Boneless Pose Editing and Animation

J. Andreas Bærentzen[1], Kristian Evers Hansen[2], and Kenny Erleben[3]

[1] Informatics and Mathematical Modelling
Technical University of Denmark
[2] 3Shape
[3] Computer Science, University of Copenhagen
jab@imm.dtu.dk, evers@exsite.dk, kenny@diku.dk

Abstract. In this paper, we propose a pose editing and animation method for triangulated surfaces based on a user controlled partitioning of the model into deformable parts and rigid parts which are denoted handles. In our pose editing system, the user can sculpt a set of poses simply by transforming the handles for each pose. Using Laplacian editing, the deformable parts are deformed to match the handles. In our animation system the user can constrain one or several handles in order to define a new pose. New poses are interpolated from the examples poses, by solving a small non-linear optimization problem in order to obtain the interpolation weights. While the system can be used simply for building poses, it is also an animation system. The user can specify a path for a given constraint and the model is animated correspondingly.

1 Introduction

Almost all modern animation systems are based on the concept of *skeletons* and *skinning*. Skinning is basically the process whereby a trained professional relates a triangle mesh (the skin) to a bone structure (the skeleton). When this work is done, the mesh may be animated by transforming the bones in the skeletal structure. The transformations of the bones, in turn, can be computed using *keyframes*, *inverse kinematics*, *motion capture*, or some other method.

While skeleton based animation is well established and used in numerous commercial systems, it is by no means an ideal solution. In particular, it is difficult to create a good correspondence between the skeleton and the mesh (skin). One goal of this paper is to convince the reader that we can do without skeletons. Instead of a skeleton, we propose to allow the user to paint rigid handles on the model and animate by transforming these handles.

1.1 Contributions and Related Work

Traditional inverse kinematics [1] combined with linear blend skinning [2] is used for character animation in packages such as Maya®and 3DStudio Max®. This combination offers the advantage of direct control and manipulation of the skin and skeletons. However, there are certain issues. Linear blend skinning is notoriously known for joint collapse and “candy wrap” artifacts. Many people have tried to fix these problems, for instance by replacing the interpolation method [3], or improving on the skin weights using statistical analysis and adding correction terms to the linear blending [4]. Also advanced methods exist for assigning skin weights, e.g. [5]. Nevertheless, animators still spend a considerable amount of time tweaking skin weights. In fact, traditional skeleton and skinning animation is known for requiring tweaking of skin-weights on a per motion basis, i.e. skin-weights that work well for a jumping motion may work badly for a running motion, and skinning for a lower arm twist probably won’t do for wrist flexing.

Skeleton based animation also falls a bit short with regard to what types of animations that can be achieved. Typically, animators work in one of two ways: Either the model is skinned and animated by rotation of bones (skeletal animation) or individual poses are sculpted simply by moving individual vertices to specific positions in specific poses (interpolation based animation). The former is more useful for overall character animation while the latter is needed for facial expressions. Ideally, and in order to fix the problems with bones based animation, they should be combined, and in [6] Lewis et al. present a system where the user can manipulate vertices directly for a given pose and at the same time use a skeleton driven deformation. Arguably, we obtain the same advantage in a simpler fashion.

Using our method, only one weight is associated with each vertex, and it has a very clear significance, namely the rigidity of a vertex. Using a simple selection tool, the user marks clusters of rigid vertices which become the *handles* of the model. Once a set of handles has been defined, the user can sculpt poses by rigidly transforming the handles. The deformable parts of the model are subsequently transformed using *Laplacian editing* which is discussed briefly in Section 2. (C.f. [7] for more details). Handle selection and pose editing are described in section 3. At least two benefits are gained from our approach

– **Simplicity:** The user only needs to paint “rigidity” and not “degree of association with any bone in a set of bones”. Moreover, there are no bones which need to be aligned with the model in its rest position.
– **Genericity:** Our method is based on rigid motion (translation or rotation) of handles followed by a smooth deformation of the remaining parts of the mesh. This technique can be used to achieve the same effects as both skeletal and interpolation based animation.

Inverse kinematics problems are usually underdetermined in the sense that different motions will lead to the same goal [8]. In order to tackle this issue, recent authors have considered reconstructing plausible motion from examples. In [9] Grochow et al. represent a dense set of poses as feature vectors in a low dimensional space and given a set of constraints finds an interpolated pose close to one of the examples. In Mesh based inverse kinematics [10] Sumner et al. also finds a model in pose space which best fits a set of constraints. Our approach is conceptually similar, but the problem to be solved is simpler since the pose space is a space of handle transformations and not vertex transformations. The technique used for animation is described in detail in section 4. Finally, in section 5, we discuss our findings, draw conclusions and point to future work.

2 Laplacian Editing

The fundamental idea in Laplacian editing is to represent mesh vertices in terms of *differential coordinates*. The differential coordinates capture the position of a vertex relative to its neighbours, and by nailing down the translational freedom, we can reconstruct the mesh directly from this representation. Moreover, we can impose several constraints and reconstruct the remaining vertices from the differential coordinates in the least squares sense which is what allows us to perform deformations.

To be more specific the differential coordinates of a vertex i are

$$\boldsymbol{\delta}_{\mathbf{i}} = \frac{1}{|N_i|} \sum_{j \in N_i} \mathbf{v}_j - \mathbf{v}_i \tag{1}$$

where j is a vertex in the neighbourhood, N_i, of i, and $\mathbf{v}_i$ and $\mathbf{v}_j$ are the positions of vertex i and its neighbour j, respectively. This operator, known as the umbrella operator, (1) constitutes a mesh Laplace operator given a simple local parameterization [7]. It is clearly possible to compute the mesh Laplacian simultaneously for all vertices if we write it as a matrix $\mathbf{L}$ where $\mathbf{L}_{ii} = -1$ and $\mathbf{L}_{ij} = \frac{1}{|N_i|}$ if there is an edge from vertex i to vertex j and zero otherwise. Provided we have all the vertices and a column vector $\mathbf{P}$, we can now compute the differentical coordinates for all vertices $\mathbf{D} = \mathbf{LP}$ The matrix does not have full rank [7] but adding just a single constrained vertex allows us to reconstruct the positions $\mathbf{P}$ from the $\mathbf{D}$ vector. In general, we would add multiple constraints and then recompute the vertex positions by solving for $\mathbf{D}$ in the least squares sense. If we wish to constrain a vertex i, we add an equation of the form $w_i \mathbf{v}_i = w_i \mathbf{c}_i$, where w_i is the weight of our constraint and $\mathbf{c}_i$ is the position of the constrained vertex. This equation is added as a row to the system $\mathbf{D} = \mathbf{LP}$. Let m be the number of constraints. In this case the system becomes

$$[\mathbf{D}|w_1\mathbf{c}_1 \; w_2\mathbf{c}_2 \; ... \; w_m\mathbf{c}_m]^T = \left[\frac{\mathbf{L}}{I_{m \times m}|0}\right] \mathbf{P} \;\;, \tag{2}$$

where we have assumed that the first m vertices are constrained. Of course, the vertex numbering is arbitrary and the constraints can be imposed on any m vertices.

To simplify notation, let the extended $\mathbf{D}$ vector of Laplacians be denoted $\tilde{\mathbf{D}}$ and the extended $\mathbf{L}$ matrix be denoted $\tilde{\mathbf{L}}$. In this case, the above system is $\tilde{\mathbf{D}} = \tilde{\mathbf{L}}\mathbf{P}$, and to reconstruct our vertices, we need to compute the least squares solution

$$\mathbf{P} = (\tilde{\mathbf{L}}^T \tilde{\mathbf{L}})^{-1} \tilde{\mathbf{L}}^T \tilde{\mathbf{D}} \tag{3}$$

Unfortunately, the differential coordinates $\mathbf{D}$ are not invariant with respect to rotation or scaling. In animation, the first problem is particularly severe. Several elegant solutions exist, e.g. [11,12,13]. Inspired by the work of Lipman et al. [13], we choose the approach we find to be the simplest. The idea is to have a smooth version of the mesh. For each vertex of the smoothed mesh, we compute a frame and represent the differential coordinates in terms of this frame. When the surface is deformed by moving the constrained vertices, we compute a new smoothed version of the mesh and corresponding frames for each vertex. The differential coordinates are then rotated simply by transferring them to the new frame.

What lends efficiency to this scheme is that the smooth solution is simply obtained by solving (3) with $\mathbf{D} = \mathbf{0}$ since this solution minimizes the membrane energy [7]. Intuitively, minimizing the Laplacians corresponds to placing each unconstrained vertex at the barycenter of its neighbours. Clearly, the constraints are also important in this case since an unconstrained mesh would simply collapse.

For a large mesh solving (3) at interactive rates is not feasible unless attention is paid to the structure of the problem. Fortunately, we are looking for a least squares solution which means that we are dealing with a positive definite matrix. Moreover, $(\tilde{\mathbf{L}}^T\tilde{\mathbf{L}})$, is fairly sparse, and a good way of solving such a linear system is by Cholesky factorization [14]. We do this using the Taucs library which is sparse matrix library known for its speed.

3 Pose Editing

In the following, we describe how an animator might define a set of handles and edit the model using these handles. The discussion is based on our test system which works well, but clearly vertex selection and handle rotation or translation can be done in a number of ways. We have chosen simple methods that work well with mouse and keyboard. First of all, It is necessary to have a relatively flexible mechanism for selecting the vertices which belong to a given handle since a single handle can be an arbitrary and not necessarily connected group of vertices. In

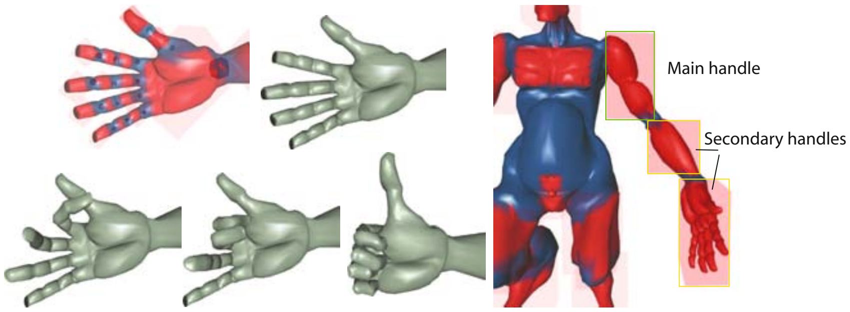

Fig. 1. Handles painted onto a hand model, and some sculpted poses (left). When we need to transform an arm it is crucial to be able to select several handles and rotate using the center of rotation for just one of these (right).

our system, handles can be defined using a paint metaphor where the user sprays the regions that are to be defined as handles. It is also possible to make a screen space box selection and the user can choose to select only the visible vertices which fall within the box or to select only one connected component. An example of handles on a hand model and corresponding sculpted poses is shown in Fig. 1 (left). As a part of defining the handles, the animator also needs to indicate the center of rotation for each handle.

Having selected the handles and rotation centers, the user proceeds to sculpting the pose. This is basically done by moving and rotating handles. A handle is moved or rotated simply by selecting and dragging with the mouse. Since we use a mouse as interface all translation is parallel to the view plane and rotation is around an axis perpendicular to the view plane. However, the user can choose arbitrary views.

Clearly, we sometimes wish to transform several handles in one operation (See Fig. 1 right). In order to do so, the user specifies a set of handles to transform (typically rotate) and then the handles are transformed using the first selected handle's center of rotation. Selecting several handles imposes some structure of the handles reminiscent of a skeletal structure, but the structure itself is not stored, only the transformations which are applied to each handle in the selection.

It is also important to state that when a handle is transformed, the actual transformation is stored (e.g. a rotation axis and the angle in degrees) and not simply a transformation matrix or, worse, the new vertex positions, since knowing the precise transformation allows us to smoothly go from rest position of the handles to the deformed state by using only a specific fraction of each transformation.

Apart from simply selecting the handles, the user also has the option of specifying that parts of the handle are less rigid than others (using the paint metaphor). This can have a large impact as shown in Fig. 2 bottom left.

4 Pose Blending and Animation

The poses which have been defined using the method outlined above can be used in a number of ways. In some computer games, models are animated simply by interpolating between the positions of corresponding vertices in two (keyframe) poses. That tends to be a better strategy than skeletal based animation for things like facial expressions and it works well if rotations are not too big, e.g. if we have a dense set of keyframe poses. Our pose blending and animation system could indeed be used simply as a tool for constructing keyframe meshes to be used in conjunction with linear interpolation.

However, the second goal of this project was to develop an animation system with a direct coupling to the pose editing system just described.

In Lewis et al. [6] a pose space is defined as the space spanned by the variations of the controls needed to specify the various poses. In the current context, our controls are simply the transformations imposed on the handles.

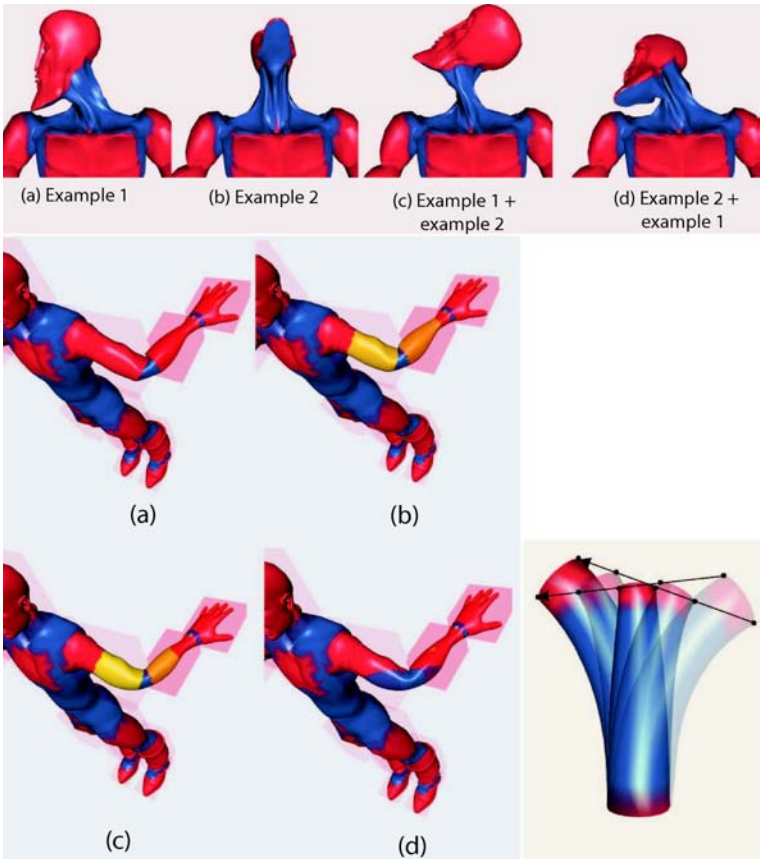

Fig. 2. This figure shows that examples do not commute (top row), and that if we use semirigid vertices we obtain different results. Here weights of 1, 2/3, 1/3, 0 were used (bottom left). Finally, simply interpolating linearly between examples does not suffice (bottom right).

Our goal is not only to interpolate between poses but also to find the best pose to match a given constraint. In this respect our work is similar to that of Sumner et al. [10]. However, the strategy is different. Sumner et al. are not concerned with how the pose is obtained whereas we employ the transformations used to define the pose when interpolating in pose space. Moreover, they constrain individual vertices whereas we constrain handles.

An issue confronted when interpolating in pose space is the fact that for large rotations linear interpolation simply does not work as shown in Fig. 2 bottom right. This means that we cannot just linearly interpolate between the vertex positions. Nor can we perform linear interpolation between transformation matrices in a meaningful fashion. This last problem has actually been addressed by Alexa [15] who proposed a scheme for meaningful commutative addition and scalar multiplication of transformations (rotations, scalings, and translations) represented as 4×4 matrices. This scheme could have been used, but since we have the benefit of knowing the actual transformations, in particular the rotation angles and axes, we decided instead to perform pose interpolation in the following fashion. For a given handle h we compute the combined transformation matrix, M^h, as follows

$$\mathbf{M}^h = \prod_{i=1}^{P} \prod_{j=1}^{N} \mathbf{T}_{i,j}^h(t^i) \tag{4}$$

where P is the number of poses, N is the number of transformations of handle h, and $T_{i,j}^h(t^i)$ is a function mapping the pose weight, t^i, to a 4×4 transformation matrix representing the j'th transformation of the the i'th pose scaled by t^i. For instance, if $T_{i,j}^h(1)$ represents rotation about a given axis with angle θ, $T_{i,j}^h(t^i)$ is simply a rotation about the same axis by the angle $t^i\theta$. Thus, given two poses with weight 0.5 we construct for each handle the transformations corresponding to half the pose transforms and concatenate the result. Since matrix multiplication does not, in general, commute the order of transformations for each pose is significant as is the order of poses when applying (4). This is illustrated in Fig. 2 top.

In our pose interpolation system, the user can grab any point on a handle and simply drag it. The system then performs an optimization in order to find an interpolated pose such that in the transformed pose, the selected point, $\mathbf{p}$ will match the position to which it has been dragged $\mathbf{p}'$. Essentially, the problem is to find a set of pose weights t^i such that

$$\|M_h\mathbf{p} - \mathbf{p}'\| \tag{5}$$

is minimized. Moreover, we should minimize for several constraints contemporaneously.

4.1 Optimization

There are two steps to this minimization, namely picking the best order of the poses and picking the best weights. Assume we are given an order, we simply need to minimize the target distances as discussed above. Unfortunately, (5) does not suffice in itself since there is no penalty for a pose which does not contribute to reaching the target. In other words, if the foot is constrained, adding in a pose which raises an arm is not going to increase the energy. Hence, we have experimented with a number of energy terms to add to (5). The most useful are 1: Sum of weights, 2: One minus sum of weights, 3: Distance to example pose, 4: Distance from last interpolated pose

1 often gives good results but effectively keeps us close to the rest pose which is often not desired. Thus, we recommend 2 which keeps the sum of weights close to a division of unity. If the pose space is dense and we wish to remain close to the examples, 3 is useful. Finally, 4 basically constrains the solution to be close to the previous solution. If we are performing an animation, this term avoids discontinuities.

A number of strategies were considered for performing the actual optimization. Bearing in mind that the energy functional is relatively complex and consists of heterogeneous terms, we decided to use a simple method (inspired by Hooke and Jeeves [16]) which only requires evaluation of the energy for various combinations of the weights: From the initial value, the weight of the first pose is iteratively increased halving the step length if no improvement is made, until the step size is below a given threshold. If increasing did not work, decreasing is attempted. Once a best weight has been found, the next pose is tried. This can be seen as a simple hill climbing method which we found to work well in practice.

Only one iteration is used to ensure program responsiveness. However, when the constraint position is moved, we rerun the optimization procedure starting from the previously found weights since they are almost always a good starting guess.

The final ingredient is the ordering of poses. We choose the simple greedy strategy of ordering the poses by the distance of the pose (by itself) to the target constraints. Note that in a polished system based on our method, the user would be exposed to few, if any, of the choices mentioned above.

5 Results and Discussion

In this paper, we have proposed an effective method for pose editing and animation which is entirely based on painting rigidity (i.e. handles) and placing centers of rotation. This is arguably much simpler than traditional skinning. Performancewise, the system can handle fairly large models at interactive frame rates. In Table 5 (left) we have listed the precomputation time and the time it takes to actually carry out the Laplacian editing for a range of model sizes. Note that even for a mesh of 45000 vertices, it takes less than a second total to compute the deformed pose given handle positions. These numbers seem to be about the same as what Sumner et al. report for their MeshIK system, but notice that the numbers cited there is for one iteration of their linear solver, and six iterations are generally needed for the solution to the nonlinear problem [10].

A particularly strong point of our method is the fact that it combines, in a homogeneous fashion, rotational motion (which is simple in bones based systems) with translational deformations. In Fig. 3 top, a bent leg is shown. On the left,

Table 1. These tables show performance numbers for deformation and the frame rate for an animation

Deformation

Model	Vertices	Precomp. (s)	Per frame (s)
Boba Fett (small)	2138	0.09	0.04
Boba Fett (medium)	10600	0.52	0.18
Armadillo	17300	1.04	0.30
Boba Fett (large)	45000	3.45	0.83

Animation

	Example poses		
handles	8	12	20
18	17.4	15.4	12.4
24	17.2	14.9	11.7
	fps		

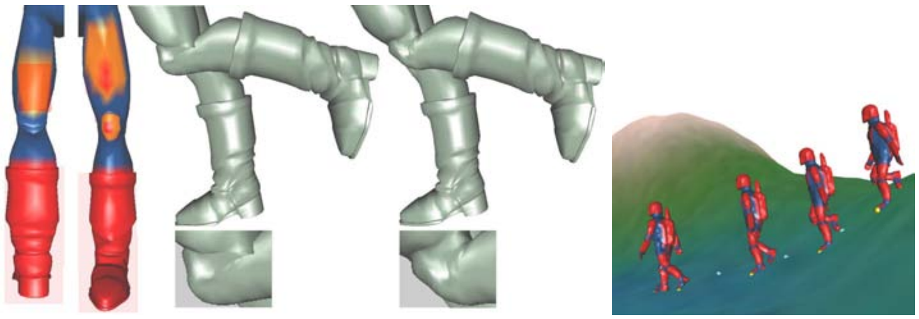

Fig. 3. Top: Using bones it is tricky to achieve an artifact free bent knee. On the left, we have shown the handles. The middle figure shows what happens if we simply rotate the lower leg. On the right is the result after some tweaking of the knee and uper leg handles. Bottom: A walk animation.

the leg is simply bent by rotating the lower leg. This leads to self intersections and an unnatural looking knee. Through minor adjustments (mostly translations) a much more natural pose is obtained. It is not clear how these adjustments could have been made in a purely bones based system.

Table 5 right contains the frame rates for animation of the small Boba Fett model (2138 vertices) which is shown walking down a grassy slope in Fig. 3 bottom. The animation is done simply by placing constraints on the feet of the model and then alternately moving the left and right constraint position. The balls indicate constraint positions. Note that for each frame, the optimal pose is found; the handles are transformed accordingly, and finally Laplacian editing is used to deform the model.

Our method leads to a homogeneous paradigm for animation and a simple workflow for the animator. Unsurprisingly, these advantages do not come for free. Unlike the simple matrix blending used in bones systems, which is easily implemented on a GPU, we need to solve a (possibly large) linear system for each frame. For this reason it may not be directly feasible to use this form of animation in a game, but it would be possible to sculpt keyposes and use the pose interpolation to create a dense set of keyframes which would be very suitable for real-time animation. One might also envision our system being used for actual animation in a non-real time setting our in a real time setting in a few years if there is a sufficient advance in the speed at which we can solve linear systems. For instance, one direction of future investigation would be to offload the solution of the linear system to graphics hardware.

References

1. Wellman, C.: Inverse kinematics and geometric constraints for articulated figure manipulation. Master's thesis, Simon Fraser University (1993)
2. Fernando, R., Kilgard, M.J.: The Cg Tutorial, The Definitive Guide to Prgrammable Real-Time Graphics (Chapter 6, Section 9.1). Addison-Wesley, London (2003)

3. Kavan, L., Zara, J.: Spherical blend skinning: a real-time deformation of articulated models. In: SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games, New York, USA, pp. 9–16. ACM Press, New York (2005)
4. Kry, P.G., James, D.L., Pai, D.K.: Eigenskin: real time large deformation character skinning in hardware. In: SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 153–159. ACM Press, New York (2002)
5. Mohr, A., Gleicher, M.: Building efficient, accurate character skins from examples. ACM Trans. Graph. 22(3), 562–568 (2003)
6. Lewis, J.P., Cordner, M., Fong, N.: Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In: SIGGRAPH '00. 27th annual conference on Computer graphics and interactive techniques, New York, USA, pp. 165–172. ACM Press/Addison-Wesley, New York (2000)
7. Sorkine, O.: Differential representations for mesh processing. Computer Graphics Forum vol. 25(4) (2006)
8. Maciejewski, A.A.: Motion simulation: Dealing with the ill-conditioned equations of motion for articulated figures. IEEE Comput. Graph. Appl. 10(3), 63–71 (1990)
9. Grochow, K., Martin, S.L., Hertzmann, A., Popović, Z.: Style-based inverse kinematics. ACM Trans. Graph. 23(3), 522–531 (2004)
10. Sumner, R.W., Zwicker, M., Gotsman, C., Popović, J.: Mesh-based inverse kinematics. ACM Trans. Graph. 24(3), 488–495 (2005)
11. Sorkine, O., Lipman, Y., Cohen-Or, D., Alexa, M., Rössl, C., Seidel, H.P.: Laplacian surface editing. In: Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing, Eurographics Association, pp. 179–188. ACM Press, New York (2004)
12. Lipman, Y., Sorkine, O., Levin, D., Cohen-Or, D.: Linear rotation-invariant coordinates for meshes. In: P.o.A.S. (ed.) Proceedings of ACM SIGGRAPH 2005, pp. 479–487. ACM Press, New York (2005)
13. Lipman, Y., Sorkine, O., Cohen-Or, D., Levin, D., Rössl, C., Seidel, H.P.: Differential coordinates for interactive mesh editing. In: Shape Modeling International 2004. (2004)
14. Botsch, M., Bommes, D., Kobbelt, L.: Efficient linear system solvers for mesh processing. In: Martin, R., Bez, H., Sabin, M.A. (eds.) Mathematics of Surfaces XI. LNCS, vol. 3604, pp. 62–83. Springer, Heidelberg (2005)
15. Alexa, M.: Linear combination of transformations. ACM Trans. Graph. 21(3), 380–387 (2002)
16. Hooke, R., Jeeves, T.A.: Direct search solution of numerical and statistical problems. Journal of the ACM 8, 212–229 (1961)