Implicit Boundary Control of Vector Field Based Shape Deformations

Wolfram von Funck[1], Holger Theisel[2], and Hans-Peter Seidel[3]

[1] MPI Informatik, D-66123 Saarbrücken, Germany
`wfunck@mpi-inf.mpg.de`
`http://www.mpi-inf.mpg.de/~wfunck`
[2] Computer Graphics Group, Bielefeld University, D-33501 Bielefeld, Germany
[3] MPI Informatik, D-66123 Saarbrücken, Germany

Abstract. We present a shape deformation approach which preserves volume, prevents self-intersections and allows for exact control of the deformation impact. The volume preservation and prevention of self-intersections are achieved by utilizing the method of Vector Field Based Shape Deformations. This method produces physically plausible deformations efficiently by integrating formally constructed divergence-free vector fields, where the region of influence is described by implicitly defined shapes. We introduce an implicit representation of deformation boundaries, which allows for an exact control of the deformation: By placing the boundaries directly on the shape surface, the user can specify precisely where the shape should be deformed and where not. The simple polygonal representation of the boundaries allows for a GPU implementation, which is able to deform high-resolution meshes in real-time.

1 Introduction

Deforming shapes under a number of constraints is a standard problem in Computer Graphics. For instance, animation of characters can be achieved by deforming the shape of the character according to its underlying skeleton, elastic bodies are deformed by performing simulations based on physical laws, or in industrial design, fair surface deformations are obtained by minimizing curvature energy.

When deforming solid objects, the constraint of volume preservation is an important and often-addressed issue: Under the assumption that the object consists of an incompressible material, its volume remains constant under deformation. While physical simulations or constrained optimization techniques can be used to achieve this goal, they are usuallycomputationally expensive and require special data structures like grids or embedding meshes. In contrast to this, the method of *Vector Field Based Shape Deformations* (VFSD) [1] constructs and integrates divergence-free vector fields on-the-fly without any simulation, optimization or special data structures and produces realistic looking volume-preserving deformations without self-intersections.

Boundary constraints, often used in the context of shape editing, are another useful aspect of shape deformations. The user often wants to specify precisely which parts of the shape should be deformed and which parts should not be deformed at all. This is usually done by placing boundary constraints on the surface, i.e. the user draws two curves on the surface. The region enclosed by the first curve undergoes a full deformation, e.g. a translation or rotation. The region enclosed by the second curve is not deformed at all. In the region between both curves, the deformation is smoothly blended between full and zero deformation.

In this paper, we present a method which brings both approaches together. The user can define the deformation impact by drawing two boundaries onto the surface of the shape and can deform the shape in a volume-preserving and foldover-free manner with respect to these boundaries. While in most existing approaches boundaries are constraints of an optimization, we introduce implicit boundaries, which are defined by closed polygons and give a direct mathematical solution of a smooth blending function which defines the amount of deformation for every point in space.

The paper is organized as follows: Section 2 reviews relevant shape deformation approaches. Section 3 describes how implicit boundaries are defined and how they can be used together with Vector Field Based Shape Deformations to deform triangle meshes. Section 4 demonstrates several application scenarios, while Section 5 goes into the details of the implementation and analyzes the performance. Finally, Section 6 discusses the presented method and possible future research.

2 Related Work

We would like to give the reader an overview of related work. We review both surface-based approaches and space deformations, since our approach is a space deformation which is constructed by defining boundaries and the surface of the shape.

Surface based methods define the deformation on the surface of the shape. A common approach, based on triangle meshes, is to specify a number of original and target vertices and compute the remaining vertex positions by a variational approach [2,3]. Boundary constraint-modeling has established itself as a standard method for deforming surfaces represented by triangle meshes. The main idea is that the user specifies boundary constraints as displacements of a number of vertices. Usually this means that two bands of vertices, the boundaries, are marked on the surface, where on one band the displacements are zero and on the other band the displacements describe a simple deformation like translation, rotation or scaling. Using variational calculus, a deformation field which is optimal with respect to curvature energy is found for the free vertices [4,5,6,7,8,9,10]. This involves solving a sparse linear Laplacian system during each modification by the user.

Space deformations are defined for all points in space, i.e. a shape is deformed by deforming the space where it is placed in. The first space deformation methods appeared in the form of free-form deformations (FFD) [11]. The idea of FFD is to define deformations by modifying coarse control structures like lattices [11,12,13], curves [14,15], or points [16,17]. Using radial basis functions, it is possible to extend the boundary constraint modeling method from the surface setting to the space setting [18]. In order to give the user the impression of real incompressible material, many space deformation approaches have been developed which preserve the volume of the shape under deformation [16,19,20,21,22,1]. Since self-intersections are irreversible using space deformations, a number of approaches have been developed to address this problem [23,24,25].

Another representative of space deformations, which addresses the issues of volume-preservation and prevention of self-intersections, is the method of Vector Field Based Shape Deformations [1]. This method relies on a formal construction of time-dependent divergence-free vector fields on which path line integrations are carried out to deform the vertices of a triangle mesh. Due to the C^1 continuity of the vector fields, the resulting deformation is smooth. Due to the path line integration, self-intersections are prevented. Since the vector fields are divergence-free, the volume of the shape remains constant under deformation. Thanks to the direct mathematical formulation of the vector fields, the deformation if independ of the shape representation, requires neither special control structures nor precomputations. In the following, we review the method. To simplify matters, we formulate the construction in a time-independent context – the extension to the time-dependent case is straightforward.

Given two C^2 continuous scalar fields $p, q : \mathbb{R}^3 \rightarrow \mathbb{R}$, a C^1 continuous divergence-free vector field $\mathbf{v}$ can be constructed from the gradients of p and q as

$$\mathbf{v}(x, y, z) = \nabla p(x, y, z) \times \nabla q(x, y, z). \tag{1}$$

Simple deformations can be constructed with this method using linear or quadratic scalar fields. In particular, a translation can be achieved by using two linear fields

$$e(\mathbf{x}) = \mathbf{u}(\mathbf{x} - \mathbf{c})^T, f(\mathbf{x}) = \mathbf{w}(\mathbf{x} - \mathbf{c})^T, \tag{2}$$

where $\mathbf{u}$ and $\mathbf{w}$ are orthogonal normalized vectors and $\mathbf{c}$ is an arbitrary center point. Since $\mathbf{u}$ and $\mathbf{w}$ are the gradients of e and f, respectively, $\mathbf{u} \times \mathbf{w}$ defines the translation direction. A rotational vector field can be constructed from a linear and a quadratic field:

$$e(\mathbf{x}) = \mathbf{a}(\mathbf{x} - \mathbf{c})^T, f(\mathbf{x}) = (\mathbf{a} \times (\mathbf{x} - \mathbf{c})^T)^2 \tag{3}$$

The rotation axis is defined by the normalized vector $\mathbf{a}$, while $\mathbf{c}$ describes the rotation center.

By performing a stream line integration (or path line integration for the time-dependent case) of each mesh vertex in the resulting vector fields, it is possible to rotate and translate a mesh arbitrarily. Obviously, such transformations can be achieved more easily and efficiently by other means. However, we can create more complex and local deformations by blending the scalar functions e, f using a third function b, the *blending function.* In [1], the blending is done in a piecewise manner, where b is the function of a distance field. Here, we define b more generally as a C^2 continuous field $b : \mathbb{R}^3 \rightarrow [0,1]$. Given b, we can define the blended fields

$$p(\mathbf{x}) = b(\mathbf{x}) \cdot e(\mathbf{x}) \tag{4}$$

$$q(\mathbf{x}) = b(\mathbf{x}) \cdot f(\mathbf{x}). \tag{5}$$

Using (2) for translations or (3) for rotations, we can insert (4) and (5) into (1) to obtain a divergence-free vector field, which describes

– a full translation/rotation at points where $b(\mathbf{x}) = 1$,
– a zero-deformation where $b(\mathbf{x}) = 0$,
– a smoothly blended deformation for points where $0 < b(\mathbf{x}) < 1$.

By defining an appropriate blending function b, it is possible to specify which points in space should be deformed by what amount. Figure 1 demonstrates this in the 2D setting. Since the blending is done before the cross product of the gradients is computed (Equation 1), the resulting vector field is still divergence-free. Since p and q are C^2 continuous, the resulting vector field $\mathbf{v}$ is C^1 continuous [1].

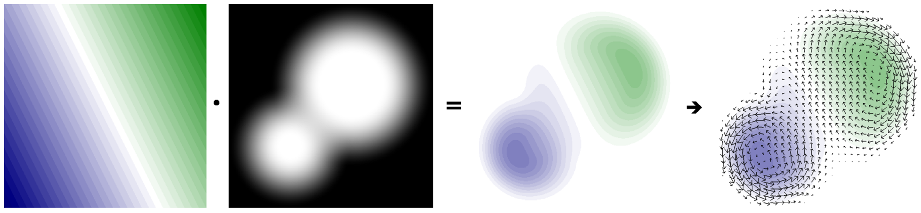

Fig. 1. Blending the deformation. A linear field (left), describing a translation along its isolines, is multiplied with a blending function (center left). The result is a blended field (center right) from which a divergence-free deformation field (right) can be computed.

3 Deformation Blending

In the following sections, we will show a method to construct a blending function which can be used for boundary constraint modeling based on VFSD.

3.1 Implicit Boundaries

From a technical point of view, we don't use the term *boundary constraints* in the sense of an optimization problem, but use boundary constraints as user defined positions in space where the surface should be deformed in a prescribed manner. From the user's point of view, our approach resembles other boundary constraint modeling approaches: the user draws two boundaries on the surface, where the outer boundary defines the support of the deformation, while the inner boundary defines the control handle of the deformation. Figure 2 illustrates this.

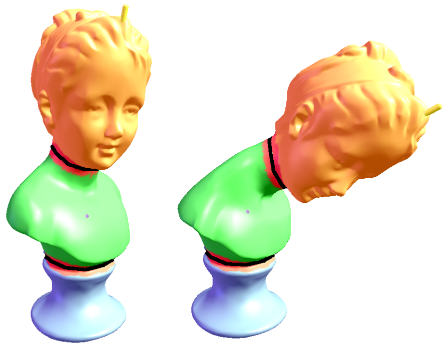

Fig. 2. By drawing boundaries onto the surface, the user can define a support region (here the body) and a handle region (head)

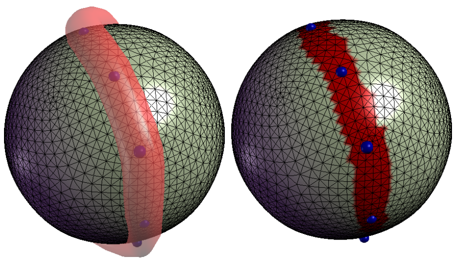

Fig. 3. Left: an implicit boundary defined by a few points. Right: the corresponding vertices marked.

Since VFSD is not a surface-based technique but describes space deformations, the boundaries have to be defined in space and not only on the surface. Therefore, we formulate them implicitly. We do so by constructing a smooth implicit function for both the inner and the outer boundary. More precisely, we define for each boundary a closed piecewise linear curve, i.e. a ring of connected line segments. Then we use an approximate smooth distance field to each curve as implicit function. Given n points $\mathbf{b}_j$, $1 <= j <= n$ with $\mathbf{b}_{n+1} := \mathbf{b}_1$, defining such a polygonal curve, we can compute an approximate distance field using the technique from [26] as follows: given the Euclidian distance fields $l_j(\mathbf{x})$ of each line segment defined by the endpoints $\mathbf{b}_j$, $\mathbf{b}_{j+1}$, we get

$$d(\mathbf{x}) = \frac{1}{\sqrt[k]{\displaystyle\sum_{i=1}^{n} \frac{1}{(l_i(\mathbf{x}))^k}}} \tag{6}$$

This corresponds to the *R-equivalence* $l_1(\mathbf{x}) \sim ... \sim l_n(\mathbf{x})$ described in [26], which joins the distance fields $l_j(\mathbf{x})$ to a smooth approximate one. k is a positive integer, which basically controls the “exactness” of the distance field: the greater k, the more the approximation approaches the real distance field of the polygonal curve, which, in general, contains discontinuities. The smaller we choose k, the smoother the approximation becomes. In our implementation, we use $k = 2$ in order to obtain smooth deformations even for coarse polygons.

Given such a scalar field for both the inner and the outer boundary, i.e. $d_i(\mathbf{x})$ and $d_o(\mathbf{x})$, we want to constrain the deformation as follows:

- if $d_i(\mathbf{x}) = t_i$, we want full deformation,
- if $d_o(\mathbf{x}) = t_o$, we want no deformation,
- else, we want a smooth blending between full and zero deformation.

t_i and t_o are user-adjustable thresholds which define the thickness of the innner and outer boundary, respectively. Visually, an implicit boundary can be seen as a closed tube with adjustable thickness running over the surface, as depicted in Figure 3. Besides the necessary implicit formulation, this has the advantage that the number of polygon vertices is independent of the mesh resolution, and the user can define smooth boundaries with only a few points, which is especially useful for parallel computation on the GPU. In order to avoid discontinuities in the deformation, the user has to avoid intersections between inner and outer boundary. For instance, when deforming the Dragon's mouth as in Figure 4, the boundary thickness has to be chosen such that the boundaries don't touch each other between upper and lower jaw. The points $\mathbf{b}_j$ and t_i, t_o have to be chosen such that the boundary area on the surface is connected, i.e., it divides the shape into two parts. For given t_i, t_o, this can always be achieved by increasing the density of the control points $\mathbf{b}_j$ and by placing them close to sharp features.

3.2 Smooth Blending

Having the implicit boundaries defined, we need to construct a function that can be used to blend smoothly from full deformation to zero deformation between inner and outer boundary. This can be accomplished in a straightforward way by interpolation with inverse distance weighting [27]. We define the blending function as

$$b(\mathbf{x}) = \frac{1/(d_i(\mathbf{x}) - t_i))^3 \cdot 1 + 1/(d_o(\mathbf{x}) - t_o))^3 \cdot 0}{1/(d_i(\mathbf{x}) - t_i))^3 + 1/(d_o(\mathbf{x}) - t_o))^3} \tag{7}$$

In the limit, the following holds: $b(\mathbf{x}) = 1$ for $d_i(\mathbf{x}) = t_i$ and $b(\mathbf{x}) = 0$ for $d_o(\mathbf{x}) = t_o$. Because of the cubic weights, $b(\mathbf{x})$ has two vanishing derivatives at points with $d_i(\mathbf{x}) = t_i$ or $d_o(\mathbf{x}) = t_o$. As we will see later, this is an important property which is needed to perform the deformation of the mesh in a piecewise manner. A further degree of freedom can be achieved by multiplying the weights with user-defined factors. This is especially useful to control bend deformations.

So far, we have a C^2 continuous blending function b which can be used together with Equations (4), (5) to construct blended scalar fields with (2) for translations or with (3) for rotations. Using (1), we can construct a divergence-free vector field, which deforms the mesh (nearly) according to the boundary constraints. Vertices on the inner boundary, i.e. with $d_i(\mathbf{x}) = t_i$, are fully deformed because for them $b(\mathbf{x}) = 1$ holds. Vertices on the outer boundary, i.e. with $d_o(\mathbf{x}) = t_o$, are not deformed because for them $b(\mathbf{x}) = 0$ holds. For all other vertices, the deformation is smoothly blended between full and zero deformation.

3.3 Piecewise Deformation

By simply applying such a deformation to the whole mesh, vertices outside of the support region will be deformed as well and vertices in the handle region won't undergo a constant deformation, in general. We therefore need to perform the deformation in a piecewise fashion, which is quite simple. Vertices belonging to the handle region and the inner boundary are deformed in full, i.e. they undergo a constant translation or rotation (e.g. the head and the boundary on the neck in Figure 2). Vertices in the support region (body between the boundaries in Figure 2), not belonging to any boundary, are deformed using the blended fields (4), (5). All other vertices are not deformed at all. Thanks to the two vanishing derives of the blending function at the boundaries, this piecewise deformation is C^1 continuous. Furthermore, the property of volume preservation still holds as long as no self-intersections occur. Self-intersections can only occur between the deforming parts of the mesh and the non-deforming parts.

Mathematically, we also could instead define the blending function in a piecewise fashion, such that the resulting deformation would be exactly the same. Technically, deforming the mesh in the piecewise manner described above is more efficent because the handle region can be deformed directly using a rigid transformation and the zero-deformation vertices are not considered at all.

3.4 Integration in Space Time

The description of the blending function is based on a time-independent context. However, since the inner boundary is actually moving over time, the blending function has to be updated within each integration step. This is straightforward: at the beginning of the integration, the inner boundary polygon is at its original position. Then, with each integration step, the position is updated by the amount corresponding to the step size. For instance, for a rotation, the polygon points are rotated step by step until the full rotation is reached. In [1], a more detailed description of the integration process can be found.

4 Applications

In principle, every deformation that is constructable as the cross product of two gradients can be used with the described approach. However, we confine ourselves to two simple, yet effective transformations: rotation and translation. Obviously, a scaling transformation wouldn't make sense, since we want to preserve the shape volume. As we will see in this section, this toolset allows for a variety of useful deformations.

In order to control translation and rotation, the user can place a *knob* somewhere on the mesh surface, and a *joint* somewhere in space. In most of the figures in this paper, the knob is depicted as a yellow stick, e.g. on top of the bust in Figure 2. The joint is a small white sphere, usually placed somewhere in the support region. The knob resembles typical *Gizmo* objects found in many shape modeling systems, which can be used to control transformations by grabbing and dragging it at different points. The knob is a simplified version because only translation and rotation are supported.

4.1 Rotation

In order to rotate the handle region, the user drags the knob, where the knob movement is constrained to a fixed radius about the joint position. From the displacement of the knob position, the rotation axis and angle can be determined

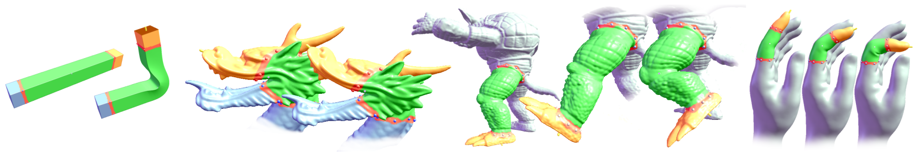

Fig. 4. Popular application scenarios for boundary constraint modeling

with respect to the joint position. Using the joint position as rotation center **c** in (3), a deformation that bends the shape can be accomplished by integrating the mesh vertices until the rotation angle is reached. In contrast to [1], where the shape is continually updated, the integration restarts from the original mesh every time the knob position changes. Figure 4, as well as Figure 2, shows this approach applied in various scenarios known from the Literature. The deformation looks rather realistic thanks to the volume-preservation and avoidance of self-intersections and even high resolution models can be deformed interactively.

Figure 5 demonstrates how local details are deformed: the "teeth" of the comb-like shape don't touch each other during deformation and their distortion seems appropriate with respect to the global deformation.

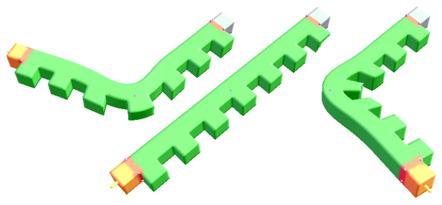

Fig. 5. Local details are slightly distorted in strongly deformed areas (left) and never intersect with each other (right)

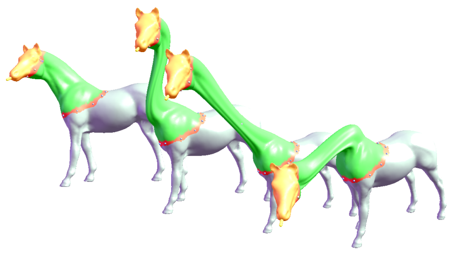

Fig. 6. By translating the horse head, the neck deforms in a natural manner

Also twisting is possible by simply using the vector between joint and knob as rotation axis. A more uniform twisting deformation can be achieved with two quadratic scalar fields [1].

4.2 Translation

When the user wants to translate the handle region, he or she can drag the knob freely in space. The joint is ignored for this deformation type. However, also the translation (2) requires a center point **c**. In this case we use the barycenter of the control points of the inner boundary.

In Figure 6, the user drags the head of a horse model. The shape of the neck automatically adapts to the new position. Due to the constant volume, the neck becomes thinner when the head is pulled.

Interesting effects can be achieved by carefully selecting boundaries and moving the handle parallel to the surface: as shown in Figure 7, the deformation automatically produces "wrinkles" on the cheek of the face, which is a result of the volume preservation and the prevention of foldovers. Although the "wrinkles" appear to be rather strong, no self-intersections of the surface occur.

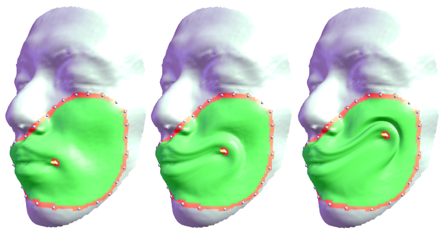

Fig. 7. "Wrinkles" can be produced by translating the handle accross the surface. Although they appear to be rather strong, no self-intersections of the surface occur.

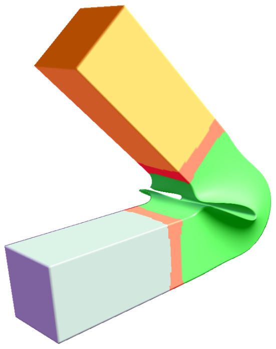

Fig. 8. When the boundaries move close to each other, the shape is distorted

5 Implementation and Performance

As shown in [1], the performance of the integration can be increased by a large amount by shifting the computation to the GPU, where multiple path lines can be integrated in parallel. This is also possible with our approach: since the number of boundary control points is usually low, these points can be passed to the shader as a simple array. We implemented two vertex programs, one for translation and one for rotation, which can be controlled by passing translation vector, rotation axis, angle etc. to them. During the integration, the polygon points of the inner boundary are updated internally with respect to the translation/rotation, as described in Section 3.4. Except for the extraction of the handle and support region (Section 3.3), no further preprocessing is required. After integration, vertex positions are read back from video memory and the mesh normals are computed. An alternative approach would be to compute normals directly on the GPU using the Jacobian of the vector field, similar to [18]. This would decrease the integration performance because of the necassary computation of Jacobians in each integration step, but would clear the CPU from doing this task and redundatize the readback of vertex positions. However, we have not tested this alternative yet. As we will see in the following performance analysis, the normal computation on the CPU makes only a small fraction of the total deformation time.

The performance of the approach strongly depends on the “amount” of deformation, i.e. how far the handle is translated or rotated. This is because the numerical path line integration adapts its step size according to the complexity of the vector field and the duration of the integration. In order to present a meaningful statement about performance, we measured the times of usual “real-world” deformations, namely the ones depicted in Figure 4. Table 1 lists the deformed shapes (from left to right in Figure 4) and the benchmark results. *v/s*

Table 1. Performance benchmark: complex meshes can be deformed interactively

shape	vertices	boundary points	*v/s* (integration)	*v/s* (integration + normals)
box	47,296	8	788,267	647,890
dragon	86,814	23	413,400	369,421
leg 1	31,014	17	449,478	382,889
leg 2	31,014	17	443,057	364,871
finger	2725	14	454,167	454,167

(integration) is the number of vertices per second for integration only and *v/s (integration + normals)* the number of vertices per second for integration plus normal computation. The measurements were made on a 2.6 GHz Opteron CPU and a GeForce 6800 GT graphics card. They show that complex meshes can be deformed interactively.

6 Conclusion

We presented a shape deformation technique based on vector field integration which incorporates implicit boundaries to steer the impact of the deformation.

By using the VFSD technique [1], our deformations are volume-preserving and foldover-free, giving the user the impression of working with real, incompressible material. While the original VFSD approach defined the regions of influence by simple implicit objects, our method constructs a smooth blending function based on implicit boundaries. That way, the user can specifiy the impact of the deformation directly on the surface of the shape.

Thanks to the polygonal representation of the implicit boundaries, they are independent of the resolution of the deformed mesh. In most cases, a small number of control points suffices to define the boundaries.

Since the description of boundaries is simple (a small set of points), the numerical path line integration can be computed on the GPU and even complex models can be deformed interactively.

There are some restrictions of our approach which should be addressed in future research.

Self-intersections. Self-intersections are only avoided for the deforming regions of the shape surface because the deformation is carried out in a piecewise fashion. It is e.g. possible to bend the finger in Figure 4 such that the finger tip intersects the thumb or other parts of the hand. An additional collision detection would solve the problem, but would also drop performance.

Close boundaries. When inner and outer boundary are close to each other, the gradient of the resulting blending function is high in these regions. This can result in unpleasing deformations. E.g., when an extreme bending is performed (Figure 8), the boundaries approach each other, and the box is distorted more and more at its center (but nervertheless preserves its volume). A possible solution would be to perform such deformations (even more) piecewise, by using for example a third boundary between inner and outer boundary and constructing two blending functions: the first blends between inner and central boundary, the other between central and outer boundary.

References

1. von Funck, W., Theisel, H., Seidel, H.P.: Vector field based shape deformations. ACM Trans. Graph. 25(3), 1118–1125 (2006)
2. Welch, W., Witkin, A.: Variational surface modeling. In: SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques, pp. 157–166. ACM Press, New York (1992)
3. Taubin, G.: A signal processing approach to fair surface design. In: SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, pp. 351–358. ACM Press, New York (1995)
4. Kobbelt, L., Campagna, S., Vorsatz, J., Seidel, H.P.: Interactive multi-resolution modeling on arbitrary meshes. In: SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques, pp. 105–114. ACM Press, New York (1998)
5. Botsch, M., Kobbelt, L.: An intuitive framework for real-time freeform modeling. ACM Trans. Graph. 23(3), 630–634 (2004)
6. Lipman, Y., Sorkine, O., Cohen-Or, D., Levin, D., Rössl, C., Seidel, H.P.: Differential coordinates for interactive mesh editing. In: Proceedings of Shape Modeling International, pp. 181–190. IEEE Computer Society Press, Los Alamitos (2004)
7. Sorkine, O., Lipman, Y., Cohen-Or, D., Alexa, M., Rössl, C., Seidel, H.P.: Laplacian surface editing. In: Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing, Eurographics Association, pp. 179–188 (2004)
8. Yu, Y., Zhou, K., Xu, D., Shi, X., Bao, H., Guo, B., Shum, H.Y.: Mesh editing with poisson-based gradient field manipulation. ACM Trans. Graph. 23(3), 644–651 (2004)
9. Lipman, Y., Sorkine, O., Levin, D., Cohen-Or, D.: Linear rotation-invariant coordinates for meshes. ACM Trans. Graph. 24(3), 479–487 (2005)
10. Zayer, R., Rössl, C., Karni, Z., Seidel, H.P.: Harmonic guidance for surface deformation. In: Computer Graphics Forum, Proceedings of Eurographics 2005, Dublin, Ireland, Eurographics, vol. 24, pp. 601–609. Blackwell, Oxford (2005)

11. Sederberg, T., Parry, S.: Free-form deformation of solid geometric models. In: SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques, pp. 151–160. ACM Press, New York (1986)
12. Coquillart, S.: 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, pp. 187–196. ACM Press, New York (1990)
13. MacCracken, R., Joy, K.: Free-form deformations with lattices of arbitrary topology. In: SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pp. 181–188. ACM Press, New York (1996)
14. Barr, A.: Global and local deformations of solid primitives. In: SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques, pp. 21–30. ACM Press, New York (1984)
15. Singh, K., Fiume, E.: Wires: a geometric deformation technique. In: SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques, pp. 405–414. ACM Press, New York (1998)
16. Hirota, G., Maheshwari, R., Lin, M.: Fast volume-preserving free form deformation using multi-level optimization. In: Proceedings Solid Modeling and applications, pp. 234–245 (1992)
17. Hsu, W., Hughes, J., Kaufman, H.: Direct manipulation of free-form deformations. In: SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques, pp. 177–184. ACM Press, New York (1992)
18. Botsch, M., Kobbelt, L.: Real-time shape editing using radial basis functions. Computer Graphics Forum (Proceedings Eurographics 2005) 24(3), 611–621 (2005)
19. Desbrun, M., Gascuel, M.P.: Animating soft substances with implicit surfaces. In: SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, pp. 287–290. ACM Press, New York (1995)
20. Rappoport, A., Sheffer, A., Bercovier, M.: Volume-preserving free-form solids. IEEE Transactions on Visualization and Computer Graphics 2(1), 19–27 (1996)
21. Aubert, F., Bechmann, D.: Volume-preserving space deformation. Comput. and Graphics 21(5), 6125–6639 (1997)
22. Angelidis, A., Cani, M.P., Wyvill, G., King, S.: Swirling-sweepers: Constant-volume modeling. In: Computer Graphics and Applications, 12th Pacific Conference on (PG'04), pp. 10–15 (2004)
23. Mason, D., Wyvill, G.: Blendeforming: Ray traceable localized foldover-free space deformation. In: CGI '01: Proceedings of the International Conference on Computer Graphics, Washington, DC, USA, p. 183. IEEE Computer Society, Los Alamitos (2001)
24. Gain, J.E., Dodgson, N.A.: Preventing self-intersection under free-form deformation. IEEE Transactions on Visualization and Computer Graphics 7(4), 289–298 (2001)
25. Angelidis, A., Wyvill, G., Cani, M.P.: Sweepers: Swept user-defined tools for modeling by deformation. In: Proceedings of Shape Modeling and Applications, pp. 63–73. IEEE, Orlando (June 2004)
26. Biswas, A., Shapiro, V.: Approximate distance fields with non-vanishing gradients. Graphical Models 66(3), 133–159 (2004)
27. Shepard, D.: A two-dimensional interpolation function for irregularly-spaced data. In: Proceedings of the 1968 23rd ACM national conference, pp. 517–524. ACM Press, New York (1968)