PB-FFD: A Point-based Technique for Free-Form Deformation

Kevin T. McDonnell
Dowling College

Hong Qin
Stony Brook University

Abstract

This paper presents PB-FFD, a novel, interactive, point-based technique for performing free-form deformation of polygonal meshes. First, a volumetric deformation space is defined as the linear combination of overlapping, ellipsoidal radial basis functions (EBFs) of compact support. Mesh vertices are then parameterized with respect to local coordinate frames centered over the origins of the EBFs. As in traditional FFD, the mesh vertices are displaced in response to changes in the control point positions. Hence, PB-FFD presents to the user an interface similar to that of traditional free-form deformation, but does not require that a deformation lattice be explicitly constructed. PB-FFD also supports multiresolution control, direct manipulation of meshes, automatic construction and refinement of the deformation space, among other benefits. The paper covers important implementation issues, discusses any special cases that may arise during use of the technique, and provides advice to practitioners who may wish to implement PB-FFD.

1 Introduction

This paper presents PB-FFD – point-based, free-form deformation – a new meshless, interactive FFD technique for multiresolution manipulation of polygonal meshes. In traditional FFD, a designer embeds a polygonal mesh in a 3D space and indirectly deforms the polygons by translating the vertices of the control mesh, thereby warping the embedding space. It is a very powerful and intuitive deformation technique, and has been incorporated into commercial 3D modeling and design software packages. In our PB-FFD technique (see Figure 1), we use ellipsoidal radial basis functions (EBFs) to define the deformation space and to serve as control points that can be manipulated to deform the embedded mesh. The major benefits of PB-FFD are: (i) the deformation space can have arbitrary geometry and topology; (ii) it does not demand that the user tediously construct and manage a complex deformation lattice; (iii) it is easy to implement; and (iv) it is very efficient. PB-FFD also supports multiresolution control, direct manipulation of meshes, automatic construction of the deformation space, curve-based deformations, and hierarchical refinement of the deformation space, among other benefits. Throughout the paper we discuss our experience in implementing and using PB-FFD, describe special cases and how to avoid them, and provide several examples of how PB-FFD works in practice. We show that our new deformation approach provides a very intuitive, point-based interface to the user that is efficient and also very easy to implement.

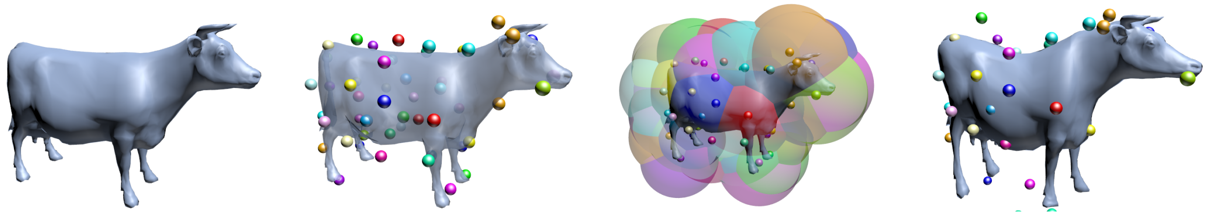

(a) (b) (c) (d)

Figure 1: Point-based free-form deformation. (a) Input cow mesh (courtesy of Viewpoint). (b) Control points that were automatically placed inside and outside the mesh. (c) Control point ROIs (zoomed out view). (d) Deformed mesh.

2 Overview of Space Deformation Techniques

Sederberg and Parry's [25] pioneering work on free-form deformation in the mid-1980s has inspired numerous extensions and improvements to FFD. For example, extended free-form deformations [9] and subdivision volume-based techniques [19] permit one to apply FFD over deformation lattices that closely match the geometry of an embedded mode. Such approaches require the construction of a potentially complex deformation lattice. This is one of the problems we have managed to solve with PB-FFD. Hsu and colleagues [12] developed approaches for directly displacing the mesh vertices, leaving it to a numerical solver to position the control points to effect the desired displacements. PB-FFD also provides a solution to this problem, but does not require a numerical solver. Other important developments in FFD techniques include dynamic free-form deformations [10], volume-preserving FFD [11], discontinuity-introducing deformations [24], sketch-driven FFD [14], and hierarchical free-form deformations [8].

PB-FFD also builds on techniques from point-based mesh manipulation, such as Scodefs [5]. Like Scodefs, our deformations are defined by a collection of unconnected points and facilitate deformation of objects. Our approach also has some aspects in common with the point-based Dirichlet Free-Form Deformation (DFFD) approach [20]. While DFFD also permits the use of arbitrarily distributed control points, (i) it relies on the explicit construction of a Delaunay triangulation; (ii) its control point regions of influence (ROIs) are defined explicitly by the triangulation; and (iii) it employs complicated, multivariate Bézier simplices to define the deformation space. In contrast, (i) our approach requires no Delaunay triangulation or Voronoi diagram; (ii) ROIs can be arbitrarily positioned and oriented; and (iii) simple basis functions are used. Furthermore, our approach facilitates hierarchical manipulation, automatic placement and simple addition/removal of control points, rapid reparameterization of embedded meshes, and other features. Botsch and Kobbelt [6] and Kojekine et al. [16] have also introduced frameworks based on radial basis functions (RBFs) for surface deformation. Whereas their approaches employ a set of RBFs as an effective vehicle for constraint imposition, our technique employs EBFs primarily to define and warp the deformation space via FFD. Noh et al. [21] showed how RBFs could be employed to automatically reproduce facial animations acquired from a video stream.

Last, there is a large body of work on 2D space warping techniques, many of which cast the shape manipulation process as the morphing of a source shape into a target shape. Virtually all these approaches make extensive use of interpolation, much as we use in PB-FFD. In contrast to these space warp approaches, however, PB-FFD employs interpolation to drive a process of space warping via control point manipulation. Generally speaking, 2D space warping techniques instead rely on constraint satisfaction methodologies to effect shape deformation. One of the seminal works in this area is that of Beier and Neely [4], who described a simple, yet effective, technique for image warping. It works by first establishing correspondences between source and target images. Then, a smooth image morphing is guaranteed by interpolating between the two images over a sequence of time-steps. Wolberg, Lee and colleagues [17, 18] have also published work in the 2D space warping area. Their 1995 paper that employs FFD for image warping [18] describes work that is closely related to ours, as it too uses a deformation function to manipulate a domain. Combined with a snake-based energy minimization technique, their approach provides a complete system for specifying the features to be warped and for controlling the speed of the warp. More recent work in image morphing includes that of Alexa et al. [1], Igarashi et al. [15], and Schaefer et al. [23]. All three classes of techniques seek to deform images in a manner that is "as-rigid-as-possible," to use Alexa's terminology [1]. Alexa's approach achieves rigid-like deformations by minimizing local distortions to the volumes of the simplices that comprise the deforming shape (i.e., triangles in 2D, tetrahedra in 3D). A similar approach was used by Igarashi and colleagues in developing an interactive shape deformation system that employs a SmartSkin touchpad. Schaefer's technique is founded on moving least squares (MLS) and provides an intuitive, point-based deformation approach that also seeks to generate realistic looking deformations.

3 Point-based Free-Form Deformation

In PB-FFD we define a volumetric deformation space as the linear combination of overlapping, ellipsoidal radial basis functions of compact support that we will collectively call the *control set*. Mesh vertices are parameterized with respect to local coordinate frames whose origins sit at the centers of the EBFs. As in traditional FFD, the mesh vertices are displaced in response to changes in the control point positions. In subsequent sections we describe how to parameterize polygonal meshes inside the deformation space, how to define the embedding space itself, and how to facilitate advanced interactive techniques like hierarchical manipulation and curve-based deformations.

3.1 Mesh Parameterization

In traditional FFD, each of the N vertices in the embedded model is assigned a parameter triple to parameterize it within the Bézier or B-spline's parametric space. In our approach, we assign to each vertex $\mathbf{v}_i$ a set of one or more parameterizations – i.e., one parameterization per basis function that influences the position of the vertex. We write $\mathbf{u}_{ij}$ to denote the parameterization of a vertex $\mathbf{v}_i$ with respect to the parametric domain of basis function ϕ_j that is associated with exactly one of the M control points $\mathbf{p}_j$. The origin of this reference frame is given as the position of the control point (i.e., $\mathbf{p}_j = (p_{jx}, p_{jy}, p_{jz})$) along with its three orthogonal basis vectors, $(\mathbf{a}_j, \mathbf{b}_j, \mathbf{c}_j)$ (where, for example, $\mathbf{a}_j = (a_{jx}, a_{jy}, a_{jz})$). These vectors form a vector space in which mesh vertices are parameterized. Specifically, we express an EBF as a scaled, rotated 3D Gaussian distribution, normalized by the value τ so that the basis function takes on a value of 1 at its center:

$$\phi(\mathbf{x}) = \frac{1}{\tau\sigma\sqrt{2\pi}} \exp(-\frac{1}{2\sigma^2}(\mathbf{x}-\boldsymbol{\mu})^\top \mathbf{R}\mathbf{S}^{-1}\mathbf{S}^{-1}\mathbf{R}^\top(\mathbf{x}-\boldsymbol{\mu})), \tag{1}$$

where the row vector $\mathbf{x} = [x\ y\ z\ 1]$ is a 3D point written in homogeneous coordinates, $\boldsymbol{\mu}$ is the center of the ellipsoid, τ is a normalization factor with the value $\tau = \frac{1}{\sigma\sqrt{2\pi}} \exp(0^2/2\sigma^2) = \frac{\sqrt{2}}{2\sigma\sqrt{\pi}}$, $\mathbf{S}$ is a scaling matrix that characterizes the ellipsoid's scaling factors (anisotropy) in the x, y and z directions, and

$$\mathbf{R} = \begin{bmatrix} a_x/|\mathbf{a}| & b_x/|\mathbf{b}| & c_x/|\mathbf{c}| & 0 \\ a_y/|\mathbf{a}| & b_y/|\mathbf{b}| & c_y/|\mathbf{c}| & 0 \\ a_z/|\mathbf{a}| & b_z/|\mathbf{b}| & c_z/|\mathbf{c}| & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

is a rotation matrix describing the orientation of the ellipsoid in terms of $(\mathbf{a}, \mathbf{b}, \mathbf{c})$. Note that we break with traditional EBF formulations by writing $\mathbf{x}$ in homogeneous coordinates. This simplifies our formulation and its subsequent implementation. We found in our experiments that a standard deviation of $\sigma = 3$ provided sufficiently local control over the deformation without causing buckling or creasing artifacts in the polygonal meshes. In theory, the rotation and scaling matrices could be utilized to create twisting and stretching effects, and it would interesting to explore these ideas as future work.

The parameterization $\mathbf{u}_{ij} = (\alpha_{ij}, \beta_{ij}, \gamma_{ij})$ for vertex i in the domain of EBF j is computed by projecting the vertex onto the reference frame associated with the EBF:

$$\mathbf{u}_{ij} = \left(\frac{(\mathbf{v}_i - \mathbf{p}_j)\cdot\mathbf{a}_j}{|\mathbf{a}_j|^2}, \frac{(\mathbf{v}_i - \mathbf{p}_j)\cdot\mathbf{b}_j}{|\mathbf{b}_j|^2}, \frac{(\mathbf{v}_i - \mathbf{p}_j)\cdot\mathbf{c}_j}{|\mathbf{c}_j|^2} \right). \tag{3}$$

Now we can define the x position of a vertex $\mathbf{v}_i = (v_{ix}, v_{iy}, v_{iz})$ in terms of the control points as

$$v_{ix} = \sum_{j=1}^{M} \widehat{\phi}_j(\mathbf{u}_{ij})(p_{jx} + \mathbf{u}_{ij}\cdot(a_{jx}, b_{jx}, c_{jx})) \tag{4}$$

where

$$\widehat{\phi}_j(\mathbf{u}_{ij}) = \phi_j(\mathbf{u}_{ij}) / \sum_{k=1}^{M} \phi_k(\mathbf{u}_{ik}) \tag{5}$$

and where the origin of the domain of each basis function is simply $[0\ 0\ 0]$. The equations for v_{iy} and v_{iz} can likewise be written. Intuitively speaking, $\widehat{\phi}_j(\mathbf{u}_{ij})$ provides the influence of control point j on vertex i, the p_{jx} term translates the x component of the vertex towards the control point, and the dot product $\mathbf{u}_{ij}\cdot(a_{jx}, b_{jx}, c_{jx})$ determines the x component of vertex i's position with respect to the reference frame of control point j.

Since $\mathbf{u}_{ij}$ and $\widehat{\phi}_j(\mathbf{u}_{ij})$ are constant as control points are moved, let us rewrite Equation 4 as

$$v_{ix} = \sum_{j=1}^{M} \widehat{\phi}_j(\mathbf{u}_{ij}) p_{jx} + \sum_{j=1}^{M} \left[\left(\widehat{\phi}_j(\mathbf{u}_{ij})\mathbf{u}_{ij} \right) \cdot (a_{jx}, b_{jx}, c_{jx}) \right]. \tag{6}$$

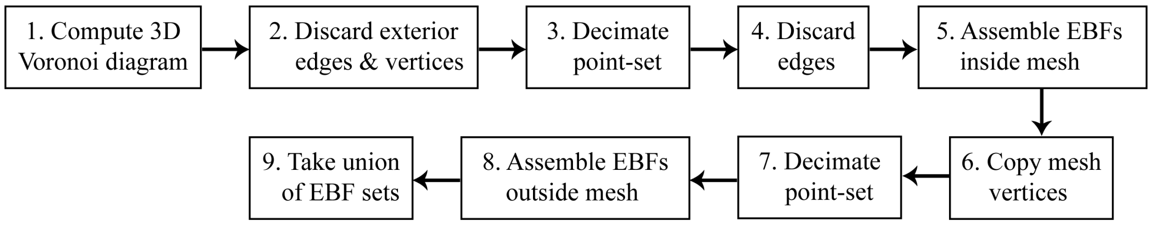
The analogous equations for y and z can likewise be rewritten. The (x, y, z) positions of all vertices can then be concisely written (and implemented) in matrix form as

$$\mathbf{V} = \mathbf{WP} + \mathbf{DA}, \tag{7}$$

where $\mathbf{V}$ and $\mathbf{P}$ are $N \times 3$ and $M \times 3$ matrices of vertex and control point positions, respectively. An entry $w_{ij} = \widehat{\phi}_j(\mathbf{u}_{ij})$ of the $N \times M$ matrix $\mathbf{W}$ specifies the influence that control point j exerts on vertex i and is computed during parameterization. The $N \times 3M$ matrix $\mathbf{D}$ serves to position the vertex with respect to the frame of reference of control point j. It can be written as three $N \times M$ sub-matrices $\mathbf{D} = \begin{bmatrix} \mathbf{E} & \mathbf{F} & \mathbf{G} \end{bmatrix}^{\top}$, where $E_{ij} = \widehat{\phi}_j(\mathbf{u}_{ij})\,\alpha_{ij}$, $F_{ij} = \widehat{\phi}_j(\mathbf{u}_{ij})\,\beta_{ij}$ and $G_{ij} = \widehat{\phi}_j(\mathbf{u}_{ij})\,\gamma_{ij}$. The $3M \times 3$ matrix $\mathbf{A}$ specifies the three orthogonal basis vectors of each coordinate frame, with all the $\mathbf{a}_j$ vectors preceding the $\mathbf{b}_j$ and $\mathbf{c}_j$ vectors. As the control points are repositioned by the user, the new mesh is computed by re-evaluating Equation 7.

3.2 Automatic Placement of Control Points

In traditional FFD, vertices are transformed indirectly by translating the control points, whose displacement causes deformation of the mesh. This functionality is supported in PB-FFD without modification and will not be discussed further. However, in traditional FFD it can be very tedious to place control points manually. Thus, we have devised an algorithm for automatically placing control points and for constructing a deformation space suitable for PB-FFD. The entire algorithm is summarized in Figures 2 and 3, with more details given in the following paragraphs.


1. Compute 3D Voronoi diagram
2. Discard exterior edges & vertices
3. Decimate point-set
4. Discard edges
5. Assemble EBFs inside mesh
6. Copy mesh vertices
7. Decimate point-set
8. Assemble EBFs outside mesh
9. Take union of EBF sets

1. Compute a 3D Voronoi diagram of the input mesh.
2. Discard all the Voronoi edges and vertices that lie outside the mesh.
3. Recursively decimate the remaining point-set using an octree until a user-specified number of points remains.
4. Discard the remaining edges.
5. Assign one EBF to each control point.
6. Create a second point set by copying the mesh's vertices and translating them by some fraction of the vertex normals.
7. Decimate this point-set until a user-specified number of points remains.
8. Assign one EBF to each control point.
9. Take the union of the two point-sets. This set defines the control points of the embedding space used for deforming polygonal meshes.

Figure 2: Algorithm for automatic placement of control points in PB-FFD.

3.2.1 The Algorithm

In the first phase (steps 1-5 in Figure 2) we assemble the control points lying in the interior of the mesh. This process provides a complete spatial decomposition for deforming the volume inside the model, much in the same way that traditional FFD lattices support this capability. This decomposition is achieved by computing an approximation of the medial axis of the input mesh [7] and by creating control points at or near the vertices of a Voronoi diagram (similar to the way they are used in surface reconstruction in [22]). We compute a 3D Voronoi diagram of the mesh and discard all edges and vertices lying outside the mesh. The discarding is done with the assistance of a kd-tree as follows. Given a Voronoi point (call it $\mathbf{q}$), we find the mesh vertex nearest to the point (call it $\mathbf{r}$ and its normal vector $\mathbf{n}$). If the

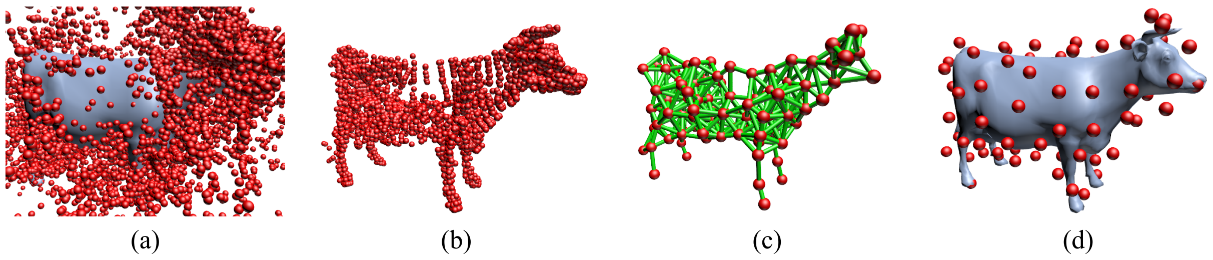


Figure 3: Major steps in our algorithm for automatic placement of control vertices of the deformation space. (a) Compute 3D Voronoi diagram of mesh (Step 1). (b) Discard all Vornoi edges and vertices that lie outside the mesh (Step 2). (c) Decimate point set and find an approximate skeleton of the mesh (Step 3). (d) Assemble EBFs inside and outside mesh to form the deformation space (Steps 4-9).

quantity $\mathbf{n} \cdot (\mathbf{q} - \mathbf{r}) \,/\, |\mathbf{q} - \mathbf{r}| > 0$ then we conclude $\mathbf{q}$ is outside the mesh and we discard it. Next, the point-set is recursively decimated using an octree. In particular, inside each octree cell we find the two closest vertices, remove them, and insert a new point lying at the mid-point of the two removed points. This is continued until the number of vertices in the cell has reached a predetermined maximum. Then, for each vertex we find the distance to its nearest neighbor and use a multiple of this value (e.g., 1.5 times) to define both the lengths of the axes of the reference frame for each control point, as well as the radius of the spherical basis function assigned to the point. We have found in our experiments that this heuristic provides a very good EBF coverage of the deformation space and virtually avoids all singularities.

In the second phase (steps 6-9 in Figure 2), we position control points lying outside the mesh by making a copy of the mesh vertices and translating each by a fraction of its normal vector (e.g., 10% in our implementation). By placing the points near the surface of the model, the system makes it easy for a designer to make large, local deformations rapidly. This point-set is then decimated until a user-specified number of vertices remains. The union of the two sets of vertices is taken and defines the control points of the deformation space. The mesh vertices can then be parameterized with respect to the control set.

3.2.2 Preventing Discontinuities in the Deformation Space

In traditional FFD, if the modeler is not careful when positioning the control points of the deformation lattice, it is possible to create a discontinuity at the mesh/lattice interface. This situation arises when the deformation lattice does not entirely enclose the mesh. The same phenomenon can occur with PB-FFD at the boundary of the ROI for a control point. If there exists a portion of the mesh not under the influence of any control point, a discontinuity will result, as can be seen in Figure 4. It is very easy to solve this problem. By inserting a control point with large ROI at the center of the 3D working space and by parameterizing all vertices with respect to that control point, we prevent discontinuities entirely. This control point – which we call the *master control point* and denote with $\mathbf{p}_1$ – is given a ROI sufficiently large that it includes all mesh vertices. We observed in our experiments that it was rare that a mesh vertex would be under the influence of no control points other than the master control point.

3.2.3 Dealing with Open Surfaces

The algorithm described in Section 3.2.1 for placing control points demands that the mesh is closed and has a clearly identifiable interior and exterior. It can be easily modified to handle open surfaces. Naturally, the first phase must be skipped since the surface has no discernible interior. During the second phase, we create two sets of control points. The first set is assembled by making a copy of the mesh vertices and translating each along the direction of its normal vector by some small, fixed distance. The second set is assembled by making a second copy of the mesh vertices and translating each along the direction of its *inverted* normal vector. As a consequence, a cloud of points is placed around the surface. Then, the union of these two sets can be taken and be decimated using the algorithm described earlier.

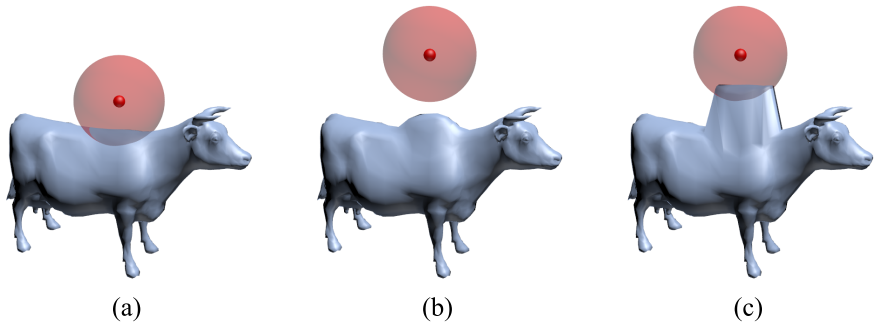


Figure 4: Significance of the master control point in preventing discontinuities. (a) Before deformation. (b) Deformation with master control point activated. (c) Deformation with master control point de-activated.

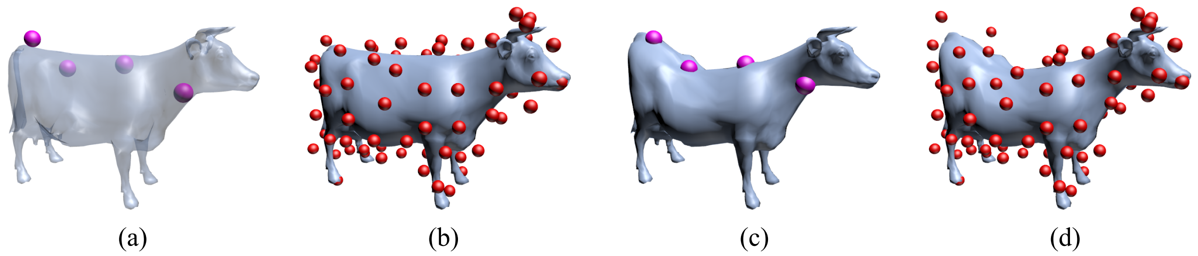


Figure 5: Example of multiple point constraints. The vertices nearest the four points will be translated to those points' positions. (a) Desired point constraints. (b) Original control set. (c) Deformed mesh satisfying the constraints. (d) Resulting control set.

3.3 Direct Manipulation

Direct manipulation of vertices in PB-FFD can be achieved by imposing point constraints. Let $\mathbf{h} = \{h_1, \ldots, h_K\}$ denote the set of indices of the vertices to be constrained. Let $\mathbf{V}' = \begin{bmatrix} v'_{h_1} & v'_{h_2} & \cdots & v'_{h_K} \end{bmatrix}^\top$ be the matrix of the desired positions of the vertices and $\Delta\mathbf{V} = \begin{bmatrix} v'_{h_1} - v_{h_1} & v'_{h_2} - v_{h_2} & \cdots & v'_{h_K} - v_{h_K} \end{bmatrix}^\top$ indicate the desired displacements. Let $\mathbf{P}_2$ denote the submatrix of $\mathbf{P}$ consisting rows 2 through M of $\mathbf{P}$. Following the approach in [13], we seek to find a matrix $\boldsymbol{\delta}$ of displacements δ_j for each control point $\mathbf{p}_j$ to effect the desired changes.

If we define an $(M-1) \times K$ constraint matrix

$$\mathbf{C} = \begin{bmatrix} w_{2h_1} & w_{2h_2} & \cdots & w_{2h_K} \\ w_{3h_1} & w_{3h_2} & \cdots & w_{3h_K} \\ \vdots & \vdots & \ddots & \vdots \\ w_{Mh_1} & w_{Mh_2} & \cdots & w_{Mh_K} \end{bmatrix}. \tag{8}$$

and set $\boldsymbol{\delta} = \mathbf{C}(\mathbf{C}^\top\mathbf{C})^{-1}\Delta\mathbf{V}$, the new value for $\mathbf{P}_2$ is given simply as $\mathbf{P}_2 + \boldsymbol{\delta}$, with $\mathbf{p}_1$ remaining unchanged. (The immovability of the master control point explains why the first subscripts on the w_{ij} terms start with 2.) As stated in [13], the existence of $\boldsymbol{\delta}$ depends on $(\mathbf{C}^\top\mathbf{C})^{-1}$, which does not always exist. This can happen, for example, if more constraints are desired than can be satisfied. We describe solutions to this problem in the Discussion section, below. An example of direct manipulation can be seen in Figure 5.

3.4 Multiresolution Editing

PB-FFD supports multiresolution control by hierarchically decomposing the deformation space. An example is given in Figure 6. At the top level of the hierarchy is a set of EBFs that provide a fine decomposition of the space, while lower levels provide coarser decompositions and are embedded within each other in a recursive structure.

Let us assume there are $L+1$ levels in the hierarchy and re-write Equation 7 as:

$$\mathbf{V} = \mathbf{P}^{L+1} = \mathbf{W}^L\mathbf{P}^L + \mathbf{D}^L\mathbf{A}^L, \tag{9}$$

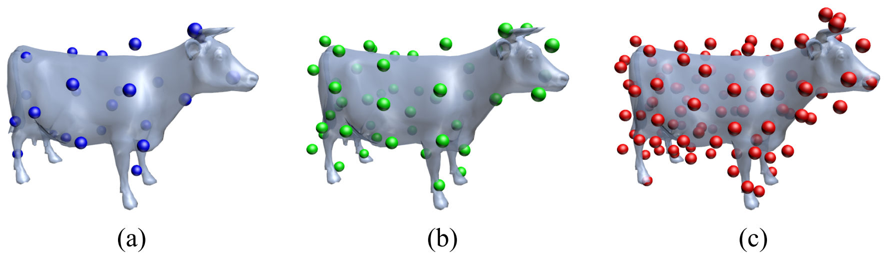


Figure 6: PB-FFD control point hierarchy. (a) Level 1: 20 control points. (b) Level 2: 40 control points. (c) Level 3: 80 control points.

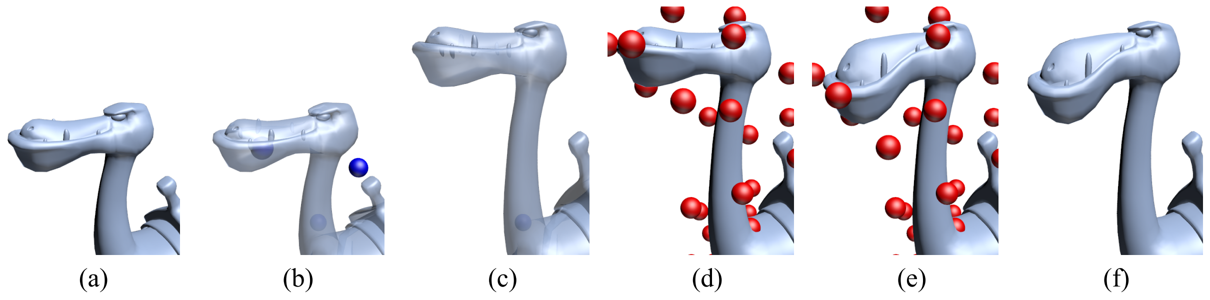


Figure 7: Multiresolution point-based deformations applied to the head of the "dinopet" model. (a)-(c) Deformations at coarsest level to elongate the neck. The surfaces in (b) and (c) have been drawn transparently to reveal the control points inside the surface. Note that some control points have been moved outside of this camera view. (d)-(f) Deformations at finest level to stretch the snout.

where the superscript indicates the level. Expressing the control points at level $\ell + 1$ (for $1 \leq \ell \leq L$) in terms of the control points at level ℓ yields a set of equations of the form:

$$\mathbf{P}^{\ell+1} = \mathbf{W}^{\ell}\mathbf{P}^{\ell} + \mathbf{D}^{\ell}\mathbf{A}^{\ell}. \tag{10}$$

A change in position of a point in the coarsest level ($\mathbf{P}^1$) cascades through the hierarchy to the finer levels.

Initially, $\mathbf{P}^L$ and $\mathbf{A}^L$ are given explicitly by the user or are assembled through the automatic control point placement algorithm described in Section 3.2.1. The matrices $\mathbf{W}^L$ and $\mathbf{D}^L$ are assembled by the system. The points $\mathbf{P}^{L-1}$ are defined by decimating a copy of the control points corresponding with $\mathbf{P}^L$ until the set of points is half its original size. Control points are merged on a pairwise basis, starting with the two points closest to each other. Given two nearby control points, $\mathbf{p}_i^L$ and $\mathbf{p}_j^L$, these points are replaced with a new one positioned at their mid-point. The new point's reference frame is computed by finding an ellipsoid (or a sphere, as in our implementation) that completely encloses the reference frames of $\mathbf{p}_i^L$ and $\mathbf{p}_j^L$. After $\mathbf{P}^L$ has been iteratively reduced in this fashion to half its original size, we have the new set of control points corresponding with $\mathbf{P}^{L-1}$. The matrices $\mathbf{W}^{L-1}$ and $\mathbf{D}^{L-1}$ can next be assembled by the system. The user can then manipulate the mesh at level $L - 1$, and points at levels greater than $L - 1$ will be displaced. This entire decimation-driven process is executed recursively to produce levels $L - 2, L - 3, \ldots, 1$ of the hierarchy, each level containing half the number of control points as the previous one.

Now we must define how changes at level ℓ affect points at levels less than ℓ. We considered several possibilities. We experimented with a hierarchical point constraint imposition technique similar to that of Section 3.3, but found that the $(\mathbf{C}^\top\mathbf{C})^{-1}$ matrix was singular too often for this option to be of practical use. Instead, we elected to use a reparameterization-based approach for imposing the constraints. Suppose we are directly editing the control points at $\mathbf{P}^\ell$. To update the positions at level $\ell - 1$, we view the control point positions at level ℓ as point constraints to be satisfied by repositioning $\mathbf{P}^{\ell-1}$. The point constraints can be imposed by reparameterizing that subset of points in level ℓ that is being moved. This provides a simple solution that can be computed very rapidly. However, one must be careful not to move a control point outside the domains of all the EBFs in a particular level. Otherwise, after reparameterization, that point will no longer influence the deformation of the mesh.

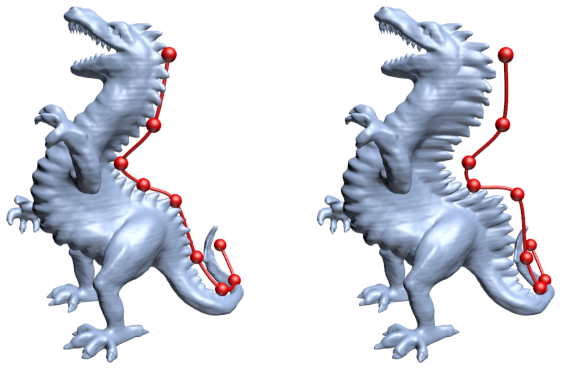


Figure 8: Spline curves can be employed to perform skeleton-drived deformations in a very intuitive manner. (a) Before deformations. (b) After deformations.

3.5 Curve-based Deformations

Curve-based deformations (Figure 8) can be simulated by defining interpolating spline curves whose ROIs enclose portions of the mesh. Once the control polygon of a curve has been defined and the spline has been discretized at a high resolution into a collection of EBFs, the enclosed portion of the mesh can be parameterized and deformed directly. Note that it should be possible to extend this functionality to skeleton-based deformations using the medial axis computed during automatic control point placement. This might prove to be an interesting deformation tool and would be a worthwhile direction for future work.

4 Implementation Issues and Performance Data

We have implemented a prototype sculpting system based on our PB-FFD framework on a generic PC workstation containing a 2.8 GHz CPU and 1 GB RAM. The algorithms and data structures can be implemented relatively easily in C++ and OpenGL with the assistance of the Qhull library [3] to compute the Voronoi diagram and the ANN library [2] to help determine which Voronoi points are inside the mesh (see Section 3.2.1). Performance data and some additional examples that were created with our system are shown below. The time for assembling the control set includes the construction of a three-level control point hierarchy with 200 control points at the finest level. Since only a fraction of a second is required to update the mesh after repositioning the control points (either directly or indirectly), such timings are not included in Table 1.

Model (# triangles)	**Qhull**	**Assemble Control Set**	**Parameterization**
Cow (5.8K)	1.062	7.204	0.157
Dinopet (15.9K)	3.250	21.374	0.375
Pit bull (25.0K)	5.032	38.375	0.657
Dragon (108.6K)	26.937	200.281	2.625

Table 1: Performance data for the preprocessing stages of PB-FFD. All times are in seconds. The somewhat costly step of automatic control set assembly can be skipped entirely if the user wishes to place control points manually.

5 Discussion

By and large the theoretical formulation of point-based free-form deformations translates well into program code. As indicated in Section 4, we have sought to use existing libraries (Qhull and ANN) wherever possible to expedite the implementation. We employed an in-house linear algebra library for performing traditional matrix operations like multiplication and inversion.

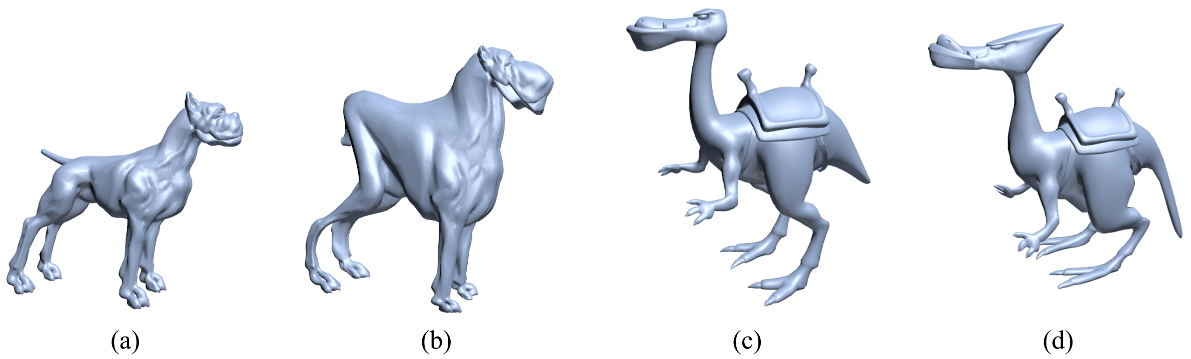


Figure 9: Additional examples of PB-FFD. (a)-(b) Pit bull model before and after deformations. (c)-(d) Dinopet model before and after deformations.

One question we did not discuss earlier is the determination of which EBFs parameterize which vertices. Since we employ compactly supported EBFs, each vertex will generally be under the influence of only a few control points. For each vertex in our implementation, we simply performed a linear search of the EBFs to determine which ones contained that vertex. Although this is a relatively inefficient $O(MN)$ algorithm, we assert that it would be overkill to use a kd-tree or similar data structure for accelerating nearest-neighbor queries for such small values of M. The efficient timings for "Parameterization" given in Table 1 demonstrate this quite clearly. Moreover, since $M \ll N$, M can be considered a constant in practice.

In most of our experiments we employed approximately 100 control points at the finest level of the control point hierarchy. This number is typical for FFD-based interaction since more than a few dozen control points tend to clutter the screen and confuse the user. Naturally, if this number is too small or too great – i.e., there are not enough degrees of freedom, or there are too many – the user could manually add or remove vertices, or simply instruct the system to adjust the number of control points automatically. We find a similar situation in traditional FFD in which the user-selected deformation lattice may be too sparse or too dense. Both in our technique and in FFD it is a simple matter of changing a few parameters to generate a new deformation space. Note that when a user adds a new control point, its behavior will be influenced by that of existing points. If this is undesirable, the designer can simply disable or delete control points near the new point so as to provide the new control point with greater control over the deformation.

Regarding the run-time usability of PB-FFD, every mesh vertex should be under the influence of at least two control points. Otherwise, the translation of a control point will cause a discontinuity in the mesh at the limit of the control point's ROI. Our solution is to use the master control point, which we advise not to render on-screen.

Another implementation concern is whether the matrix $\boldsymbol{\delta}$ of displacements always exists (see Section 3.3). We found that for single-level manipulation, it was very rare that $\boldsymbol{\delta}$ could not be computed because $\mathbf{C}^\top\mathbf{C}$ was singular. We experimented with various iterative techniques for finding an approximate solution to $\boldsymbol{\delta}$ in those situations, but found that their relatively slow convergence rates hampered the interactive design experience. In the end we decided that since this situation arose so infrequently, it would not be an undue burden on the user to request that he modify the point constraint slightly and try again.

References

[1] M. Alexa, D. Cohen-Or, and D. Levin. As-rigid-as-possible shape interpolation. In *Computer Graphics (Proceedings of ACM SIGGRAPH 2000)*, pages 157–164, July 2000.

[2] S. Arya and D. M. Mount. Approximate nearest neighbor searching. In *Proceedings of the Fourth Annual Symposium on Discrete Algorithms*, pages 271–280, 1993.

[3] C. Barber, D. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996.

[4] T. Beier and S. Neely. Feature-based image metamorphosis. In *Computer Graphics (Proceedings of ACM SIGGRAPH '92)*, pages 35–42, July 1992.

[5] P. Borrel and A. Rappoport. Simple constrained deformations for geometric modeling and interactive design. *ACM Transactions on Graphics*, 13(2):137–155, 1994.

[6] M. Botsch and L. Kobbelt. Real-time shape editing using radial basis functions. *Computer Graphics Forum*, 24(3):611–621, 2005.

[7] G. Bradshaw and C. O'Sullivan. Adaptive medial-axis approximation for sphere-tree construction. *ACM Transactions on Graphics*, 23(1):1–26, 2004.

[8] S. Capell, S. Green, B. Curless, T. Duchamp, and Z. Popovic. A multiresolution framework for dynamic deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 41–47, 2002.

[9] S. Coquillart. Extended free-form deformation: A sculpturing tool for 3D geometric modeling. In *Computer Graphics (Proceedings of ACM SIGRRAPH '90)*, pages 187–196, Aug. 1990.

[10] P. Faloutsos, M. van de Panne, and D. Terzopoulos. Dynamic free-form deformations for animation synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):201–214, 1997.

[11] G. Hirota, R. Maheshwari, and M. C. Lin. Fast volume-preserving free-form deformation using multi-level optimization. In *Proceedings of the Fifth ACM Symposium on Solid Modeling and Applications*, pages 234–245, 1999.

[12] W. M. Hsu, J. F. Hughes, and H. Kaufman. Direct manipulation of free-form deformations. In *Computer Graphics (Proceedings of ACM SIGGRAPH '92)*, pages 177–184, July 1992.

[13] S.-M. Hu, C.-L. T. H. Zhang, and J.-G. Sun. Direct manipulation of FFD: efficient explicit solutions and decomposible multiple point constraints. *The Visual Computer*, 17(6):370–379, 2001.

[14] J. Hua and H. Qin. Free-form deformations via sketching and manipulating scalar fields. In *Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications*, pages 328–333, June 2003.

[15] T. Igarashi, T. Moscovich, and J. F. Hughes. As-rigid-as-possible shape manipulation. In *Computer Graphics (Proceedings of ACM SIGGRAPH 2005)*, pages 1134–1141, July 2005.

[16] N. Kojekine, V. Savchenko, M. Senin, and I. Hagiwara. Real-time 3D deformations by means of compactly supported radial basis functions. In *Proceedings of Eurographics 2002*, pages 35–42, 2002.

[17] S. Lee, G. Wolberg, and S. Y. Shin. Polymorph: Morphing among multiple images. *IEEE Computer Graphics and Applications*, 18(1):58–71, 1998.

[18] S.-Y. Lee, K.-Y. Chwa, S. Y. Shin, and G. Wolberg. Image metamorphosis using snakes and free-form deformations. In *Computer Graphics (Proceedings of ACM SIGGRAPH '95)*, pages 439–448, Aug. 1995.

[19] R. MacCracken and K. I. Joy. Free-form deformations with lattices of arbitrary topology. In *Computer Graphics (Proceedings of ACM SIGGRAPH '96)*, pages 181–188, Aug. 1996.

[20] L. Moccozet and N. Magnenat-Thalmann. Dirichlet free-form deformations and their application to hand simulation. In *Proceeding of IEEE Computer Animation '97*, pages 93–102, 1997.

[21] J.-Y. Noh, D. Fidaleo, and U. Neumann. Animated deformations with radial basis functions. In *Proceedings of the 2000 ACM Symposium on Virtual Reality Software and Technology*, pages 166–174, 2000.

[22] M. Samozino, M. Alexa, , P. Alliez, and M. Yvinec. Reconstruction with Voronoi centered radial basis functions. In *Symposium on Geometry Processing*, pages 51–60, 2006.

[23] S. Schaefer, T. McPhail, and J. Warren. Image deformation using moving least squares. In *Computer Graphics (Proceedings of ACM SIGGRAPH 2006)*, pages 533–540, July 2006.

[24] S. Schein and G. Elber. Discontinuous free-form deformations. In *Proceedings of the Twelfth Pacific Conference on Computer Graphics and Applications*, pages 227–236, 2004.

[25] T. W. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. In *Computer Graphics (Proceedings of ACM SIGRRAPH '86)*, pages 151–160, Aug. 1986.