International Journal of Shape Modeling
© World Scientific Publishing Company

Mean-Value Geometry Encoding

VLADISLAV KRAEVOY
University of British Columbia
201-2366 Main Mall,
Vancouver, BC V6T1Z4,Canada
vlady@cs.ubc.ca

ALLA SHEFFER
University of British Columbia
201-2366 Main Mall,
Vancouver, BC V6T1Z4,Canada
sheffa@cs.ubc.ca
http://www.cs.ubc.ca/∼sheffa/

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Communicated by (xxxxxxxxxx)

Geometry editing operations commonly use mesh encodings which capture the shape properties of the models. Given modified positions for a set of anchor vertices, the encoding is used to compute the positions for the rest of the mesh vertices, preserving the model shape as much as possible. In this paper, we introduce a new shape preserving and rotation invariant mesh encoding. We use this encoding for a variety of mesh editing applications: deformation, morphing, blending and motion reconstruction from Mocap data. The editing algorithms based on our encoding and decoding mechanism generate natural looking models that preserve the shape properties of the input.

Keywords: mesh editing, rotation invariant shape representation, local shape representation, shape preservation, multiresolution, shape blending and morphing, Mocap data reconstruction

1. Introduction

The standard encoding of discrete geometric models is done by describing the 3D Euclidean coordinates of the vertices and the adjacency/connectivity relationships between them. Several alternative representations have been proposed as better suited for specific processing needs, such as:

- *Compression* techniques (for a review see Alliez and Gotsman[1]) utilize prediction based encodings, where the position of a vertex with respect to its neighbors is stored as a difference vector from a position predicted based on

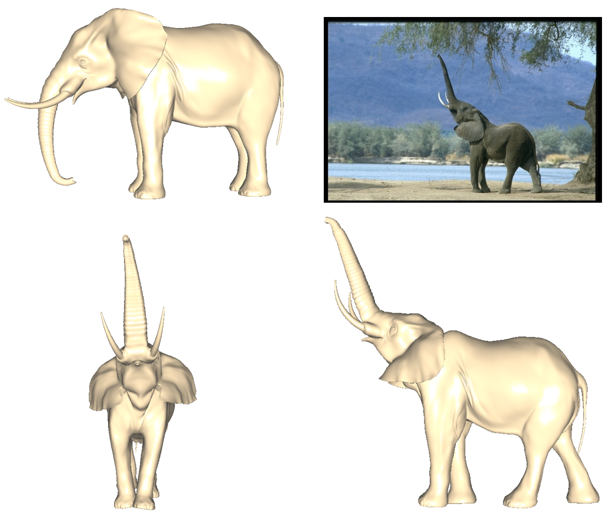

Fig. 1. Bringing elephants into nature: (top) original 3D model and inspiration image, (bottom) deformation using mean-value encoding.

its neighbors. The main motivation for this alternative representation is to make the encoding more compact while preserving the original geometry.

- *Geometry Editing* operations also commonly utilize alternative representations. Here the motivation is to encode local and global shape properties of the models and maintain those as much as possible while the models undergo global changes.

In either case, the representations typically combine local encoding of the model vertices with an explicit Euclidean encoding of a sub-set of the vertices, referred to as *anchors*. For compression, the Euclidean positions of the anchors are strictly preserved. For editing, the anchors are typically used as the control mechanism. Namely, the positions of the anchors are modified and the local encoding is used to establish the new positions in 3D space for the rest of the vertices. Since the focus in editing is on shape preservation, it is desirable for the local encoding to satisfy the following requirements:

(1) If the positions of the anchors are unchanged, the positions of the rest of the vertices should remain unchanged.
(2) If the positions of the anchors are a rigid transformation of the original, the positions of the rest of the vertices should transform similarly.
(3) In all other cases, the encoding should strive to minimize the difference in shape between the new and original models. There seems to be no established numerical measure of shape preservation. Therefore, visual inspection appears to be the major criterion used.

Based on the properties above, and in particular (2), a good encoding should be rotation invariant. However this is not the case with the majority of existing representations (Section 1.1).

In this work we present a rotationally invariant local representation, the *mean-value encoding*, which describes the position of each vertex with respect to its neighbors using a local coordinate frame. The description is based solely on continuous shape properties of the model such as angles and distances. Therefore, models with the same shape have identical encoding, and the encoding for models with close shape is very similar. Thus our encoding/decoding mechanism satisfies all three of the requirements above. Using our hierarchical encoding and decoding procedure, the decoding is performed in near interactive frame rates, taking less than a second for models of up to 100K faces.

We use the new encoding for a variety of editing applications: deformation, morphing, blending, and fitting of motion captured data. We introduce a new fully automated technique that can realistically reconstruct complex human motion based on Mocap data alone (Figures 8, 9). We demonstrate that compared to several other recent methods our algorithm constructs more natural looking result models and better preserves the shape properties of the input. An additional advantage of our approach is that although it does not explicitly prevent model self-intersections, the shape preservation property of our encoding drastically reduces the risk of local self-intersection.

The rest of the paper is organized as follows. Section 1.1 reviews editing techniques and associated geometry encoding schemes. Section 2 introduces the mean-value encoding and decoding formulations. Section 3 provides an efficient hierarchical decoding algorithm, developed based on these formulations. Section 4 demonstrates the applications of the new encoding and provides comparison with previous encodings. Finally, Section 5 summarizes the paper.

1.1. *Previous Work*

Earlier mesh editing techniques (e.g. Zorin et al.[2], Kobbelt et al.[3]) often used hierarchical mesh encodings. The idea is to decompose the mesh into two or more levels of detail, such that each level is encoded with respect to the previous one. The editing is performed on the coarsest level and then propagated to higher levels. Zorin et al.[2] propose such an encoding for meshes with subdivision connectivity. Guskov et al.[4] develop an encoding where a vertex is encoded as a distance in the normal direction, from the average of the neighbor vertices. This provides a rotation-invariant representation of details. However, most existing meshes can't be encoded using such a representation since the vertices are typically not positioned strictly above the average of the neighbors. Thus, this method requires quality remeshing and a variety of heuristics to treat vertices that even after remeshing do not satisfy the positioning requirement. Kobbelt et al.[5,3] use a different remeshing technique to obtain a similar type of encoding. Bischoff and Kobbelt[6,7] propose a volumetric detail encoding, providing more natural behavior at the expense of a more involved reconstruction operator. In order to use the encoding for editing purposes, most of these methods use a smooth base mesh as the coarsest level of the hierarchy. Thus, the general editing problem is reduced to the challenge of editing the smooth base mesh. Recent methods, such as Bischoff and Kobbelt[7], propose linear techniques for modifying the base mesh. However, since simple linear formulations are unable to distinguish between rotational and other linear transformations (Sorkine et al.[8]) the editing can lead to undesired artifacts.

A skeleton based encoding is an encoding where the position of each vertex is defined with respect to the links of the model's skeleton. This encoding can facilitate simple editing operations (e.g. Yoshizawa et al.[9]). Although skeletons exist for any model in theory, they are generally hard to compute. Moreover, binary editing operations usually require skeletons with identical connectivity. The construction of such skeletons in 3D remains, to our knowledge, an open problem.

The Laplacian coordinates[10] provide a local encoding which can be computed for any mesh. The Laplacian coordinates of each vertex are defined as a displacement vector between the average of the neighbor vertices and the actual 3D position of the vertex. Using this encoding, mesh editing becomes extremely efficient, since the decoding procedure only requires solving a simple linear system. Regrettably, since these coordinates are not invariant under rotation and scaling, the technique introduces visible artifacts for large deformations (Figure 6 (b)). Sorkine et al.[8] extend the use of Laplacian coordinates by linearly approximating local rotation and solving repeatedly for small rotational updates (Figure 6(c)). Yu et al.[11] and Zhou et al.[12] combine Laplacian coordinates with a different rotation approximation mechanism. Since all the above mentioned techniques use only an estimations of rotations the results are still suboptimal.

Lipman et al.[13] propose to split the editing problem into two separate linear systems. The first linear system estimates the local rotations at each vertex and the second liner system reconstructs the vertex positions using those estimates. This is only an approximation and not an exact solution.

This work builds upon the work of Sheffer and Kraevoy[14] who introduced *Pyramid coordinates*, a rotation invariant local mesh representation based on mean-value weighting[15] of neighbor vertices combined with a normal distance encoding. The authors fail to provide a closed form formulation for obtaining Euclidean coordinates from the encoding, leading to visible artifacts near anchor vertices (Figure 6 (d)). The method is also quite time consuming with deformation operations on 50K vertices taking 2 to 3 minutes to compute. Like the pyramid coordinates, our mean-value encoding is based on a set of angles and lengths describing the position of a vertex with respect to its neighbors. However, in contrast to Sheffer and Kraevoy[14], by using a different local frame, we develop a closed form decoding formulation. Thus we do not encounter any of the drawbacks mentioned above.

2. Mean-value Encoding

2.1. *Encoding one vertex*

Given a mesh with vertices V and edges E the mean-value encoding for each vertex $v_i \in V$ is computed from the Euclidean coordinates of the vertex and its m neighbor vertices v_j, where $(i,j) \in E$.

To compute the local coordinate frame we enumerate the neighbor vertices counter-clockwise around v_i as $v_{j_1}, \ldots, v_{j_m}$. For each vertex v_i we define a corresponding local projection plane

$$P_i = n_x x + n_y y + n_z z + d_i \tag{1}$$

using the normal at v_i, $n_i = (n_x, n_y, n_z)$. The normal n_i is computed as

$$n_i = \frac{\sum_{k=1}^{m} (v_{j_{k+1}} - l) \times (v_{j_k} - l)}{\| \sum_{k=1}^{m} (v_{j_{k+1}} - l) \times (v_{j_k} - l) \|} \tag{2}$$

where

$$l = \frac{1}{m} \sum_{(i,j)\in E} v_j . \tag{3}$$

In other words, we use an area averaged normal to a local Laplacian mesh as the normal of the projection plane. By using a normal formulation based solely on the neighbor vertex positions v_j, we are able to obtain an explicit formula for decoding v_i (Section 2.2), leading to a closed form global decoding formulation. This enables us to achieve much better results in terms of stability, speed, and shape preservation compared to Pyramid coordinates[14], where the normal estimate was based on the current position of v_i. Moreover, thanks to the area averaging, the computed normal is a continuous and bounded derivative function for all non-degenerate vertex configurations. [a]

We compute d_i, the average distance from origin, as:

$$d_i = -\frac{1}{m} \sum_{(i,j)\in E} n_i \cdot v_j . \tag{4}$$

Given n_i and d_i, the shape encoding of the vertex coordinates is separated into a tangential component computed in the projection plane and a normal component based on the vertex offset from the plane.

Given the vertex v_i and its neighbor vertices v_j we encode v_i in the following manner. First, we project v_i and its neighbors v_j onto the projection plane:

$$v_i' = v_i - (d_i + (v_i \cdot n_i))n_i \tag{5}$$

$$v_j' = v_j - (d_i + (v_j \cdot n_i))n_i \tag{6}$$

[a]The degenerate situations in which the normal is ill-defined can arise when either all the neighbor vertices are nearly collinear or when the mesh has very sharp creases. Both situations can be prevented during the mesh decoding.

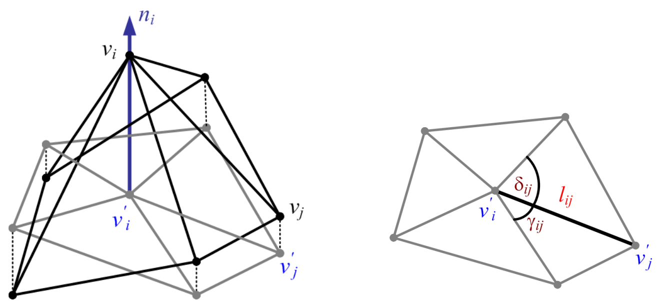

Fig. 2. Mean-value encoding: The 3D mesh is shown in black, the normal n_i is shown as a vertical vector, the projected mesh in the local projection plane is shown in gray, and the values $(\delta_{ij}, \gamma_{ij}, l_{ij})$ used to compute w_{ij} are shown on the right figure.

We then compute the mean-value weights[15] of v'_i with resepect to v'_j:

$$w'_{ij} = \frac{tan(\gamma_{ij}/2) + tan(\delta_{ij}/2)}{l_{ij}}$$

$$w_{ij} = \frac{w'_{ij}}{\sum_{(i,k)\in E} w'_{ik}} \tag{7}$$

The angles γ_{ij}, δ_{ij} and the lengths l_{ij} are shown in Figure 2.

To represent the normal component of v_i with respect to the local frame, we calculate the signed cosine of the angle between each edge incident to v_i and the normal n_i

$$c_{ij} = \frac{(v_i - v_j) \cdot n}{\|v_i - v_j\|}.$$

We calculate and store the cotangent of the angle between each edge and the normal

$$b_{ij} = \frac{c_{ij}}{\sqrt{1 - c_{ij}^2}}.$$

The encoding of the entire model consists of the set of coefficients w_{ij} and b_{ij} defined for each half-edge (note that $w_{ij} \neq w_{ji}$ and $b_{ij} \neq b_{ji}$).

In the next section, we demonstrate how to use the encoding to explicitly obtain the 3D coordinates of v_i from those of the adjacent vertices.

2.2. *Decoding one vertex*

The 3D positions of the neighbor vertices v_j and the encoding coefficients w_{ij} and b_{ij} uniquely define the position of the vertex v_i in the Euclidean space. We now sketch the numerical derivations leading to the explicit formulation for v_i (Equation 13). Using the mean-value weights w_{ij}, we obtain v'_i (Equation 5) from v'_j

$$v'_i = \sum_{(i,j)\in E} w_{ij} v'_j = \sum_{(i,j)\in E} w_{ij}(v_j - (d_i + (v_j \cdot n_i))n_i) \tag{10}$$

where n_i and d_i are calculated using Equations 2 and 4 respectively. Due to the reconstruction property of the mean-value weights[16], this formula exactly reconstructs v'_i for any set of neighbor vertices and mean-value weights, including cases where v'_i is outside the kernel of the projected neighbor vertices. Given v'_i and using any one of the coefficients b_{ij}, v_i is given by

$$v_i = v'_i + (\|v'_i - v'_j\| b_{ij} + (v_j - v'_j) \cdot n_i) n_i \tag{11}$$

Since the mean-value weights sum to one, we can rewrite this as

$$v_i = v'_i + \sum_{(i,j)\in E} w_{ij}(\|v'_i - v'_j\| b_{ij} + (v_j - v'_j) \cdot n_i) n_i. \tag{12}$$

Using basic numerical substitutions, we obtain the following formula for v_i as function of the rest of the vertices using the encoding coefficients w_{ij}, and b_{ij},

$$v_i = F_i(V) = \sum_{(i,j)\in E} w_{ij}(v_j + \|N_i \sum_{(i,k)\in E} w_{ik}(v_k - v_j)\| b_{ij} n_i) \tag{13}$$

where n_i is given by Equation 2 and N_i is the 3×3 matrix (column notations)

$$N_i = I_3 - n_i n_i^T, \tag{14}$$

where I_3 is a 3×3 identity matrix. This formula uniquely defines v_i, for any set of neighbor vertices and any mean-value encoding.

It is trivial to show that if the neighbor vertex positions are a rigid transformation of the original the obtained position of the vertex v_i is the original subject to the same rigid transformation. Hormann[16] proves that mean-value weights are continuous everywhere in the plane. This implies that a small variation in v'_j will result in small variation in v'_i. Since $F_i(V)$ (Equation 13) is C^∞ nearly everywhere with respect to v_j, a small variation in v_j will result in a small variation in v_i. Given constant w_{ij} and b_{ij}, F is non-smooth only if n_i is ill-defined, that is, the mesh contains extreme degeneracies, which we assume is not the case. Those observations demonstrate that if the neighbor vertex positions are a small deformation of the original, so will be the position of v_i. Hence the decoding preserves the shape of the model under local deformation.

2.3. *Encoding and Decoding of Models*

To uniquely encode 3D models, we must eliminate the degrees of freedom provided by rigid transformation and global scaling. Hence, in addition to the mean-value encoding of each vertex (w_{ij} and b_{ij}), the full encoding must contain the 3D Euclidean coordinates of four anchor vertices V_a. Additional anchors may be defined based on application requirements.

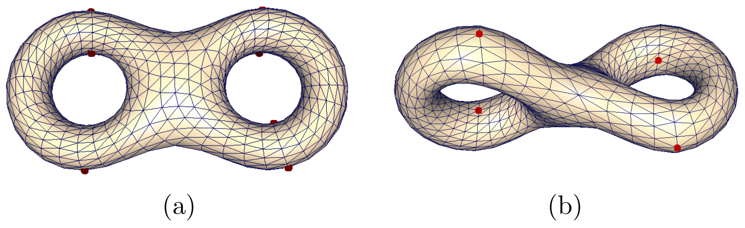

Fig. 3. Deformation of a figure eight model by directly minimizing Equation 15: (a) Original model. (b) The result of twisting the model by 90 deg.

To decode a 3D model from the encoding above, we formulate and solve the following non-linear least squares minimization problem

$$\arg\min_{V'} G(V') = \frac{1}{2}\sum_{v_i \in V}(v_i - F_i(V))^2 \tag{15}$$

where $V' = V \setminus V_a$. Note that while we search for the coordinates of non-anchor vertices only, the sum on the right-hand-side of the formula runs over all the vertices in the mesh. If the anchor vertex positions are unchanged, the set of original vertex positions is clearly a solution to the minimization problem.

We can solve this problem using standard non-linear least-squares minimization techniques. We use Levenberg-Marquardt[17] minimization combined with line search and trust-region as implemented in MATLAB. Since for all of our applications the original model is available, the original coordinates can always be used to provide the initial guess for the optimization. Figure 3 shows the deformation of a figure eight model using our minimization code with 8 anchor vertices placed as shown in the figure. The solution required 9 iterations to converge. It is possible to further accelerate the solution procedure using advanced numerical techniques and using a C++ implementation instead of MATLAB. However, as the size of the system grows, it would be unrealistic to expect real-time editing interaction using this solution approach. Therefore, we incorporate a multiresolution structure into the encoding and decoding procedures (Section 3), interleaving it with the numerical minimization, to speed up the decoding.

3. Hierarchical Encoding and Decoding of Meshes

To efficiently decode the model given the mean-value encoding, we modify the encoding and decoding procedures. During the encoding, a multiresolution hierarchy for the model is constructed and the encoding for the different levels in the hierarchy is stored. The hierarchical encoding is then used by the decoding procedure. This approach makes the encoding slightly more time consuming, but dramatically speeds up the decoding. For all the applications described below (Section 4), the encoding can be done once as a pre-processing step, while decoding is often performed repeatedly and ideally should be done in real-time. The hierarchical approach is therefore very suitable for these scenarios. In contrast to previous hierarchical approaches, such as Kobbelts et al.[7,3], the base mesh for the hierarchy contains only the anchor vertices. Hence, our algorithm does not need any time-consuming remeshing in the pre-processing stage.

3.1. ***Hierarchical Encoding***

The multi-resolution hierarchy is constructed using a simplification procedure that removes the non-anchor vertices one by one, until only a base mesh connecting the anchors remains. The simplification is performed using a sequence of half-edge collapse operations. A mesh hierarchy is constructed, keeping track of all the individual edge collapse operations. When selecting the edge to be collapsed, we use a mixture of volume preservation (see Lindstrom and Turk[18]) and minimal angle maximization metrics in order to preserve the mesh shape and to avoid degenerate configurations throughout all the levels of the hierarchy. Before each collapsed vertex is removed, the mean-value encoding of the vertex in the current mesh is computed and stored for reconstruction purposes.

3.2. ***Hierarchical Decoding***

At the beginning of the decoding process, a base mesh is constructed by placing the anchor vertices at the specified locations (e.g., Figure 4 (b)). For most applications those locations differ from the original ones, as explained below (Section 4). The subsequent decoding procedure involves two major operations: vertex split and optimization.

3.2.1. *Vertex split*

Reversing the simplification order, collapsed vertices are added to the mesh one at a time. We use Equation 13 to obtain the position for each new vertex. Note that at the time of insertion, the positions of adjacent vertices in the current mesh are well defined. Since we use the volume and minimal-angle maximization metrics during simplification, the surrounding mesh is reasonably well shaped and does not contain degeneracies. As a result the functions F_i in Equation 13 are well defined. If the anchor positions are unchanged or a rigid transformation of the original position, this placement gives the exact desired position of the vertex in 3D.

3.2.2. *Optimization*

If the anchor positions are modified, each split introduces some error. While after each vertex split operation, $v_i - F_i(V)$ equals 0 at the inserted vertex v_i, this is not necessarily the case for the adjacent vertices. Hence $G(V')$ (Equation 15) is not optimized.

To find the minimizer of $G(V')$, after performing a sequence of vertex splits, we use a Gauss-Newton minimization procedure combined with line-search. The

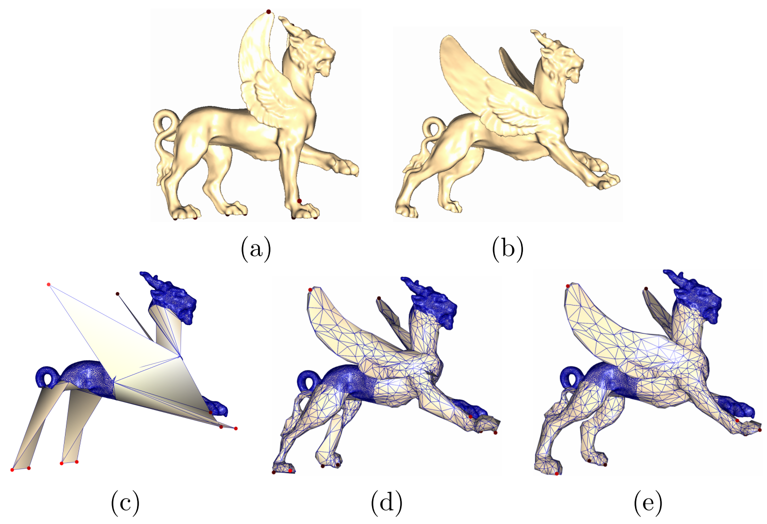

Fig. 4. Deformation using mean-value encoding and decoding: (a) Original model. (b) Final mesh. (c) Base mesh (anchors and fixed parts) with modified anchors. (d) Intermediate mesh after edge-splits. (d) Intermediate mesh after relaxation. Note the smoothing effect on the legs and the wings.

minimum is obtained when the Jacobian of $G(V')$ is zero. The rows of the Jacobian are:

$$\frac{\partial G}{\partial v_i} = v_i - F_i(V) - \sum_{(i,j)\in E} (v_j - F_j(V)) \frac{\partial F_j(V)}{\partial v_i} \tag{16}$$

Defining the Jacobian of $G(V')$ as J, the vector of the functions F_i as F, and the matrix of partial derivatives $\frac{\partial F_j(V)}{\partial v_i}$ as ΔF, the system can be rewritten in matrix format as

$$J(V') = (I' - \Delta F)^T (V' - F) = 0 \tag{17}$$

where I' is an $|V| \times |V'|$ sparse matrix with 1's on the diagonal. Using Gauss-Newton method we ignore the second order terms in the Hessian, defining

$$H = (I' - \Delta F)^T (I' - \Delta F) \tag{18}$$

Thus at each iteration of the procedure, we solve the linear system

$$H\delta = -J(V') \tag{19}$$

and update $V' = V' + \alpha\delta$ $(0 \leq \alpha \leq 1)$. We use standard bisection line-search to compute α. We perform numerical derivation to compute the matrix of partial derivatives ΔF for the three coordinates x, y and z. We use the conjugate gradient method to solve the linear system (Equation 19).

For the models we edited, we found it is sufficient to perform optimization only once during the reconstruction procedure for an intermediate mesh with about 3%

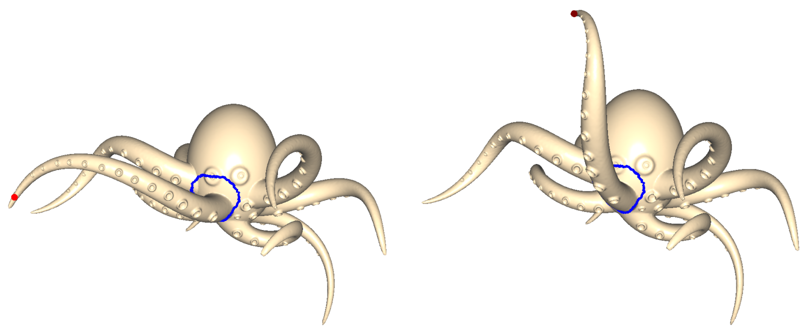

Fig. 5. Lifting an octopus tentacle using one control vertex. The boundary of ROI is indicated by the blue dots.

of the vertices. On average, the optimization for this intermediate mesh required about 7 iterations to converge.

For extreme deformations, after the model is fully reconstructed, we apply several Gauss-Seidel iterations, typically four, toward solving Equation 17. The Gauss-Seidel procedure uses Equation 16 (equating it to zero) to set the value for v_i.

Figure 4 shows the decoding stages for a deformed feline model. The parts of the model that remain fixed, such as the head, are treated as anchors. The error introduced by performing edge-splits alone is clearly visible on the intermediate mesh (Figure 4(c)). For this 100K triangle model the decoding took 0.86 seconds. The hierarchical encoding took 6.6 seconds.

The applications of the encoding and decoding mechanism described above are based on modifying the positions of the anchor vertices, and reconstructing the mesh subject to the new anchor positions. The next section describes several of these applications.

4. Applications

The applications of our encoding and decoding mechanism include editing operations such as deformation, morphing and blending, as well as animation and fitting of motion capture data. To control the change in the model shape for some of those operations, we need to define a *region of influence* (ROI) for the modification performed (Figure 5). The region defines which mesh vertices are recomputed using the decoding mechanism and which are left in their original locations. The vertices outside the region of influence are simply treated as additional anchors.

4.1. *Deformation*

Mesh deformation is probably the most useful and straightforward application of the encoding/decoding mechanism we developed. A typical mesh deformation interface modifies the locations of a set of control vertices and updates the coordinates of the

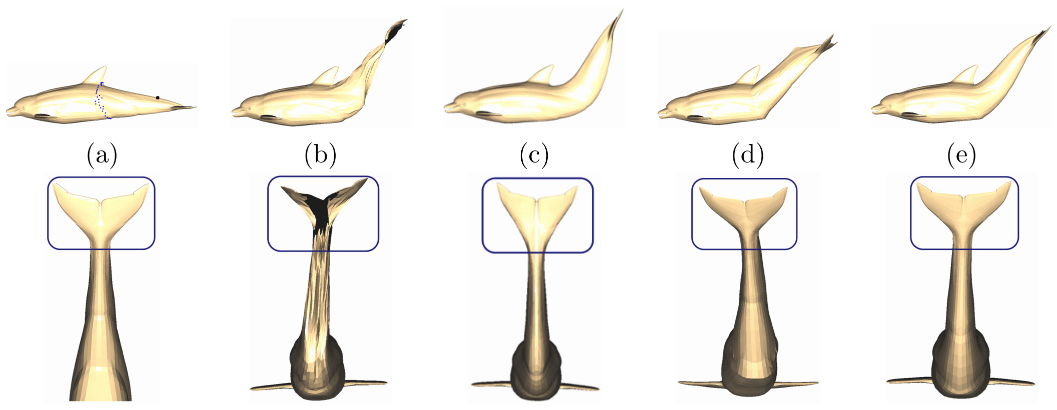

Fig. 6. Comparison of deformation methods, with details (zoom in on the tail): (a) Original model. (b) Laplacian coordinates; (c) Extended Laplacian coordinates[8]; (d) Pyramid coordinates[14]; (e) Mean-Value encoding. Note that only the last example preserves the original shape of the tail fins.

rest accordingly. In contrast to recent methods such as Yu at al.[11], we require no normal estimates, or curves of vertices to control the deformation. In our setting, the control vertices become the anchors of the encoding. When desired, we can use the region of influence to limit the deformation to only a part of the model. Given the selected set of anchors, the hierarchical encoding procedure is applied, constructing the appropriate multiresolution hierarchy. Since the hierarchy construction stage does not depend on the positions of the anchors after deformation, the hierarchy is precomputed only once per anchor and ROI selection. During the deformation itself, the decoding procedure is applied on-the-fly when the user interactively modifies the positions of the anchors.

Figures 1, 4, 5, and 6 show model deformation performed using mean-value encoding. In Figure 1 we successfully recreate a realistic roaring elephant pose using only 17 anchor vertices. Figure 4 shows the algorithm stages for deforming the feline model. Figure 5 demonstrates the preservation of high frequency details during deformation and the smooth transition between the deformed and undeformed regions of the mesh. Note that although the ROI boundary passes through one the octopus's eyes, the shape of the eyes is not affected.

Figure 6 uses a simple example to compare our deformation technique to deformations generated by several recent techniques. As expected, a purely linear deformation technique, such as Laplacian coordinates[10] (Figure 6(b)) leads to extreme shear, when the anchor position undergoes rotational displacement. The method of Sorkine et. al[8] (Figure 6(c)) significantly reduces the distortion, but still leads to visible shearing artifacts near the tail fins. The Pyramid coordinates[14] method (Figure 6(d)) causes less shearing artifacts, but exhibits discontinuities near the anchor and along the boundaries of the ROI. In contrast, our mean-value encoding and decoding mechanism produces a smooth and intuitive deformation with no undesirable artifacts (Figure 6(e)). To perform the comparison we reimplemented

Laplacian coordinates[10]. For Sorkine et. al[8] and Pyramid coordinates[14] we used software provided by the authors.

Table 1 provides statistics and runtime results for the examples described above. All runs were performed on a P4 3GHz machine. We use $G(V')$ to measure the difference in shape between the original and deformed models. The value of $G(V')$ for all the models deformed using mean-value encoding is less than $1e^{-3}$ (Table 1). This provides a numerical indicator that our deformation procedure preserves the local shape of the models. In contrast, when using other methods for the dolphin deformation the error is one or more orders of magnitude larger.

Model	#vert.	ROI #vert.	#anchors	$G(V')$	Enc. (sec.)	Dec. (sec.)
Feline	49864	34759	9	0.000232	6.65	0.863
Octopus	149678	19196	1	0.000003	3.52	0.745
Dolphin	5937	1156	1	0.000146	0.190	0.054
Sheffer and Kraevoy[14]				0.001061		
Sorkine et al.[8]				0.006401		
Alexa[10]				0.011042		

Table 1. Deformation statistics. $G(V')$ measures the value of the function on the deformed model, given the original mean-value encoding.

4.2. *Animation*

The mean-value encoding mechanism can be used to animate models by prescribing space-time trajectories for anchor vertices. The encoding and the appropriate hierarchy are computed in the pre-processing step. During animation, the Euclidean positions of the vertices at each time-step are computed based on the anchor positions and the encoding.

We use this algorithm to create animated sequences from motion capture data. Mocap data provides trajectories for real human/animal motion by capturing the motion of a set of markers on moving subjects. Reconstruction of fully realistic complex motion from such data requires deep knowledge of anatomy and is beyond the scope of this work. Nevertheless, using our encoding based approach, we are able to create convincingly realistic motion using only a mean-value encoding of a human shape and the Mocap trajectories.

We define a corresponding anchor on the mesh for each marker in the Mocap data (Figure 7). We can now treat the Mocap data as the trajectory defined for the anchor vertices and compute the animation sequence as described above.

The resulting sequences demonstrated in Figures 8 and 9 and in the companion movie look very realistic: the full body motion is accurately reconstructed from the movement of the anchors and the high-frequency details of the male model, such as muscles and facial features, are fully preserved during animation. Note that

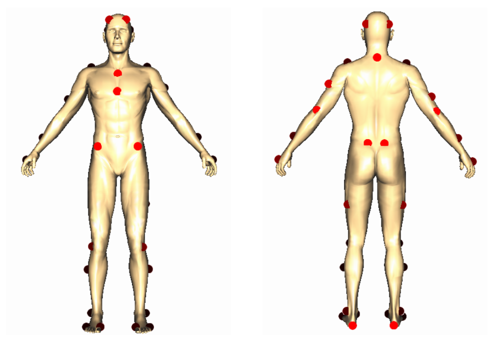

Fig. 7. For each Mocap marker we define a corresponding anchor on the mesh.

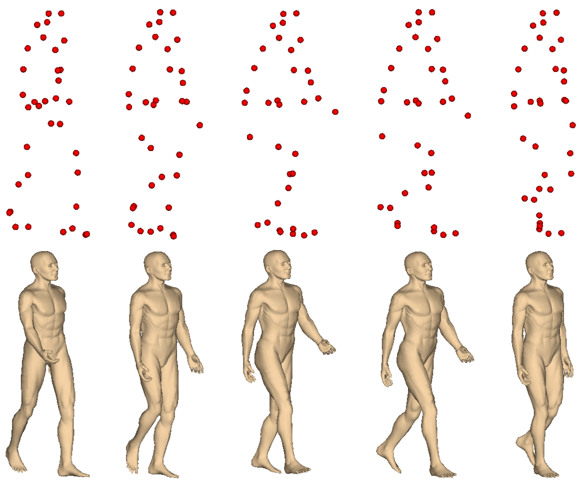

Fig. 8. Reconstruction of walking motion from Mocap data. (top) Original Mocap data. (bottom) Animation sequence.

animation reconstruction is significantly more challenging than regular deformation, since no ROI is given and the coordinates for all the vertices in the mesh need to be computed solely on the basis of the anchor coordinates. Since most previous editing methods rely on a ROI specification, they would not be suitable for this task.

4.3. *Morphing and Blending*

Morphing and blending operations generate the vertex coordinates for intermediate models as a function of the coordinates of the input models. A blending operation computes one intermediate model which is seen as a weighted average of the inputs while morphing operations compute a sequence of such intermediate models. Both operations typically consist of three stages [19]. The first two stages compute a correspondence and a common connectivity for the input models. At the end of the

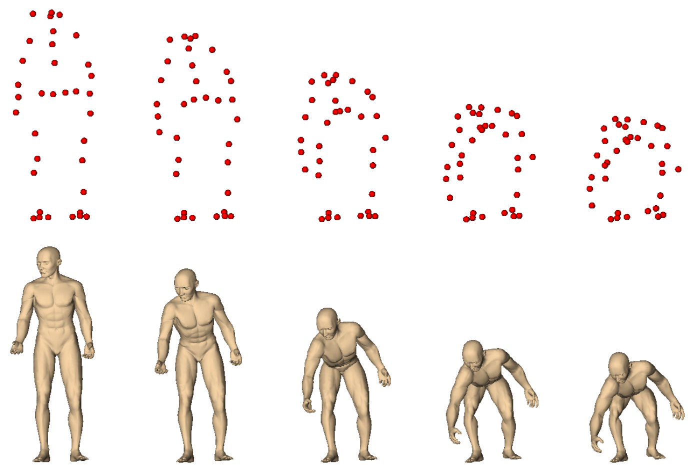

Fig. 9. Reconstruction of more complex sitting motion from Mocap data. (top) Original Mocap data. (bottom) Animation sequence.

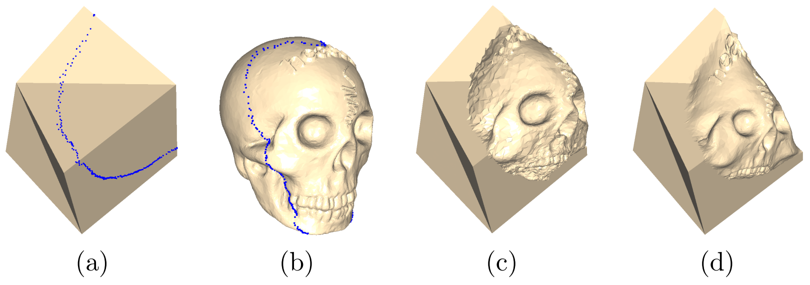

Fig. 10. Transferring the face of (b) to (a)): (a) & (b) Original models. (c) Blending using Alexa[10]. Note the discontinuity around the face and the high frequency noise particularly noticeable on the teeth. (d) Blend using mean-value encoding.

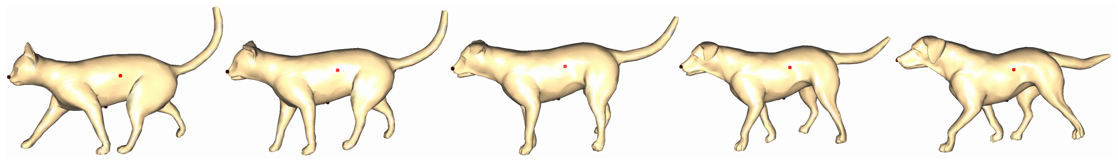

Fig. 11. Turning a cat into a dog. Note the preservation of local details and the smooth rotation of the legs and tail.

second stage, we have several models with the same connectivity with different geometries. For local morphing and blending, the common connectivity computation can be restricted to only a part of the model [20,8]. We focus on the final stage of the process where the geometry for the intermediate models is computed. This is typically done by computing weighted averages of the geometry encoding of the inputs. For standard morphing, the averaging is based on the time-step. For blending, the averaging captures which parts of the output model should be more similar to one or the other of the inputs.

To blend or morph models, we average the mean-value encodings of the models based on the appropriate weights. This provides an encoding which can be seen as a local shape average between the inputs. We then use the decoding mechanism to obtain the intermediate model(s). The results are demonstrated in Figures 10 and 11. In Figure 10 we use blending to paste a skull onto a complex polyhedron. Using mean-value encoding the blend is seamless and the details on the skull, such as the writeup and the teeth are perfectly preserved (Figure 10(d)). In contrast, performing the blending using Laplacian coordinates [10] results in visible artifacts (Figure 10(c)).

The cat to dog morphing sequence (Figure 11) uses four anchors on the animal bodies for registration. The mean-value based morphing generates rotational motion for the tail and legs to perform a smooth transition between these features on the two models. The timings for performing both operations are comparable to those for deformation. For example, the blended model in Figure 10 took 2.8 seconds to compute, including encoding and decoding.

5. Conclusions

We introduce a new, robust method for mesh editing based on a novel local representation, the mean-value encoding. The encoding captures the local shape properties of the mesh and is invariant under rigid transformations. Even more important, the encoding is a continuous function of the model geometry with similar models having similar encodings. Using this encoding we developed an efficient and robust decoding algorithm. We demonstrated the use of mean-value encoding for several mesh editing applications, including deformation, motion reconstruction from Mocap data, morphing and blending. Using our algorithm the editing times are comparable to those of recent state-of-the art mesh editing techniques, while the quality of our results is better.

Acknowledgements

To appear in the final version of the paper.

References

1. Pierre Alliez and Craig Gotsman. Recent advances in compression of 3D meshes. In *Proceedings of the Symposium on Multiresolution in Geometric Modeling*, 2003.
2. Denis Zorin, Peter Schröder, and Wim Sweldens. Interactive multiresolution mesh editing. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 259–268, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.
3. Leif Kobbelt, Jens Vorsatz, and Hans-Peter Seidel. Multiresolution hierarchies on unstructured triangle meshes. *Computational Geometry*, 14(1-3):5–24, 1999.

4. Igor Guskov, Wim Sweldens, and Peter Schröder. Multiresolution signal processing for meshes. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 325–334. ACM Press/Addison-Wesley Publishing Co., 1999.
5. Leif Kobbelt, Swen Campagna, Jens Vorsatz, and Hans-Peter Seidel. Interactive multi-resolution modeling on arbitrary meshes. *Computer Graphics*, 32(Annual Conference Series):105–114, 1998.
6. Stephan Bischoff and Leif Kobbelt. Sub-voxel topology control for level-set surfaces. *Computer Graphics Forum*, 22(3):273–280, 2003.
7. Stephan Bischoff and Leif Kobbelt. A remeshing approach to multiresolution modeling. In *Symposium on Geometry Processing*, pages 189–196, 2004.
8. Olga Sorkine, Yaron Lipman, Daniel Cohen-Or, Marc Alexa, Christian Rössl, and Hans-Peter Seidel. Laplacian surface editing. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 179–188. Eurographics Association, 2004.
9. Shin Yoshizawa, Alexander G. Belyaev, and Hans-Peter Seidel. Free-form skeleton-driven mesh deformations. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 247–253. ACM Press, 2003.
10. Marc Alexa. Local control for mesh morphing. In *Proceedings of the International Conference on Shape Modeling & Applications*, pages 209–215. IEEE Computer Society, 2001.
11. Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Mesh editing with Poisson-based gradient field manipulation. *ACM Transactions on Graphics*, 23(3):644–651, 2004.
12. Kun Zhou, Jin Huang, John Snyder, Xinguo Liu, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Large mesh deformation using the volumetric graph Laplacian. *ACM Transactions on Graphics*, 24(3):496–503, 2005.
13. Yaron Lipman, Olga Sorkine, David Levin, and Daniel Cohen-Or. Linear rotation-invariant coordinates for meshes. In *Proceedings of ACM SIGGRAPH 2005*, page accepted for publication. ACM Press, 2005.
14. Alla Sheffer and Vladislav Kraevoy. Pyramid coordinates for morphing and deformation. In *3DPVT '04: Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium on (3DPVT'04)*, pages 68–75. IEEE Computer Society, 2004.
15. Michael S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27, 2003.
16. K. Hormann. Barycentric coordinates for arbitrary polygons in the plane. Technical report, Clausthal University of Technology, September 2004.
17. William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, Cambridge (UK) and New York, 2nd edition, 1992.
18. P. Lindstrom and G. Turk. Fast and memory efficient polygonal simplification. In *Proceedings of IEEE Visualization*, pages 279–286, October 1998.
19. Marc Alexa. Recent advances in mesh morphing. *Computer Graphics Forum*, 21(2):173–196, 2002.
20. Henning Biermann, Ioana Martin, Fausto Bernardini, and Denis Zorin. Cut-and-paste editing of multiresolution surfaces. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 312–321. ACM Press, 2002.