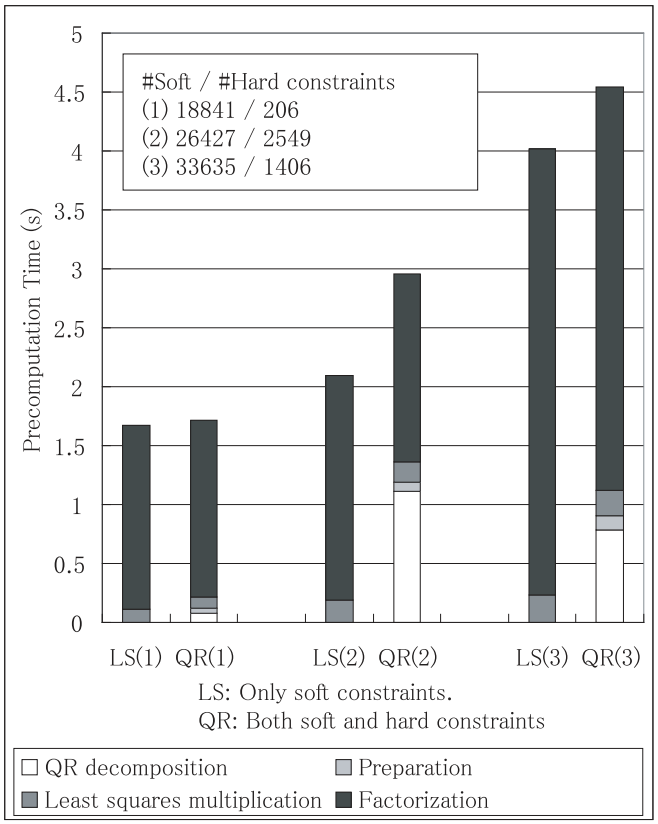



**Figure 8. Breakdown of precomputation time. All hard constraints are to fix vertex positions. Preparation is to solve $R_1^T\mathbf{y} = \mathbf{d}$ and to compute $\mathbf{c} - AQ_1y$ and $AQ_2$.**

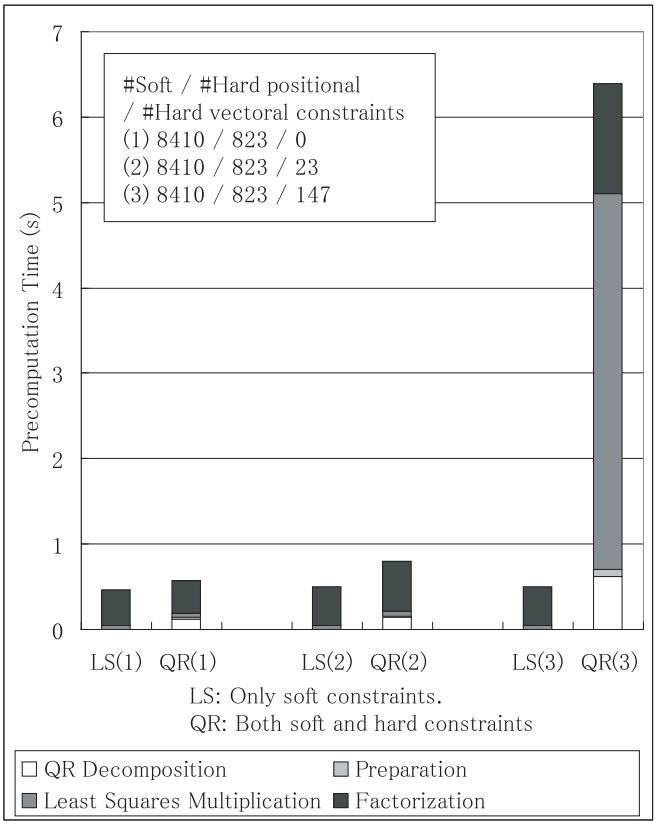
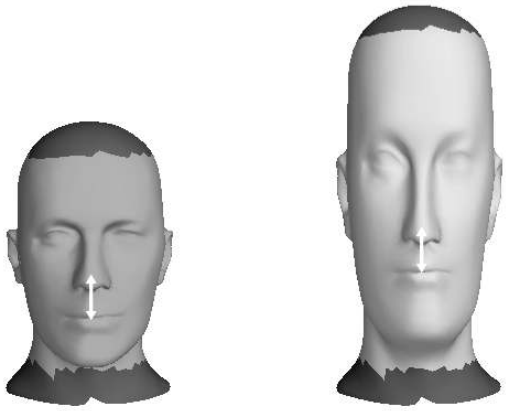
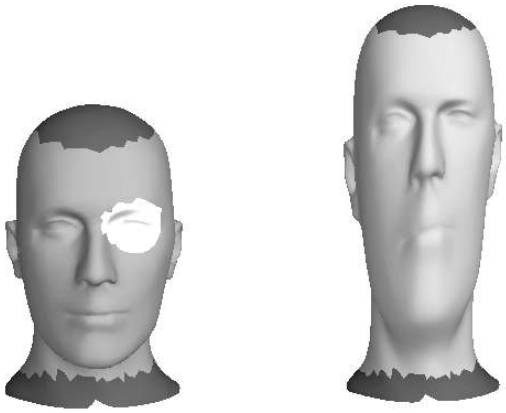



**Figure 9. Breakdown of precomputation time for various hard constraints. Preparation is to solve $R_1^T\mathbf{y} = \mathbf{d}$ and to compute $\mathbf{c} - AQ_1y$ and $AQ_2$.**



(a) (b)

**Figure 10. An example of vector-preserved deformation. (a) An original mesh. The red areas consist of constrained vertices. The white arrow is a constrained vector. (b) A deformed mesh. The relative position between the mouth and the nose is preserved.**



(a) (b)

**Figure 11. An example of feature-preserved deformation. (a) An original mesh. The red areas consist of constrained vertices. The white areas consist of vertices constrained by each other. (b) A deformed mesh. The left eye is preserved in its original shape and the right eye is stretched.**

more stable than existing least-squares methods. In addition, we show a method for resolving redundant and conflicting constraints. The performance of our method is as good as that of the method that manages only soft constraints. Since our method can strictly satisfy hard constraints, it can be applied to applications that require precise positional constraints.

In future work, we need to improve the performance when many vectorial constraints are added. Since our implementation is not optimized yet, we will be able to tune the solver for QR decomposition and matrix multiplications.

## Acknowledgements

The Armadillo, Bunny and Dragon models are courtesy of Stanford University, the Horse model is courtesy of the Georgia Tech Large Geometric Models Archive, and the Mannequin model is courtesy of the Graphics and Imaging Laboratory, University of Washington.

## References

[1] M. Alexa. Differential coordinates for local mesh morphing and deformation. *The Visual Computer*, 19(2-3):105–114, 2003.

[2] M. I. G. Bloor and M. J. Wilson. Using partial differential equations to generate free-form surfaces. *Computer-Aided Design*, 22(4):202–212, 1990.

[3] M. Botsch, D. Bommes, and L. Kobbelt. Efficient linear system solvers for mesh processing. In *IMA Conference on the Mathematics of Surfaces*, pages 62–83, 2005.

[4] M. Botsch and L. Kobbelt. An intuitive framework for real-time freeform modeling. *ACM Trans. Graph.*, 23(3):630–634, 2004.

[5] M. Botsch and L. Kobbelt. A remeshing approach to multiresolution modeling. In *Symposium on Geometry Processing*, pages 189–196, 2004.

[6] S. Coquillart. Extended free-form deformation: A sculpturing tool for 3d geometric modeling. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 187–196. ACM Press, 1990.

[7] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 317–324. ACM Press/Addison-Wesley Publishing Co., 1999.

[8] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 173–182. ACM Press, 1995.

[9] G. H. Golub and C. F. V. Loan. *Matrix Computations*. Johns Hopkins Series in the Mathematical Sciences. Johns Hopkins University Press, 3rd edition, 1989.

[10] I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 325–334. ACM Press/Addison-Wesley Publishing Co., 1999.

[11] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 105–114. ACM Press, 1998.

[12] S. Lee. Interactive multiresolution editing of arbitrary meshes. *Comput. Graph. Forum*, 18(3):73–82, 1999.

[13] Y. Lipman, O. Sorkine, D. Cohen-Or, D. Levin, C. Rössl, and H.-P. Seidel. Differential coordinates for interactive mesh editing. In *SMI 2004: Proceedings of the international conference on Shape Modeling and Applications*, pages 181–190, 2004.

[14] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph.*, 24(3):479–487, 2005.

[15] R. MacCracken and K. I. Joy. Free-form deformations with lattices of arbitrary topology. In *SIGGRAPH*, pages 181–188, 1996.

[16] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In H.-C. Hege and K. Polthier, editors, *Visualization and Mathematics III*, pages 35–57. Springer-Verlag, 2003.

[17] A. Nealen, O. Sorkine, M. Alexa, and D. Cohen-Or. A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph.*, 24(3):1142–1147, 2005.

[18] R. Schneider and L. Kobbelt. Generating fair meshes with g1 boundary conditions. In *GMP*, pages 251–261, 2000.

[19] T. W. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 151–160. ACM Press, 1986.

[20] O. Sorkine. Laplacian mesh processing. In Y. Chrysanthou and M. Magnor, editors, *STAR Proceedings of Eurographics 2005*, pages 53–70. Eurographics Association, Sept. 2005.

[21] O. Sorkine, Y. Lipman, D. Cohen-Or, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian surface editing. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 175–184. ACM Press, 2004.

[22] G. Taubin. A signal processing approach to fair surface design. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 351–358. ACM Press, 1995.

[23] S. Toledo, D. Chen, and V. Rotkin. Taucs: A library of sparse linear solvers. http://www.tau.ac.il/ stoledo/taucs/, 2003.

[24] A. Yamada, T. Furuhata, K. Shimada, and K.-H. Hou. A discrete spring model for generating fair curves and surfaces. In *Pacific Conference on Computer Graphics and Applications*, pages 270–279, 1999.

[25] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph.*, 23(3):644–651, 2004.

[26] K. Zhou, J. Huang, J. Snyder, X. Liu, H. Bao, B. Guo, and H.-Y. Shum. Large mesh deformation using the volumetric graph laplacian. *ACM Trans. Graph.*, 24(3):496–503, 2005.

[27] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *SIGGRAPH*, pages 259–268, 1997.