Discrete Parametrization for Deforming Arbitrary Meshes

Shigeru Kuriyama Toyohisa Kaneko

Department of Information and Computer Sciences
Toyohashi University of Technology
Tenpaku-cho, Toyohashi, Aichi, 441-8580, Japan
{kuriyama,kaneko}@tutics.tut.ac.jp

Abstract

Techniques for deforming polygonal meshes are demonstrated by using two-dimensional lattices of control points or functions for pasting features. The deformations use a shape-preserving parametrization that embeds the mesh's vertices in a normalized two-dimensional space while preserving shape consistency for non-flat surfaces. A discrete smoothing used for the parametrization has inefficient iterative calculations, which is unsuitable for manipulations of dense meshes, and an initial approximation for the smoothing is therefore proposed in order to reduce the number of iterations. The approximation uses a graph-searching algorithm and a discrete normalization whose computational costs are negligible in comparison with that of the iterative calculations.

Key words: Deformations, polygonal meshes, shape-preserving parametrization, shortest path, free-form deformation, feature pasting.

1 Introduction

Polygonal meshes are the most basic representations of surfaces, and the geometrical independence of their vertices allows their shapes to be flexibly manipulated. However, direct editing of polygonal meshes requires tedious efforts by designers to preserve their visual smoothness and continuity, because such meshes have no shape controls based on their own analytical models. Therefore, some continuous functions are generally used to control deformations.

Sederberg [16] proposed a space deformation technique called free-form deformation (or FFD). The technique smoothly deforms a local 3D space by manipulating a parallelepiped 3D lattice whose vertices, called control points, represent the coefficients of trivariate basis functions. Because of the usability and versatility of FFD, many methods have been proposed for enhancing intuitive controls or extending the deformable space [4, 9, 15]. These space deformations embed vertices in local 3D parameters defined in a linear space of a convex hull, and are effective when a mesh is globally deformed in coarser-level editing. Local deformations in finer-level editing, however, often lack intuitiveness because the deformations tend to affect a mesh according to 2D coordinates defined on the mesh's surface rather than 3D coordinates defined in a lattice. Moreover, deformations with a 3D lattice inherently require more control points to be manipulated than those with a 2D lattice. Singh [18] proposed a deformation, called WIRES, that flexibly controls the shapes of meshes by manipulating a network of curves spread near the meshes. The deformation embeds vertices according to their Euclidean distances from the curves (or wires), and thus lacks isometric correspondence between the surface's geometry and the parametric space if a deformable region has a non-flat shape.

Recently, techniques for parametrizing polygonal meshes were proposed for the purpose of reconstructing surfaces and mapping textures. Eck [5] proposed a method of embedding vertices by using harmonic maps that minimize metric distortion, in order to fit B-spline patches to polygonal meshes. This method, however, reconstructs triangular domains as quadrilateral domains by using a combinatorial graph-matching algorithm with a heuristic measure of distortion, which is time-consuming and difficult to implement. Krishnamurthy [11] proposed a method of fitting B-spline patches to meshes by constructing uniform grids on the meshes with spring isocurves. This method lacks robustness for very peaked shapes and is not adaptable to the local changes of the mesh's resolution. Lee [13] proposed an adaptive parametrization using a hierarchical smoothing based on Loop subdivision in a parameter domain. This method hierarchically embeds vertices into barycentric coordinates on a mesh of the coarsest level for adaptive remeshing. However, like harmonic maps [5], it requires mapping from triangular domains to quadrilateral ones when it embeds vertices into Cartesian coordinates.

Floater [6] proposed a discrete parametrization using convex combinations of nearby vertices' parameters. The way of determining weights for the combinations categorizes the parametrization as uniform, chord-length, or shape-preserving, and he showed that the last parametrization can most smoothly reconstruct shapes. Levy [14] introduced another type of discrete parametrization for non-distorted texture mapping. Their parametrization is more flexible than Floater's method in controlling the gradients of parameters by adding minimization constraints, called homogeneity and orthogonality, to the roughness criterion. The additional constraints depend on each other and some tradeoff must be considered for distorted surfaces. The degrees of influences between these constraints must be controlled by designers, which is intuitive for controlling a texture map but not for deforming a shape. Moreover, the orthogonality constraint is less intuitive for deforming a shape, and the homogeneity constraint can be omitted by introducing a surface metric to the roughness criterion, which leads to the same condition as Floater's parametrization. For these reasons, we use shape-preserving parametrization proposed by Floater for constructing a deformation mechanism.

We here consider that meshes tend to consist of a large number of polygons (or facets) when they are used for designing shapes with deformations. However, the computational costs of the abovementioned parametrizations are enormous for such dense meshes, because the order of calculation is not linear to the mesh's resolution. Therefore, we propose a technique of accelerating Floater's parametrization on the basis of a graph-searching algorithm. We then apply the discrete parametrization to two types of deformations: one that deforms shapes by manipulating a 2D lattice and another that pastes featured shapes. These techniques deform meshes by translating the vertices with offset functions through deformations. Deformations of this type allow local shape modifications to be more simple and intuitive than those used in space deformations, and the ways in which the shapes are deformed are not restricted by criteria such as functional minimization [20]. Moreover, the computational cost of the offset functions is proportional to the number of constituent vertices, which ensures that shapes are deformed at interactive rates.

In the following sections, we first introduce a parametrization of polygonal meshes based on discrete smoothing, and then propose a technique for reducing the number of the iterative calculations needed to attain convergence for the smoothing. We next explain about manipulations of deformable regions on a mesh. After that, we present some deformations that effectively utilize the parametrization.

2 Discrete Parametrization

A polygonal mesh is a piecewise-linear approximation of a surface, which is formally represented by a set of vertices' positions and connectivity that define the shape and topology. Let $\mathbf{P}_i \in R^3$ be the 3D position of the i-th vertex that forms a mesh, and let $(u_i, v_i) \in D$ be corresponding parameters defined in local 2D Cartesian coordinates $D \subset R^2$ on a mesh. The value of u_i (or v_i) forms a scalar valued surface, called a u-surface (or v-surface) that has the same connectivity as a parametrized mesh, and both u- and v-surfaces are generically called uv-surfaces. In this section, we explain a method of generating uv-surfaces only for a u-surface, because a v-surface can be obtained in the same way.

2.1 Discrete smoothing

A u-surface must be smooth and reflect the surface metric of a mesh. Floater [6] presented a condition for ensuring smoothness on triangular meshes by using convex combinations as follows:

$$u_i = \sum_{j \in \mathbf{C}_i} \lambda_j \, u_j \ , \quad \sum_{j \in \mathbf{C}_i} \lambda_j = 1 \ , \qquad (1)$$

where $\mathbf{C}_i$ is a set of index numbers for nearby vertices, which represents the connectivity of $\mathbf{P}_i$.

The iterative system for smoothing the u-surface is composed so that the value of u_i is updated to satisfy equation (1):

$$u_i \leftarrow k \, u_i + (1-k) \sum_{j \in \mathbf{C}_i} \lambda_j \, u_j \ , \qquad (2)$$

where the constant k is set as $0 < k < 1$ to ensure convergence of the iterative calculation. The constant is empirically set at about $k \approx 0.01 - 0.1$ for dense meshes and is gradually increased as the resolution of a mesh is decreased, in order to reduce the number of iterations. The above iterative calculation is continued until the magnitudes of correctional values for u_i are below a given threshold on all vertices; $\forall \, \mathbf{P}_i \ , \ |\sum_{j \in \mathbf{C}_i} \lambda_j \, u_j - u_i| < \tau$, where the threshold τ is determined according to the mesh's resolution. Similar iterative techniques are introduced in smoothing of polygonal meshes [19, 12] and editing of multi-resolution meshes [10].

2.2 Shape-preserving parametrization

The weights λ_j can have arbitrary positive values that divide a unity. One way to set them is to use chord lengths between nearby vertices as $\lambda_j = \|\mathbf{P}_j - \mathbf{P}_i\|^{-1} / \sum_{l \in \mathbf{C}_i} \|\mathbf{P}_l - \mathbf{P}_i\|^{-1}$, which can reflect the surface metric for non-flat meshes, where $\| \ \|^{-1}$ denotes the inverse of the geometric length of a vector. The chord-length criterion linearly approximates the arc length, and the efficiency of the approximation depends on the degree of detail in representing curved shapes with constituent polygons. However, the linear approximation is sufficient for deforming meshes, because the meshes are usually minutely subdivided on deformable regions in order to obtain smooth shapes.

Floater proposed a more sophisticated technique [6], called shape-preserving parametrization, that determines the values of the weights by combining the barycentric mapping. Briefly, the nearby vertices $\mathbf{P}_{j \in \mathbf{C}_i}$ are projected onto $\mathbf{X}_j \in R^2$ so as to preserve the Euclidean distances from the vertex $\mathbf{P}_i$ and so that the angles between the edges sharing $\mathbf{P}_i$ are proportionally reduced as follows:

$$\begin{aligned} \mathbf{X}_j &= (\, L_j \, \cos(2\,\pi\,\omega_j),\; L_j \, \sin(2\,\pi\,\omega_j)\,) \ , \\ L_j &= \|\mathbf{P}_j - \mathbf{P}_i\| \ , \quad \omega_j = \theta_j / \sum_{l \in \mathbf{C}_i} \theta_l \ , \\ \theta_0 &= 0,\; \theta_{j+1} - \theta_j = (\mathbf{P}_{j+1} - \mathbf{P}_i) \wedge (\mathbf{P}_j - \mathbf{P}_i) \ , \end{aligned}$$

where $\mathbf{A} \wedge \mathbf{B}$ denotes the angle between two vectors $\mathbf{A}$ and $\mathbf{B}$, and the cyclic order of indices j is defined so that $\mathbf{P}_j$ and $\mathbf{P}_{j+1}$ are contained in a common polygon. Next, the barycentric coordinates at $\mathbf{X}_i$ are calculated with respect to a triangle $\triangle_{j \in \mathbf{C}_i}$ whose vertices denoted by $\mathbf{X}_j$, $\mathbf{X}_{\varphi_j}$, and $\mathbf{X}_{\varphi_j+1}$, where the consecutive vertices $\mathbf{X}_{\varphi_j}$, $\mathbf{X}_{\varphi_j+1}$ are selected so as to enclose $\mathbf{X}_i$ with $\mathbf{X}_j$. Let a triplet $(\epsilon_0,\, \epsilon_1,\, \epsilon_2)$ be defined by cyclic permutations of $(j,\, \varphi_j,\, \varphi_j+1)$; the weights λ_j in equation (1) are then obtained as follows:

$$\lambda_j = \frac{\sum_{l=1}^{N_i} b_j^l}{N_i} \quad , \quad b_{\epsilon_0}^j = \frac{area(\mathbf{X}_i,\, \mathbf{X}_{\epsilon_1},\, \mathbf{X}_{\epsilon_2})}{area(\mathbf{X}_j,\, \mathbf{X}_{\varphi_j},\, \mathbf{X}_{\varphi_j+1})} \ ,$$

where $b_{\epsilon_0}^j$ represents the barycentric coordinates for the nearby vertex $\mathbf{X}_{\epsilon_0}$ on the triangle $\triangle_j$, $area()$ represents the area of the triangle whose vertices are given in the arguments, and N_i denotes the number of nearby vertices of $\mathbf{P}_i$. Notice that $\forall\, j,\ b^j_{l \neq j, \varphi_j, \varphi_j+1} = 0$. Smoothing based on the shape-preserving criterion is more effective than that based on the chord-length criterion for highly irregular meshes that consist of the polygons whose shapes and sizes are extremely dissimilar. For a more detailed theoretical discussion, see [6].

2.3 Boundary conditions

Parametrization based on discrete smoothing requires imposition of linear constraints along all boundaries. We fix the values of $(u,\, v)$ parameters at the vertices on four boundaries that correspond to the edges of a quadrilateral enclosing a deformable region. A pair of opposite boundaries imposes the constant values of $u = 0$ and $u = 1$ on their vertices, and the other pair of boundaries imposes the linearly interpolated values of $0 \leq u \leq 1$ with respect to the chord-length between successive vertices as

$$u_0 = 0 \ , \quad u_{i+1} - u_i = \frac{\|\mathbf{Q}_{i+1} - \mathbf{Q}_i\|}{\sum_{l=0}^{N^B-1} \|\mathbf{Q}_{l+1} - \mathbf{Q}_l\|} \ ,$$

where $\mathbf{Q}_i,\ i = 0, 1, \ldots, N^B$ represents the i-th vertex that composes a boundary polyline. These boundary conditions normalize the u-parametric space, and the boundary conditions for v-surfaces are similarly imposed by exchanging the constraints between two pairs of boundaries.

The four boundaries are arbitrarily formed on a mesh by tracing the connections of vertices, which often causes jagged patterns (or aliasing). This jagging breaks the continuity of the u-surface across the boundaries. On the other hand, the ways of forming quadrilateral boundaries are limited if the vertices are traced along the given connections so as to avoid such jagging. We therefore add vertices on the edges of the polygons, as shown by the white circles in Figure 3, so that the boundaries are formed by smooth polylines. The values of the $(u,\, v)$ parameters of the additional vertices are fixed according to the type of boundary conditions, and the vertices' connectivity along the edges is reconstructed. Notice that the vertices on the boundaries are added only on uv-surfaces and not on a parametrized (or deformed) mesh. The way of generating boundary polylines is after explained in subsection 4.1.

The iterative calculation in equation (2) can be localized by calculating only vertices inside a deformable region. However, the resulting u-surface has unnatural gradients on the boundaries if the vertices outside the deformable region are removed from the calculation. We therefore include the vertices that reside on several external layers, tracing the mesh's connectivity from the boundaries in the directions of outer half spaces. The redundant external vertices serve to correct the gradients of uv-surfaces on the boundaries through smoothing.

3 Acceleration of parametrization

The parametrization must be efficiently calculated when a deformable region is interactively changed. The number of iterative updates for generating uv-surfaces, however, depends on the number of vertices in the region, and thus the calculation cost for parametrization exceeds the linear order of the number. The iterative calculations can be reduced by selecting the optimal value of the constant k in equation (2) on the basis of eigenvalue analysis of the linear system [8]. However, the effects of correctional updates propagate from the boundaries to the center, which inherently requires a number of iterations proportional to the mesh's resolution. Hierarchical smoothing of the uv-

surfaces dramatically reduces the number of iterations - in optimal cases, below some constant [12, 10]. This is because the coarser-level smoothing provides well-approximated initial values for the finer-level smoothing. However, hierarchical structures severely limit the ways of selecting deformable regions. On the other hand, arbitrary selection of the regions forces reformation of the structures, which cannot be performed at interactive rates. For these reasons, the hierarchical approach is unsuitable for efficient and flexible parametrization, and we therefore introduce another way of simply and quickly determining the initial guesses of $(u_i,\, v_i)$ in order to reduce the number of iterative calculations needed for smoothing.

Shape-preserving (or chord-length) parametrization is a homeomorphic mapping $(R^3 \rightarrow R^2)$, denoted by Ω, and can be conceptually decomposed into two processes as $\Omega = \Psi * \Phi$. The notation Φ is an isometric mapping of a part of a non-flat mesh enclosed by a quadrilateral onto a flat surface while preserving the proportions of all edges' lengths, and the notation Ψ is a mapping $(R^2 \rightarrow R^2)$ of the flat surface onto a square diagram representing a normalized $(u,\, v)$ parametric space, as shown in the following figure.

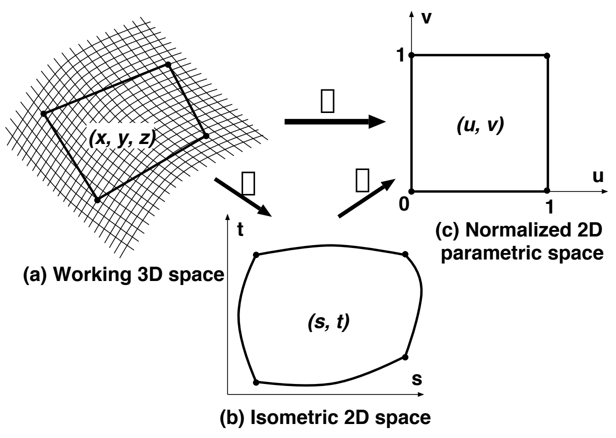


Figure 1: Decomposition of mapping for parametrization

3.1 Graph-searching algorithm

Let the mapping Φ be approximately computed by searching for the minimum distance from a boundary, and let the mapping Ψ be approximately computed by using discrete normalization, where both computational costs are far less than that needed for exactly solving equation (1). The resulting values are then used as initial values of iterative calculations in equation (2) in order to obtain accurate values. We first evaluate the distance of each vertex from the boundary corresponding to $u = 0$ (or $v = 0$), using a one-source shortest-path algorithm proposed by Dijkstra [1]. Assume that the vertices on the boundary $u = 0$ are connected to the virtual starting vertex and that the cost at the vertices are set to zero, and assume also that the cost of the other vertices are initially set to infinity. Then the total cost (or distance) at each vertex is calculated by regarding the mesh's connectivity as a bidirectional graph and by searching for shortest paths, where the cost along each edge is determined by its geometric length. Theoretically, the calculation cost of Dijkstra's algorithm with a priority queue is $O(E\ \log N)$, where E and N represent the numbers of edges and vertices, respectively. The number of connections of each vertex can be regarded as being below some constant, and therefore E is proportional to N, which leads to $O(E \log N) \rightarrow O(N \log N)$. Therefore the cost of calculating shortest paths is small enough in comparison with the iterative calculations for smoothing, which is after evaluated in Table 1. The total cost at each vertex along a shortest path approximates the minimum distance from the $u = 0$ (or $v = 0$) boundary, and represents the value of s (or t) defined on an isometric plane as shown in Figure 1 (b).

3.2 Discrete normalization

We here evaluate the normalized parameters $(u_i,\, v_i)$ assuming that $(s_i,\, t_i)$ are obtained by bilinear transformations of $(u_i,\, v_i)$ as follows:

$$\begin{aligned} s_i &= (s_0 + (s_1 - s_0)\, v_i)\, u_i\ , \\ t_i &= (t_0 + (t_1 - t_0)\, u_i)\, v_i\ , \end{aligned} \tag{3}$$

where $(s_0,\, 0)$, $(0,\, t_0)$, and $(s_1,\, t_1)$ denote the values of isometric parameters $(s,\, t)$ at the corner vertices whose normalized values $(u,\, v)$ correspond to $(1, 0)$, $(0, 1)$, and $(1, 1)$, respectively. Equation (3) leads to quadratic equations with respect to u_i or v_i whose coefficients are determined by the values of $(s_i,\, t_i)$ at each vertex. The values of $(u_i,\, v_i)$ are first obtained by solving the quadratic equations, and are next iteratively corrected as follows:

$$\begin{aligned} u_i &\leftarrow k\, u_i + (1-k)\, s_i / S(v_i)\ , \\ v_i &\leftarrow k\, v_i + (1-k)\, t_i / T(u_i)\ , \end{aligned} \tag{4}$$

where the constant k is determined in a similar manner to that used in equation (2). The function $S(v_i)$ (or $T(u_i)$) returns the normalization factor of s_i (or t_i) at the parameter v_i (or u_i), and is composed by linearly interpolating the values of s (or t) at the vertices that reside on the boundary of $u = 1$ (or $v = 1$). The values of $(u_i,\, v_i)$ are iteratively updated by equation (4) until the correctional values are below a given threshold for all vertices, similarly in equation (2). We empirically confirmed that several iterations are enough for the values to converge except for highly irregular meshes.

3.3 Evaluation

In Table 1, we show the effects of acceleration for various meshes. The first column denotes sample meshes shown in Figure 2, where the mesh (c) has two deformable regions on the forehead and nose, and the second column denotes the number of vertices in the deformable regions. The third and fourth columns indicate the amount of CPU time and the number of iterative updates needed to attain convergence for the smoothing with and without acceleration, respectively, where the threshold τ for determining convergence is set to 10^{-4}, and the times spent for acceleration, which include search of shortest paths and discrete normalization, are enclosed in brackets.

Data	No.	With acceleration		Without acceleration	
	vertex	Time	No.iter	Time	No.iter
(a)	5356	1.87 [0.33]	$(56, 37)$	23.71	$(717, 504)$
(b)	4501	4.21 [0.25]	$(42, 192)$	20.61	$(653, 484)$
(c) Nose	1052	0.55 [0.04]	$(87, 135)$	0.77	$(152, 167)$
(c) Head	755	0.1 [0.02]	$(30, 30)$	0.45	$(159, 115)$

Table 1: Computational costs of parametrization

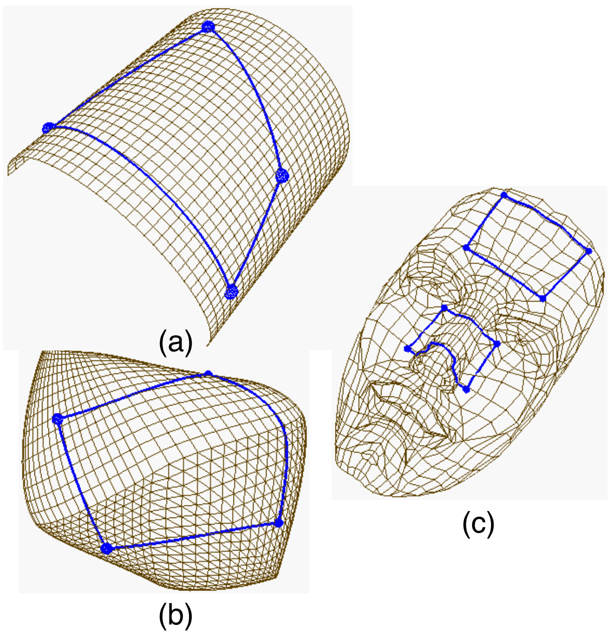


Figure 2: Examples of meshes and deformable regions

The CPU times, listed in units of a second, were measured on a R5000 (180-MHz) machine, and the numbers of iterations were counted for u- and v-surfaces separately as indicated in this order in parentheses. These examples show that the computational costs of calculating initial guesses are negligible, and the effect of our acceleration is conspicuous for mesh (a), which has a large number of vertices and high regularity. Notice that the wireframes are drawn in every four spaces to visually clarify the pattern of the meshes, and thus the actual meshes are obtained by subdividing the polygons drawn by the wireframes 4×4 times.

4 Manipulations of deformable regions

4.1 Generation of boundary polylines

A deformable region is arbitrarily set by constructing edges of a quadrilateral; it is, however, difficult to draw curved polylines corresponding to the edges directly on a curved mesh. We therefore offer a simple way of forming the polylines by merely selecting four corner vertices $\mathbf{W}_i$, $i = 1, 2, 3, 4$. Four planes are then created so as to include each pair of the vertices $(\mathbf{W}_i, \mathbf{W}_{i+1})$, and their normal vectors $\mathbf{H}_i$ are determined as $\mathbf{H}_i = (\mathbf{W}_i - \mathbf{W}_{i+1}) \times (\mathbf{N}_i + \mathbf{N}_{i+1})$. Here, $\mathbf{N}_i$ denotes the normal direction of a tangent plane at the vertex $\mathbf{W}_i$, which is determined by averaging those of the polygons sharing $\mathbf{W}_i$, and $\times$ denotes an outer product. The polylines are then generated by connecting the intersections of polygons' edges with the planes, as shown in Figure 3. The intersections are merged into a mesh's vertices if the geometric distance between them is very small, in order to avoid degeneracy of an edge, as shown by the gray circles in Figure 3.

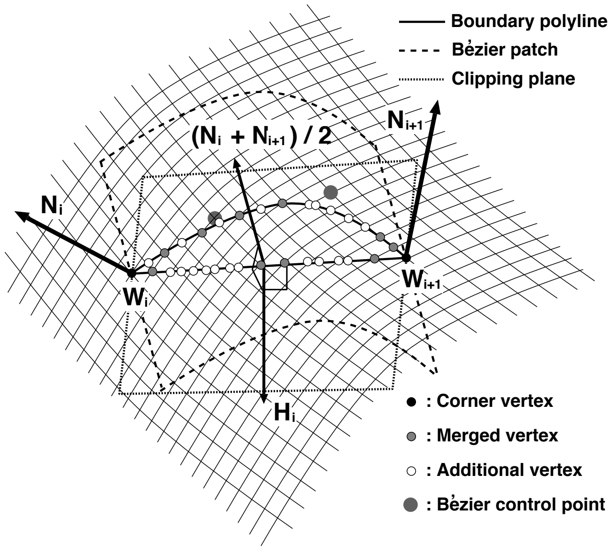


Figure 3: Generation of boundary polylines

Boundary polylines are simply and intuitively generated by clipping a mesh with planes; they should, however, be more flexibly formed on a mesh, to extend the definitions of deformable regions. We therefore introduce sweep surfaces of Bézier curves for flexible formations of the boundary polylines. Bézier curves of the n-th order are generated by selecting n sequential control points on a mesh, and by sweeping the curves along the averaged normal $(\mathbf{N}_i + \mathbf{N}_{i+1})/2$, as shown in Figure 3. The intersections of the sweep surfaces and the polygons' edges are efficiently calculated by using the Bézier clipping method [17]. Krishnamurthy's method [11] of painting boundary curves is also usable, which enables more flexible drawing of boundary polylines in compensation for the computational cost of fitting B-spline curves on a mesh.

4.2 Conversions of coordinate systems

Thus far we have proposed discrete parametrization only for Cartesian coordinates; however, it is expected to be used for managing other types of coordinates, such as polar coordinates and barycentric coordinates, in order to extend the definitions of deformable regions.

Polar coordinates, whose parameters are denoted by $(r,\,\theta)$, should generate scalar-valued $r\,\theta$-surfaces by fixing vertices on a circular boundary and a central pole. However, the condition of equation (1) cannot generate r-surfaces as conics that linearly diffuse the values of r from a pole to a circular boundary. Moreover, regions of circles or ellipses are hard to draw directly on a curved surface, and they are generally drawn with rectangular regions enclosing them, as in most 2D drawing tools. For these reasons, we simply convert the Cartesian coordinates obtained from a quadrilateral region into polar coordinates, using a common transformation between the coordinates.

Barycentric coordinates are defined on a triangular region and their values at $\mathbf{P}_i$, denoted by $(\alpha_i, \beta_i, \gamma_i)$, are generally determined by the proportions of the area between the region's triangle and the sub-triangles whose vertices consist of two corner vertices and $\mathbf{P}_i$. Therefore we select three corner vertices on a mesh, denoted by $\mathbf{P}_\alpha,\, \mathbf{P}_\beta,\, \mathbf{P}_\gamma$, and calculate a scalar valued surface by regarding the vertex $\mathbf{P}_\alpha$ as a degenerate boundary of $\alpha = 1$ and regarding the edge connecting $\mathbf{P}_\beta$ and $\mathbf{P}_\gamma$ as a boundary of $\alpha = 0$, and call the resulting value $\hat{\alpha}_i$. We next similarly calculate $\hat{\beta}_i$ and $\hat{\gamma}_i$ by shifting the boundary conditions among the triangular edges, and determine the values of barycentric coordinates at $\mathbf{P}_i$ as follows:

$$\chi_i = \frac{\hat{\chi}_i}{\hat{\alpha}_i + \hat{\beta}_i + \hat{\gamma}_i} \quad , \quad \chi = \alpha, \beta, \gamma \quad .$$

It is noteworthy that these barycentric coordinates inherit the isometric property from shape-preserving parametrization.

5 Shape Deformations

In this section, we describe deformations that make effective use of our parametrization. We introduce an offset function, denoted by $\mathbf{F}$, that translates the vertex $\mathbf{P}_i$ to a new position $\mathbf{P}'_i$ with respect to its parameters $(u_i,\, v_i)$ as

$$\mathbf{P}'_i = \mathbf{P}_i + \mathbf{F}(u_i,\, v_i) \ .$$

Deformations can thus be categorized according to the way in which they construct the function $\mathbf{F}$. We introduce two types of deformation: one constructs $\mathbf{F}$ with a 2D lattice of control points, and the other does so with a pattern of an offsetting feature.

5.1 Two-dimensional FFD

FFD embeds vertices in a 3D parametric space using linear transformations [16], and requires management of dense control points arranged in a 3D lattice. If we locally create a bulge or a dent in a mesh, control points arranged in a 2D space are sufficient for intuitively controlling the shape; the traditional FFD, however, must construct a redundant 3D lattice to enclose the deformable region. We therefore introduce a 2D version of FFD using our parametrization.

We here introduce offset vectors, denoted by $\mathbf{O}_{pq}$, that are arranged in a 2D $N_p \times N_q$ lattice. Control points of traditional FFD determine vertices' positions on deformed surfaces; the offset vectors, in contrast, determine the extent of their variation from the initial shape. The offset function is then obtained as follows:

$$\mathbf{F}(u_i, v_i) = \sum_{p=1}^{N_p} \sum_{q=1}^{N_q} \mathbf{O}_{pq}\, B_p(u_i)\, B_q(v_i) \quad ,$$

where $B_p(u_i)$ and $B_q(v_i)$ are basis functions such as Bernstein polynomials or B-splines for the u and v directions, respectively. As for B-splines, we give the non-decreasing knot vectors k^u_i and k^v_i in the u and v directions with the boundary conditions of $k^u_1 = k^v_1 = 0$ and $k^u_{N_p} = k^v_{N_q} = 1$. The continuity conditions across the boundaries are preserved by fixing the positions of the offset vectors on outer layers as $\mathbf{O}_{pq} = 0$ for $1 \le p, q \le 1 + L,\ N_p - L \le p \le N_p,\ N_q - L \le q \le N_q$, where L denotes the depth of the layer, which is determined by the degree of continuity and the property of the basis functions. For example, use of cubic B-splines as bases fixes two layers $L = 2$ for second-order continuity.

Controlling shapes with a 2D lattice allows polygonal meshes to be intuitively and hierarchically modified. Such modifications are widely used, especially in spline-based surface modelers, which control shapes by means of hierarchically arranged control points [7]. Our FFD with a 2D lattice allows similar hierarchical deformations for arbitrary polygonal meshes. Moreover, concentric circular lattices [4] can be introduced by converting the coordinate system from Cartesian (u, v) to polar $(r,\, \theta)$. Use of barycentric coordinates also has the potential to extend 2D lattices so that they have arbitrary topology [15].

Figure 4 shows examples of deformations with 2D lattices, where uniform cubic B-splines are used as bases and deformable regions are shown in Figure 2(c). Figure 4(a) shows the arrangement of the lattices that are defined by rectangular and circular grids on the nose and the forehead, respectively, and arrows indicate offset vectors $\mathbf{O}_{pq}$, where the origins of the vectors, indicated by spheres, are located at the vertices whose parameters are nearest to the corresponding knot values of $\mathbf{O}_{pq}$; $(u_i,\, v_i) \approx (k^u_p,\, k^v_q)$. It is noteworthy that the locations of the spheres merely indicate the visual cue in manipulating offset vectors, and numerically have no effect on the calculations of deformations. Figure 4(b) demonstrates the deformed mesh with the lattices on Figure 4(a), where the smoothing is accelerated and the threshold for convergence is set to $\tau = 10^{-4}$, as is used for the evaluation of accelerations in subsection 3.3. Notice that the meshes are flat-shaded to emphasis the facets.

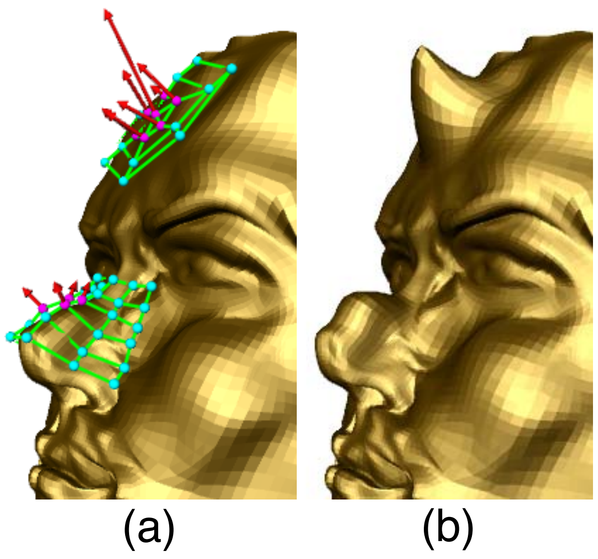


Figure 4: Free-form deformations with 2D lattices

5.2 Feature pasting

It is often useful for designers to supply offsetting patterns as features, if local deformations are created as combinations of simple bulges and dents. Designers can then simply deform a shape by pasting the features in arbitrary locations, scales, and directions on a mesh. Pasting of surfaces has been proposed for B-spline surfaces [2] with useful interfaces [3] for manipulating the pasting domain in a world space. This pasting method uses a parametrization based on bilinear transformation, and the parametrization is not isometric because of the non-linearity in projecting the 3D position onto a quasi-planar region spanned by four points. This non-isometric property is not suitable for pasting features on non-flat meshes, and we therefore introduce a way of pasting for arbitrary meshes, using our parametrization.

The pasting based on B-splines [2] introduces vector-valued mapping of offset surfaces, where the vectors are interpreted in a coordinate frame calculated at each vertex by using the surface geometry before pasting. Such a coordinate frame for polygonal meshes requires the gradients to be evaluated [14] with respect to the parameters u and v; our parametrization, however, do not include such gradients. Therefore, we use a normal direction $\mathbf{N}_i$ at each vertex $\mathbf{P}_i$ instead of the coordinate frame, and introduce a scalar-valued function $G(u_i,\, v_i)$ that determines the offset along $\mathbf{N}_i$, like scalar displacement maps [11], as follows:

$$\mathbf{F}(u_i, v_i) = \mathbf{N}_i\, G(u_i,\, v_i) \enspace .$$

Use of the normal direction allows bulged or dented shapes to be naturally pasted with respect to the orientations of the curved shapes. However, it often causes undesirable overlaps or folds in deformed shapes when deformable regions have highly concave shapes.

Figure 5 shows the shapes of the offsetting features defined on regular squares and the deformed meshes after pasting, where deformable regions are shown in Figure 2(a) and (b), and parametrization is done with the same conditions as those used in Figure 4.

6 Conclusions

We have proposed a method of discretely parametrizing arbitrary meshes, and shown how it can be applied to shape deformations with offset functions. The discrete approach of parametrization allows greater flexibility in controlling deformable regions with quadrilaterals on a mesh, regardless of the mesh's net pattern and resolution, where the topology must be homeomorphic to a disc. The parametrization is accelerated by discretely smoothing uv-surfaces with initial approximations that are computed by using shortest-path algorithm and discrete normalization. As a result, our model of parametrization and deformations can avoid fatal delays even in manipulating dense meshes, which is inevitable in constructing interactive systems. The acceleration of the parametrization is not so effective for highly irregular meshes; such meshes, however, are rarely used for designing shapes of surfaces in common CG and CAD applications.

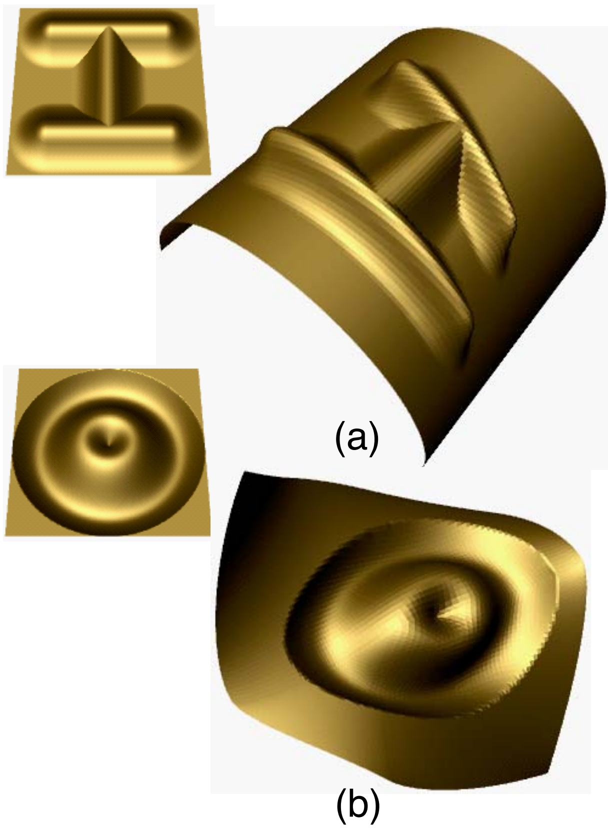


Figure 5: Examples of feature pastings

7 References

[1] Aho, A. V., Hopcroft, J. E., and Ullman, J. D., *Data Structures and Algorithms*, Addison-Wesley, 1983.

[2] Barghiel, R., Bartels, R., and Forsey, D. Pasting Spline Surfaces, In *Mathematical Method for Curves and Surfaces*, Vanderbilt University Press, pages 31–40, 1995.

[3] Chan, L. K. Y., Mann, S., and Bartels, R. World Space Surface Pasting, In *Proceedings of Graphics Interface '97*, pages 146–154, 1997.

[4] Coquillart, S. Extended Free-Form Deformation: A Sculpting Tool for 3D Geometric Modeling, In *Proceedings of SIGGRAPH '90*, *Comput. Graph.*, 24, 4, pages 187–196, 1990.

[5] Eck, M. and Hoppe, H. Automatic Reconstruction of B-Spline Surfaces of Arbitrary Topological Type, In *Proceedings of SIGGRAPH '96*, pages 325–334, 1996.

[6] Floater, M. S. Parametrization and Smooth Approximation of Surface Triangulations, *Computer Aided Geometric Design*, 14, 3, pages 231–250, 1997.

[7] Forsey, D. R. and Bartels, R. H. Hierarchical B-Spline Refinement, In *Proceedings of SIGGRAPH '88*, *Comput. Graph.*, 22, pages 205–212, 1988.

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

[9] Hsu, W. M., Hughes, J. F., and Kaufman, H. Direct Manipulation of Free-Form Deformations, In *Proceedings of SIGGRAPH '92*, *Comput. Graph.*, 26, 2, pages 177–184, 1992.

[10] Kobbelt, L., Campagna, S., Vorsatz, J., and Seidel, H. Interactive Multi-Resolution Modeling on Arbitrary Meshes, In *Proceedings of SIGGRAPH '98*, pages 105–114, 1998.

[11] Krishnamurthy, V. and Levoy, M. Fitting Smooth Surfaces to Dense Polygon Meshes, In *Proceedings of SIGGRAPH '96*, pages 313–324, 1996.

[12] Kuriyama, S. and Tachibana, K. Polyhedral Surface Modeling with a Diffusion System, In *Proceedings of Eurographics '97*, *Comput. Graph. Forum*, 16, 3, pages 39–46, 1997.

[13] Lee, A. W. F., Sweldens, W., Schröder, P., Cowsar, L., and Dobkin, D. MAPS: Multiresolution Adaptive Parametrization of Surfaces, In *Proceedings of SIGGRAPH '98*, pages 95–104, 1998.

[14] Lévy, B. and Mallet, J. L. Non-distorted Texture Mapping for Sheared Triangulated Meshes, In *Proceedings of SIGGRAPH '98*, pages 343–352, 1998.

[15] Maccracken, R. and Joy, K. I. Free-Form Deformations with Lattices of Arbitrary Topology, In *Proceedings of SIGGRAPH '96*, pages 181–188, 1996.

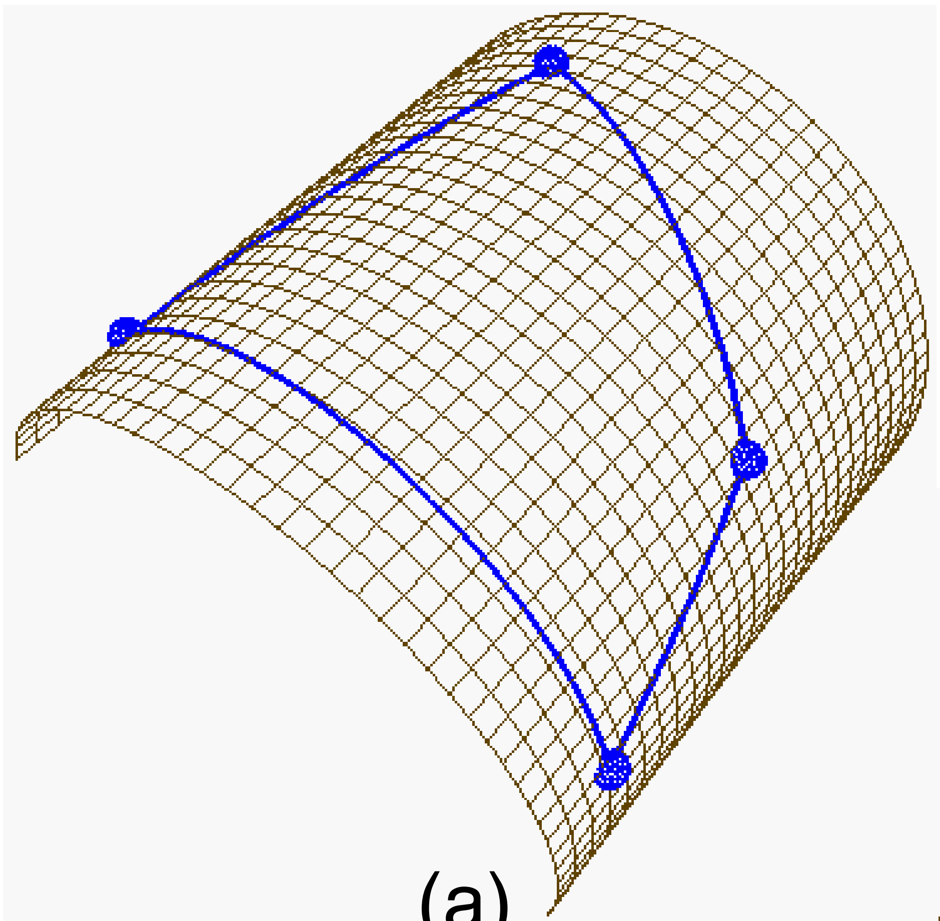
[16] Sederberg, T. W. and Parry, S. R. Free-Form Deformation of Solid Geometric Models, In *Proceedings of SIGGRAPH '86*, in *Comput. Graph.*, 20, 4, pages 151–160, 1986.

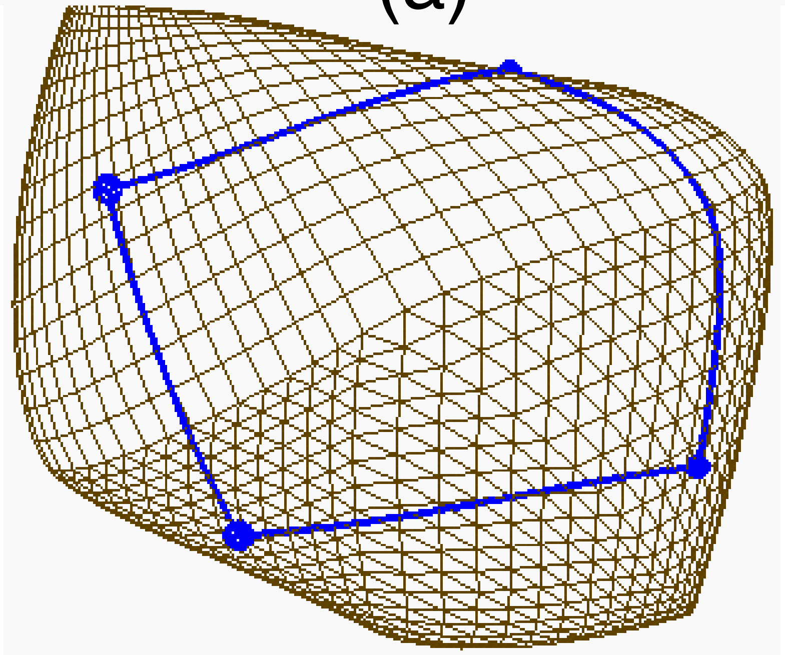
[17] Sederberg, T. and Nishita, T. Curve Intersection Using Bézier Clipping, *Computer Aided Design*, 22, 9, pages 538–549, 1990.

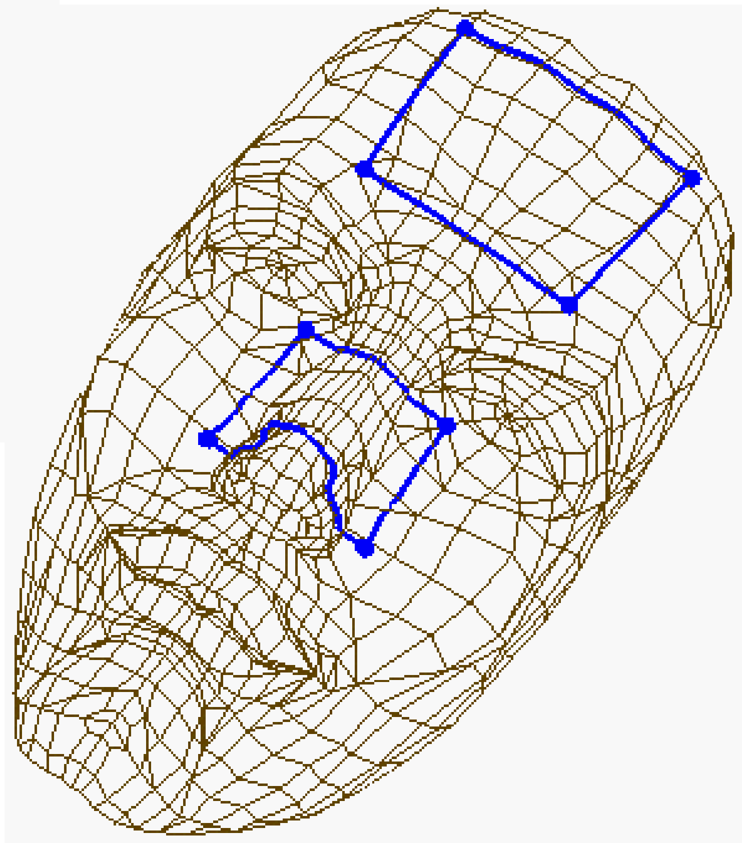
[18] Singh, K. and Fiume, E. Wires: A Geometric Deformation Technique, In *Proceedings of SIGGRAPH '98*, pages 405–414, 1998.

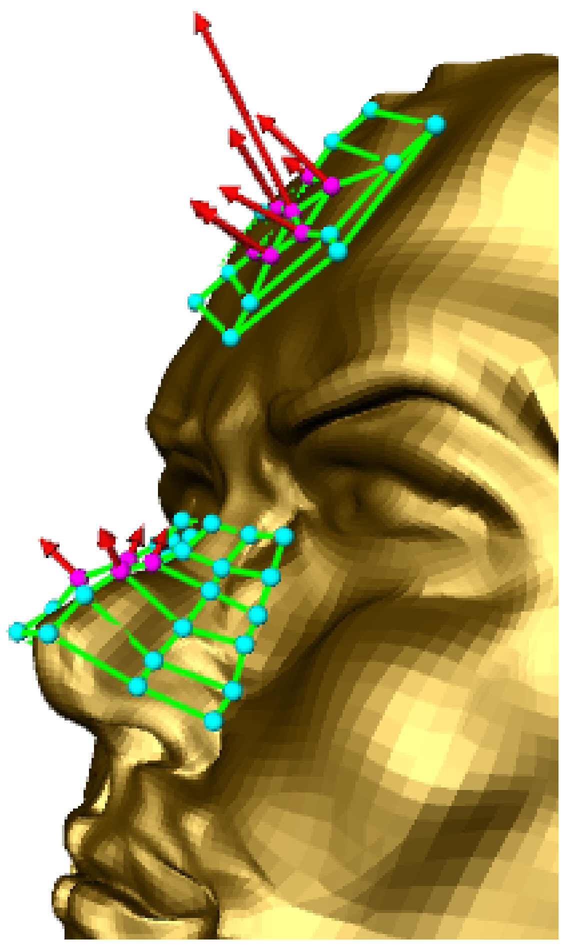
[19] Taubin, G. A Signal Processing Approach to Fair Surface Design, In *Proceedings of SIGGRAPH '95*, pages 351–358, 1995.

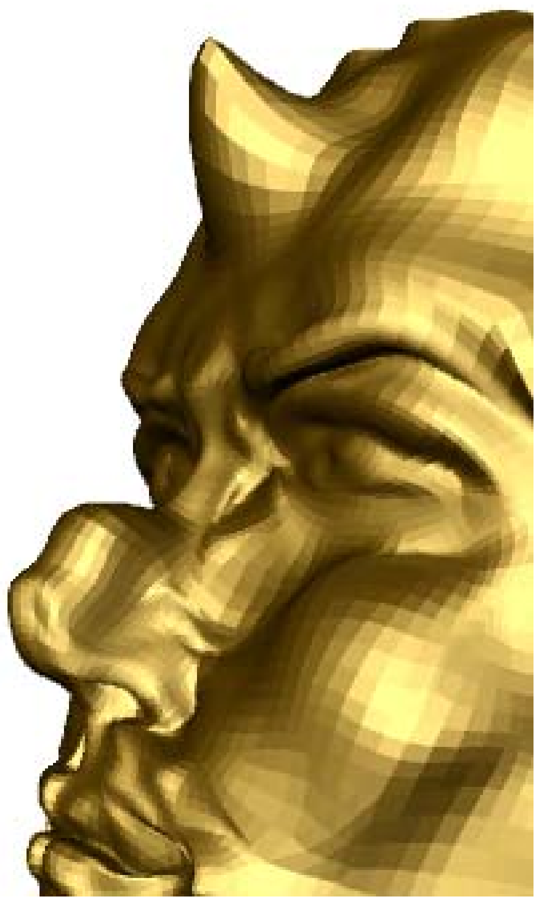
[20] Welch, W. and Witkin, A. Free–Form Shape Design Using Triangulated Surfaces, In *Proceedings of SIGGRAPH '94*, pages 247–256, 1994.

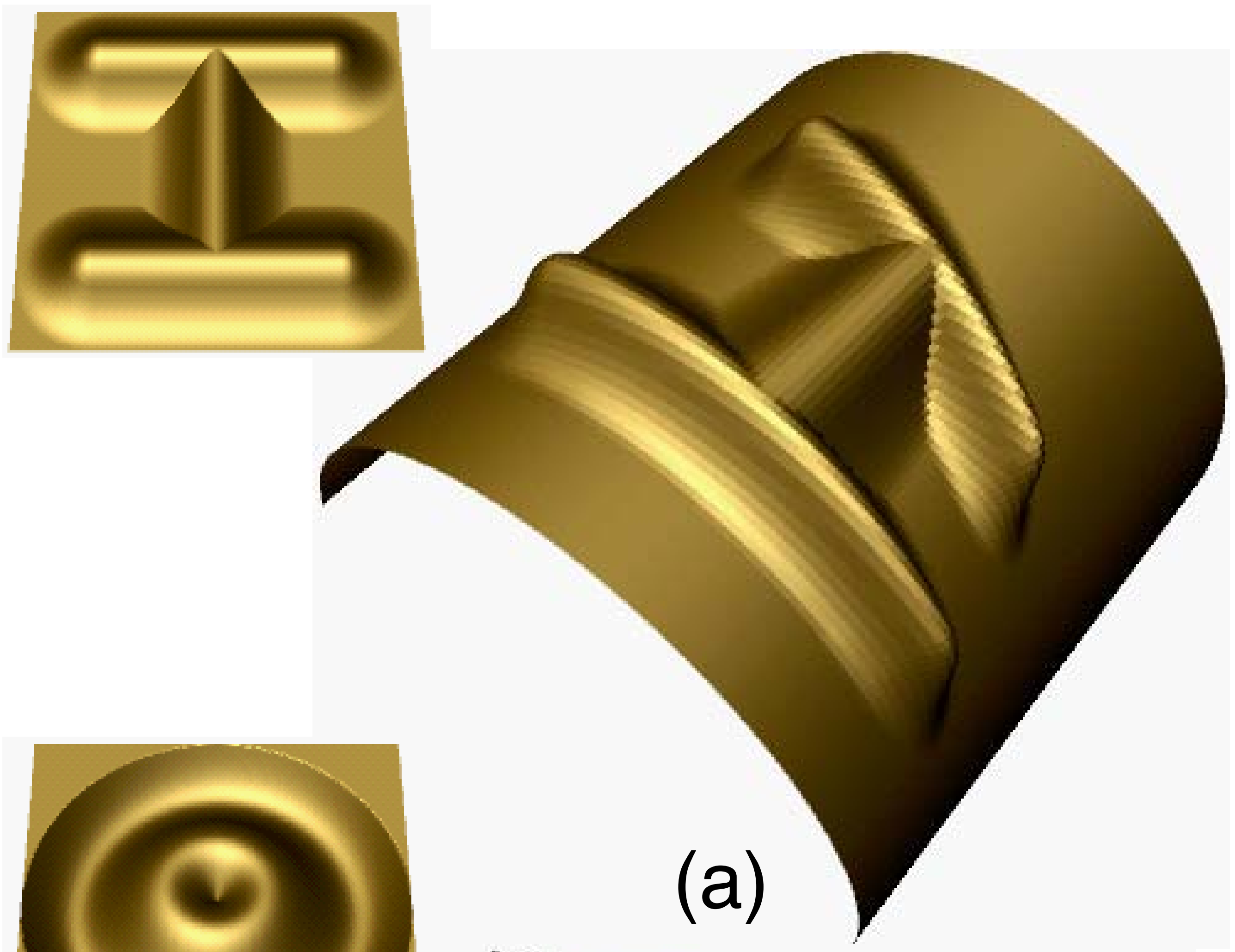

(a)

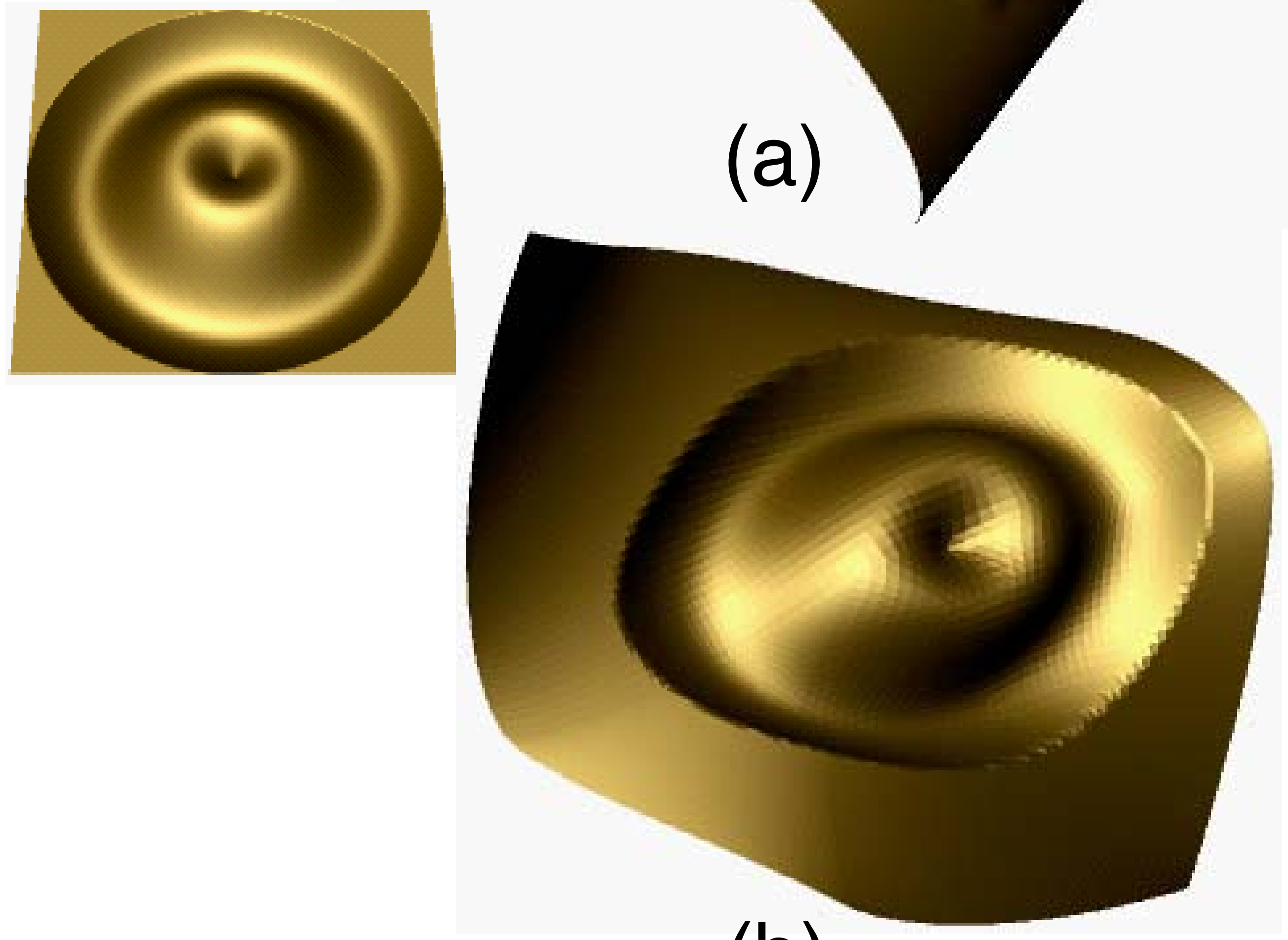

(b)


(c)


(a)


(b)


(a)


(b)