Sweepers & Swirling-sweepers

Alexis Angelidis

Sweepers & Swirling-Sweepers

Alexis Angelidis

University of Otago, New Zealand

Abstract

Sweepers and swirling-sweepers are operations for modeling by space deformation. The artist describes a deformation as paths through which tools are moved. The movement of a tool causes a deformation of the working shape along the path of the tool. This is in accordance with a clay modeling metaphor, easy to understand and predict. It is desirable that deformations for modeling are 'foldover-free', that is parts of deformed space cannot overlap so that the deformations are reversible. Both sweepers and swirling-sweepers satisfy this criteria. In addition, swirling-sweepers preserve the shape's volume.

1. Introduction

In Computer Graphics, in the context of interactive shape modeling, a common characteristic of popular techniques is the possibility for the artist to operate on a shape by modifying directly the shape's mathematical description. But with the constant increase of computing power, it is realistic and more effective to insert some interface between the artist and the mathematics describing a shape.

Space deformation is a family of techniques that permits describing operations on a shape independently from that shape's description. With this separation, new shape descriptions can easily benefit from existing space deformation, and further development can be carried in parallel. While space deformation has been used for solving a wide range of problems in Computer Graphics, they are missing a framework specific to interactive shape modeling. Sweepers is a framework for defining shape operations, in which the basis of operations is simply gesture.

Shapes produced with sweepers are coherent because sweepers are foldover-free: there is no ambiguity as to which points of space belong to the shape. A non foldover-free deformation would produce a self intersection of the shape, which cannot be cured with any space deformation. Sweepers were introduced in [AWC04], and are presented in Section 4.

In addition to *gesture as the basis of creation* and *shape coherency*, another concept familiar to most users is the *preservation of material*. A shape modeling technique that preserves volume would take even more advantage of user a priori knowledge of shapes. Swirling-sweepers is a technique that preserves a shape's volume independently from its description. In conjunction with any other sweeper operation, the volume of a shape can be increased, decreased or preserved. Swirling-sweepers were introduced in [ACWK04] and are presented in Section 5.

Sweeper and swirling-sweeper operations are independent from any shape description, since they are deformation of space. In order to visualize their effect on a shape, we propose in Section 6 a shape description which is suitable for the task of interactive shape modeling, and animation to some extent [AW04].

2. Modeling with deformation

A shape is the result of repeated deformation of the space in which the initial shape is embedded. A convenient formalism can be used for specifying any modeling operation by deformation: the *modeling equation* gives the final shape $S(t_n)$ as a function the initial shape $S(t_0)$:

$$S(t_n) = \{ \bigOmega_{i=0}^{n-1} f_{t_i \mapsto t_{i+1}}(\mathrm{p}) | \mathrm{p} \in S(t_0) \} \quad (1)$$

$$\text{where } \bigOmega_{i=0}^{n-1} f_{k_i \mapsto k_{i+1}}(\mathrm{p}) = f_{k_{n-1} \mapsto k_n} \circ \cdots \circ f_{k_0 \mapsto k_1}(\mathrm{p})$$

The operator Ω expresses the finite repeated composition of functions. Each function $f_{t_i \mapsto t_{i+1}} : \mathbb{R}^3 \mapsto \mathbb{R}^3$ is a deformation that transforms every point p of space at time t_i into a point of space at time t_{i+1}. Sections 3, 4 and 5 will focus on defining functions $f_{t_i \mapsto t_{i+1}}$.

Normal Deformation: Computing accurate normals to the surface is very important, since the normals' level of quality will dramatically affect the visual quality of the shape. Let us recall that in order to compute the new normals, the previous normals are multiplied by the co-matrix[†] of the Jacobian [Bar84]. The Jacobian of f at p is the matrix $J(f,\mathrm{p}) = (\frac{\partial f}{\partial x}(\mathrm{p}), \frac{\partial f}{\partial y}(\mathrm{p}), \frac{\partial f}{\partial z}(\mathrm{p}))$. Let us also recall that the following expression is a convenient way to compute the co-matrix of $J = (\vec{\mathrm{j}}_x, \vec{\mathrm{j}}_y, \vec{\mathrm{j}}_z)$, where the vectors $\vec{\mathrm{j}}_x$, $\vec{\mathrm{j}}_y$ and $\vec{\mathrm{j}}_z$ are column vectors:

$$J^C = \left(\vec{\mathrm{j}}_y \times \vec{\mathrm{j}}_z, \vec{\mathrm{j}}_z \times \vec{\mathrm{j}}_x, \vec{\mathrm{j}}_x \times \vec{\mathrm{j}}_y\right) \tag{2}$$

3. Related work

This section overviews several existing space deformation techniques, organized in three groups: axial deformations, lattice-based deformations and tool-based deformations. For the sake of clarity, we present existing space deformations aligned with the axes $\vec{\mathrm{e}}_x$, $\vec{\mathrm{e}}_y$ and $\vec{\mathrm{e}}_z$ and within the unit cube $[0,1]^3$, whenever possible. But a mere change of coordinates enables the artist to place the deformation anywhere in space. Note that affine transformations are the simplest case of space deformations.

3.1. Axial space deformations

Axial space deformations are a subset of space deformations whose control-points are geometrically connected along a curve. The curve may be initially straight or bent. To compare existing deformation techniques from the same point of view, we use $\vec{\mathrm{e}}_z$ as the common axis of deformation, which leads to slight reformulation in a few cases.

3.1.1. Global and local deformations of solid primitives

A. Barr defines space tapering, twisting and bending via matrices whose components are functions of one space coordinate [Bar84]. We denote $(x,y,z)^\top$ the coordinates of a point. We show in Figures 1, 2, and 3 the effects of these operations, and we give their formula in the form of 4×4 homogeneous matrices to be applied to the coordinates of every point in space to be deformed.

3.1.1.1. Tapering operation: The function r is monotonic in an interval, and is constant outside that interval.

$$\begin{pmatrix} r(z) & 0 & 0 & 0 \\ 0 & r(z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 1: *Taper deformation of a super-ellipsoid shape. A description of the shape can be found in [Gla89].*

[†] Matrix of the co-factors

3.1.1.2. Twisting operation: The function θ is monotonic in an interval, and is constant outside that interval.

$$\begin{pmatrix} \cos(\theta(z)) & -\sin(\theta(z)) & 0 & 0 \\ \sin(\theta(z)) & \cos(\theta(z)) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

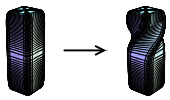


Figure 2: *Twist deformation of a super-ellipsoid.*

3.1.1.3. Bending operation: This operation bends space along the axis y, in the $0 < z$ half-space. The desired radius of curvature is specified with ρ. The angle corresponding to ρ is $\theta = \hat{z}/\rho$. The value of $\hat{z}$ is the value of z, clamped in the interval $[0, z_{\max}]$.

$$\begin{pmatrix} \cos\theta & 0 & \sin\theta & \rho - \rho\cos\theta - \hat{z}\sin\theta \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & \rho\sin\theta - \hat{z}\cos\theta \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 3: *Bend deformation of a super-ellipsoid.*

A. Barr observes that rendering the deformed shape with rays of light is equivalent to rendering the undeformed shape with curves of light. The curves of light are obtained by applying the inverse of the deformation to the rays. Because the deformation he proposes are not local, the portions of the rays to deform can be quite large.

3.1.2. A generic implementation of axial procedural deformation techniques

C. Blanc extends A. Barr's work to mold, shear and pinch deformations [Bla94]. Her transformations use a function of one or two components. She calls this function the *shape function*. Examples are shown in Figures 4, 5, and 6.

$$\begin{pmatrix} r(z) & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

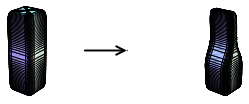


Figure 4: *Pinch deformation of a super-ellipsoid.*

$$\begin{pmatrix} r(\tan^{-1}(x,y)) & 0 & 0 & 0 \\ 0 & r(\tan^{-1}(x,y)) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

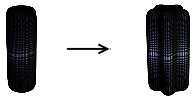


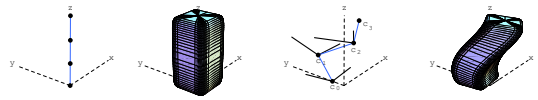
Figure 5: *Mold deformation of a super-ellipsoid.*

$$\begin{pmatrix} 1 & 0 & 0 & s(z) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 6: *Shear deformation of a super-ellipsoid.*

3.1.3. A generalized de Casteljau approach to 3d free-form deformation

Y.K. Chang and A.P. Rockwood propose a polynomial deformation that efficiently achieves "Barr"-like deformations and more [CR94], using a Bézier curve with coordinate sets defined along $\vec{e}_z$ at the curve's control knots $(z_0, z_1 \ldots, z_n) \in [0,1]^{n+1}$. A reference straight segment, $z \in [0,1]$, is deformed by specification of coordinate sets $(c_i, \vec{u}_i, \vec{v}_i, \vec{w}_i)$ along that segment. The shape follows the deformation of the segment, as shown in Figure 7.



straight axis | initial shape | control points and handles | deformed shape

Figure 7: *Example of the deformation of Y.K. Chang and A.P.Rockwood applied to a super-ellipsoid. There is no need to define a pair of handles for the end control point.*

To compute the image q of a point p of the original shape, the matrix transforming a point to a local coordinate set is needed:

$$M_i = \begin{pmatrix} u_{i,x} & v_{i,x} & w_{i,x} & c_{i,x} \\ u_{i,y} & v_{i,y} & w_{i,y} & c_{i,y} \\ u_{i,z} & v_{i,z} & w_{i,z} & c_{i,z} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (3)$$

where $\vec{w}_i = c_{i+1} - c_i$, and $\vec{u}_i$, $\vec{v}_i$ are the handles.

Using this matrix, the deformation of a point is obtained recursively with the de Casteljau algorithm for evaluating a Bézier curve:

$$f_i^j(p) = (1 - p_z) f_i^{j-1}(p) + p_z f_{i+1}^{j-1}(p) \quad (4)$$

$$\text{where } f_i^0(p) = M_i \cdot p$$

The original generalized de Casteljau algorithm presented by Y.K. Chang and A.P. Rockwood is a recursion on affine transformations rather than on points. They remark that their recursion simplifies to the classic de Casteljau algorithm when the affine transformations are degenerate, and use the degenerate case in all their examples. As we show in Figure 8, this method is capable of performing "Barr"-like deformations and more.

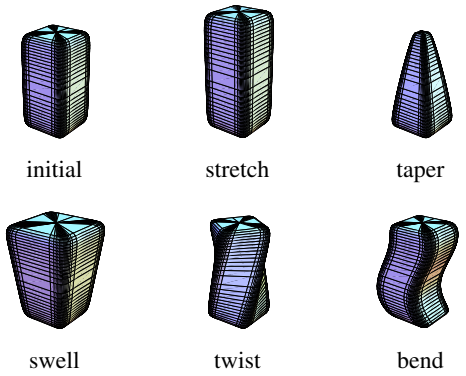


Figure 8: *Deformation of a super-ellipsoid.*

3.1.4. Axial deformation

The limitation of the methods presented so far is the initial rectilinear axis. If the shape is initially excessively bent, the manipulation of an initially straight control axis will not induce a predictable behavior of the shape. F. Lazarus et al. develop an extension of axial-based deformations using an initially curved axis [LCJ94]. Let us define a parametric curve $c(u)$. A point p in space is attached to local coordinates along the curve. The origin of this local coordinate system is $c(u_p)$, the closest point to p on the curve, and the axes are those of an extended Frenet frame that discards vanishing points [Blo90]. To find the closest point to p on curves, B. Crespin proposes an efficient algorithm based on subdivision [Cre99]. The axes are computed by propagating along the curve a frame defined at one extremity of the curve. The axes consist of three vectors: a tangent $\vec{t}(u)$, a normal $\vec{n}(u)$ and a binormal $\vec{b}(u)$. The propagated frame is computed as follows:

- the unit tangent at the origin is given by the equation of the curve:
  $\vec{t}(0) = \frac{dc(0)}{du} / \| \frac{dc(0)}{du} \|$.
- the normal and binormal are given by the Frenet frame, or can be any pair of unit vectors such that the initial frame is orthonormal.

To compute the next frame, a rotation matrix is needed. The purpose of this matrix is to minimize torsion along the curve. Numerous constructions of the rotation matrix require a sim-

ple formula:

$$R = \begin{pmatrix} a_{xx}+\theta & a_{xy}+b_z & a_{zx}-b_y \\ a_{xy}-b_z & a_{yy}+\theta & a_{yz}+b_x \\ a_{zx}+b_y & a_{yz}-b_x & a_{zz}+\theta \end{pmatrix} \quad (5)$$

$$\text{where} \quad \begin{array}{ll} (a_x,a_y,a_z)^\top = \frac{\vec{t}(u_i)\times\vec{t}(u_{i+1})}{\|\vec{t}(u_i)\times\vec{t}(u_{i+1})\|} & \alpha = 1-\theta \\ \theta = \vec{t}(u_i)\cdot\vec{t}(u_{i+1}) & \beta = \sqrt{1-\theta^2} \end{array} \quad (6)$$

$$\begin{array}{lll} a_{xx} = \alpha a_x^2 & a_{xy} = \alpha a_x a_y & b_x = \beta a_x \\ a_{yy} = \alpha a_y^2 & a_{yz} = \alpha a_y a_z & b_y = \beta a_y \\ a_{zz} = \alpha a_z^2 & a_{zx} = \alpha a_z a_x & b_z = \beta a_z \end{array} \quad (7)$$

Given a frame at parameter u_i, the next axes of a frame at u_{i+1} are computed as follows:

- the tangent is defined by the equation of the curve: $\vec{t}(u_{i+1}) = \frac{dc(u_{i+1})}{du} / \| \frac{dc(u_{i+1})}{du} \|$.
- the normal is given by the rotation of the previous normal: $\vec{n}(u_{i+1}) = R \cdot \vec{t}(u_i)$.
- the binormal is given by a cross product: $\vec{b}(u_{i+1}) = \vec{t}(u_i) \times \vec{n}(u_i)$.

The choice of the size of the step, $u_{i+1} - u_i$, depends on the trade-off between accuracy and speed. B. Crespin extends the axial deformation to surface deformation [Cre99].

3.1.5. Wires: a geometric deformation technique

K. Singh and E. Fiume introduce *wires*, a technique which can easily achieve a very rich set of deformations with curves as control features [SF98]. Their technique is inspired by armatures used by sculptors.

A wire is defined by a quadruple (R, W, s, r): the reference curve R, the wire curve W, a scaling factor s that controls bulging around the curve, and a radius of influence r. The set of reference curves describes the armature embedded in the initial shape, while the set of wire curves defines the new pose of the armature.

On a curve C, let p_C denote the parameter value for which $C(p_C)$ is the closest point to p. Let us also denote $C'(p_C)$ the tangent vector at that parameter value.

The reference curve, R, generates a scalar field $F : \mathbb{R}^3 \mapsto [0,1]$. The function F which decreases with the distance to R, is equal to 1 along the curve and equals 0 outside a neighborhood of radius r. The algorithm to compute the image q of a point p influenced by a single deformation consists of three steps, illustrated in Figure 9:

- Scaling step. The scaling factor is modulated with F. The image of a point p after scaling is: $p_s = R(p_R) + (p - R(p_R))(1 + sF(p))$, where p_R denotes the parameter value for which $R(p_R)$ is the closest to p.
- Rotation step. Let θ be the angle between the tangents $R'(p_R)$ and $W'(p_R)$. The point p_s is rotated around axis $R'(p_R) \times W'(p_R)$ about center $R(p_R)$ by the modulated angle $\theta\, F(p)$. This results in point p_r
- Translation step. Finally, a translation is modulated to produce the image $p_{def} = p_r + (W(p_R) - R(p_R))$.

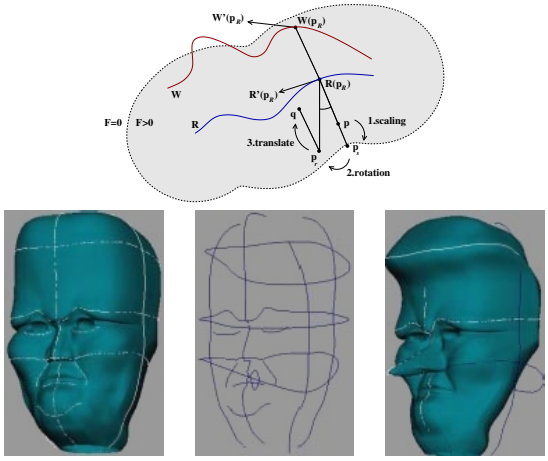


Figure 9: *Top: deformation of a point by a single wire. Bottom: deformation of a shape with multiple wires (courtesy of [SF98]).*

They propose different blending methods in the case when a point is subject to multiple wires. These methods work by taking weighted combinations of the individually deformed point. Let us denote p_i the deformation of p by wire i. Let $\Delta p_i = p_i - p$. The simplest deformation is:

$$p_{def} = p + \frac{\sum_{i=1}^{n} \Delta p_i \|\Delta p_i\|^m}{\sum_{i=1}^{n} \|\Delta p_i\|^m}$$

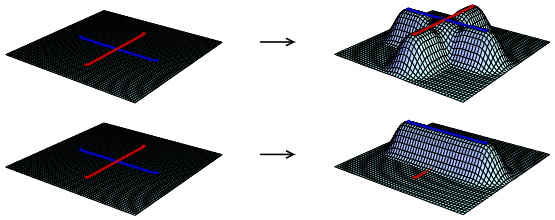


Figure 10: *Blending weights based on summed displacement magnitudes. This blending is not free from artifacts: notice the creases around the intersection in the upper-right figure.*

The scalar m is defined by the artist. This expression is not defined when m is negative and $\|\Delta p_i\|$ is zero. To fix this, they suggest to omit the wires for which this is the case. Their second solution is to use another blending defined for both positive and negative values of m:

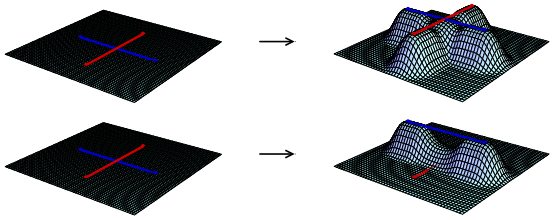
$$\mathrm{p}_{def} = \mathrm{p} + \frac{\sum_{i=1}^{n} \Delta\mathrm{p}_i \prod_{j\neq i} \|\Delta\mathrm{p}_j\|^{|m|}}{\sum_{i=1}^{n} \prod_{j\neq i} \|\Delta\mathrm{p}_j\|^{|m|}}$$

Reference curves Wire curves

Figure 11: *Blending weights based on multiplied displacement magnitudes. The deformation is defined at the intersection of the reference curves.*

In order to use unmoved wires as anchors that hold the surface, they use $\mathrm{F}_i(\mathrm{p})$ instead of $\Delta\mathrm{p}_i$ as a measure of proximity:

$$\mathrm{p}_{def} = \mathrm{p} + \frac{\sum_{i=1}^{n} \Delta\mathrm{p}_i \mathrm{F}_i(\mathrm{p})^m}{\sum_{i=1}^{n} \mathrm{F}_i(\mathrm{p})^m}$$



Reference curves Wire curves

Figure 12: *Blending weights based on influence function. The unmoved wire holds space still. This blending is not free from artifacts: notice the creases around the intersection in the upper-right figure.*

Other capabilities of wires can be found in the original paper [SF98]. Note that the expensive part of the algorithm is computing the distance from each curve to each deformed surface point.

3.1.6. Blendeforming: ray traceable localized foldover-free space deformation

As explained in the introduction, there are practical reasons for which a space deformation should be foldover-free. D. Mason and G. Wyvill introduce blendeforming [MW01]. A deformation is specified by moving a point or the control points of a curve along a constrained direction. Space follows the deformation of these control features in a predictable manner.

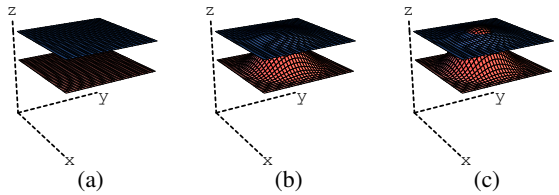
They define the blendeforming deformation as a bundle of non-intersecting streamlines. The streamlines are parallel, and described by a pair of functions: $b_{\mathrm{x,y}} : \mathbb{R}^2 \mapsto [-d_{\max}, d_{\max}]$ and $b_{\mathrm{z}} : [0,1] \mapsto [0,1]$. Function $b_{\mathrm{x,y}}$ controls the amount of deformation for each individual z-streamlines, and the choice of function b_{z} affects the maximum compression of space along the streamlines. The deformation of point $\mathrm{p} = (x,y,z)^\top$ is

$$\mathrm{p}_{def} = (x, y, z_{def})^\top \qquad (8)$$
$$\text{where } z_{def} = z + b_{\mathrm{x,y}}(x,y)\; b_{\mathrm{z}}(z)$$

It is the definition of b_{z} together with a corresponding threshold $d_{\max}$ that prevents foldovers, as shown in Figure 13. The following function is a possible choice for $b_{\mathrm{z}}(z)$, used in the example:

$$b_{\mathrm{z}}(z) = \begin{cases} 16z^2(1-z)^2 & \text{if } z \in [0,1] \\ 0 & \text{otherwise} \end{cases} \qquad (9)$$
$$\text{with } d_{\max} = \frac{3\sqrt{3}}{16} \simeq 0.324$$

Functions permitting larger values for $d_{\max}$ can be found in the original paper. Since $b_{\mathrm{x,y}}$ is independent of z, any function with values in $[-d_{\max}, d_{\max}]$ can be used for it, regardless of the slope. Because the amplitude of the effect of a blendeforming function is bounded by the $d_{\max}$ threshold, it is obvious that modeling an entire shape uniquely with blendeforming functions can be rather tedious. In the original paper, the authors also propose bending blendeforming functions, defined in cylindrical coordinates.



(a) (b) (c)

Figure 13: *(a) Initial scene: two parallel planes. (b) Blendeforming, with $b_{\mathrm{x,y}}(x,y) = (x^2 - x + y^2 - y - 1/2)^2$. The value of $d_{\max}$ guarantees that the two planes will never intersect. (c) With $d_{\max} < d$, foldover occurs: the lower plane intersects the higher plane.*

3.2. Lattice-based space deformations

The limitation of axial-based or surface-based space deformation is the arrangement of the controls along a curve or on a surface. Note that this statement is untrue only for wires, which permits the blending of the controls [SF98]. Lattice-based space deformations are techniques that allow control points to be connected along the three dimensions of space. There are two ways of understanding lattice-based deformation, related to the manner in which the artist expresses the deformation. Let us denote the space deformation function by f.

In the first interpretation of lattice-based deformations, the artist provides pairs of points: a source point and a destination point, $(\mathrm{p}_i, \mathrm{q}_i)$. The deformation f will interpolate or approximate the pairs in this way $f(\mathrm{p}_i) = f_{\mathrm{p}}(\mathrm{p}_i) \approx \mathrm{q}_i$. The function f_{p} is a position field. A position field does not have any physical equivalent to which the artist or scientist can relate, and requires a certain amount of imagination to be visualized.

In the second interpretation of lattice-based deformations, the artist provides a source point and a displacement of that

point, $(\mathrm{p}_i, \vec{\mathrm{v}}_i)$. The deformation f will interpolate or approximate the pairs in this way $f(\mathrm{p}_i) = \mathrm{p}_i + f_{\vec{\mathrm{v}}}(\mathrm{p}_i) \approx \mathrm{p}_i + \vec{\mathrm{v}}_i$. The function $f_{\vec{\mathrm{v}}}$ is a vector field. There is a convenient physical analogy to a vector field. Vector fields are used in fluid mechanics to describe the motion of fluids or to describe fields in electromagnetics [Rut90, Gri]. This analogy is of great help for explaining and creating new space deformations.

While the effect of using either a position field or a vector field is equivalent, the vector field also gives more insight in the process of deforming space: in lattice-based space deformations, the path that brings the source point onto the desired target point is a straight translation using a vector. In this section on lattice-based space deformation, we will therefore consider the construction of a vector field rather than a position field whenever possible.

3.2.1. Free-form deformation of solid geometric models

The effect of Free-Form Deformation (FFD) on a shape is to embed this shape in a piece of flexible plastic. The shape deforms along with the plastic that surrounds it [SP86].

The idea behind FFD is to interpolate or approximate vectors defined in a 3d regular lattice. The vectors are then used to translate space. In their original paper, T. Sederberg and S. Parry propose to use the trivariate Bernstein polynomial as a smoothing filter. Let us denote by $\vec{\mathrm{v}}_{ijk}$ the $(l+1) \times (m+1) \times (n+1)$ *control vectors* defined by the artist. The smoothed vector field is a mapping $\mathrm{p} \in [0,1]^3 \mapsto \mathbb{R}^3$.

$$\vec{\mathrm{v}}(\mathrm{p}) = \Sigma_{i=0}^{l}\binom{i}{l}(1-x)^{l-i}x^i\left(\Sigma_{j=0}^{m}\binom{j}{m}(1-y)^{m-j}y^j \left(\Sigma_{k=0}^{n}\binom{k}{n}(1-z)^{n-k}z^k\right)\right)\vec{\mathrm{v}}_{ijk} \tag{10}$$

Then the deformation of a point is a translation of that point

$$\mathrm{p}_{def} = \mathrm{p} + \vec{\mathrm{v}}(\mathrm{p}) \tag{11}$$

In order for the deformation to be continuous across the faces of the FFD cube, the boundary vectors should be set to zero. A drawback of using the Bernstein polynomial is that a control vector $\vec{\mathrm{v}}_{ijk}$ has a non-local effect on the deformation. Hence updating the modification of a control vector requires updating the entire portions of shape within the lattice. For this reason, J. Griessmair and W. Purgathofer propose to use B-Splines [GP89].

In commercial software, the popular way to let the artist specify the control vectors is to let him move the control points of the lattice, as shown in Figure 14(c). A drawback often cited about this interface is the visual self occlusion of the control points. This problem increases with the increase in resolution of the lattice. Another drawback is the manipulation of control points, which requires high skills in spacial apprehension from the artist. Clearly, practical FFD manipulation through control-points can only be done with reasonably small lattices.

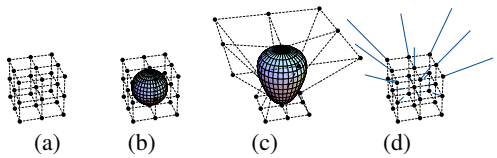


Figure 14: *FFD. (a) Lattice of size 3^3. (b) Initial shape. (c) The popular interaction with an FFD lattice consists of displacing the control points. (d) The discrete vectors.*

3.2.2. Extended free-form deformation (EFFD)

Due to the practical limit of the size of the FFD-lattice, the major restriction of an FFD is strongly related to the arrangement of control-points in parallelepipeds. The parallelepipeds are also called *cells*. To provide the artist with more control, S. Coquillart proposes a technique with non-parallelepipedic and arbitrarily connected *cells*. The technique is called Extended Free-Form Deformation (EFFD) [Coq90].

To model with EFFD, the artist first builds a lattice by placing the extended cells anywhere in space, and then manipulates the cells to deform the shape. An extended cell is a small FFD of size 4^4. The transformation from the cell's *local coordinates* $\mathrm{s} = (u,v,w)^\top$ to world coordinates is:

$$\mathrm{p}(\mathrm{s}) = \Sigma_{i=0}^{3}\binom{i}{3}(1-u)^{3-i}u^i\left(\Sigma_{j=0}^{3}\binom{j}{3}(1-v)^{3-j}v^j \left(\Sigma_{k=0}^{3}\binom{k}{3}(1-w)^{3-k}w^k\right)\right)\mathrm{p}_{ijk} \tag{12}$$

The eight corners $\mathrm{p}_{ijk\in\{0,3\}^3}$ of a cell are freely defined by the artist. The position of the remaining $4^4 - 8$ are constrained by the connection between cells, so that continuity is maintained across boundaries. This is done when the artist connects the cells. Because the lattice is initially deformed, finding a point's coordinates s in a cell is not straightforward. The local coordinates of a point p in a cell are found by solving Equation (12) in s using a numerical iteration. This can be unstable in some cases, although the authors report they did not encounter such cases in practice. Once s is found, the translation to apply to p is found by substituting in Equation (12) the control points p_{ijk} with the control vectors $\vec{\mathrm{v}}_{ijk}$. Note that specifying the control points, the cells and the control vectors is rather tedious, and results shown in the paper consist essentially of imprints.

3.2.3. Free-form deformations with lattices of arbitrary topology (SFFD)

R.A. MacCracken and K.I. Joy have established a method that allows the user to define lattices of arbitrary shape and topology [MJ96]. The method is more stable than EFFD since it does not rely on a numerical iteration technique.

Their method is based on subdivision lattices. We will refer to it as SFFD, for subdivision FFD. The user defines a

control lattice, L: a set of vertices, edges, faces and cells. A set of refinement rules are repeatedly applied to L, creating a sequence of increasingly finer lattices $\{L_1, L_2, \dots L_l\}$. The union of cells define the deformable space. After the first subdivision, all cells can be classified into cells of different type: type-n cells, $n \geq 3$. See [MJ96] for the rules.

Although there is no available trivariate parameterization of the subdivision lattice, the correspondence between world coordinates and lattice coordinates is possible thanks to the subdivision procedure. The location of a vertex embedded in the deformable space is found by identifying which cell contains it. Then, for a type-3 cell, trilinear parameterization is used. For a type-n cell, the cell is partitioned in $4n$ tetrahedra, in which the vertex takes a trilinear parameterization. Each point is tagged with its position in its cell.

Once a point's location is found in the lattice, finding the point's new location is straightforward. When the artist displaces the control points, the point's new coordinates are traced through the subdivision of the deformed lattice.

3.2.4. Direct manipulation of free-form deformations (DMFFD)

The manipulation of individual control points makes FFD and EFFD tedious methods to use. Two groups of researchers, P. Borrel and D. Bechmann, and W.M. Hsu et al. propose a similar way of doing direct manipulation of FFD control points (DMFFD) [BB91, HHK92]. The artist specifies translations $\vec{\mathrm{v}}_i$ at points p_i in the form $(\mathrm{p}_i, \vec{\mathrm{v}}_i)$. The DMFFD algorithm finds control vectors that satisfy, if possible, the artist's desire. Let us define a single input vector $\vec{\mathrm{v}}$ at point p. The FFD Equation (10) must satisfy

$$\vec{\mathrm{v}} = \mathrm{B}(\mathrm{p})(\vec{\mathrm{v}}_{ijk}) \tag{13}$$

Let $\nu = (3(l+1)(m+1)(n+1))$. The matrix B is the $3 \times \nu$ matrix of the Bernstein coefficients, which are functions of point p. Note that their method is independent of the chosen filter: instead of the Bernstein polynomials, W.M. Hsu et al. use B-Splines and remark that Bernstein polynomials can be used. P. Borrel and D. Bechmann on the other hand found that using simple polynomials works just as well as B-Splines. The size of the vector of control vectors $(\vec{\mathrm{v}}_{ijk})$ is $3(l+1)(m+1)(n+1)$. When the artist specifies μ pairs $(\mathrm{p}_i, \vec{\mathrm{v}}_i)$, the FFD Equation (10) must satisfy a larger set of equations:

$$\begin{pmatrix} \vec{\mathrm{v}}_1 \\ \vdots \\ \vec{\mathrm{v}}_\mu \end{pmatrix} = \mathbf{B} \cdot \begin{pmatrix} \vec{\mathrm{v}}_{ijk} \\ \vdots \\ \vec{\mathrm{v}}_{ijk} \end{pmatrix} \quad \text{where } \mathbf{B} = \begin{pmatrix} \mathrm{B}(\mathrm{p}_1) \\ \vdots \\ \mathrm{B}(\mathrm{p}_\mu) \end{pmatrix} \tag{14}$$

This set of equations can either be overdetermined or under determined. In either case, the matrix $\mathbf{B}$ cannot be inverted in order to find the $\vec{\mathrm{v}}_{ijk}$. The authors use the Moore-Penrose pseudo-inverse, $\mathbf{B}^+$. If the inverse of $\mathbf{B}^\top \cdot \mathbf{B}$ exists, then

$$\mathbf{B}^+ = (\mathbf{B}^\top \cdot \mathbf{B})^{-1} \cdot \mathbf{B}^\top \tag{15}$$

It is however preferable to compute the Moore-Penrose pseudo-inverse using singular value decomposition (SVD). The $\mu \times \nu$ matrix $\mathbf{B}$ can be written

$$\mathbf{B} = U \cdot D \cdot V^\top \tag{16}$$

where U is an $\mu \times \mu$ orthogonal matrix, V is an $\nu \times \nu$ orthogonal matrix and D is an $\mu \times \nu$ diagonal matrix with real, non-negative elements in descending order.

$$\mathbf{B}^+ = V \cdot D^{-1} \cdot U^\top \tag{17}$$

Here, the diagonal terms of D^{-1} are simply the inverse of the diagonal terms of D.

The size of the basis, or, equivalently the number of control points, has a direct effect on the locality of the deformation around the selected point. In their approach, P. Borrel and D. Bechmann pursue the reasoning even further, and define a technique suitable for n-dimensional objects [BB91]. In the context of shape animation, i.e. in $\mathbb{R}^4$ with time as the fourth dimension, the Bernstein, B-Splines or simple polynomials are inappropriate. They propose to use a basis that does not change the initial time, t_0, and final time, t_f, of an object:

$$\mathrm{B}_t(\mathrm{p}, t) = \Big((t-t_0)(t-t_f)\ ,\ (t-t_0)(t-t_f)t\ ,\ (t-t_0)(t-t_f)t^2\ ,\ \dots \Big)^\top$$

3.2.5. Simple constrained deformations for geometric modeling and interactive design (scodef)

In simple constrained deformations (scodef), P. Borrel and A. Rappoport propose to use DMFFD with radial basis functions (RBF) [BR94]. The artist defines constraint triplets $(\mathrm{p}_i, \vec{\mathrm{v}}_i, r_i)$: a point, a vector that defines the desired image of the point, and a radius of influence. Let $\phi_i(\mathrm{p})$ denote the scalar function $\phi(\frac{\|\mathrm{p}-\mathrm{p}_i\|}{r_i})$ for short. The motivation of using RBF is to keep the deformation local, in the union of spheres of radius r_i around the points p_i. A naive vector field would be:

$$\vec{\mathrm{v}}(\mathrm{p}) = \sum_{i=1}^{n} \vec{\mathrm{v}}_i \phi_i(\mathrm{p}) \tag{18}$$

Unless the points p_i are far apart enough, Equation (18) will not necessarily satisfy the artist's input $\vec{\mathrm{v}}(\mathrm{p}_i) = \vec{\mathrm{v}}_i$ if the functions ϕ_i overlap. However, this can be made possible by substituting the vectors $\vec{\mathrm{v}}_i$ with suitable vectors $\vec{\mathrm{w}}_i$.

$$\vec{\mathrm{v}}(\mathrm{p}) = \sum_{i=1}^{n} \vec{\mathrm{w}}_i \phi_i(p) \tag{19}$$

These vectors $\vec{\mathrm{w}}_i$ can be found by solving a set of $3n$ equations:

$$\vec{\mathrm{v}}_i = (\vec{\mathrm{w}}_1 \dots \vec{\mathrm{w}}_n) \cdot \begin{pmatrix} \phi_1(\mathrm{p}_i) \\ \vdots \\ \phi_n(\mathrm{p}_i) \end{pmatrix} \text{ where } i \in [1, n] \tag{20}$$

Let us take the transpose, and arrange the n equations in rows. The following equation is the equivalent of Equation (14), but with radial basis functions:

$$\begin{pmatrix} \vec{v}_1^{\top} \\ \vdots \\ \vec{v}_n^{\top} \end{pmatrix} = \begin{pmatrix} \phi_1(p_1) & \dots & \phi_n(p_1) \\ \vdots & & \\ \phi_1(p_n) & \dots & \phi_n(p_n) \end{pmatrix} \cdot \begin{pmatrix} \vec{w}_1^{\top} \\ \vdots \\ \vec{w}_n^{\top} \end{pmatrix} \quad (21)$$

where $i \in [1,n]$

Let $\mathbf{\Phi}$ be the $n \times n$ square matrix of Equation (21). This matrix takes the role of $\mathbf{B}$ in Equation (14). Since $\mathbf{\Phi}$ can be singular, the authors also use its pseudo-inverse $\mathbf{\Phi}^{+}$ to find the vectors $\vec{w}_i$.

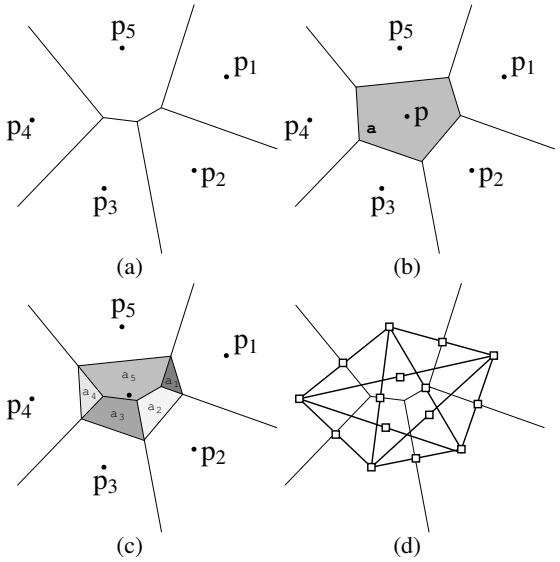


Figure 15: *2D illustration of the Sibson coordinates (a) Voronoï cells of the control points. (b) Voronoï diagram is updated after the insertion of point* p. *(c) The areas stolen by the point* p *from its natural neighbors give the Sibson coordinates a_i/a. (d) Local control net, with Bézier abscissa.*

3.2.6. Dirichlet free-form deformation (DFFD)

With DFFD, L. Moccozet and N. Magnenat-Thalmann propose a technique that builds the cells of a lattice automatically [MMT97], relieving the artist from a tedious task. The lattice cells are the cells of a Voronoï diagram of the control points, shown in Figure 15. The location of a point within a cell is neatly captured by the Sibson coordinates. The naive deformation of a point p is given by interpolating vectors defined at the control points with the Sibson coordinate.

$$p \mathrel{+}= \sum_{i=1}^{n} \frac{a_i}{a} \vec{v}_i \quad (22)$$

Where a_i is the volume of cell i stolen by p, and a is the volume of the cell of p. This interpolation is only C^0. They use a method developed by G. Farin [Far90] to define a continuous parameterization on top of the Sibson coordinates. The interpolation is made of four steps:

- build the local control net
- build *Bézier abscissa*
- define *Bézier ordinates* such that the interpolant is C^1
- evaluate the multivariate Bernstein polynomial using Sibson coordinates.

3.2.7. Preventing self-intersection under free-form deformation

In FFD, EFFD and DMFFD, if the magnitude of a control-vector is too high, the deformation may produce a self-intersection of the shape's surface (see a self-intersection in Figure 13). Once the shape's surface self-intersects, there is no space deformation that can remove the self-intersection. The appearance of this surface incoherency is the result of a space foldover: the deformation function is a surjection of $\mathbb{R}^3$ onto $\mathbb{R}^3$, not a bijection. J. Gain and N. Dodgson present foldover detection tests for DMFFD deformations that are based on uniform B-Splines [GD01]. They argue that a necessary and sufficient test is too time consuming, and present an alternative sufficient test. Let us define q_{ijk}, the deformed control points of the lattice. If the determinants of all the following 3×3 matrices are all positive, there is no foldover.

$$\phi_{ijk} = s \ \det\left(q_{i\pm1\,jk} - q_{ijk}, q_{ij\pm1k} - q_{ijk}, q_{ijk\pm1} - q_{ijk}\right)$$
where the sign s is 1 if $(i \pm 1, j \pm 1, k \pm 1)$ are clockwise, else -1. (23)

The idea underlying the test is that the determinant of three column vectors is the volume of the parallelepiped defined by these vectors. A negative volume detects a possible singularity in the deformation. A technique for efficiently testing several determinants at once can be found in the original paper.

This test can then be used to repair the DMFFD. Let us define $(p_i, \vec{v}_i)$, the pairs of points and vectors defining the DMFFD. If a foldover is detected, the DMFFD is recursively

split into two parts: $(p_i, \vec{v}_i/2)$ and $(p_i + \vec{v}_i/2, \vec{v}_i/2)$. The procedure eventually converges, and the series of DMFFDs obtained are foldover-free and can be applied safely to the shape.

3.3. Tool-based space deformations

Lattice-based techniques are capable of building a wide range of vector fields. But when dealing with a problem in animation, modeling or visualization, a technique tailored for that specific problem will be more suitable. This section is about techniques that focus on a particular unresolved problem of space deformation, and solve it in an original way.

3.3.1. Interactive space deformation with hardware assisted rendering

Y. Kurzion and R. Yagel present *ray deflectors* [KY97]. The authors are interested in rendering the shape by deforming the rays, as opposed to directly deforming the shape. To deform the rays, one needs the inverse of the deformation that the artist intends to apply to the shape. Rather than defining a deformation and then trying to find its inverse, the authors directly define deformations by their inverse. Their tool can translate, rotate and scale space contained in a sphere, locally and smoothly. Moreover they define a discontinuous deformation that allows the artist to cut space, and change a shape's topology. A tool is defined within a ball of radius r around a center c. Let ρ be the distance from the center of the deflector c and a point p.

$$\rho = \|p - c\| \tag{24}$$

3.3.1.1. Translate deflector: To define a translate deflector, the artist has to provide a translation vector, $\vec{t}$. The effect of the translate deflector will be to transform the center point, c, into $c + \vec{t}$.

$$f_T(p) = \begin{cases} p - \vec{t}(1 - \frac{\rho^2}{r^2})^2 & \text{if } \rho < r \\ p & \text{otherwise} \end{cases} \tag{25}$$

where $\theta \in \mathbb{R}$

3.3.1.2. Rotate deflector: To define a rotate deflector, the artist has to provide an angle of rotation, θ, and a vector, $\vec{n}$, about which the rotation will be done. The reader can find the expression of a rotation matrix, $R_{\theta',\vec{n},c}$, in Appendix **??**. Let us call θ' an angle of rotation that varies in space:

$$\theta' = -\theta(1 - \frac{\rho^2}{r^2})^4$$

$$f_R(p) = \begin{cases} R_{\theta',\vec{n},c} \cdot p & \text{if } \rho < r \\ p & \text{otherwise} \end{cases} \tag{26}$$

where $\|\vec{t}\| \in [0, \frac{3\sqrt{3}r}{8}]$

3.3.1.3. Scale deflector: To define a scale deflector, the artist has to provide a scale factor s. The scale deflector acts like a magnifying glass.

$$f_S(p) = \begin{cases} p - (p - c)(1 - \frac{\rho^2}{r^2})^4 s & \text{if } \rho < r \\ p & \text{otherwise} \end{cases} \tag{27}$$

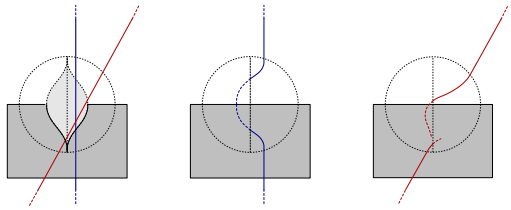
where $s \in [-1, 1]$

3.3.1.4. Discontinuous deflector: To define a discontinuous deflector, the artist has to provide a translation vector, $\vec{t}$. The deflector is split into two halves, on each side of a plane going through c and perpendicular to $\vec{t}$. In the half pointed at by $\vec{t}$, the discontinuous deflector will transform c, into $c + \vec{t}$, while in the other half, the discontinuous deflector will transform c, into $c - \vec{t}$. The effect will be to cut space. The deformation applied to the rays is:

$$f_D(p) = \begin{cases} p - \vec{t}(1 - \frac{\rho^2}{r^2})^2 & \text{if } \rho < r \text{ and } 0 < (p - c) \cdot \vec{t} \\ p + \vec{t}(1 - \frac{\rho^2}{r^2})^2 & \text{if } \rho < r \text{ and } (p - c) \cdot \vec{t} < 0 \\ p & \text{otherwise} \end{cases} \tag{28}$$

where $\theta \in \mathbb{R}$

Since this deformation is discontinuous on the disk separating the two halves of the deformation, a ray crossing that disk will be cut in two, as we show in Figure 16(c). Thus a shape intersection algorithm will have to march along the ray from the two sides of the ray, until each curve crosses the separating disk. This deformation assumes that the shape's representation has an inside and outside test. Note that other authors have extended FFD for dealing with discontinuities [SE04].



(a) (b) (c)

Figure 16: *(a) Discontinuous deflector as observed by the artist. Two arbitrary rays are shown. (b) Simple case, where the ray of light crosses only one hemisphere. (c) When the ray of light changes hemisphere, the curve of light is subject to a discontinuity.*

3.3.2. Geometric deformation by merging a 3D object with a simple shape

P. Decaudin proposes a technique that allows the artist to model a shape by iteratively adding the volume of simple 3D shapes [Dec96]. His method is a metaphor of clay sculpture by addition of lumps of definite size and shape. His deformation function is a closed-form, as opposed to a numerical method that would explicitly control the volume [HML99].

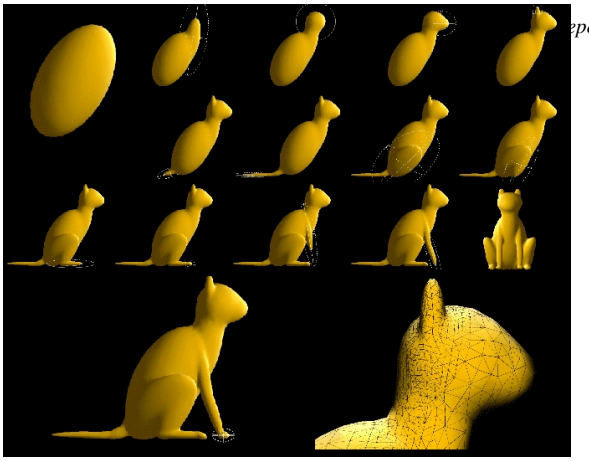


Figure 17: *Steps of the modeling of a cat (courtesy of [Dec96]).*

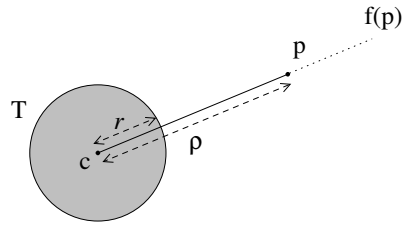


Figure 18: *The insertion of a tool at center* c *affects the position of point* p. *See the deformation in Equations (29).*

Loosely speaking, this technique inflates space by blowing up a tool in space through a hole. This will compress space around the point in a way that preserves the outside volume. Hence if the tool is inserted inside the shape, the tool's volume will be added to the shape's volume. On the other hand, if the tool is inserted outside the shape, the shape will be deformed but its volume will remain constant. This is illustrated for the 2D case in Figure 19. A restriction on the tool is to be star-convex with respect to its center c . The deformation function is[‡] (see Figure 18):

$$f_{3D}(\mathrm{p}) = \mathrm{c} + \sqrt[3]{\rho(\mathrm{p})^3 + r(\mathrm{p})^3}\,\vec{\mathrm{n}} \qquad (29)$$

- $\rho(\mathrm{p})$ is the magnitude of the vector $\vec{\mathrm{u}} = \mathrm{p} - \mathrm{c}$.
- $r(\mathrm{p})$ is the distance between c and the intersection of the tool with the half-line $(\mathrm{c}, \vec{\mathrm{u}})$.
- $\vec{\mathrm{n}} = \vec{\mathrm{u}}/\|\vec{\mathrm{u}}\|$ is the unit vector pointing from c to p.

If the tool was not a star-convex in c, then $r(\mathrm{p})$ would be ambiguous. The deformation is foldover-free. It is continuous everywhere except at the center c. The effect of the deformation converges quickly to the identity with the increasing distance from c. The deformation can be considered local, and is smooth everywhere except at c. An example in $3D$ is shown in Figure 17. A feature of this space deformation which is rare, is that it has an exact yet simple inverse in the space outside the tool:

$$f_{3D}^{-1}(\mathrm{p}) = \mathrm{c} + \sqrt[3]{\rho(\mathrm{p})^3 - r(\mathrm{p})^3}\,\vec{\mathrm{n}} \qquad (30)$$

[‡] The 2D case is obtained by replacing 3 with 2.

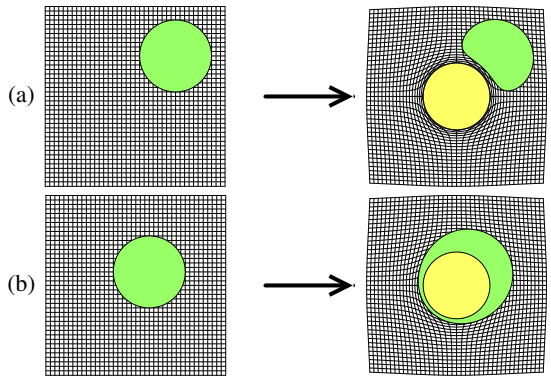


Figure 19: *(a) Deformation of a shape (green) by blowing up a tool (yellow) outside the shape. The shape's area is preserved. (b) Deformation of a shape by blowing up a tool inside the shape. The shape's area is increased by that of the tool.*

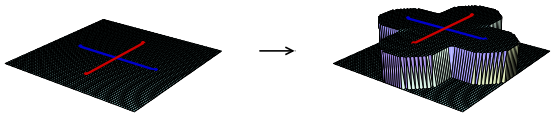
3.3.3. Implicit free-form deformations (IFFD)

B. Crespin introduces Implicit Free-Form Deformations (IFFD) [Cre99]. Note that though it is called implicit, the deformation is explicit. IFFD is rather a technique inspired by implicit surfaces, a vast branch of computer graphics whose presentation is beyond the scope of this manuscript [BBB*97]. The field $\phi \in [0,1]$ generated by a skeleton modulates a transformation, M, of points. The deformation of point p with a single function is:

$$f(\mathrm{p}) = \mathrm{p} + \phi(\mathrm{p})(M \cdot \mathrm{p} - \mathrm{p}) \qquad (31)$$

He proposes two ways to combine many deformations simultaneously. Let use denote p_i the transformation of p with deformation f_i. The first blending is shown in Figure 20. For M, we have used a translation matrix.

$$p_{def} = p + \frac{\sum_{i=1}^{n}(p_i - p)\phi_i(p)}{\sum_{i=1}^{n}\phi_i(p)}$$

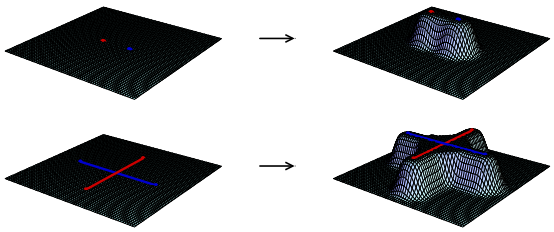


Reference segments Translated segments

Figure 20: *Blending weights based on summed displacement magnitudes. The deformation is only defined where the amounts ϕ are not zero, and is discontinuous at the interface $\sum_i \phi_i = 0$. This blending is useful when the deformed shape is entirely contained within the field.*

The second blending attempts to solve the continuity issue, but requires the definition of supplementary profile functions, γ_i. The purpose of the index i is to assign individual profiles to skeletons.

$$p_{def} = p + \frac{\sum_{i=1}^{n}(p_i - p)\phi_i(p)\gamma_i(p)}{\sum_{i=1}^{n}\phi_i(p)}$$



Reference segments Translated segments

Figure 21: *Blending weights based on displacement magnitudes and profile functions. For control points, the technique works well. For segments, there is a discontinuity near their intersection.*

In order to produce Figure 21, the following γ_i function was used:

$$\gamma_i(p) = \begin{cases} 1-(1-\sigma^2)^2 & \text{if } \sigma \in [0,1], \\ & \text{where } \sigma = \sum_{i=1}^{n}\phi_i(p) \\ 1 & \text{otherwise} \end{cases} \quad (32)$$

3.3.4. Twister

I. Llamas et al propose a method called twister in which a twist transformation of points is weighted with a scalar function [LKG*03], i.e. in a similar way to IFFD but with a transformation restrained to a twist. With this restriction, they propose to weight single twists along the trajectory of transformation rather than weighting the displacement. They define a twist by transforming an orthonormal coordinate system $(o, \vec{u}, \vec{v}, \vec{w})$ into $(o', \vec{u}', \vec{v}', \vec{w}')$. The axis of the twist is defined by a direction $\vec{d}$ and point a on the axis, while the angle of rotation around the axis is α and the translation factor along the axis is d:

$$\begin{aligned} \vec{d} &= \frac{\vec{g}}{\|\vec{g}\|} \\ \text{where } \vec{g} &= (\vec{u}'-\vec{u})\times(\vec{v}'-\vec{v}) + \\ &\quad (\vec{v}'-\vec{v})\times(\vec{w}'-\vec{w}) + \\ &\quad (\vec{w}'-\vec{w})\times(\vec{u}'-\vec{u}) \\ \alpha &= 2\arcsin\left(\frac{\|\vec{u}'-\vec{u}\|}{2\|\vec{d}\times\vec{u}\|}\right) \\ d &= \vec{d}\cdot(o'-o) \\ a &= \frac{o+o'-d\vec{d}}{2} + \frac{\vec{d}\times(o-o')}{2\tan(\alpha/2)} \end{aligned} \quad (33)$$

Their procedure for deforming a point p with a twist parameterized in t is:

1. Bring p into local coordinates: translate by $-\vec{a}$ and then rotate by a rotation that maps $\vec{d}$ onto $\vec{z}$.
2. Apply the twist in local coordinates: translate by $t\ d$ along $\vec{z}$ and rotate by $t\ \alpha$ around $\vec{z}$
3. Finally bring p back into world coordinates: rotate by a rotation that maps $\vec{z}$ onto $\vec{d}$ and translate by $\vec{a}$

To weight the twist, they propose to use a piecewise scalar function:

$$t(p) = \cos^2(\pi\|p-o\|/2r) \quad (34)$$

For operations that require simultaneous twists, they propose simply to add the displacement of the weighted twist. Details for defining a two-point constraint can be found in the paper.

3.3.5. Scalar-field guided adaptive shape deformation and animation (SFD)

J. Hua and H. Qin create a technique called SFD [HQ04]. They define a deformation by attaching space to the level-sets of an animated scalar field. The artist is offered three different techniques for animating a scalar field. Since there are many ways of attaching a point to a level-set of a scalar field, the authors choose the way that keeps the shape as rigid as possible.

They define $\phi(t, p(t))$, the scalar field which is animated in time, t. Since a moving point, $p(t)$, is attached to a level-set of the scalar field, the value of ϕ at p is constant in time:

$$\frac{d\phi}{dt} = 0 \quad (35)$$

The square of Equation (35) gives a constraint:

$$\left(\frac{d\phi}{dt}\right)^2 = 0 \quad (36)$$

There are several ways of attaching a point to a level set while the scalar field is moving. The simplest way would be to make a point follow the shortest path, found when the magnitude of the point's speed, $\|\vec{v}(t)\|$, is minimized. Another possibility, chosen by the authors, is to minimize the variation of velocity, so that the deformation is as rigid as possible. Instead of using the divergence of the speed to measure rigidity, they use an estimate by averaging the variation

of speed between that point's speed, $\vec{v}$, and its neighbors' speed, $\vec{v}_k$:

$$(\nabla \cdot \vec{v})^2 \approx \frac{1}{k} \sum_k (\vec{v} - \vec{v}_k)^2 \tag{37}$$

Since this is a constrained optimization problem [Wei04], there exists a Lagrange multiplier λ such that:

$$\frac{d}{d\vec{v}} \left(\frac{d}{dt} \phi(t, p(t)) \right)^2 + \lambda \frac{d}{d\vec{v}} (\nabla \cdot \vec{v})^2 = \vec{0} \tag{38}$$

According to the authors, λ is an experimental constant, used to balance the flow constraint and speed variation constraint. Its value ranges between 0.05 and 0.25. We rearrange this equation and expand the derivative of ϕ with the chain rule:

$$\frac{d}{d\vec{v}} \left((\nabla\phi \cdot \vec{v} + \frac{\partial \phi}{\partial t})^2 + \lambda (\nabla \cdot \vec{v})^2 \right) = \vec{0} \tag{39}$$

Let us define $\hat{\vec{v}}$, the average of the velocity of all the adjacent neighbors connected with edges to point p. If we substitute $(\nabla \cdot \vec{v})^2$ for its approximate given by Equation (37), and then apply the derivative with respect to $\vec{v}$, we obtain:

$$(\nabla\phi \cdot \vec{v} + \frac{\partial \phi}{\partial t}) \nabla\phi + \lambda (\vec{v} - \hat{\vec{v}}) = \vec{0} \tag{40}$$

By solving the system of Equation (40), the updated position is:

$$\vec{v} = \hat{\vec{v}} - \frac{\hat{\vec{v}} \cdot \nabla\phi + \frac{\partial \phi}{\partial t}}{\lambda + (\nabla\phi)^2} \nabla\phi \tag{41}$$

The algorithm deforms a set of vertices in n sub-steps. If n is set to one, the deformation takes one step:

for i = 1 to n **do**
  for all p_k in the list of vertices to update **do**
    Update the scalar field $\phi(t + \Delta t, p_k)$.
    Deduce $\frac{\partial \phi}{\partial t} = \frac{\phi(t+\Delta t, p_k) - \phi(t, p_k)}{\Delta t}$
    Calculate $\nabla\phi$, possibly with finite differences.
    Compute $\hat{\vec{v}}$ according to neighbors' velocities.
    Deduce $\vec{v}$ according to Equation (41).
    Update vertex positions with $p_k(t + \Delta t) = p_k(t) + \vec{v} \frac{\Delta t}{n}$
    Improve surface representation using a mesh refinement and simplification strategy.
    if $\phi(t + \Delta t, p_k(t + \Delta t)) \approx \phi(t, p_k(t))$ **then**
      remove p_k from the list of vertices to update.
    end if
  end for
end for

In the first step, since all the speeds are zero, we suggest that they could be initialized with:

$$\vec{v} = -\frac{\frac{\partial \phi}{\partial t}}{\lambda + (\nabla\phi)^2} \nabla\phi \tag{42}$$

Firstly, this technique is not a very versatile space deformation technique since it requires an explicit surface in order to compute the divergence of the speed. Secondly, the advantage of a large set of possible SFD shape operations (as large as the set of possible animated scalar fields) is at the cost of making the artist's task rather tedious: specifying the animated field does not permit quick and repeated operations on the shape. Also, results show the editing of imported shapes rather than shapes entirely modeled from scratch.

3.4. Limitations

The large number of space deformation techniques can lead quickly to the naive conclusion that in any shape modeling by deformation scenario, the limitation of a technique may be simply circumvented by using another technique. This reasoning presents several flaws. Firstly, from the point of view of a programmer, the amount of effort required to implement a space deformation Swiss-army knife for shape modeling would be considerable. Secondly, from the point of view of an artist, choosing quickly the most appropriate space deformation would require a vast amount of knowledge of the underlying mathematics of many techniques, which is a skill that should not be required. Thirdly, from a researcher's point of view, all space deformation techniques are not necessarily designed for the specific purpose of shape modeling, and there are surely efficient ways of dealing with specific problems. We will discuss this last point in the remainder of this section, i.e. we will overview the suitability of individual space deformation techniques for the purpose of interactive shape modeling.

Firstly, the subset of space deformations, whose effect on a shape is not local, makes these techniques unsuitable for the task of modeling shapes, since an artist's operation on a visible portion of the shape will have effects on portions that are further away [Bar84, Bla94, CR94, LCJ94].

Secondly, a large number of space deformation techniques requires the artist to specify a rather large number of control parameters [SP86, Coq90, MJ96, MMT97, HML99, HQ04]. We believe that increasing the number of parameters does not increase the amount of control by an artist, but rather it makes the task longer and more tedious. Many techniques illustrate their capabilities on imported models, that were either digitized or pre-modeled with conventional modeling techniques with a few exceptions [Dec96, HHK92, LKG*03]. We believe that the absence of a model entirely developed in one piece with a single technique is some evidence that the technique is tedious to use for the dedicated purpose of modeling shapes.

Finally, many space deformation techniques do not prevent a surface from self-intersecting after deformation, aside from a few exceptions [Dec96, MW01, GD01]. A self-intersecting surface is a rather annoying situation in modeling with deformation, since it is impossible for a space deformation to remove a previously introduced self-intersection. Thus we believe that space deformation operations for shape modeling should satisfy all the following criteria:

- Its effective span should be controllable.
- Its input parameters should be reduced to their strict minimum: a gesture.
- It should be predictable, in accordance with a metaphor.
- It should be foldover-free, and even revertible.
- It should be sufficiently fast for existing computing devices.

To our knowledge, the literature does not contain techniques satisfying all the above criteria. Rather than defining a set of unrelated techniques, we will specify a framework in which we will define deformation operations that satisfy the above. We will illustrate the modeling capabilities of our framework with techniques and examples.

4. Modeling with gesture

In the following, the input that defines transformations is a gesture, obtained with a mouse or hand tracking device. For the sake of simplicity, we will denote by f the function $f_{t_i \mapsto t_{i+1}}$.

4.1. Naive deformation

A simple space deformation can be defined with a transformation (translation, rotation, scale, etc.) whose effect is spatially weighted. Thus two entities suffice:

- a transformation: 4×4 matrix M, defined by a gesture (mouse move, tracked hand)
- the amount of transformation at $\mathrm{p} \in \mathbb{R}^3$: scalar field $\varphi(\mathrm{p}) \in [0,1]$, defined for instance using the distance to a shape. We call *tool* the field.

The most straightforward way of weighting M with φ is to weight the displacement induced by M at p:

$$\dot{f}(\mathrm{p}) = \mathrm{p} + \varphi(\mathrm{p})\ (M \cdot \mathrm{p} - \mathrm{p}) \tag{43}$$

This weighting of M produces however poor results in several cases, including when M is a rotation matrix. To compute fractions of a transformation, we rather use the formalism of M. Alexa [Ale02], i.e. the multiplication operator $\odot$ which behaves essentially like $\cdot$ for scalars (see Appendix A). Note that although we use Alexa's operator, we do not necessarily evaluate it numerically as proposed in his paper, since some cases reduce to more efficient and elegant closed-form formulas, as we will show. Thus the transformation M can be weighted with φ as follows:

$$\ddot{f}(\mathrm{p}) = (\varphi(\mathrm{p}) \odot M) \cdot \mathrm{p} \tag{44}$$

The deformation $\ddot{f}$ is however naive, since it can create a foldover. For example, if M is a translation of large magnitude, it can map points within the support of φ onto points outside from the support of φ, thus folding space onto itself as shown in Figure 22(left).

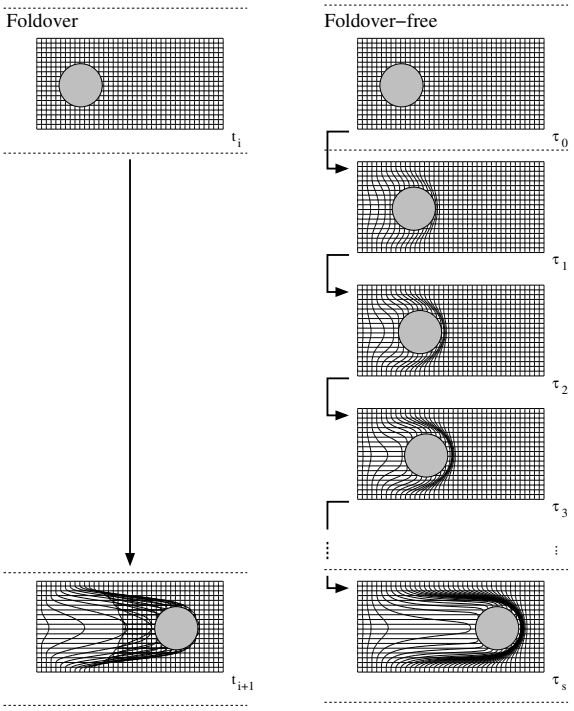


Figure 22: *2D illustration of our solution to foldovers. Left: the deformation maps space onto itself. Right: the deformation is decomposed into small foldover-free steps.*

4.2. Defining simple tools

To define a tool, a smooth function μ can be composed to the distance to a shape. We chose to use the following C^2 piecewise polynomial, in which λ controls the size of the influence of the tool:

$$\mu_\lambda(d) = \begin{cases} 0 & \text{if} \quad \lambda \le d \\ 1 + (\frac{d}{\lambda})^3 (\frac{d}{\lambda}(15 - 6\frac{d}{\lambda}) - 10) & \text{if} \quad d < \lambda \end{cases} \tag{45}$$

Ball tool: The distance to a ball has a simple expression in local coordinates:

$$d_{sphere}(\mathrm{p}) = \begin{cases} 0 & \text{if } \|M_{t_i}^{-1} \cdot \mathrm{p}\|^2 \le 1 \\ \det(M_{t_i}^{\frac{1}{3}})(\|M_{t_i}^{-1} \cdot \mathrm{p}\| - 1) & \text{otherwise} \end{cases} \tag{46}$$

If the artist wishes to apply a non-uniform scale to the sphere, it would turn into an ellipsoid, and Equation (46) would not be usable.

Filled ellipsoid tool: The ellipsoid is defined in local coordinates as a unit sphere, whose position in world coordinates is encoded in a possibly non-uniform matrix M_{t_i}. To compute the distance to a filled ellipsoid, we use the numerical method described in [Ebe01].

$$d_{\text{ellipsoid}}(\mathrm{p}) = \begin{cases} 0 & \text{if } \|M_{t_i}^{-1} \cdot \mathrm{p}\|^2 \le 1 \\ \min\limits_{\mathrm{q} \in S} \|\mathrm{p} - M_{t_i} \cdot \mathrm{q}\| & \text{otherwise,} \\ \text{where } S \text{ is the unit sphere at the origin} \end{cases} \tag{47}$$

Mesh tool: It is convenient for an artist to choose or manufacture his own tools. For this purpose, we propose the possibility of *baking* pieces of clay in order to use them as tools. By baking, we mean precomputing a data structure such that the distance field can be computed efficiently. We propose one way in [Ang05] (see Figure 23). More information on distance computation can be found in [Gué01].

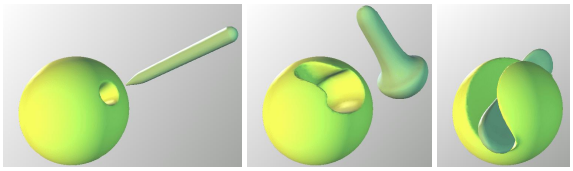


Figure 23: *Example of customized tools deforming a sphere.*

4.3. Single sweeper

As shown in Figure 22(right), if we decompose the transformation into a series of s small enough transformations, and apply each of them to the result of the previous one, foldovers are avoided. The decomposition in s steps for a general transformation is expressed as follows:

$$\begin{array}{rcl} f(\mathrm{p}) & = & \bigcirc_{k=0}^{s-1} f_k(\mathrm{p}) \\ \text{where } f_k(\mathrm{p}) & = & (\frac{\varphi^k(\mathrm{p})}{s} \odot M) \cdot \mathrm{p} \\ \text{and } \varphi^k(\mathrm{p}) & = & \varphi((\frac{k}{s} \odot M^{-1}) \cdot \mathrm{p}) \end{array} \quad (48)$$

The value returned by φ^k is that of the scalar field φ transformed by $\frac{k}{s} \odot M$, a fraction of M. It can be shown that there exists a finite number of steps such that the deformation is foldover-free (see [Ang05]). We propose the following as a lower bound to the required number of steps s:

$$\max_{\mathrm{p}} \|\nabla\varphi(\mathrm{p})\| \max_{l\in[1,8]} \|\log(M) \cdot \mathrm{p}_l\| < s \quad (49)$$

where $\mathrm{p}_{l\in[1,8]}$ are the corners of a box oustide which the function φ equals zero.

4.4. Simultaneous sweepers

Applying more than one operation at the same time and the same place has applications in modeling: for instance for modeling a symmetric object, or to define a tool composed of several tools. The simultaneous manipulation of tools also allows the artist to pinch a shape. Let us consider n operations, defined with $M_{i\in[1,n]}$ and $\varphi_{i\in[1,n]}$. The following is a naive way to achieve simultaneous deformations, using the formalism of M. Alexa (see Appendix A):

$$f(\mathrm{p}) = (\bigoplus_{i=1}^{n} \varphi_i(\mathrm{p}) \odot M_i) \cdot \mathrm{p} \quad (50)$$

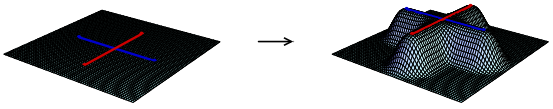
This function is naive because it adds the effect of each operation. The following expression provides a *normalized* and *smooth*[§] combination of all the transformations at any point p in space[¶]:

$$\begin{cases} \mathrm{p} & \text{if } \sum_k \varphi_k = 0 \\ \bigoplus_{i=1}^{n} \left(\left(\frac{1-\prod_k(1-\varphi_k)}{\sum_k \varphi_k} \varphi_i \right) \odot M_i \right) \cdot \mathrm{p} & \text{otherwise} \end{cases} \quad (51)$$

where:

- $\frac{1}{\sum_k \varphi_k}$ is required to produce a **normalized** combination of the transformations. This prevents for instance two translations of vector $\vec{d}$ producing translations of vector $2\vec{d}$, which would send some points far away from the tools. This unwanted behaviour was also identified by K. Singh and E. Fiume [SF98].
- $1 - \prod_{k=1}^{n}(1 - \varphi_k(\mathrm{p}))$ **smooths** the deformation in the entire space. Smoothness would be lost between the regions where $\sum_k \varphi_k = 0$ and $\sum_k \varphi_k \neq 0$ if we only used the normalization above.

Figure 25 shows a comparison between additive blending of Equation (50) and the correct one of Equation (51). In Figure 24, we show our blending in a scenarion similar to existing blending methods, presented in Section 3.



Reference segments Translated segments

Figure 24: *Blending with sweepers. The surface appears nice and smooth, as opposed to surfaces in Figures 10,11,12, 25,21 and 20.*

Equation (51) however may produces foldovers for similar reasons to the case of a single tool, with Equation (44). If we decompose it into small steps, foldovers can be avoided:

$$\begin{array}{rcl} f(\mathrm{p}) & = & \bigcirc_{k=0}^{s-1} f_k(\mathrm{p}) \\ \text{where } f_k(\mathrm{p}) & = & \begin{cases} \mathrm{p} & \text{if } \sum_j \varphi_j^k = 0 \\ \bigoplus_{i=1}^{n} \left(\left(\frac{1-\prod_j(1-\varphi_j^k)}{\sum_j \varphi_j^k} \varphi_i^k \right) \odot M_i \right) \cdot \mathrm{p} & \text{otherwise} \end{cases} \\ \text{and } \varphi_j^k(\mathrm{p}) & = & \varphi_j((\frac{k}{s} \odot M_j^{-1}) \cdot \mathrm{p}) \end{array} \quad (52)$$

Note that the value returned by φ_j^k is that of the scalar field φ_j transformed by $\frac{k}{s} \odot M_j$, a fraction of M_j. The following expression is a lower bound to the required number of steps, generalizing the single tool condition (see justification in [Ang05]):

$$\sum_j \max_{\mathrm{p}}(\|\nabla\varphi_j(\mathrm{p})\|) \max_{l\in[1,8]} \left\| \log M_j \cdot \mathrm{p}_{l_j} \right\| < s \quad (53)$$

§ as smooth as the φ_i.

¶ The operator $\oplus$ expresses a repetive sum: $\bigoplus_{i=1}^{n} M_i = M_1 \oplus M_2 \oplus \cdots \oplus M_n$.

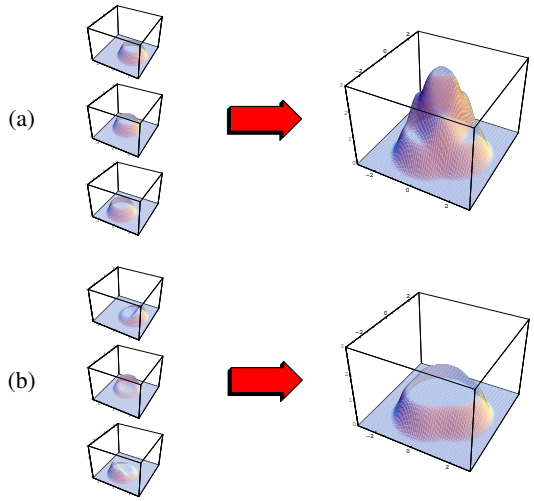


Figure 25: *Blending of three scalar fields. To illustrate the behaviour of our blending in this figure, we directly combine the scalar fields instead of using them to modulate a transformation. (a) Adding the scalar fields. (b) By multiplying each field with* $(1-\prod(1-\varphi_k))/\sum\varphi_k$*, the sum of the fields is normalized.*

where $p_{l_j\in[1,8]}$ are the corners of a bounding box oustide which the function φ_j equals zero.

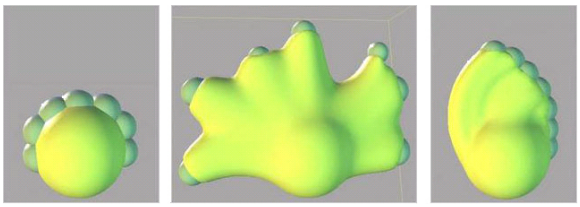


Figure 26: *Simultaneous sweepers*

4.5. Fast single sweeper

In a single tool scenario, some transformations are convenient to input by the artist: translations, non-uniform and uniform scaling and rotations. With these simple transformations, the deformations of a point is much simpler to compute, as there is a closed-form to the logarithm of simple matrices. In this section, in addition to efficient expressions for computing the number of required steps, we provide fast deformation functions for a vertex and its normal. For deforming the normal, computing the Jacobian's co-matrix is not always required: $J^C \cdot \vec{n}$ leads to much simpler expressions. Note that the normal's deformations do not preserve the normal's length. It is therefore necessary to divide the normal by its magnitude. We denote $\vec{\gamma}_k = (\gamma_x, \gamma_y, \gamma_z)^\top$ the gradient of φ_k at p and $\vec{\gamma}$ the gradient of φ at p

If *M* is a translation: The use of $\odot$ simplifies, using translation vector $\vec{d}$. The minimum number of steps is:

$$\max_p \|\vec{\gamma}(p)\| \; \|\vec{d}\| < s \tag{54}$$

The s vertex deformations are:

$$f_k(p) = p + \frac{\varphi_k(p)}{s}\vec{d} \tag{55}$$

The s normal deformations are:

$$g_k(\vec{n}) = \vec{n} + \frac{1}{s}(\vec{\gamma}_k \times \vec{n}) \times \vec{d} \tag{56}$$

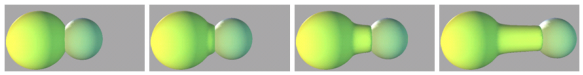


Figure 27: *Translation*

If *M* is a uniform scaling operation: Let us define the center of the scale c, and the scaling factor σ. The minimum number of steps is:

$$\max_p \|\vec{\gamma}(p)\| \; \sigma\log(\sigma) d_{\max} < s \tag{57}$$

where $d_{\max}$ is the largest distance between a point in the deformed area and the center c, approximated using a bounding box. The s vertex deformations are:

$$f_k(p) = p + (\sigma^{\frac{\varphi_k(p)}{s}} - 1)(p - c) \tag{58}$$

Let $\vec{\chi} = \frac{\log(\sigma)}{s}(p-c)$. The s normal deformations are:

$$g_k(\vec{n}) = \vec{n} + (\vec{\gamma}_k \times \vec{n}) \times \vec{\chi} \tag{59}$$

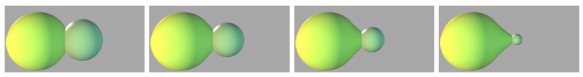


Figure 28: *Scale*

If *M* is a non-uniform scaling operation: Let us define the center of the scale c , its direction of scale, unit vector $\vec{n}$, and its scaling factor, σ. The minimum number of steps is:

$$\max_p \|\vec{\gamma}(p)\| \; \sigma\log(\sigma) d_{\max} < s \tag{60}$$

where $d_{\max}$ is the largest distance between a point in the deformed area and the plane of normal $\vec{n}$ passing through c. The s vertex deformations are:

$$f_k(p) = p + (\sigma^{\frac{\varphi_k(p)}{s}} - 1)((p-c)\cdot\vec{n})\vec{n} \tag{61}$$

Let $\vec{\chi} = \frac{\log(\sigma)}{s}(p-c)$. The s normal deformations are:

$$g_k(\vec{n}) = \vec{n} + \sigma^{\frac{\varphi_k(p)}{s}}((\vec{v} + (\vec{v}\cdot\vec{\chi})\vec{\gamma}_k) \times \vec{n}) \times \vec{v} \tag{62}$$

It is appropriate to remark here that the tool is also subject to the scale, and that the influence function φ_t must be defined in an appropriate way, as described in Section **??**.

If *M* is a rotation: Let us define a rotation angle θ, center of

rotation r and vector of rotation $\vec{v} = (v_x, v_y, v_z)^\top$. The minimum number of steps is:

$$\max_{p} \|\vec{\gamma}(p)\| \, \theta r_{\max} < s \tag{63}$$

where $r_{\max}$ is the distance between the axis of rotation and the farthest point from it, approximated using a bounding box. The s vertex deformations are:

$$f_k(p) = p + (\cos\frac{\varphi_k\theta}{s} - 1)\vec{\xi} \times \vec{n} + \sin\frac{\varphi_k\theta}{s}\vec{\xi} \tag{64}$$
$$\text{where } \vec{\xi} = \vec{v} \times (p - r)$$

The s normal deformations are:

$$\begin{aligned} g_k(\vec{n}) &= (\vec{n}\cdot\vec{v})\vec{v} + \vec{v} \times (\cos(h)\vec{n} \times \vec{v} - \sin(h)\vec{n}) \\ &\quad + \theta\vec{\gamma} \times (\vec{n} \times \vec{\xi} \\ &\quad + ((\cos(h) - 1)(\vec{n} \times \vec{\xi}) \cdot \vec{v} \\ &\quad + \sin(h)\vec{n} \cdot \vec{\xi})\vec{v}) \end{aligned} \tag{65}$$
$$\text{where } h = \frac{\varphi_k\theta}{s}$$

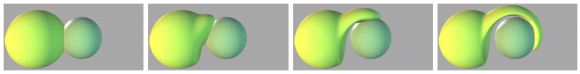


Figure 29: *Rotation*

4.6. Fast symmetric sweepers

For an operation symmetric about a plane, the transformation matrices are of the same type, thus blending them leads to simple expressions. Let us consider two tools φ_0 and φ_1. If the influence of both tools is zero at p, that is if $\varphi_0(p) = 0$ and $\varphi_1(p) = 0$, then the deformation is the identity. If the influence of one tool is zero at p, that is if $\varphi_0(p) = 0$ or $\varphi_1(p) = 0$, then the deformation equation is that of a single tool. When both influences are not zero at p, that is if $\varphi_0(p) \neq 0$ and $\varphi_1(p) \neq 0$, then the deformation induced at p by the tools' motion must be computed using Equation (51). In the rest of this section, we have simplified the blending equation for simple symmetric transformations of the same type.

Translation: The number of steps is:

$$\max_{p} \|\vec{\gamma}(p)\| \, (\|\vec{d}_0\| + \|\vec{d}_1\|) < s \tag{66}$$

The deformation of a point is:

$$f_k(p) = p + \frac{1}{s}(1 - \frac{\varphi_0 + \varphi_1}{\varphi_0\varphi_1})(\varphi_0\vec{d}_0 + \varphi_1\vec{d}_1) \tag{67}$$

Rotation, scale and non-uniform scale: The deformation of a point is:

$$f_k(p) = \exp(\frac{1}{s}(1 - \frac{\varphi_0 + \varphi_1}{\varphi_0\varphi_1})(\varphi_0 \log M_0 + \varphi_1 \log M_1)) \tag{68}$$

4.7. Results

Although a few simple transformations were combined (translation, uniform scale and rotation), the set of possible deformations is very high because of the arbitrary shape of the tools, and also because many tools' deformations can be blended. The shapes shown in Figure 30 were modeled in real-time in *one hour* at most, and were all made starting with a sphere.

Figures 30(a) and 30(b) show the use of the multi-tool to achieve smooth and symmetric objects. Figure 30(d) shows that sharp features can be easily modeled. Figures 30(c) and 30(i) show the advantage of foldover-free deformations, as the artist did not have to concentrate on avoiding self-intersections: our deformations do not change the topology of space and thus preserve the topology of the initial object.

5. Modeling with constant volume

In a non-virtual modeling context, one of the most important factors which affects the artist's technique is the amount of available material. This aspect was ignored in the previous sections. The notion of an amount of material is not only familiar to professional artists, but also to children, who may experience it with Play-Doh® at kindergarten, and to adults through everyday life experience. A shape modeling technique that preserves volume will take advantage of this, and will hopefully be genuinely intuitive to use.

5.1. Swirl

We define a particular case of sweeper, a *swirl*, by using a point tool c, together with a rotation of angle θ around an axis $\vec{v}$ (see Figure 31). A scalar function, φ, and a deformation are defined as before. Informally, a swirl twists space locally around axis $\vec{v}$ without compression or dilation (see proof in [ACWK04]): it preserves volume.

5.2. Ring of swirls

Many deformations of the above kind can be naively combined to create a more complex deformation:

$$f(p) = \left(\bigoplus_{i=0}^{n-1}(\varphi_i(p) \odot M_i)\right) \cdot p \tag{69}$$

It is important here to remark that the above blending is *not* the blending formula of simultaneous tools defined in Equation (51), and only uses simple weights. The reason for using the above simple blending equation as opposed to Equation (51) is that the latter modulates the amount of individual transformations locally, and attempting to control the volume with it would be inappropriate. We provide a convenient way for the artist to input n rotations, by specification of a single translation $\vec{t}$. Let us consider n points, c_i, on the circle of center h, and radius r lying in a plane perpendicular to $\vec{t}$. To these points correspond n consistently-oriented

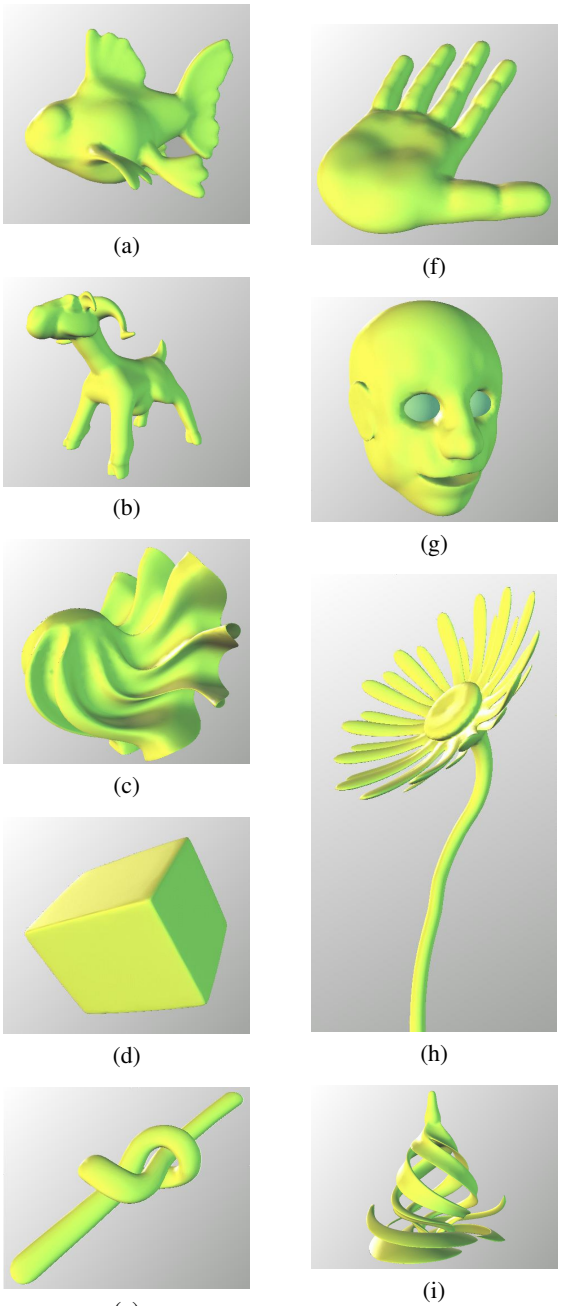


Figure 30: *All these shapes were modeled starting with a sphere, in at most one hour. In (c), the first modeling step was to squash the sphere into a very thin disk. In (g), eyeballs were added.*

unit tangent vectors $\vec{v}_i$ (see Figure 32). Each pair, $(c_i, \vec{v}_i)$, together with an angle, θ_i, define a rotation. Along with radii of influence $\lambda_i = 2r$, we can define n swirls. The radius of the circle r, is left to the user to choose. The following value for θ_i will transform h exactly into $h + \vec{t}$ (see justification in [ACWK04]):

$$\theta_i = \frac{2\|\vec{t}\|}{nr} \tag{70}$$

With this information, the deformation of Equation (69) is now a tool capable of transforming a point into a desired target. We show in Figure 32 the effect of the tool for different values of n; in practice, we use 8 swirls.

5.2.0.1. Preserving coherency and volume If the magnitude of the input vector $\vec{t}$ is too large, the deformation of Equation (69) will produce a self-intersecting surface, and will not preserve volume accurately. The reason for self-intersection is explained in Section 4.1. The volume is not accurately preserved because the blending operator, $\oplus$, blends the transformation matrices, and not the deformations. To correct this, it is necessary to subdivide $\vec{t}$ into smaller vectors. Thus foldovers and volume preservation are healed with the same strategy. The number of steps must be proportional to the speed and inversely proportional to the size of the tool. We use:

$$s = \max(1, \lceil 4\|\vec{t}\|/r \rceil) \tag{71}$$

As the circle sweeps space, it defines a cylinder. Thus the swirling-sweeper is made of ns basic deformations. Figure 33 illustrates this decomposition applied to a shape.

5.3. Swirling-Sweepers

We summarize here the swirling-sweepers algorithm:

Input point h, translation $\vec{t}$, and radius r
Compute the number of required steps s
Compute the angle of each step, $\theta_i = \frac{2\|\vec{t}\|}{nrs}$
for each step k from 0 to $s-1$ **do**
  for each point p in the tool's bounding box **do**
    $M = 0$
    for each swirl i from 0 to $n-1$ **do**
      $M \mathrel{+}= \varphi_k^i(p) \log M_{i,k}$
    end for
    $p = (\exp M) \cdot p$
  end for
end for

The point c_{ik} denotes the center of the i^{th} swirl of the k^{th} ring of swirls. For efficiency, a table of the basic-swirl centers, c_{ik}, and a table of the rotation matrices, $\log M_{i,k}$, are precomputed. We have a closed-form for the logarithm of the involved matrix, given in Equations (72) and (73), saving an otherwise expensive numerical approximation:

$$\begin{aligned} \vec{n} &= \theta_i \vec{v}_i \\ \vec{m} &= c_{i,k} \times \vec{n} \end{aligned} \tag{72}$$

$$\log M_{i,k} = \begin{pmatrix} 0 & -n_z & n_y & m_x \\ n_z & 0 & -n_x & m_y \\ -n_y & n_x & 0 & m_z \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{73}$$

Note that for the sake of efficiency, we handle these matrices as mere pairs of vectors, $(\vec{n},\vec{m})$. Once M is computed, we use a closed-form for computing $\exp M$. Since the matrix M is a weighted sum of matrices $\log M_{i,k}$, the matrix M is of the form of Equation (73), and can be represented with a pair $(\vec{n}_M,\vec{m}_M)$. If $\vec{n}_M = 0$, then $\exp M$ is a translation of vector $\vec{m}_M$. Else, if the dot product $\vec{m}_M \cdot \vec{n}_M = 0$, then $\exp M$ is a rotation of center c, angle θ axis $\vec{v}$, as given by Equation (74):

$$\begin{array}{rcl} c & = & \frac{\vec{\omega}\times\vec{m}}{\|\vec{\omega}\|^2} \\ \theta & = & \|\vec{n}_M\| \\ \vec{v} & = & \vec{n}_M/\theta \end{array} \qquad (74)$$

Finally, in the remaining cases, we denote $l = \|\vec{n}_M\|$, and we use Equation (75) (see Appendix B for efficiency):

$$\exp M = I + M + \frac{1-\cos l}{l^2}M^2 + \frac{l-\sin l}{l^3}M^3 \qquad (75)$$

Symmetrical objects can be easily modeled by introducing a plane of symmetry about which the tool is reflected (see Figure 35).

5.4. Results

In Figure 35, we compare the shapes' volume with unit spheres on the right. The shapes volumes are respectively 101.422%, 99.993%, 101.158% and 103.633% of the initial sphere. This error is the result of accumulating smaller errors from each deformation. For instance 80 swirling-sweepers have been used to model the alien. The small errors are due to the finite number of steps, and to our choice of shape representation. The shapes shown in Figure 35 were modeled in real-time in *half an hour* at most, and were all made starting with a sphere.

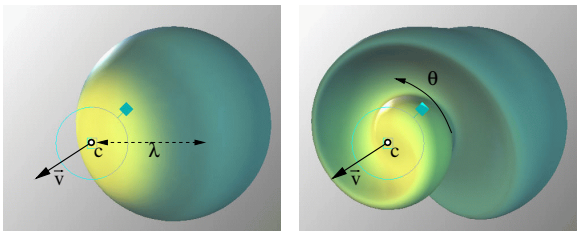


Figure 31: *The effect on a sphere of a swirl centered at c, with a rotation angle θ around $\vec{v}$. The two shapes have the same volume.*

6. A shape description

With sweepers and swirling-sweepers, shape modeling operations based on gesture can be conveniently described, while coherency and volume of the shape are maintained. By using both operation types, the artist can increased, decreased or preserved the volume of a shape (see Figure 36). Because these operation types are indepent from the shape description, several choices are available: mesh, particles, discrete grid of deformed raytracing (see [Ang05]). In the context of shape modeling, the number of deformations is possibly excessively large, and issues related to such excess have to be taken into consideration when defining a shape description. We provide in this section a shape description for interactive modeling which supports high deformation and does not break when highly stretched.

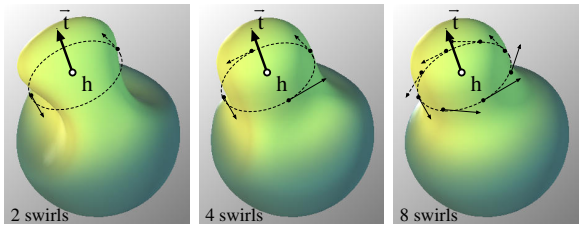


Figure 32: *By arranging n basic swirls in a circle, a more complex deformation is achieved. In the rightmost image: with 8 swirls, there are no visible artifacts due to the discrete number of swirls.*

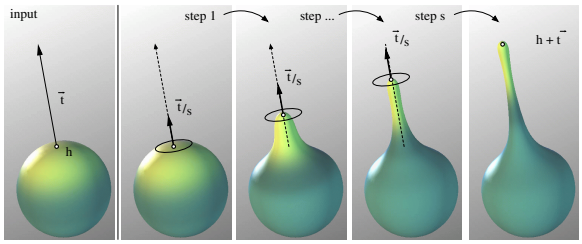


Figure 33: *A volume preserving deformation is obtained by decomposing a translation into circles of swirls. 3 steps have been used for this illustration. As the artist pulls the surface, the shape gets thinner. The selected point's transformation is precisely controlled.*

A simple way of representing a deformable shape is to place a set of samples on the surface of the shape: this makes the task of deforming the shape as straightforward as deforming the points on its surface. Points are discrete surface samples, and need to be somehow connected using splatting, interpolation or approximation scheme in order to display a continuous surface.

Our method uses an updated mesh, i.e. vertices connected with triangles. Connectivity provides convenient $2D$-boundary information for rendering the surface as well as surface neighborhood information, which enables the artist to define very thin membranes without having them vanish, as shown in Figure 30(c). The use of triangular "C^0 patches" circumvents issues related to non-regular vertices and smoothness maintenance across the boundaries that join patches. Also, current hardware handles polygons very efficiently, which is relevant to us since interactivity is among our objectives. The reader however should be aware that point-sampled geometry has recently ignited a lot of interest from researchers [PKKG03].

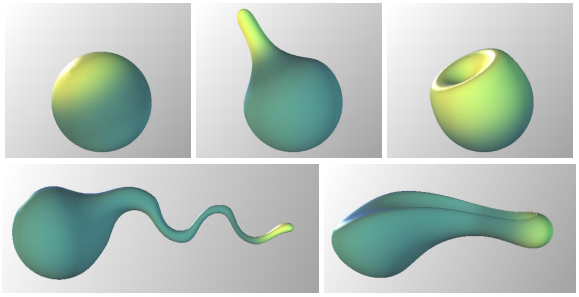


Figure 34: *When pushed or pulled, a sphere will inflate or deflate elsewhere.*

The possibly large number of deformations applied by an artist requires some minimum surface sampling density. Thus, the scene is initialized with a polygonal model, e.g. a sphere with a homogeneous density of nearly equilateral triangles[‖]. In order to quickly fetch the vertices to be deformed and the edges that require splitting or collapsing, these are inserted into a 3D grid. Note that this spatial limitation is not too restrictive for the artist, as our deformations allow us to translate the entire model rigidly and scale it uniformly.

To fetch the vertices that are deformed, a query is done with the tool's bounding box. Conveniently, this bounding box is also used in Eq. (49). Since the principle of our swept deformations is to subdivide the input gesture into a series of smaller ones, all the transformations applied to the vertices are bounded. To take advantage of this decomposition in steps, we apply a modified version of a more generic algorithm [GD99]. Our method requires keeping two vertices and two normals per vertex, corresponding to the previous and following state of some small step operation f_k. Loosely speaking, our surface-updating algorithm assumes that smooth curves run on the surface, and that the available information, namely vertices and normals, should be able to represent them. If this is not the case after deformation, then it means the surface is under-sampled. On the other hand, if an edge is well enough represented by a single sample, then it is collapsed.

Let us consider an edge e defined by two vertices (v_0, v_1) with normals $(\vec{n}_0, \vec{n}_1)$, and the deformed edge e' defined by vertices (v'_0, v'_1) with normals $(\vec{n}'_0, \vec{n}'_1)$. In addition to the conditions in [GD99] based on edge length and angle between normals, we also base the choice of splitting edge e' on the error between the edge and a fictitious vertex, which belongs to a smooth curve on the surface. The fictitious vertex is used only for measuring the error, and is not a means of interpolating the vertices. If the error between the fictitious vertex and the edge is too large, the edge e is split, and the new vertex and normal are deformed. On the other hand if the fictitious vertex represents the edge e' well enough, then edge e is collapsed, and the new vertex is deformed. We

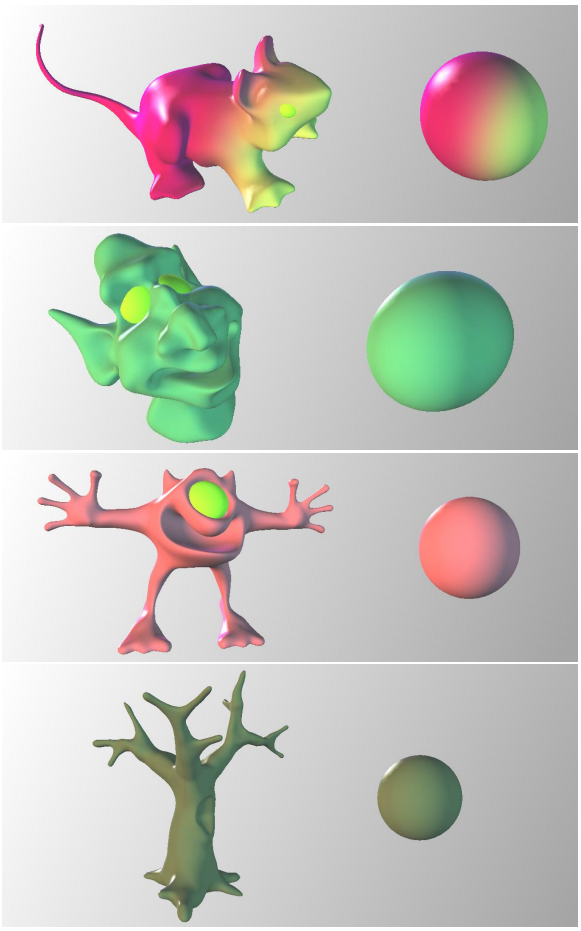


Figure 35: *Examples of models "sculpted" with swirling-sweepers. The mouse, the goblin, the alien and the tree have respectively 27607, 25509, 40495 and 38420 vertices. These objects were modeled in less than 30 min by one of the authors. Eyeballs have been added.*

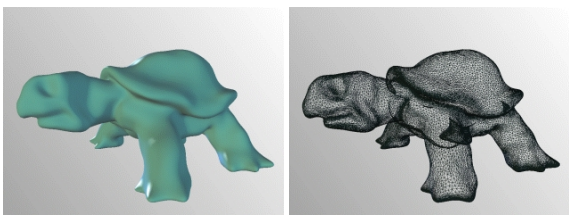


Figure 36: *Shape modeled with sweepers and swirling-sweepers.*

[‖] A simple way to obtain an homogeneous sphere polygonization consists of starting with an icosahedron, putting all its edges longer than h in a queue, splitting them and putting the pieces longer than h back in the queue. Each time a split is performed, the new edges are flipped to maximize the smallest angle.

define the fictitious vertex as the mid-vertex of a C^1 curve, since vertices and normals only provide first order information about the surface. The following cubic polynomial curve interpolates the vertices v'_0 and v'_1 with corresponding shape tangents $\vec{t}_0$ and $\vec{t}_1$, defined below:

$$\begin{aligned} c(u) \quad = \quad & (v'_0(1+2u)+\vec{t}_0 u)(1-u)^2 + \\ & (v'_1(1+2(1-u))-\vec{t}_1(1-u))u^2 \end{aligned} \tag{76}$$

The only constraint on tangent $\vec{t}_i$ is to be perpendicular to the corresponding normal $\vec{n}_i$. The following choice defines tangents of magnitude proportional to the distance between the vertices:

$$\begin{aligned} \vec{t}_0 &= \vec{g} - \vec{g}\cdot\vec{n}'_1 \\ \vec{t}_1 &= \vec{g} - \vec{g}\cdot\vec{n}'_0 \\ \text{where} \quad \vec{g} &= v'_1 - v'_0 \end{aligned} \tag{77}$$

With the above tangents, the expression of the middle vertex simplifies:

$$c(0.5) = (\, v'_0 + v'_1 + (\vec{g}\cdot\vec{n}'_0 - \vec{g}\cdot\vec{n}'_1)/4 \,) \,/\, 2 \tag{78}$$

With the fictitious vertex $c(0.5)$, the tests to decide whether an edge should be split or collapsed can now be defined:

Too-long edge: An edge e' is too long if *at least one* of the following conditions is met:

- The edge is longer than $L_{\max}$, the size of a grid-cell. This condition keeps a minimum surface density, so that the deformation can be caught by the net of vertices if the coating thickness λ_j is greater than $L_{\max}$.
- The distance between the fictitious vertex and the mid-vertex of e' is too large (we used $L_{\max}/20$). This condition prevents the sampling from folding on itself, which would produce multiple sampling layers of the same surface.
- The angle between the normals $\vec{n}'_0$ and $\vec{n}'_1$ is larger than a constant $\theta_{\max}$. This condition keeps a minimum curvature sampling.

Too-short edge: An edge e' is too short if *all of* the following conditions are met:

- The edge's length is shorter than $L_{\min}$ (we used $L_{\max}/2$).
- The angle between the normals $\vec{n}'_0$ and $\vec{n}'_1$ is smaller than a constant $\theta_{\min}$.
- The distance between the fictitious vertex and the mid-vertex of e' is too small (we used $L_{\min}/20$).

Also, to avoid excessively small edges, an edge is merged regardless of previous conditions if it is too small (we used $L_{\min}/20$).

We stress that the procedure for updating the mesh is applied at each small step, rather than after the user's deformation function has been applied. Because vertex displacements are bounded by the foldover-free conditions, the update of our shape description does not suffer from problems related to updating a greatly distorted triangulation. Figure 37 shows a twist on a simple U-shape. Figure 38 shows the algorithm preserving a fine triangulation only where required. Figure 39 shows the algorithm at work in a more practical situation. The procedure outline is:

```
Compute the number of steps required, s.
for each step k do
  Deform the points, and hold their previous values
  for each too-long edge do
    split the edge and deform the new point.
  end for
  for each too-short edge do
    collapse the edge and deform the new point.
  end for
end for
```

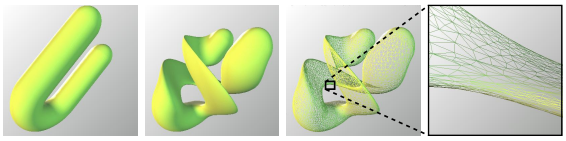


Figure 37: *Example of our mesh-updating algorithm on a highly twisted U-shape. The close-up shows a sharp feature, with finer elongated triangles.*

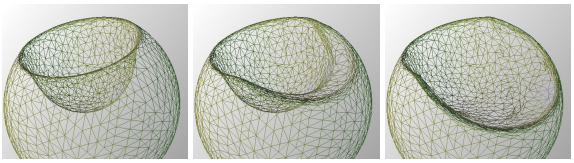


Figure 38: *Behaviour of our mesh-updating algorithm on an already punched sphere. The decimation acompanying the second punch simplifies the small triangle of the first punch. The tool has been removed for better visualization.*

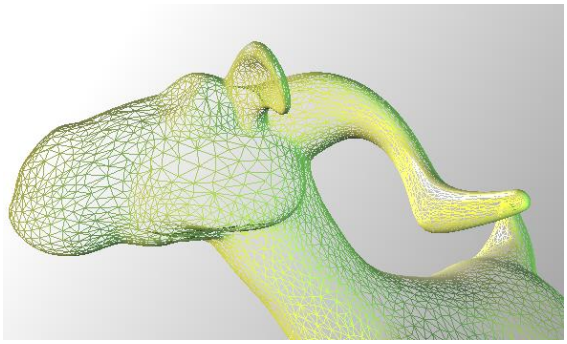


Figure 39: *Close-up of the goat. Notice the large triangles on the cheek and the fine ones on the ear. The initial shape is a sphere.*

Limitation: With the updated mesh method, we choose to ignore the history of functions applied to the shape by the artist. Thus we "collapse" the history by freezing it in the current shape. To explain the major consequence of this, let us suppose the scene at a time t_k, such that the shape $S(t_k)$ is shown to the user. The next deformation produced by the artist with the mouse is function $f_{t_k \mapsto t_{k+1}}$, and all the

mesh refinements and simplifications are performed in $S(t_k)$. This is however an approximation: ideally the last operation should be concatenated to the history of deformations, and the whole series should be applied to the initial shape $S(t_0)$, i.e. $\bigcirc_{i=0}^{n} f_{t_i \mapsto t_{i+1}}$ should be applied to each new vertex. This would however become more and more time consuming as the sequence of deformations gets longer (n gets larger), and the modeling software would eventually become unusable.

7. Conclusion

Sweepers, is a framework for defining swept deformation operation for shape modeling. It permits the description of a family of shape operations based on gesture between the artist and the mathematics describing the shape, and enables an artist to handle shapes in a more efficient way than modifying directly a shape's mathematical description. Because sweepers are foldover-free, they maintain easily a shape coherency. Swirling-sweepers is a type of swept-deformation for describing shape modeling operations that preserve implicitly the shape's volume. Subjectively, swirling-sweepers is the most effective modeling technique defined in the sweepers framework. Further work on volume-preservation outlines that there is in fact a link between swirling-sweepers and fluid mechanics [AN]. The separation of the shape's operations and the shape's description leads to the exploration of alternative ways to describe a shape's surface or volume for rendering. While our proposed method is sufficient in a wide range of situations, more research should be done in this area.

8. Acknowledgments

Many thanks to Marie-Paule Cani, Geoff Wyvill, Scott King and Brendan McCane for contributing to the work on sweepers and swirling-sweepers presented in this chapter.

Appendix A: Linear Combination of Transformations

The multiplication operator $\odot$ and addition operator $\oplus$, which behave essentially like $\cdot$ and $+$ for scalars. The operator $\odot$ is defined as $\alpha \odot M = \exp(\alpha \log M)$ and the operator $\oplus$ is defined as $M \oplus N = \exp(\log M + \log N)$. The following series defines the exponential of a matrix:

$$\exp M = \mathrm{I} + M + \frac{1}{2}M^2 + \frac{1}{6}M^3 \cdots = \sum_{k=0}^{\infty} \frac{M^k}{k!} \qquad (79)$$

The logarithm of a matrix is defined as an inverse of the exponential, as follows:

$$\log(\mathrm{I} - M) = -M - \frac{1}{2}M^2 - \frac{1}{3}M^3 \cdots = -\sum_{k=1}^{\infty} \frac{M^k}{k} \qquad (80)$$

In a similar that repeting $+$ can be expressed with $\sum$, the repetition of $\oplus$ can be expressed as follows:

$$\bigoplus_{i=1}^{n} M_i = M_1 \oplus M_2 \oplus \cdots \oplus M_n \qquad (81)$$

Appendix B: Exponential

Applying the exponential of the matrix to a point does not require to compute the exponential of the matrix explicitly. Let us define the matrix M with a pair of vectors, $(\vec{n}, \vec{m})$.

$$\begin{aligned} \exp(M) \cdot \mathrm{p} &= \mathrm{p} + (\vec{\mathrm{m}} + \vec{\mathrm{n}} \times \mathrm{p})b + (\tfrac{\vec{\mathrm{n}} \times \vec{\mathrm{m}}}{l^2} - \mathrm{p})a \\ &\quad + \vec{\mathrm{n}}((\vec{\mathrm{n}} \cdot \mathrm{p})a + (\vec{\mathrm{n}} \cdot \vec{\mathrm{m}})(1-b))\tfrac{1}{l^2} \\ \text{where } l &= \|\vec{\mathrm{n}}\| \\ a &= 1 - \cos(l) \\ b &= \tfrac{\sin(l)}{l} \end{aligned} \qquad (82)$$

References

[ACWK04] ANGELIDIS A., CANI M.-P., WYVILL G., KING S.: Swirling-sweepers: Constant-volume modeling. In *Pacific Graphics 2004* (October 2004), IEEE, pp. 10–15. Best paper award at PG04. 1, 16, 17

[Ale02] ALEXA M.: Linear combination of transformations. In *Proceedings of SIGGRAPH'02* (July 2002), vol. 21(3) of *ACM Transactions on Graphics, Annual Conference Series*, ACM, ACM Press / ACM SIGGRAPH, pp. 380–387. 13

[AN] ANGELIDIS A., NEYRET F.: Simulation of smoke based on vortex filament primitives. In *Symposium on Computer Animation'05*. http://www-evasion.imag.fr/Publications/2005/AN05. 21

[Ang05] ANGELIDIS A.: *Shape Modeling by Swept Space Deformation*. PhD thesis, University of Otago, 2005. 14, 18

[AW04] ANGELIDIS A., WYVILL G.: Animated sweepers: Keyframed swept deformations. In *CGI'04* (July 2004), IEEE, pp. 320–326. 1

[AWC04] ANGELIDIS A., WYVILL G., CANI M.-P.: Sweepers: Swept user-defined tools for modeling by deformation. In *Proceedings of Shape Modeling and Applications* (June 2004), IEEE, pp. 63–73. Best paper award at SMI04. 1

[Bar84] BARR A. H.: Global and local deformations of solid primitives. In *Proceedings of SIGGRAPH'84* (July 1984), vol. 18(3) of *Computer Graphics Proceedings, Annual Conference Series*, ACM, ACM Press / ACM SIGGRAPH, pp. 21–30. 2, 12

[BB91] BORREL P., BECHMANN D.: Deformation of n-dimensional objects. In *Proceedings of the first symposium on Solid modeling foundations and CAD/CAM applications* (1991), pp. 351–369. 7

[BBB*97] BAJAJ C., BLINN J., BLOOMENTHAL J., CANI-GASCUEL M.-P., ROCKWOOD A., WYVILL B., WYVILL G.: *Introduction to Implicit Surfaces*. Morgan-Kaufmann, 1997. 10

[Bla94] BLANC C.: A generic implementation of axial procedural deformation techniques. In *Graphics Gems* (1994), vol. 5, pp. 249–256. Academic Press. 2, 12

[Blo90] BLOOMENTHAL J.: Calculation of reference frames along a space curve. *Graphics gems* (1990), 567–571. 3

[BR94] BORREL P., RAPPOPORT A.: Simple constrained deformations for geometric modeling and interactive design. In *ACM Transactions on Graphics* (April 1994), vol. 13(2), pp. 137–155. 7

[Coq90] COQUILLART S.: Extended free-form deformation: A sculpturing tool for 3d geometric modeling. In *Proceedings of SIGGRAPH'90* (July/August 1990), vol. 24(4) of *Computer Graphics Proceedings, Annual Conference Series*, ACM, ACM Press / ACM SIGGRAPH, pp. 187–195. 6, 12

[CR94] CHANG Y.-K., ROCKWOOD A.: A generalized de Casteljau approach to 3d free-form deformation. In *Proceedings of SIGGRAPH'94* (July 1994), Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, pp. 257–260. 3, 12

[Cre99] CRESPIN B.: Implicit free-form deformations. In *Proceedings of the Fourth International Workshop on Implicit Surfaces* (1999), pp. 17–24. 3, 4, 10

[Dec96] DECAUDIN P.: Geometric deformation by merging a 3d object with a simple shape. In *Graphics Interface* (May 1996), pp. 55–60. 9, 10, 12

[Ebe01] EBERLY D. H.: *3D Game Engine Design*. Morgan Kaufmann, 2001. 13

[Far90] FARIN G.: Surfaces over Dirichlet tessellations. *Computer Aided Geometric Design 7(1-4)* (June 1990), 281–292. 8

[GD99] GAIN J. E., DODGSON N. A.: Adaptive refinement and decimation under free-form deformation. *Eurographics'99 7*, 4 (April 1999), 13–15. 19

[GD01] GAIN J. E., DODGSON N. A.: Preventing self-intersection under free-form deformation. *IEEE Transactions on Visualization and Computer Graphics 7*, 4 (October-December 2001), 289–298. 8, 12

[Gla89] GLASSNER A. (Ed.): *An Introduction to Ray traycing*. Morgan Kaufmann, 1989. 2

[GP89] GRIESSMAIR J., PURGATHOFER W.: Deformation of solids with trivariate b-splines. In *Eurographics Conference Proceedings* (1989), Elsevier Science, pp. 137–148. 6

[Gri] GRIFFITHS D. J.: *Introduction to Electrodynamics*, third ed. Prentice Hall. 6

[Gué01] GUÉZIEC A. P.: "Meshsweeper": Dynamic point-to-polygonal-mesh distance and applications. *IEEE Transactions on Visualization and Computer Graphics 7*, 1 (January/March 2001), 47–61. 14

[HHK92] HSU W. M., HUGHES J. F., KAUFMAN H.: Direct manipulation of free-form deformations. In *Proceedings of SIGGRAPH'92* (July 1992), vol. 26(2) of *Computer Graphics Proceedings, Annual Conference Series*, ACM, ACM Press / ACM SIGGRAPH, pp. 177–184. 7, 12

[HML99] HIROTA G., MAHESHWARI R., LIN M. C.: Fast volume-preserving free form deformation using multi-level optimization. In *Proceedings of the fifth ACM symposium on Solid modeling and applications* (June 1999), ACM, pp. 234–245. 9, 12

[HQ04] HUA J., QIN H.: Scalar-field-guided adaptive shape deformation and animation. *The Visual Computer 1*, 1 (April 2004), 47–66. 11, 12

[KY97] KURZION Y., YAGEL R.: Interactive space deformation with hardware assisted rendering. *IEEE Computer Graphics and Applications 17(5)* (September/October 1997), 66–77. 9

[LCJ94] LAZARUS F., COQUILLART S., JANCÃĹNE P.: Axial deformations: an intuitive deformation technique. In *Computer-Aided Design* (1994), vol. 26, 8, pp. 607–613. 3, 12

[LKG*03] LLAMAS I., KIM B., GARGUS J., ROSSIGNAC J., SHAW C. D.: Twister: A space-warp operator for the two-handed editing of 3d shapes. In *SIGGRAPH* (August 2003), vol. 22(3) of *ACM Transactions on Graphics, Annual Conference Series*, ACM, pp. 663–668. 11, 12

[MJ96] MACCRACKEN R. A., JOY K. I.: Free-form deformations with lattices of arbitrary topology. In *Proceedings of SIGGRAPH'96* (August 1996), Computer Graphics Proceedings,

Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, pp. 181–188. 6, 7, 12

[MMT97] MOCCOZET L., MAGNENAT-THALMANN N.: Dirichlet free-form deformation and their application to hand simulation. In *Computer Animation'97* (June 1997), pp. 93–102. 8, 12

[MW01] MASON D., WYVILL G.: Blendeforming: Ray traceable localized foldover-free space deformation. In *Proceedings of Computer Graphics International (CGI)* (July 2001), pp. 183–190. 5, 12

[PKKG03] PAULY M., KEISER R., KOBBELT L. P., GROSS M.: Shape modeling with point-sampled geometry. In *Proceedings of SIGGRAPH'03* (July 2003), vol. 22(3), ACM, pp. 641–650. 18

[Rut90] RUTHERFORD A.: *Vectors, Tensors and the Basic Equations of Fluid Mechanics*. Dover, 1990. 6

[SE04] SCHEIN S., ELBER G.: Discontinuous free form deformations. In *Proceedings of Pacific Graphics* (October 2004), IEEE, pp. 227–236. 9

[SF98] SINGH K., FIUME E.: Wires: a geometric deformation technique. In *Computer graphics, Proceedings of SIGGRAPH'98* (July 1998), Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, pp. 405–414. 4, 5, 14

[SP86] SEDERBERG T., PARRY S.: Free-form deformation of solid geometric models. In *Proceedings of SIGGRAPH'86* (August 1986), vol. 20(4) of *Computer Graphics Proceedings, Annual Conference Series*, ACM, ACM Press / ACM SIGGRAPH, pp. 151–160. 6, 12

[Wei04] WEISSTEIN E.: Lagrange multipliers. From Mathworld – A Wolfram Web Ressource `mathworld.wolfram.com/LagrangeMultipliers.html`, 2004. 12