Free Form Deformations or Deformations Non-Constrained by Geometries or Topologies

Romain Raffin

Abstract Free-form deformations are widely used to model 3D objects. In these methods "free-form" designates: "*whatever the object is, whatever its description and topology, we are able to deform it*". They limit the user interaction to pull some points of an embedding rough mesh. From the user point-of-view, it does not matter if the object manipulated is 3-dimensional, of 0-genus or a parametric surface, he or she always uses the same process to model a complex object: load an initial object from a library and deform it via FFD methods to follow his (her) needs. A large number of deformation methods have been published, allowing new deformations, new kinds of controls or enhancing the description of resulting objects. In fact advantage of deformation non-constrained by geometries is also a drawback: as it only manipulates points it could only result in points, so its necessary to use and maintain the neighborhood (the topology) or the surface expression on a second hand, as these methods do not care of object description.

1 Free-Form Deformations from the Beginning

The work of creating an object from an empty scene is quite tedious, first methods of mesh manipulation [28, 31] highlight the future needs of user-oriented, efficient modeling methods. They first deform an object from a single vertex, diffusing the deformation on neighbors according to edges distance weights. The work of Barr [2] initiates free-form deformation techniques. It permits to deform an existing object globally (see Fig. 1), with mathematical functions (taper, bend, twist). Since it was not a "vertex by vertex" displacement method it simplifies the creation process. This method does not act on topology, keeping the neighbors as they were, with the edges

R. Raffin (✉)
Aix-Marseille University, LSIS UMR 7296, Domaine universitaire de Saint Jérôme,
Av. Escadrille Normandie-Niemen, Marseille Cedex 13397, France
e-mail: romain.raffin@lsis.org

M. González Hidalgo et al. (eds.), *Deformation Models*, Lecture Notes in Computational Vision and Biomechanics 7, DOI: 10.1007/978-94-007-5446-1_2,

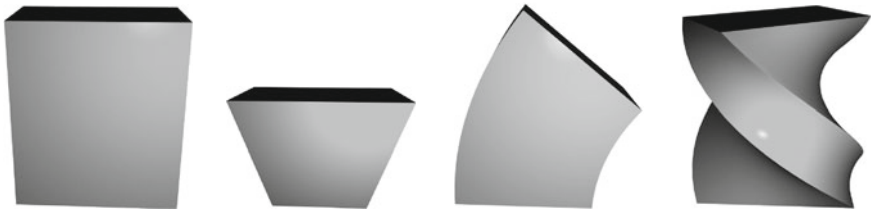


Fig. 1 Barr [2] global deformations. **a** Initial object; **b** taper; **c** bend; **d** twist

and faces (in a B-rep model) providing links between vertices. The deformation functions are described by matrices, as transformations (rotate, scale, translate) and the method allows composition and application of a stack of deformations, compliantly with a CSG modeling environment.

As Barr methods are made to be applied globally, locality of a deformation is obtained by the developer with conditional instructions, as this is not described by the transformation matrices.

2 FFD: Free Form Deformation

The method created by Sederberg and Parry [37] tends to give a virtual sculpture approach. In a global view the method considers the object to be deformed embedded in a parallelepipedical grid (see Fig. 2). Once the local coordinates of all vertices of the object (the teapot in Fig. 2) are expressed in the grid frame, pulling a grid point transmits the deformation to the initial object by a simple re-expression of the coordinates of the object in the world frame.

Authors described (in words) the process of FFD:

> *A good physical analogy for FFD is to consider a parallelpiped of clear, flexible plastic in which is embedded an object, or several objects, which we wish to deform. The object is imagined to also be flexible, so that it deforms along with the plastic that surrounds it.*

More formally expressed, a FFD is a $\mathbb{R}^3 \rightarrow \mathbb{R}^3$ application using a trivariate product of Bernstein polynomials. First step is to define object vertices coordinates in grid space. It needs an origin X_0 and frame vectors: $\overrightarrow{S}$, $\overrightarrow{T}$ and $\overrightarrow{U}$ (in 3D space but 2D or 4D adaptation can be made trivially).

Figure 3a shows the local frame and expression of local object coordinates (s, t and u) of a point X, following:

$$\overrightarrow{X_0X} = s\,\overrightarrow{S} + t\,\overrightarrow{T} + u\,\overrightarrow{U}$$

A way to compute (s, t, u) values is:

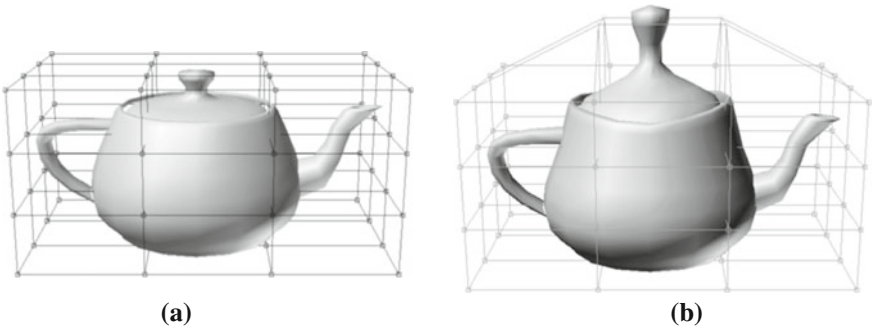


Fig. 2 FFD deformation of a teapot. **a** Initial object and surrounding FFD mesh; **b** changes in mesh configuration involve object deformation

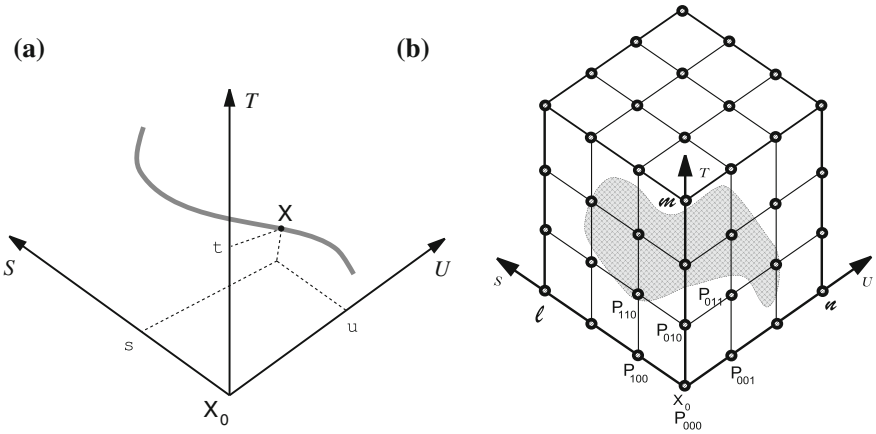


Fig. 3 FFD schema. **a** FFD local associated frame; **b** FFD embedding mesh construction

$$s = \frac{\overrightarrow{T} \wedge \overrightarrow{U} \bullet \overrightarrow{X_0X}}{\overrightarrow{T} \wedge \overrightarrow{U} \bullet \overrightarrow{S}} \quad t = \frac{\overrightarrow{S} \wedge \overrightarrow{U} \bullet \overrightarrow{X_0X}}{\overrightarrow{S} \wedge \overrightarrow{U} \bullet \overrightarrow{T}} \quad u = \frac{\overrightarrow{S} \wedge \overrightarrow{T} \bullet \overrightarrow{X_0X}}{\overrightarrow{S} \wedge \overrightarrow{T} \bullet \overrightarrow{U}}$$

If a point X lies in the grid it verifies: $0 < s < 1, 0 < t < 1$ and $0 < u < 1$. The grid is cut in l, m, n parts according to $\overrightarrow{S}$, $\overrightarrow{T}$ and $\overrightarrow{U}$ dimensions respectively. In the example of Fig. 3b, we defined a uniform $l = m = n = 3$ grid.

Each vertex P_{ijk} of the embedding grid is associated to a control point of a parametric volume. In the frame $(X_0, \overrightarrow{S}, \overrightarrow{T}, \overrightarrow{U})$, their locations are given by:

$$\overrightarrow{X_0P_{ijk}} = \frac{i}{l}\overrightarrow{S} + \frac{j}{m}\overrightarrow{T} + \frac{k}{n}\overrightarrow{U} \text{ with } i \in [0 \ldots l],\ j \in [0 \ldots m],\ k \in [0 \ldots n]$$

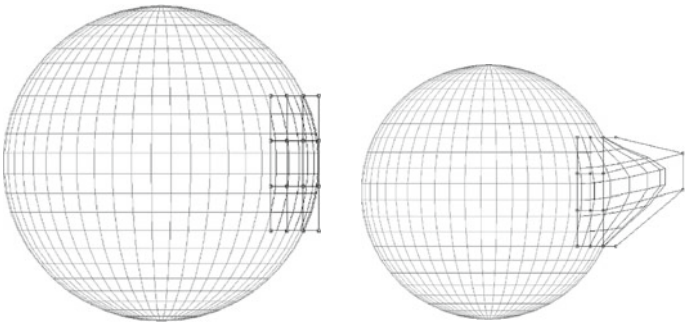


Fig. 4 Local FFD. **a** Initial object and local range grid; **b** deformation by a local grid

We can express the position of X in the grid frame (denoted X_{ffd}) with these control points, and weights given by Bernstein polynomial values:

$$X_{ffd} = \sum_{i=0}^{l} \binom{l}{i} (1-s)^{l-i} s^i \left[\sum_{j=0}^{m} \binom{m}{j} (1-t)^{m-j} t^j \left[\sum_{k=0}^{n} \binom{n}{k} (1-u)^{n-k} u^k P_{ijk} \right] \right] \tag{1}$$

Computing local coordinates of each point of the initial object is made with formulae (1) and attach the object's points to control points of the grid. Local influence of these control points is underlaid in the Bernstein polynomial weights. The more a control point is close from an object vertex, the more it weighs in local coordinates expression, and the more its displacement acts on point deformation. Conversely control points influence all object vertices: even if an extreme control point is displaced, the entire deformed object must be recomputed (an analogy is evident with Bézier curve control points).

Initially the FFD method does not describe local deformation of an object, but authors propose to use a restricted grid on the local part to be deformed (as illustrated at Fig. 4). Continuity between unmoved and deformed parts is not defined (kept as initially), as the deformation continuity exists only in embedding grid. Attention must be paid to not create stretched faces by displacing the whole local grid far from the object, or twisting it until self-intersection.

2.1 FFD *Properties*

FFD method is interesting because:

1. It conserves the object visual aspect. Even though its topology is not managed by the deformation, it is not possible to create or remove holes, even if inverting exterior and interior can be possible.

2. It mainly uses polyhedral B-rep object but one can use parametric object as input to increase potential resulting shapes. If this object obey to control network displacement, deforming such an object consists in deforming the control points net. As it deforms a control net of an approximating parametric surface, the deformation effect is then nothing but user-friendly.
3. An information on the volume of the deformed part of the object can be computed studying the jacobian of the FFD [37] and volume-preserving methods have been published [17, 35].

To conclude, the FFD method is a rapid and user-friendly method to deform an object. Locality can be obtained with a restricted embedding grid. As the method is easy to implement, it is mainly used in a wide range of applications. An important drawback is the parallelepiped shape of the grid with an influence zone of a deformation difficult to place, to localize on the initial object or to manipulate finely. Starting from this, another methods have been implemented, modifying some steps or tools of the process (grids shapes, coordinates functions, objects that can be deformed).

2.2 *Extensions and Classifications of* FFD

The initial method proposed par Sederberg [37] initiate a lot of extensions. In this section, we propose to list some of them sorted by geometry dimension as these methods lie on point "grid", curves or volumes (like the initial FFD).

2.3 *0-Dimensional Methods*

First dimension listed, it involves the deformation tool to be a single point. Vertices of the initial object are expressed regarding this frame by a simple distance value (mostly euclidean). The methods of [3, 4, 18, 36] or [29] can be cited, an other definition of local coordinates can be found in [30]. We will develop some of these methods later in the document. Hsu et al. [18] propose a direct manipulation of the initial mesh, hiding to the user the complexity of the embedding grid. The latter exists but the model interacts with the grid diffusing the initial object point displacement to the grid and computing the resulting deformation on the entire object. B-splines basis functions are used to ensure local action and continuity. Coordinates of a point in space are obtained by:

$$q_{i,j,k}(s,t,u) = \sum_{l,m,n=-3}^{0} P_{i+l,j+m,k+n} B_l(s) B_m(t) B_n(u) \tag{2}$$

As previously seen, local coordinates (s, t, u) are computed for each object point. For one single object point displacement with relative distances it moves the grid

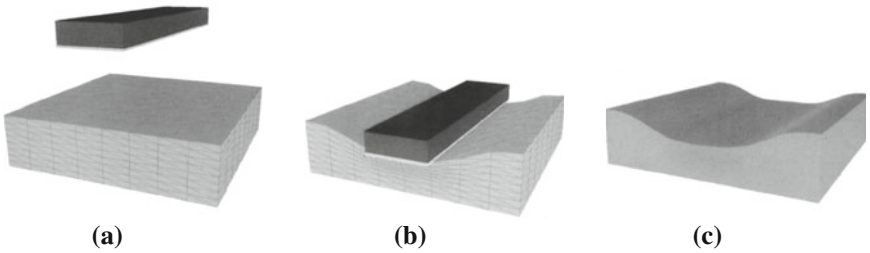


Fig. 5 FFD direct manipulation [18]. **a** Initial object and deformation profile; **b** profile placement; **c** resulting deformation

of control points solving a least-square problem (LSQ). Denoting B the B-spline coefficients matrix and P the control points matrix, this can be written as:

$$q = BP$$

q_{new} new location of point q is given by (ΔP contains control point displacement):

$$\begin{aligned} q_{new} &= B(P + \Delta P) \\ &= BP + B\Delta P \\ &= q + \Delta q \quad \text{with} \quad \Delta q = B\Delta P \end{aligned}$$

Then, finding ΔP is done by:

$$\begin{aligned} \Delta P &= B^{-1}\Delta q \\ &\equiv B^{+}\Delta q \end{aligned}$$

B^{+} is the pseudo-inverse matrix of B, obtained with LSQ method. It may occurs that the resulting control points configuration obtained decrease continuity by repeating control points, authors add a matrix characterizing control points self-dependence that solves this problem.

The preceding method works well if one move two object points that not share the same control point. In the contrary, solving the constraint equations can not be done, and a way to overcome this problem is to refine the initial mesh. It involves more B-splines description but permits to separate their influence zones. Figure 5 shows a deformation obtained by direct manipulation of object points. A tool (profile) is put on the object, once pushed it displaces some object points, updates grid points location and results in the object deformation.

An other method: DFFD [9] stands for *Dirichlet based Free-Form Deformation*. As the preceding method it hides the embedding grid, using Sibson coordinates [13] to control the weights of points. The grid is based on a set of points and influence zones obtained automatically by Voronoï diagram of these points. Once done, the

other points of the objects are inserted in this spatial arrangement to compute their locale coordinates. Computing the difference between the two Voronoï configurations before and after point deformation permits to diffuse deformation to regions (sets of object points). This method is more local than the classical FFD as a point acts on its Delaunay-neighbors and support easily multiresolution objects. It has been applied to hand movements in virtual environments [20, 30].

2.4 Dimension 1: Curves

Curves can define deformation tools, local parametrization is obtained by distances computations (so it can raise problems if points are projected on the same site on the curve). Two principal methods are detailed here: AXDF [26] and WIRES [38]. AXDF [26] main idea is to express local coordinates according a curvilinear axis. Apart of this, the process is merely the same:

1. creation of an axis passing through the object,
2. expression of object coordinates in axis ones with projection from $\mathbb{R}^3$ space to $\mathbb{R}$ curve parameter,
3. axis transformation,
4. computation of object's vertices new locations.

End-user interacts with the deformation by defining the axis and its modification. It remains mandatory to keep a well defined sliding frame, whose displacement along the axis is constrained (Frenet based [6, 7]) to avoid axis inversions (see Fig. 6).

To solve this continuity problem, authors use a minimization method [21]. One can retrieve in AXDF deformations the results of classical ones: rotation, torsion (see Fig. 6b, c). An interesting contribution of the method is that it defines an influence area that restricts deformation spatially and preserves continuity, as on Fig. 7c. Three zones are used with two radii R_{min} and R_{max}. In the first one (if distance from the axis is less than R_{min}) the deformation is maximal. From R_{min} to R_{max} influence decreases as the distance increases. Beyond R_{max} the deformation vanishes (Fig. 7a).

The WIRES [38] method is similar to the latter. It deforms a curve in another curve in space, with all the belonging geometries. Two parametric curves are defined by the user: $R(t)$ lies on the object and $W(t)$ figures the destination of $R(t)$. For each point P in space, if we can compute the distance $\|R(p_r) - P\|$ we can assign a deformation value at this point. Denoting p_r the parameter that correspond to the projection of P onto $R(t)$, we have: $\|R(p_r) - P\| = \min(\|R(t) - P\|)$ (see Fig. 9a). The value of p_r is obtained by solving the equation $R'(t) \cdot (R(t) - P) = 0$. The deformation manage the constraint linked to $R(t)$ according to a deformation function f.

An additional parameter s is introduced, allowing to "pinch" the deformation along its axis. It attracts or repulses the deformed points following their distances to $R(t)$ curve.

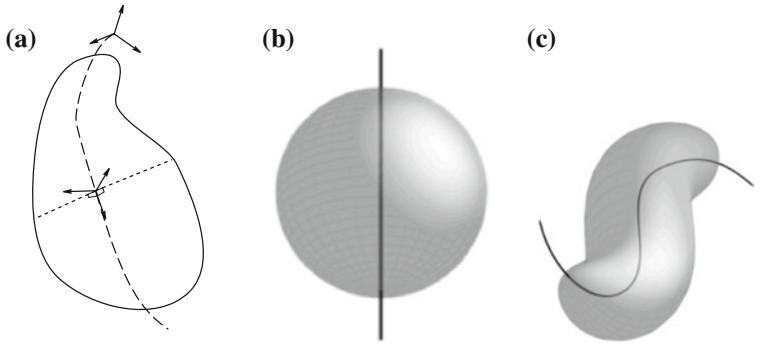


Fig. 6 AxDF construction [26]. **a** Example of construction; **b** object to be deformed and AxDF axis; **c** result of axis transformation

Fig. 7 Influences' radii of AxDF [26]. **a** Radii examples; **b** influence's radii projected on a plane; **c** effects of axis displacement

Curves shown at Fig. 8a–c illustrate the impact of different s values. The location of a deformed point P_s from its initial position P (see Fig. 9b) is now given by:

$$P_s = R(p_r) + (P - R(p_r)) \left(1 + (s-1) f \left(\frac{\|R(p_r) - P\|}{r}\right)\right)$$

In the preceding equation r is a radius that creates an isotropic influence around the curve $R(t)$. The path from $R(t)$ curve to $W(t)$ is discretized to a set of positions that preserves tangential continuity and assure the displacement constraint. Tangent $\overrightarrow{R'(p_r)}$ is collinear to $\overrightarrow{W'(p_r)}$ according to the f function. Translation $T_{RW} = (W(p_r) - R(p_r)) f \left(\frac{\|R(p_r)-P\|}{r}\right)$ is then applied. The P_{def} image of P by the deformation (see Fig. 9d) is obtained by:

$$P_{def} = P_r + (W(p_r) - R(P_r)).f \left(\frac{\|R(p_r) - P\|}{r}\right)$$

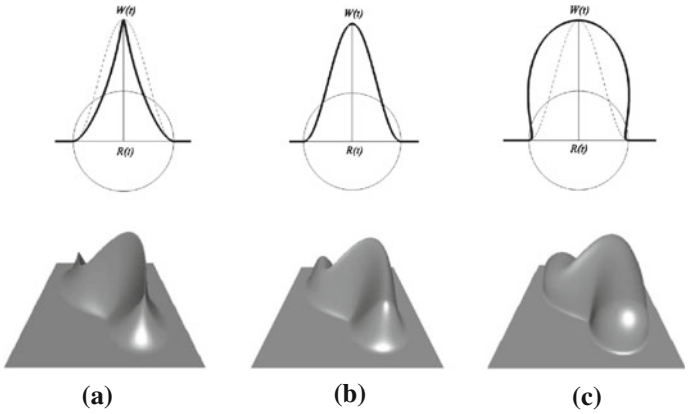


Fig. 8 WIRES deformations obtained by different values of s. **a** $s < 1$; **b** $s = 1$; **c** $s > 1$

P_r is an intermediate image of P_s by the rotations α and θ around the axis $R(p_r)$ and $\overrightarrow{R'(p_r)} \wedge \overrightarrow{W'(p_r)}$ (see Fig. 9c), where θ is the angle $\left(\overrightarrow{R'(p_r)}, \overrightarrow{W'(p_r)}\right)$ and $\alpha = f(P)$.

One can see that if $\overrightarrow{W'(p_r)} = \overrightarrow{R'(p_r)}$ the displacement is reduced to a single translation.

2.5 Dimension 2: Surfacic Deformations

In the third stage of free-form deformations we will study surfacic deformation tools [14, 22, 30]. We can include in this section the DFFD deformations [30] as they use a neighborhood with the edges joining each vertex and describe a discrete mesh. We develop herein the [14] method. If a parametric surface is used as a deformation tool, a parametric surface $S(u, v)$ must be defined and the projection of object's points on these surface gives the local coordinates in the tool space (as in AXDF for curves). The $H(u, v)$ surface ("height" surface) allows the modification of the distance between point P and initial surface $S(u, v)$ (see Fig. 10).

More formally, the deformation is expressed as follow:

- The two parametric surfaces that construct the deformation tool

$$S(u,v) = \sum_{i=0}^{m_s} \sum_{j=0}^{n_s} S_{ij} B_{i,k_{su}}(u) B_{j,k_{sv}}(v) \text{ and } H(u,v) = \sum_{i=0}^{m_h} \sum_{j=0}^{n_h} S_{ij} B_{i,k_{hu}}(u) B_{j,k_{hv}}(v)$$

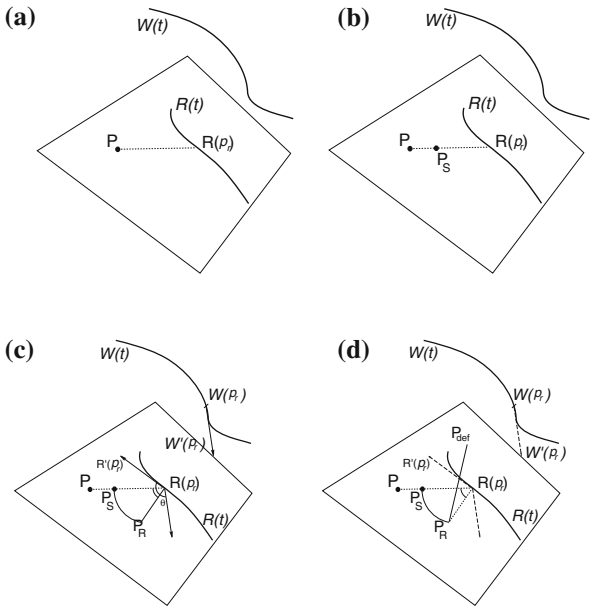


Fig. 9 WIRES deformation schema. **a** Construction of $R(p_r)$; **b** construction of P_s; **c** construction of P_R; **d** deformation

Initially, Feng et al.[14] use B-spline surfaces with node constraints to fix borders curves.

- $P(x, y, z)$ a point of the object to be deformed and $P'(x', y', z')$ the projection of P on $S(u, v)$. With the normal $\overrightarrow{N_S}$ of S at P', P is:

$$P = P' + h_p \overrightarrow{N}_S = S(u_p, v_p) + h_p \overrightarrow{N}_S(u_p, v_p)$$

where h_p is the distance from point P to surface $S(u, v)$. The parameters u_p and v_p are those of P' on S. If we denote P_{new} the new position of P after surface deformations of S and H in S_{new} and H_{new} respectively:

$$P_{new} = S_{new}(u_p, v_p) + H_{new}(u_p, v_p) h_p \overrightarrow{N}_{new}(u_p, v_p)$$

where $\overrightarrow{N}_{new}$ is the normal vector of S_{new}. Figure 10 shows construction and deformations process. This deformation is limited to vertical displacements of the surface H. Its extension to other displacements will increase rapidly the computational costs to minimize displacements.

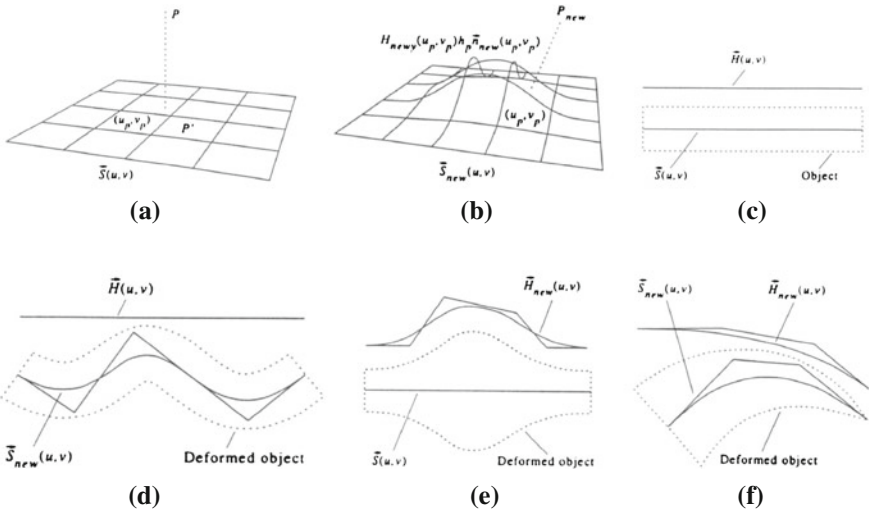


Fig. 10 Surfacic deformation [14]. **a** P' projection of P; **b** deforming S, obtaining P_{new}; **c** configuration of S and H; **d** S displacement; **e** H modification; **f** deformation by S and H

2.6 Dimension 3: Volumes

"Last" tool type (as we can increase easily space dimension of the embedding grid), volumic deformations were the initiators of free-form deformations. Most of works defines new volumes [16, 20, 23, 27], grid construction to define a wide range of tools [8], continuous definition of the deformation [1] or dynamic characterization to animate objects [12]. We will first describe the method of [8] which composes FFD meshes. We will also see [23] for a new embedding grid definition. Last, we will present a method combining implicit objects and deformations [10].

2.6.1 EFFD

Any intuitive it can be, the FFD method is disadvantaged by its simplicity. The construction of a complex object and the definition of a precise deformation are difficult with a parallelepiped embedding grid. As an example, the Fig. 11 shows a deformation applied on a sphere, the rectangular definition of the grid projection on the spherical surface produces a non-isotropic deformation (Fig. 11a, b). Conversely, a cylindrical grid definition would override this problem (see Fig. 11c, d).

EFFD is an extension of FFD (*Extended Free-Form Deformation*), described by Coquillart [8]. The deformation process is kept but it can use non-prismatic grids (see Figs. 11, 12) or a combination of grids. The usable meshes are more interesting for the final user, but their complexity can lead challenges to manipulate them. As the grid manipulate the embedded object, if some details are needed one can increase

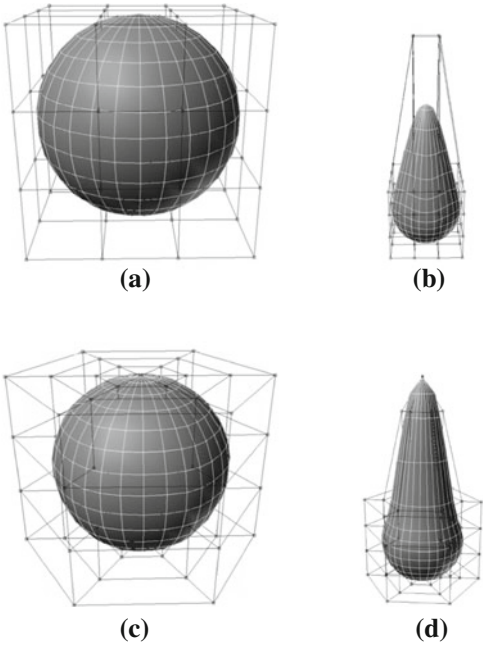


Fig. 11 Isotropy failure in FFD. **a** FFD parallepiped grid; **b** deformation result of this FFD; **c** cylindrical grid; **d** deformation result (EFFD)

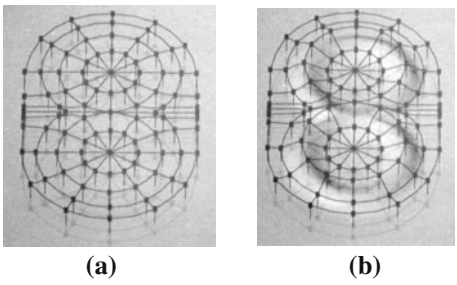
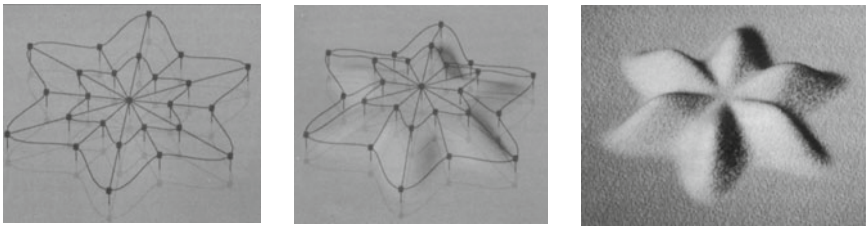


Fig. 12 Non-prismatic grids combination [8]. **a** EFFD mesh building; **b** resulting deformation

locally the grid density, but as joints between grids also manage continuity, a balance must be found (it also increases computational cost).

A hard part of the EFFD process is to express the global coordinates of the embedded object in the grid. It requires to found the chunk an object point lies in and to obtain the local coordinates in the grid volume. Fortunately, the grid is defined by Bézier tensor products and it is possible to use the control networks convex hulls. Once the chunk is found authors use a Newton approximation to compute coordinates in this local grid.



(a) **(b)** **(c)**

Fig. 13 EFFD deformation example [8]. **a** Initial mesh; **b** grid modification; **c** resulting object

This method is a great improvement of initial FFD. It brings intuitive control, various grid shapes and grids combinations that serve the user wishes. Figure 13 shows the ease of construction of a star obtained from a deformation applied on a plane.

2.6.2 NFFD

To control locality of the deformation Lamousin and Waggenspack [23] propose to use NURBS instead of Bézier volumes (NFFD method). The embedding grid is no longer divided in a uniform way but can be refined in areas where the deformation is important and kept rough where the user want only to conserve the initial object shape. The process is based again on FFD, where the local expression of grid coordinates is the only module that is modified. If the grid mesh $V_{i,j,k} = (x_{i,j,k}, y_{i,j,k}, z_{i,j,k})$ is used, a weight $W_{i,j,k}$ is given to each vertex (initially 1). With u, v and w the axis of the local coordinate system, and a, b and c the divisions of these axis, it gives $(a+1) \times (b+1) \times (c+1)$ grid points that embed the initial object. B-splines basis functions follow the rules:

$$\begin{aligned} 2 \leq p \leq a+1 \\ 2 \leq m \leq b+1 \\ 2 \leq n \leq c+1 \end{aligned}$$

Nodal vectors are expressed for each coordinates by:

$$\begin{aligned} U &= (u_0, \ldots, u_q) \text{ with } q = a + 2(p-1) \\ V &= (v_0, \ldots, v_r) \text{ with } r = b + 2(m-1) \\ W &= (w_0, \ldots, w_s) \text{ with } s = c + 2(n-1) \end{aligned}$$

These vectors are non-uniforms and the nodes multiplicity is equal to basis functions order at extremities to ensure borders interpolation. As in EFFD method, a minimization is necessary to obtain, once the cell of the Nurbs volume is determined, the local coordinates of every object vertices.

2.6.3 IFFD

This method is described in [11] and is based on the use of implicit volumes instead of polyhedral ones for the embedding grid (in addition a more global method can be found in [19], manipulating scalar fields). Here again, the main difficulty is raised by the local coordinates computation, as the deformation tool can be defined with $n+1$ objects $P_0, \ldots, P_n$ with for each P_i:

- a local coordinates system and its associated function $\phi_i : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ that gives for every point M in global space its coordinates $\widetilde{M}_i$ in P_i by $\widetilde{M}_i = \phi_i(M)$. This function is reversible and bijective to ensure the return from local to global coordinates $M_i = \phi_i^{-1}(\widetilde{M}_i)$,
- a transformation function $\Delta_i : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ that permits, in local space, to displace $\widetilde{M}_i$ to $\widetilde{M}'_i$,
- a density function $F_i : \mathbb{R}^3 \rightarrow \mathbb{R}^+$ that gives the influence of the implicit primitive. It is a decreasing function with finite support.

The process is the same as in FFD:

$$M \begin{bmatrix} x \\ y \\ z \end{bmatrix} \underset{\text{Freezing}}{\Longrightarrow} \widetilde{M} \begin{pmatrix} \widetilde{M}_0 \\ \vdots \\ \widetilde{M}_n \end{pmatrix} \underset{\text{Deforming}}{\Longrightarrow} \widetilde{M}' \begin{pmatrix} \widetilde{M}'_0 \\ \vdots \\ \widetilde{M}'_n \end{pmatrix} \underset{\text{Combining}}{\Longrightarrow} M' \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

Influence functions balance displacements according to the following two formulations:

- Globally,

$$M' = M + \frac{\sum_{i=0}^{n} F_i(M)(M'_i - M)}{\sum_{i=0}^{n} F_i(M)}$$

- Locally, a function $\Gamma_i : \mathbb{R} \rightarrow [0, 1]$ is added to implement deformation range. The blending function f_i is based on [10]:

$$f_i(t) = \begin{cases} \frac{3-4t}{2} & 0 < t \leq \frac{1}{2} \\ 2(1-t)^2 & \frac{1}{2} < t \leq 1 \\ 0 & \text{else} \end{cases}$$

with:

$$M' = M + \frac{\sum_{i=0}^{n} \Gamma_i\left(\frac{3}{2} - F(M)\right) F_i(M)(M'_i - M)}{F(M)}$$

where $F(M) = \sum_{i=0}^{n} F_i(M)$

Images of Fig. 14 show some deformations that can be obtained with IFFD.

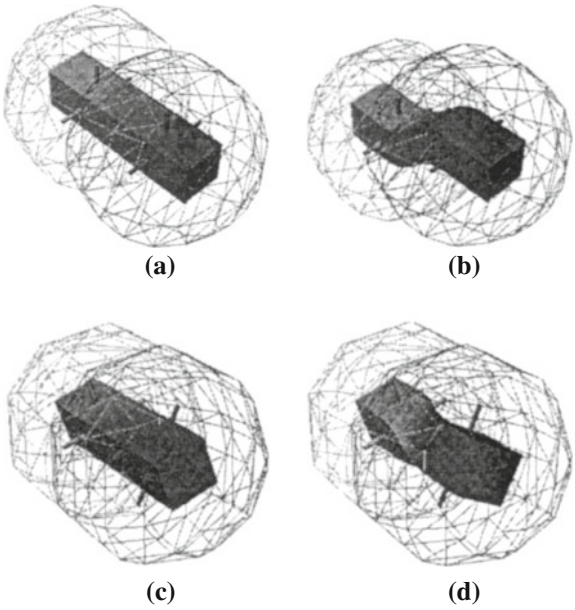


Fig. 14 IFFD: implicit free-form deformation examples [10]. **a** Grid translation; **b** translation result; **c** grid rotation; **d** rotation result

2.7 FFD *Conclusion*

This short survey of FFD methods declination shows that these methods are intuitive, quite easy to implement and easy to use. The complexity of the inner object to be deformed is hidden by the embedding grid. A drawback is due to user needs to go further in the definition of complex grids. Since the grids definitions must ensure that every vertex coordinates of the initial object can be expressed in the grid space, the algorithmic complexity increases as costs of computational process and can raise singularity situation. This has the opposite effect to that intended: the user can not understand this complexity and leaves the method. Another drawback is however the lack of constrained deformations (by positions, continuity, etc). We propose to study in the next section free-form deformations that permits this constraints description.

3 *n*D-Deformation

The previous methods we have compared have in common a description of a deformation but without displacement constraint satisfaction, this prevents the user to easily place objects relatively. Considering the numerous variations based on FFD methods and the hardness to overcome polyhedral grids in 2 or 3 dimensions, Bechmann [3]

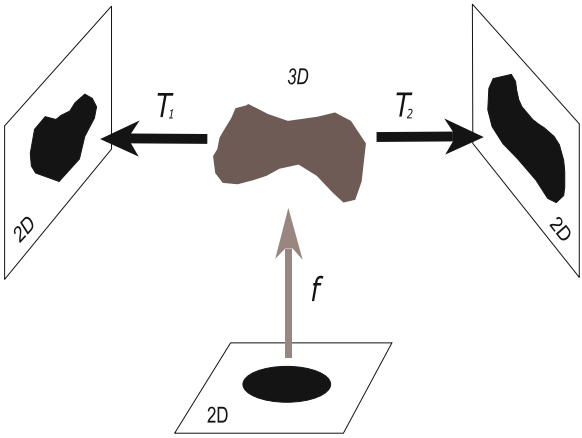


Fig. 15 nD-deformation schema for a 2D deformation

creates a more generic deformation model, named "nD-deformation" that express the deformation whatever the space dimensions. In this system, deformations are considered as constraints satisfactions, initial object is embedded in a space that permits to solve deformation constraints and then reprojected in three or four dimensions (with 4D deformations one can construct animations by deformation of space-time). Figure 15 presents the main principle of nD-deformation. An initial object is embedded in a space where more freedom-degrees exists (an upper dimensional space), in this space constraints solving is made easier and can be processed. Once achieved, the object is projected in a 3D space, eventually by different projection methods that result, for the same constraints solving system, in a set of matching resulting objects.

In the Fig. 15 the deformation is applied on object vertices, and is mathematically described by a polynomial function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$, with $n = 2$ the initial space dimensions and $m = 3$ the embedding one. There is m degrees of freedom to solve the deformation's constraints and f defines locality controls on the deformation. $T : \mathbb{R}^m \rightarrow \mathbb{R}^n$ is a linear inverse projection, from the upper space $\mathbb{R}^m$ to the initial one. The projection matrix corresponding to T is computed using pseudo-inverse methods to satisfy constraints or least-squares ones to give approximate deformation. If the space $\mathbb{R}^m$ is of sufficient high-order, user can also provides attracting or repulsing constraints.

Formally, if we denote:

- $Q(x_1, x_2, x_3, \ldots, x_n)$ a point in space $\mathbb{R}^n$,
- $\Delta Q(\Delta x_1, \Delta x_2, \Delta x_3, \ldots, \Delta x_n)$ the displacement values of Q, with

$$\Delta Q = d(Q) = (d(x_1), d(x_2), d(x_3), \ldots, d(x_n))^T$$

If M is the projection matrix associated to T, the deformation can be written as:

$$d(Q) = M.f(Q) \tag{3}$$

The deformation process is:

1. User gives r constraints $C_i, i \in [1, r]$ in space $\mathbb{R}^n$ and their new positions. Authors denote $d(C_i) = [d_1(C_i), d_2(C_i), \ldots, d_n(C_i)]^T$ the coordinates variation.
2. Matrix M is obtained by solving a system of $n \times r$ linear equations (each constraint raises n equations):
$$d(C_i) = M.f(C_i) \quad \forall i \in [1, r] \tag{4}$$
3. Once M computed, the displacement $d(Q)$ of a point Q in the initial space is given by:
$$d(Q) = M.f(Q)$$

If we consider each line M_j of M separately (with j a coordinate in $\mathbb{R}^n$), we can solve the lines of M independently:

$$d_j(C_i) = M_j.f(C_i) = f^T(C_i).M_j^T, \quad \forall i \in [1, r] \text{ and } j \in [1, n]$$

For the j th coordinate, the set of constraints gives:

$$D_j = \begin{bmatrix} d_j(C_1) \\ d_j(C_2) \\ \vdots \\ d_j(C_r) \end{bmatrix} = \begin{bmatrix} f^T(C_1) \\ f^T(C_2) \\ \vdots \\ f^T(C_r) \end{bmatrix} . M_j^T = X \cdot M_j^T \tag{5}$$

with:

$$X = \begin{bmatrix} f_1(C_1) \ldots f_m(C_1) \\ \vdots \\ f_1(C_r) \ldots f_m(C_r) \end{bmatrix}$$

We can now aggregate these results to construct the matrix M:

$$M_j^T = X^{-1} \begin{bmatrix} d_j(C_1) \\ d_j(C_2) \\ \vdots \\ d_j(C_r) \end{bmatrix}$$

One difficulty is to find the values for inverse matrix X^{-1}, because of its size $r \times m$. This matrix is generally non-invertible. This led three cases:

- If $m > r$ there is more unknown factors than equations, an infinity of solutions can be computed whatever the rank of matrix X, each of them satisfying the deformation constraints,
- If $m = r$ and rank of X is r, there is only one solution, the set of constraints defines fully the deformation,

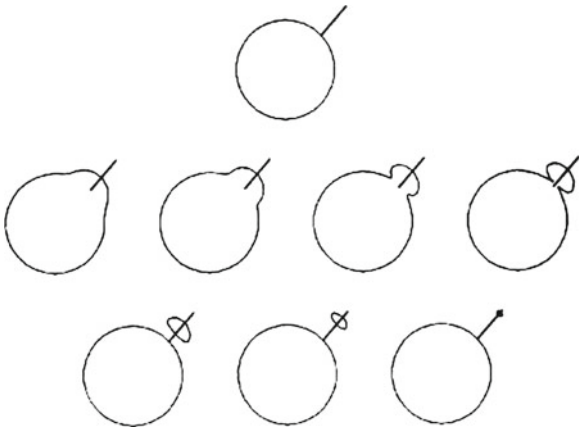


Fig. 16 4D deformation [3] of a *circle*

- If $m < r$ no solution can be found, one can use approximation method to find a "best-satisfying" solution according to the constraints.

The preceding list shows that it is necessary to use a value of m that is greater than (or at least equal to) the constraints number. Authors [3] use a pseudo-inverse matrix and a numerical method [5].

The initial 3D geometrical space can be extended to a 4D one that describe space-time deformations and animations. Figure 16 shows a deformation of a circle defined at initial time (up) and solved at end time (right down). Intermediate times explores the set of transitional shapes from the initial one to the deformed one.

The function f is chosen to preserve deformed (and resulting) object continuity. To obtain independent columns r of X, the f components must also be composed of non-linear combinations of the n coordinates. They can be point-oriented with m sub-functions, or axis-oriented with a tensor product of n vectors, or a mix between these two definitions. Basis formulation of f with m sub-functions is:

$$f(Q) = (f_1(Q), f_2(Q), \ldots, f_m(Q))$$

The f_i functions are defined as $f_i : \mathbb{R}^n \to \mathbb{R}$ for $i \in [1, m]$. If we only use points as constraints, the deformation is defined in $\mathbb{R}^n$. Each component f_i of f is a product of n values:

$$f_i(Q) = f_i^1(x_1) f_i^2(x_2) \ldots f_i^n(x_n)$$

With $f_i^j : \mathbb{R} \to \mathbb{R}$ for $i \in [1, m]$ and $j \in [1, n]$. We can then write f as:

$$f(Q) = f^1(x_1) \ldots f^n(x_n) \text{ with } f^j = \left(f_1^j(x_j) \ldots f_m^j(x_j)\right)$$

Each function can be managed separately and gives availability to specialize the deformation on a specific axis, for example to favor a direction or a temporal axis. f would be a tensor product of functions $f^k, k \in [1, n]$ defined from $\mathbb{R}^{p_k}$ into u_k if:

$$\forall U \in \mathbb{R}^n, \quad f(U) = \bigotimes_{k=1}^{n} f^k(u_k) \text{ with } f^k : \mathbb{R} \to \mathbb{R}^{p_k}$$

The dimension of the deformed object is then $m = \prod_{k=1}^{n} p_k$ and we could compute the component of $f(U)$ by:

$$\forall j \in [1, m],\ f_j(U) = \prod_{k=1}^{n} f^k_{s(j,k)}(u_k) \text{ with } 1 \leq s(j,k) \leq p_k$$

If $m = r$ and B-spline basis functions are used for f_i, we obtain the method SCODEF developed by [4] and presented hereafter.

3.1 SCODEF

Basing their works on the preceding method, Borrel [4] introduces a radius of influence for each constraint: it is the SCODEF model (*Simple Constrained Deformation*). As previously seen, it permits to fix the dimension value of $\mathbb{R}^m$ as the constraints number, facilitating constraint solving and hiding the embedding process (even it is always done). In the SCODEF model defined from $\mathbb{R}^n$ to $\mathbb{R}^n$ we have:

- n the dimension of the space,
- r the number of constraints,
- C_i the initial position of point,
- D_i the point displacement in the deformed space,
- R_i the influence radius associated with the C_i constraint,
- f_i the deformation function linked to C_i.

The deformation is point-oriented, its formulation is the same than Eq. 3, it says that:

$$d(Q) = \sum_{i=1}^{r} M_i . f_i(Q) \tag{6}$$

This can be rewritten with the addition of spherical influences. Each f_i function gives the contribution of the ith constraint to the displacement $d(Q)$ of a point Q. It is a scalar function that depends on each C_i constraint with its radius R_i:

$$f_i(Q) = B_i \left(\frac{\| Q - C_i \|}{R_i} \right)$$

Where B_i is a B-spline basis function, centered at 0 and decreasing to value 0 in 1. If we denote $U_i(Q) = \|Q - C_i\|/R_i$ and $f_i(Q) = B_i(U_i(Q))$ we can consider $U_i(Q)$ as a local parametrization of Q coordinates in C_i constraint influence. The value $f(0) = 1$ ensure constraint satisfaction. In a similar vein, the zero value of the function after passing 1 cancel the deformation outside the range of the influence radius. As in the Eq. 5 we can write for the jth coordinate:

$$d_j(C_i) = D_{ij} = \begin{bmatrix} D_{1j} \\ D_{2j} \\ \vdots \\ D_{rj} \end{bmatrix} = \begin{bmatrix} f^T(C_1) \\ f^T(C_2) \\ \vdots \\ f^T(C_r) \end{bmatrix} . M_j^T = X \cdot M_j^T$$

Following the definition, a constraint C_i influences a point Q if the distance $\|C_i - Q\|$ is lower than the constraint radius R_i: $\|C_i - Q\| < R_i$. Three situations must be evaluated:

- First case, the constraints are disjoints, the influence hull of each constraint does not overlap an other one (see Fig. 17). Influence of each constraint is isotropic. This mean for the constraint C_i:

$$U_j(C_i) = \frac{\|C_i - C_j\|}{R_j} = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{if } i \neq j \end{cases}$$

  As $f_j(C_i) = f(U_j(C_i)) = \delta_{ij}$ this gives $f(C_i) = (0, \ldots, \underbrace{1}_{\text{rank } i}, \ldots, 0)^T$. As M_i is the ith vector of M, $D_i = d(C_i) = (M_1, \ldots, M_r).f(C_i) = M_i$. This column is also the displacement $d(C_i)$ of constraint C_i. Based on Eq. 6 we have:

$$d(Q) = \sum_{i=1}^{r} B_i \left(\frac{\|Q - C_i\|}{R_i} \right) D_i$$

  When the constraints are disjoints the displacement of a point Q is the weighted average of the displacements of the constraints it is under influence. This weight of a constraint C_i is inversely proportional to the distance $\|Q - C_i\|$,
- Second case, the constraints are disjoints but there is an overlapping influence. A point Q will be under control of all of these constraints if it lies inside the overlapping area. It does not compromise the computation of matrix M and can be treated by the first case,
- If two constraints self-influence each other, their radii and centers are overlapping, this situation raises vectors dependance problem for the matrix M (see Fig. 18). If we take an example with two constraints, and identical radius and deformation function, i.e. $R_1 = R_2$ and $f_1(C_2) = f_2(C_1) = \alpha$:

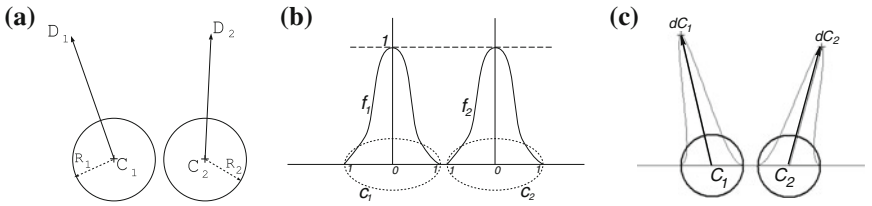


Fig. 17 Disjoints constraints in SCODEF method. **a** Constraints configuration; **b** showing associated functions; **c** deforming a line by these 2 constraints

$$(d(C_1) \quad d(C_2)) = M \begin{pmatrix} 1 & \alpha \\ \alpha & 1 \end{pmatrix}$$

It implies:

$$M_1 = \frac{1}{1-\alpha^2} d(C_1) - \frac{\alpha}{1-\alpha^2} d(C_2) \quad \text{and} \quad M_2 = \frac{-\alpha}{1-\alpha^2} d(C_1) + \frac{1}{1-\alpha^2} d(C_2)$$

In the worst case $d(C_1) = -d(C_2)$, that gives:

$$M_1 = -M_2 = \left(\frac{1}{1-\alpha} \right)$$

And finally the displacement of a point Q is given by:

$$d(Q) = \frac{d(C_1)}{1-\alpha} \left(f_1(Q) - f_2(Q) \right)$$

As much as the distance between the constraints decreases, i.e. $\alpha \to 1$, displacement vanishes, i.e. $d(Q) \to \infty$. The deformation raises a singularity, named as "Space-tearing" in [4].

To solve the "Space-tearing" problem, authors propose to add another influence radius for each constraint, and it looks like a constraint duplication (see Fig. 18d). The larger radius permits to define the constraint influence on space, the smaller one manage the co-influences between constraints. In the preceding example, if C_{1bis} is the new constraint, duplicated from C_1, we have $f_1(C_{1bis}) = 1$, $f_2(C_{1bis}) = \alpha$, $f_{1bis}(C_1) = 1$ and $f_{1bis}(C_2) = 0$, that gives for calculations of f:

$$f = \begin{pmatrix} 1 & \alpha & 1 \\ \alpha & 1 & 0 \\ 1 & \alpha & 1 \end{pmatrix}$$

This matrix can be inverted by the same method we see in [3], a large number of solutions can be found varying the smaller radius of influence.

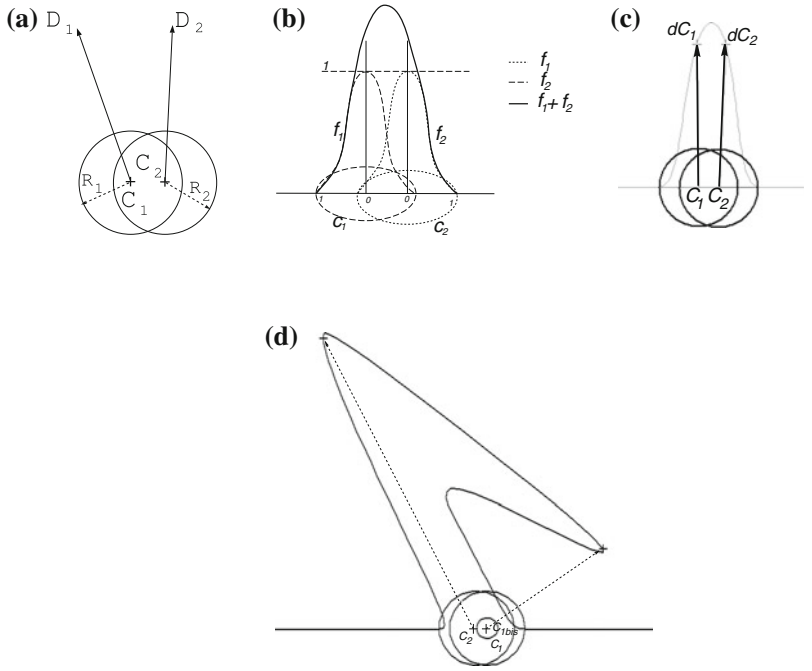


Fig. 18 Non-disjoints constraints. **a** Constraints; **b** Functions; **c** *line* deformation; **d** constraint duplication avoids SCODEF singularity

3.1.1 Extensions of the SCODEF Model

Extensions of the SCODEF model with geometric tools to handle the deformation have been made by [24, 32, 34] to form "Extended-SCODEF". Authors propose a generic expression of SCODEF model. They add new functions definitions, to vary the pinch of the object part deformed. As another extension of the deformation function, authors propose to consider the displacement to be curvilinear. The example of Fig. 19a presents a Bézier curve to construct the deformation path from the constraint point C to the displacement constraint $d(C)$.

The deformation is then expressed by:

$$T\left(d(Q)\right) = T\left(\sum_{i=1}^{r} M_i\, f_i(Q)\right)$$

Where T is a transformation of the deformation according to curve displacement. To prevent twisted curve authors use a sliding Frenet-frame along the displacement curve (see Fig. 19). This approach is quite near of sweeping techniques (see [39]).

They use this formal expression to complete influence hull's definition and to permit non-isotropic influences, Raffin et al. [32] use star-shaped shapes as a proof of concept. As the center of this shape is always defined and interior, the computations

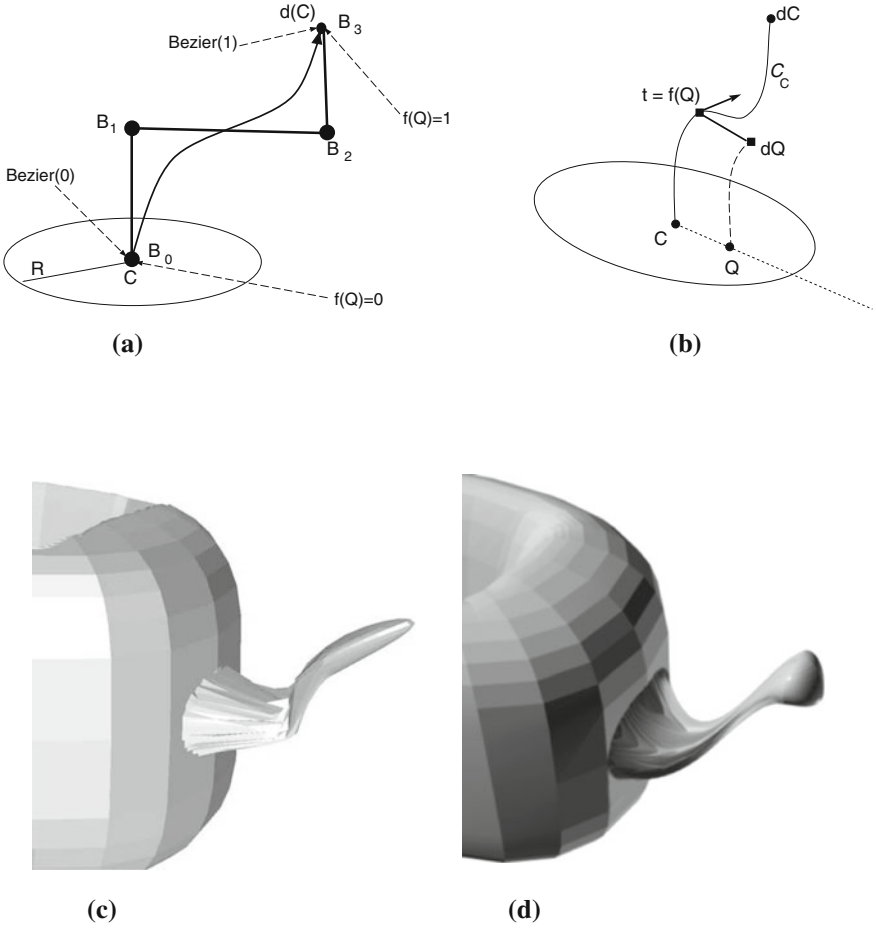


Fig. 19 Avoiding stretching in a curvilinear displacement. **a** Curvilinear displacement using a Bézier curve. $B_0 \ldots B_3$ are the control points of the parametric curve; **b** frenet sliding frame; **c** initial method; **d** frenet based curvilinear construction

of relatives distances is possible and extended-scodef can be applied. They also work with polyhedral or implicit hulls.

Another important add is the use of curvilinear constraints (see Fig. 21). As in WIRES's method, user can describe a curve to specify the trace of the deformation on the object's surface, taking into account of some neighboring points with influence hull, and associated displacement (or curvilinear path [25, 33]). This method is more user-friendly, as one can easily see the relation between deformation functions and deformed object pinch, influence hull with object's part deformed, etc.

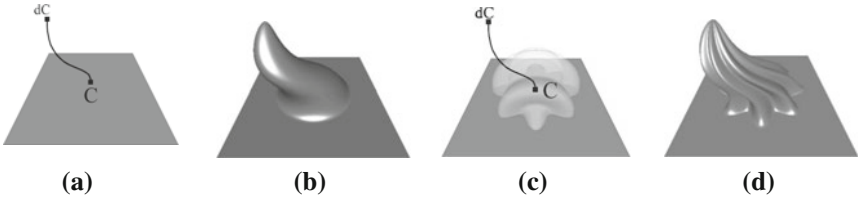


Fig. 20 Complex deformation with a non-isotropic hull. **a** Initial plane and constraint; **b** curvilinear displacement and *spherical* hull; **c** same displacement and *star-shaped* hull; **d** resulting deformed object

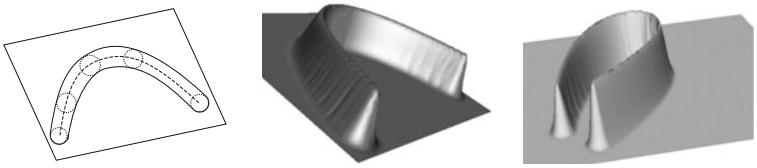


Fig. 21 Skeleton based constraint definition

4 Conclusion

All of the methods we have seen expect to deform objects of arbitrary topologies or geometries. We do not pretend to be exhaustive in this paper, as dozens of other methods and implementations exist. They are commonly used because they provide easy ways to control a deformation by giving access to a rough mesh that embeds the object and diffuse the grid displacement to some object's points lying in the same zone. This zone can be defined by parallelepiped mesh, spherical ones, compositions, vertices's distances, . . . and first initialized with the bounding box of the object. The user can handle a set of intuitive tools to control influence range of a constraint (and its shape), to describe a path of deformation the constraint will follow, or manage the pitch of the deformed part like he (or she) would do with plasticine. We saw deformations with freedom in deformed object and others with displacement constraint satisfaction. Nevertheless, the methods presented here are non reversibles and do not provide a way to recover the initial object from the deformed one (apart of saving initial object obviously). Some works have also been made with the construction of a deformation tree, that allow modifying, suppressing or moving a deformation, but in case of close but non-dependent constraints maintaining that kind of construction tree is not trivial (need to separate deformations influences).

Next things to be done on FFD related deformations is to take into account distance on surface (geodesic) or mesh. Free-form deformations of surfaces is still difficult as the user manipulate a grid that embeds the control points network of the surface. This gives two level of abstractions for the manipulation of the initial surface and its not "natural". As subdivision surfaces can express simultaneously discrete mesh and parametric surface in a single definition, [27] initiate free-form deformation on

surfaces. The main problem of singularity is persistent, as the deformation is based on constraints satisfaction with linear solving (to express vertex in local grid or to ensure displacements), if the constraints are close the system becomes unstable or its computational cost rapidly increases [15]. A very promising way for point-based methods, without location constraints, has started with "cage-deformations" methods.

References

1. Aubert F, Bechmann D (1997) Volume-preserving space deformation. Comput Graph 21(5):625–639 ISSN 0097–8493
2. Barr A (1984) Global and local deformations of solid primitives. Comput Graph 18(3):21–30
3. Borrel P, Bechmann D (1991) Deformation of n-dimensional objects. Int J Comput Geom Appl 1(4):427–453
4. Borrel P, Rappoport A (1994) Simple constrained deformations for geometric modeling and interactive design. ACM Trans Graph 13(2):137–155
5. Boullion T, Odell P (1971) Generalized inverse matrices. Wiley, New York
6. Coquillart S (1987) Computing offsets of B-spline curves. Comput Aided Des 19(6):305–309
7. Coquillart S (1987) A control-point-based sweeping technique. IEEE Comput Graph Appl 7(11):36–45
8. Coquillart S (1990) Extended free-form deformation: a sculpturing tool for 3D geometric modeling. Comput Graph 24(4):187–196
9. Coquillart S, Jancène P (1991) Animated free-form deformation: an interactive animation technique. Comput Graph 25(4):23–26
10. Crespin B (1999) Implicit free-form deformations. In: Proceedings of implicit surfaces, pp 17–23
11. Crespin B, Blanc C, Schlick C (1996) Implicit sweep objects. Comput Graph Forum 15(3):165–174
12. Faloutsos P, van de Panne M, Terzopoulos D (1997) Dynamic free-form deformations for animation synthesis. IEEE Trans Visual Comput Graph 3(3):201–214
13. Farin G (1990) Surface over dirichlet tessellations. Comput Aided Des 7:281–292
14. Feng J, Ma L, Peng Q (1996) A new free-form deformation through the control of parametric surfaces. Comput Graph 20(4):531–539
15. Gain J, Bechmann D (2008) A survey of spatial deformation from a user-centered perspective. ACM Trans Graph 27(107):1–21
16. Griessmair J, Purgathofer W (1989) Deformation of solids with trivariate B-splines. In: Hansmann W, Hopgood FRA, Strasser W (eds) Eurographics '89, pp 137–148
17. Hirota G, Maheshwari R, Lin MC (1999) Fast volume preserving free form deformation using multi-level optimization. In: Proceedings of the 5th ACM symposium on solid modeling and applications (SMA99), pp 234–245, ACM Press
18. Hsu W, Hughes J, Kaufman H (1992) Direct manipulation of free-form deformations. Comput Graph 26(2):177–184
19. Hua J, Qin H (2003) Free-form deformations via sketching and manipulating scalar fields. In: Proceedings of the eighth ACM symposium on solid modeling and applications (SPM), pp 328–333
20. Kalra P, Mangili A, Thalmann NM, Thalmann D (1992) Simulation of facial muscle actions based on rational free form deformations. Comput Graph Forum 11(3):C59–C69, C466
21. Klok F (1986) Two moving coordinate frames from sweeping along a 3D trajectory. Comput Aided Geom Des 3:217–229

22. Kobayashi K, Ootusubo K (2003) t-FFD: free form deformation by using triangular mesh. In: Proceedings of the ACM symposium on solid modeling and application, pp 226–234
23. Lamousin H, Waggenspack W (1994) Nurbs-based free-form deformations. IEEE Comput Graphics Appl 14(6):59–65
24. Lanquetin S, Raffin R, Neveu M (2006) Generalized scodef deformations on subdivision surfaces. In: Proceedings of 4th international conference articulated motion and deformable objects AMDO, LNCS 4069, pp 132–142
25. Lanquetin S, Raffin R, Neveu M (2010) Curvilinear constraints for free form deformations on subdivision surfaces. Math Comput Model 51(3–4):189–199
26. Lazarus F, Coquillart S, Jancène P (1994) Interactive axial deformations: an intuitive deformation technique. Comput Aided Des 26(8):607–613
27. MacCraken R, Joy K (1996) Free-form deformations with lattices of arbitrary topology. In: Proceedings of Siggraph'96 conference, Comput Graph, pp 181–188
28. Mäntylä M (1988) An introduction to solid modeling. Principles of computer science series. Computer Science Press, Rockville
29. McDonnell KT, Qin H (2007) PB-FFD: a point-based technique for free-form deformation. J Graph Tools 12(3):25–41
30. Moccozet L, Magnenat-Thalmann N (1997) Dirichlet free-form deformations and their application to hand simulation. In: Proceedings computer animation 97, IEEE Computer Society, pp 93–102
31. Parent R (1977) A system for sculpting 3D data. Comput Graph 11(2):138–147
32. Raffin R, Neveu M, Derdouri B (1998) Constrained deformation for geometric modeling and object reconstruction. In: 6th international conference in central europe on computer graphics and visualization (WSCG), vol 2, pp 299–306
33. Raffin R, Neveu M, Jaar F (1999) Curvilinear displacement of free-from based deformation. The Visual Computer, vol 16. Springer, pp 38–46
34. Raffin R, Neveu M, Jaar F (1999) Extended constrained deformations: a new sculpturing tool. In: Proceedings of international conference on shape modeling and applications, pp 219–224
35. Rappoport A, Sheffer A, Bercovier M (1996) Volume-preserving free-form solids. IEEE Trans Vis Comput Graph 2(1):19–27
36. Hu S, Zhang HCT, Sun J (2000) Direct manipulation of ffd. The Vis Comput 17:370–379
37. Sederberg T, Parry S (1986) Free-form deformation of solid geometrics models. ACM Trans Graph 20(4):151–160
38. Singh K, Fiume E (1998) Wires: a geometric deformation technique. In: Proceedings of Siggraph'98 conference, Comput Graph, pp 405–414
39. Yoon SH, Kim MS (2006) Sweep-based freeform deformations. Comput Graph Forum (Eurographics) 25(3):487–496

http://www.springer.com/978-94-007-5445-4

Deformation Models
Tracking, Animation and Applications
González Hidalgo, M.; Mir Torres, A.; Varona Gómez, J.
(Eds.)
2013, XIV, 297 p., Hardcover
ISBN: 978-94-007-5445-4