Efficient Non-Linear Editing of Large Point Clouds

Fabian Aiteanu Patrick Degener Reinhard Klein

Universität Bonn
Institut für Informatik II - Computergraphik
D-53117 Bonn, Germany
{aiteanu, degener, rk}@cs.uni-bonn.de

ABSTRACT

Editing 3D models is often performed on triangular meshes. We generalize editing operations based on differential coordinates to work on point clouds without explicit connectivity information. This allows a point cloud to be interpreted as a surface or volumetric body upon which physically plausible deformations can be applied. Our multiresolution approach allows for a real-time editing experience of large point clouds with 1M points without any offline processing. We tested our method on a range of synthetic and real world data sets acquired by laser scanner. All of them were interactively editable and produced intuitive deformation results within few minutes of editing.

Keywords: Point clouds, non-linear editing, real-time, differential coordinates, subsampling.

1 INTRODUCTION

Virtual 3D objects are a common asset for movies and games. In former years those were predominantly created manually, but in later years it has become more common to scan real objects and then edit and modify their virtual counterparts. For some application areas, e.g. structured light scanning, data is available only as point clouds, though editing 3D models is predominantly performed on triangular meshes. Despite ongoing efforts in the area of surface reconstruction, triangulating point sets remains computationally demanding and is error-prone in the presence of noise and outliers. Additionally, point clouds are usually sampled much denser than meshes and thus contain amounts of data, which are several times larger than for comparable meshes. This in turn can negatively influence the interactivity of editing operations.

To address this problem specialized editing approaches for point sampled models have been proposed. However, to handle the larger complexity of point clouds, these methods resort to relatively simple deformation models like transformation interpolation or linear models based on differential coordinates. As recently analysed by Sorkine and Botsch [SB09], a common problem of these simple deformation models is a counterintuitive deformation behaviour. For deformation of triangle meshes this problem is well known, and subsequently non-linear deformation models have been developed that do not show these problems. The higher intuitivity though comes at the cost of a significantly higher computational effort.

Space deformations are a different approach towards editing operations. They can conceptually be calculated quickly, but lack the ability to respond to the structure of the edited model. Points which are far apart in geodesic distance can be close in space and thus be influenced by editing operations unintentionally. To overcome this limitation, cage-based techniques try to separate such geodesically distant regions from each other, but the user is required to create or at least adjust the cage manually. In contrast, our solution does not require any manual intervention.

We present a novel method to enable direct real-time editing of point clouds without the need for prior

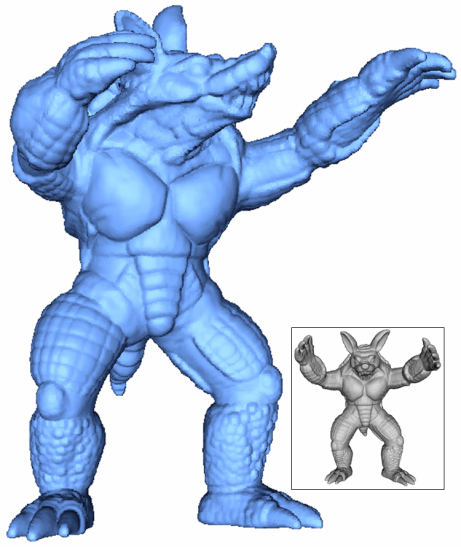

Figure 1: Armadillo model after 3 editing steps: head has been twisted by 45°, left arm twisted by 90°, right arm bent by 30°.

meshing. Our method shows intuitive, physically plausible deformation behaviour and does not require manual preprocessing like cage construction. It is based on geodesic distances only and avoids the problems of space deformation based approaches described above. Using our method, large point sampled geometry (Figure 1) can be edited as a whole at interactive frame rates.

In summary our contribution is this:

- We present an approach for editing point clouds while preserving local surface details. Due to an underlying non-linear deformation model, the obtained deformations are physically plausible, even for large handle transformations.
- Our algorithm does not pose any specific assumptions upon the structure of the point cloud being edited. The input requires solely the vertex positions in 3D space, but no explicit connectivity information. The point cloud may be irregularly sampled or contain outliers. It is even possible to handle point sampled volumes.
- As point clouds are usually sampled much denser than comparable meshes, we present a means of performing the deformation calculations on a coarse scale and transfer the results to the fine scale representation of the point cloud at interactive frame rates. To the best of our knowledge, no other system has been presented which could perform similar non-linear deformations without offline processing.

1.1 Related Work

As mentioned in the introduction, several methods exist for editing meshed input data, but which are not directly applicable to point clouds. In the following we concentrate on methods targeting point clouds.

The challenge of editing point clouds interactively has been pursued in a number of papers. In an early work of Pauly et al. [PKKG03] they deform point clouds in real-time using linear interpolations between the handles. Miao et al. [MFXP08] use a more sophisticated method with differential coordinates. Like all linear methods they both produce counterintuitive results for large deformations.

Especially for editing large models the required storage can exceed the available main memory, or the computation time can become prohibitively high. It is thus often necessary to use a simplified model upon which the editing is performed. Wand et al. [WBB+07] describe a method to visualize data sets with a size of several gigabytes in a multi-resolution out-of-core method. The direct editing operations are limited to simple translations or deletions of vertices. More complex operations can only be performed in offline computations. Boubekeur et al. [BSS07] use a streaming method to perform an offline simplification of a large mesh into a smaller point cloud, which can then be edited interactively. A second offline streaming step applies the modifications to the original mesh.

Wicke et al. [WSG05] use the concept of thin shells to construct a network of so-called fibres on the surface of a point cloud, which are then used to model and calculate possible deformations. The fibres provide a mesh representation at a coarse scale, and the deformations are later applied to the detailed representation. As both of the methods of Boubekeur and Wicke require time-consuming steps before and after the user performs the actual editing, the user has to wait for the post-editing steps to complete in order to view the final result. We deem this unsuitable for ad-hoc editing, because any corrections or further editing steps require repeating the whole procedure.

In our method, the actual editing of the point cloud is performed on a differential representation of its surface. This has recently gained much attention in the context of mesh editing. Sorkine and Botsch [SB09] compare gradient-based methods and Laplacian-based methods as the predominant methods used for differential representation-based deformations. They identify two classes: linear methods can provoke a counterintuitive behaviour for deformations, since they cannot handle both rotations and translations of local frames simultaneously, resulting in artifacts of different kinds. Non-linear methods [BPGK06, SA07] solve these issues, but are time-consuming to compute. The mesh editing framework presented by Paries et al. [PDK07] represents local frames using quaternions. By enforcing additional frame constraints they produce intuitive results for translations, rotations and scaling. Their non-linear approach is computationally demanding and thus applicable only to medium sized meshes. We generalize their approach to be applicable to large point clouds.

Space deformations (see e.g. [HSL+06, XZY+07, BPWG07, AOW+08]) are a different approach towards editing operations. They do not directly manipulate the points' positions, but warp the space and thus indirectly manipulate the points. The space deformations can conceptually be calculated quickly, but they lack the ability to respond to the structure of the edited model. Points which are far apart in geodesic distance can be close in space and thus be influenced by editing operations unintentionally. To overcome this limitation, Huang et al. [HSL+06] used a control mesh enclosing the model to separate such geodesically dis-

tant regions from each other. In their approach the user is required to define the control mesh. In contrast, our solution does not require any manual intervention.

The multiscale representation introduced by Pauly et al. [PKG06] and extended by Duranleau et al. [DBP08] encodes the displacement of points between the representations at different scales. It allows to edit the representation for a specific detail level using a space warping function and then apply the deformation to the other detail levels. This enhances interactivity and controllability of the result, but still has the mentioned limitations of space deformations.

Meshless methods, of which space deformations are one specialization, provide further aspects. The phyxels proposed by Müller et al. [MKN+04] are used to fill a volume and preserve it during editing operations. The small amount of phyxels required to fill a volume, allow for physically plausible deformations which can be calculated quickly. However, as Müller et al. already mention, such volumetric approaches are not applicable to point clouds representing a surface.

1.2 Overview

First we present a brief explanation of the mesh editing framework of Paries et al. [PDK07] in Section 2 as it constitutes the basis upon which our work is built. Our generalization to point clouds follows, including the concepts developed to use the mesh editing framework without the need for meshing the input data. In Section 3 we explain our multiresolution method, which allows editing point cloud data sets which are considerably larger than comparable meshes. Although the amount of data points may be up to two scales of magnitude higher, our parallel GPU implementation still provides several frames per second during editing operations. We conclude this paper with an overview of the results (Section 4) and our planned extensions for the future (Section 5).

2 EDITING

2.1 Mesh Editing

As the underlying model for a mesh Paries et al. [PDK07] employ a differential representation of local surfaces. For each vertex in the region of interest $x_i \in R$ they define an orthonormal local frame $F_i = (t_i^1, t_i^2, N_i)$ with right hand orientation. F_i can be interpreted as a rotation matrix and thus be expressed in terms of a unit quaternion q_i.

For each pair of adjacent vertices (i, j) in the mesh M the difference between the quaternions is calculated as $q_i^j = \bar{q}_i \cdot q_j$. During editing and user interaction the mesh surface can be reconstructed from the differential representation and the equation

$$q_i \cdot q_i^j = q_j \tag{1}$$

which is a linear system. Fixing a single unit quaternion q_{i_0} is sufficient for getting a unique solution (except for its rotation), by iteratively solving the remaining equations.

Reconstructing the mesh solely based on the local frames and ignoring the geometry can lead to unintuitive results, because translations are not accounted for. To overcome this, Paries et al. added constraints which impede a change of coordinates of 1-ring neighbors in the local frames F_i. They formulate the constraints as

$$q_i(1, c_i^j)^t \bar{q}_i = (1, x_j - x_i)^t \tag{2}$$

where $c_i^j := F_i^{-1}(x_j - x_i)$ are the local coordinates of x_j in frame F_i.

User input consists of a set of handle vertices H which specify additional frame constraints q_k^{const} and positional constraints x_k^{const}. Given those two sets of constraints, the frame differences q_i^j and the local coordinates c_i^j, the constrained reconstruction problem is to find a set of local frames q_l and absolute vertex coordinates x_l for all vertices within the region of interest R that satisfy Equations (1), (2) and

$$q_k = q_k^{const} \qquad x_k = x_k^{const} \qquad \forall k \in H. \tag{3}$$

As the resulting equation system is overconstrained, Paries et al. solve it in least squares sense using a non-linear optimization procedure. They alleviate the complexity issue by parting the original non-linear problem into several linear equation systems of which the LU-matrices remain constant throughout one editing step. Thus after specifying a region of interest and editing handles, a precomputation step factorizes the original matrices in parallel on the GPU. During the actual editing the linear equation systems are solved on the CPU with backsubstitution, which can be performed at several frames per second even for medium sized meshes with up to 100k vertices.

2.2 Editing Point Clouds

The intention to seek for a generalization of the editing operations from mesh data structures to arbitrary point clouds, comes from the observation that in practice raw data from range scanning devices often is available only in form of 3D vertex coordinates. Depending on the device, color information may also be available, but it plays only a minor role for editing operations.

Though advanced triangulation techniques like MLS surface approximation exist, they are usually dependant upon a specific structure of the point cloud, e.g. a closed surface. Margins, outliers, or in general arbitrarily distributed points pose severe problems. To be more robust, most methods have the undesired effect of smoothing the surface represented by the point cloud, thus degrading the quality of the data set. Furthermore, the computation time for the triangulation becomes notably high for large data sets.

Although the editing framework of Paries et al. relies upon mesh data structures to ensure connectivity of the vertices and to find a vertex's 1-ring, the basic equation systems are formulated without the need for an explicit mesh. They do though require the definition of a local neighborhood in order to calculate the local frames. The simplest choice for a local neighborhood, which comprises each vertex's nearest neighbors in 3D space, is sufficient for our needs and very fast to calculate.

In our approach we use the k nearest neighbors of each vertex, which are computed in a pre-processing step. To determine the nearest neighbors, we use an octree to partition the point cloud, which can be calculated in $O(n \log n)$ time. The choice of k was experimentally determined, and may theoretically be any $k \geq 2$, since we need at least two neighboring points to define the local frame for x_i. In practice, choosing k too small, like e.g. $k = 2$ leads to line-shaped disconnected components, which are not treatable well as a surface by the later algorithm steps. On the other hand, choosing k too large linearly slows down all calculations which iterate the nearest neighbors. We found $k = 5$ or $k = 6$ to provide a good tradeoff between speed and stability, and it is also the average number of neighbors in a regular triangulation.

Using only the initially found nearest neighbors for each vertex can lead to non-symmetric relations, especially if the input point cloud is irregularly sampled. This can prevent a complete traversation of the graph induced by the nearest neighbors, and split it into several seemingly disconnected components. Although removing the non-symmetric neighbors reduces the amount of data to process, it may lead to more fragmentation. Our choice of inserting the missing links to create bi-directional nearest neighbors relations counteracts the possible fragmentation. At the same time it ties outliers to the main connected component(s). The problem with auxiliary connected components which do not include user-defined constraints is that they render Equation system (1)-(3) underdetermined, and thus provide no stable solution. During the preprocessing step these unconstrained components are detected and removed from the editable part of the model.

2.3 Area and Volume Preservation

Volume preservation is a key feature used in mesh editing and has also been applied to point cloud editing [MKN+04]. It is however not unambiguous how to define this property, since a point cloud does not possess an inherent volume, as opposed to a closed mesh. This is especially the case for point clouds originating from single-image 3D capturing devices, which can only capture one side of any given object at a time. As our method strives to preserve the local neighborhood of each point, the volume is only an affected property, not one which is calculated or optimized.

In particular, if a surface is stretched using handles on opposite sides, the surface is also enlarged in perpendicular direction to preserve the local shape of each point's 1-ring. This is a rather counterintuitive behaviour, since most materials in nature exhibit the opposite behaviour of shrinking in perpendicular direction when stretched in the other direction (e.g. rubber and metal). Nevertheless, auxetic materials exist, which do enlarge in perpendicular direction (e.g. special foams), and others which do not change their perpendicular size at all (e.g. cork). Our method can be parameterized to exert any one of these elasticity behaviours by scaling the local frames q_i in each iteration before solving the equation systems, as detailed in [PDK07].

3 SAMPLING

The matrix factorization step performed during precomputation is necessary to reduce the calculation time during the actual editing of a model. We employed the SuperLU library [DEG+99] which is suited well to decompose the sparse matrices. Although the matrix size grows squarely with the number of handle vertices, the sparsity leads to a computation effort which grows below quadratic. Nevertheless, from 100k vertices onwards this requires several minutes of precomputation and eventually the matrix size grows too large to be handled in main memory.

Since complex operations on large data sets are today still limited in their speed by the available computation power, we devised a multiresolution scheme similar to Wicke et al. [WSG05]. The time-consuming solving of the Equation systems (1)-(3) is performed on a coarse-scale subsample of the input point cloud, and the fine-scale representation is then interpolated. Pauly et al. [PGK02] survey different point based simplification methods. For our subsampling we chose a method from the category of hierarchical clustering methods, since they adapt well to point clouds with varying point sampling densities. To generate the subsamples, we use an octree partitioning the original

point cloud. Taking the sample points from each leaf of the octree allows us to handle arbitrary point clouds, not only those representing implicit surfaces. At the same time, areas of the original point clouds which have a high sampling density will receive a higher amount of sample points, thus preserving local details.

Calculating the nearest neighbors for the sample points can be performed with the octree that was already calculated. For some models like the dragon model this can yield undesired connections between points (Figure 2b), which cannot be considered adjacent in the original, but become adjacent in the sampled point cloud due to the lower point density. In the work presented by Xu et al. [XZY+07] the user is required to manually create a control mesh which prevents such connections between originally unconnected parts. Our connected sample points are in effect similar to the control mesh used by Xu et al., but we sought for an alternative working without manual intervention. The idea is to find the nearest neighbors in a geodesic sense instead of the euclidean distance. As the geodesic distance conceptually requires a surface to be applicable, using the nearest neighbors relation of the original point cloud satisfies this need, but still makes it applicable to non-surface-like point clouds. With a breadth first search (Algorithm 1) for each point in the original point cloud, the geodesic nearest neighbors can be found in $O(n \cdot r)$ time, where n is the number of original points and r is the original-to-sample number ratio. The result can be seen in Figure 2c.

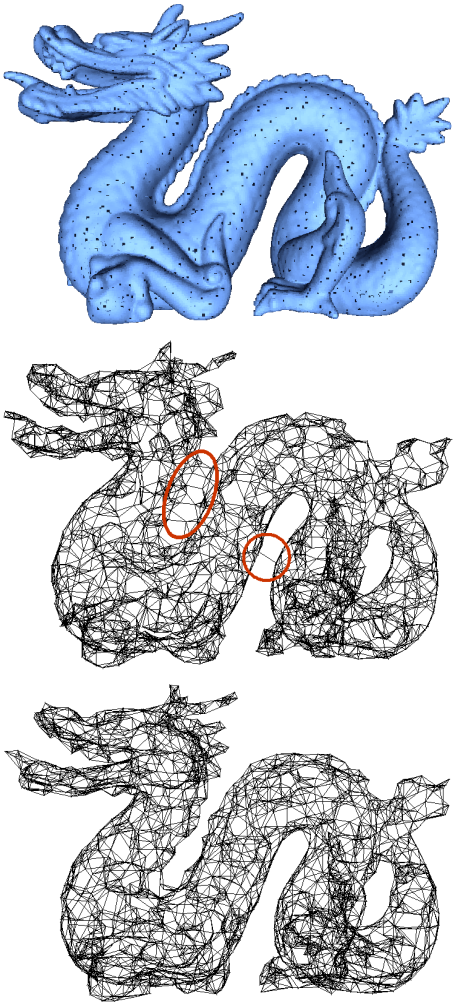

Figure 2: a) Original point cloud with sample points drawn in black. b) Sample points connected using nearest neighbors. Wrong connections between originally separated parts are created due to proximity. c) Calculating nearest neighbors on the original point cloud with a breadth first search prevents wrong connections.

After each iterative solving step for the basic equations, the new position p_i' and normal n_i' of each original point is interpolated from the translations and rotations of its nearest sample points:

$$p_i' = \sum_{s \in neighbors(i)} w_i^s \cdot (q_s \cdot p_i^s + p_s) \tag{4}$$

$$n_i' = \sum_{s \in neighbors(i)} w_i^s \cdot (q_s \cdot n_i^0) \tag{5}$$

with

$$w_i^s = \frac{1}{|p_i^s|^2} \Big/ \sum_{s \in neighbors(i)} \frac{1}{|p_i^s|^2} \tag{6}$$

where $p_i^s = p_i^0 - p_s^0$ is the coordinate of point p_i in the local coordinate system of p_s, and q_s is the quaternion describing the rotation of s from its original to its new orientation. The weights w_i^s account for a smaller influence of neighbors which have a greater distance.

We implemented this interpolation method on the CPU first, but the involved quaternion multiplications proved to be a limiting factor to the achievable

Input: point i_0; number of sample neighbors to find k; Samples S; Original nearest neighbors relation N
init queue $q \Leftarrow i_0$
init geodesic neighbors $G(i_0) = \emptyset$
repeat
  $j = q.dequeue()$
  for all $n \in N(j)$ **do**
    if n not visited **then**
      $q.enqueue(n)$
      mark n as visited
  if $j \in S$ **then**
    add j to $G(i_0)$
until $|G(i_0)| \geq k$ or $q = \emptyset$

Algorithm 1: Geodesic nearest sample neighbors for a point

Model	# points	# sample points	Subsampling	Precomputation	Equation solving	Upsampling
Bunny	35k	100	4.0	0.1	0.001	0.003
Bunny	35k	1000	0.5	2.0	0.019	0.004
Armadillo	172k	200	101.3	0.1	0.002	0.016
Armadillo	172k	5000	3.2	18.3	0.098	0.018
Dragon	437k	200	390.2	0.1	0.002	0.037
Dragon	437k	5000	13.7	23.3	0.116	0.040
Plain	1000k	200	1923.0	0.1	0.002	0.083

Table 1: Computation times in seconds on a 2.4GHz CPU and a NVidia GeForce 8800 GTX GPU. Subsampling and precomputation are executed once, while equation solving and upsampling occur each frame.

speed, since multiplying two quaternions includes 16 floating point multiplications, and multiplying a quaternion with a 3D vector includes 27 floating point multiplications. The sheer amount of operations required to perform the interpolations for large data sets with over 1M points does not allow for a true real-time editing experience using this approach, as the frame rates can drop to 1-2 frames per second. The implementation of the upsampling step in CUDA for parallel computation on the GPU is straightforward, since p_i' and n_i' can be computed independently for each point on the available GPU processors. Our current implementation running on an off-the-shelf graphics card (NVidia GeForce 8800 GTX) performs on average 5-8 times faster than the CPU version. During the editing operation, for 1M points we measured 8 frames per second for the GPU implementation, while the CPU variant yielded only 1.4 frames per second.

4 RESULTS

We tested our method on a variety of point clouds, including both artificial and scanned models. All of the models have been used without manual preprocessing. Table 1 gives an overview of the computation time for setup and each iteration step during the editing phase. Usually the model converges to its final shape during 3-5 iterations.

As we have explained before, the equation solving time consumption grows slower than $O(n^2)$ where n is the number of sample points, while the upsampling step requires only linear time depending on the number of points to be interpolated.

The visual quality of the result is largely independant of the number of sample points, unless it falls below a certain threshold depending on the complexity of the model. This means e.g. for the Stanford Bunny (Figure 3), which does not have a complex structure, that the resulting point positions have very small deviations when calculated with 100, 1,000 or 10,000 sample points. Only with fewer than about 100 sample points the upsampling step produces artifacts for large deformations (Figure 4). For the Dragon model (Figure 5), which is more complex due to its winding body, sampling artifacts can be observed for less than 500 points (Figure 4). Those artifacts could be circumvented by a more sophisticated interpolation scheme which does take into account not only the nearest neighboring sample points, but a selection of sample points which are evenly distributed on the surface around the point to be interpolated. A too low sample density does however defeat the purpose of our nonlinear editing method by reducing it to the linear interpolation.

One minor drawback of our implementation using quaternions is the inability to perform handle rotations of more than 360° within one editing step. This is due to the normalization of quaternions which we perform for stability reasons when solving the equation systems. In practice however this is not of concern, as it is possible to perform several editing steps with smaller rotations to achieve the desired result.

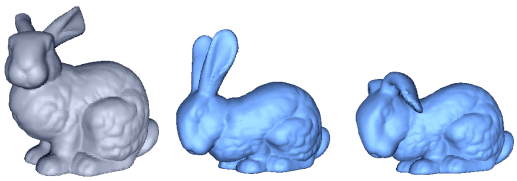

Figure 3: Bunny model with 2 editing steps.

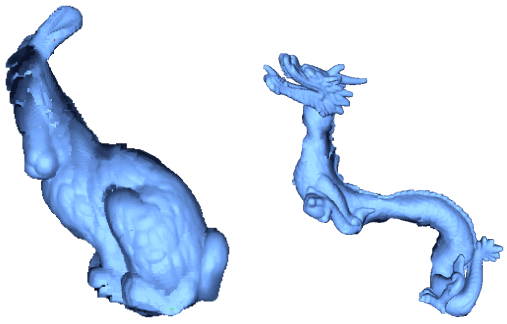

Figure 4: Bunny and Dragon models edited with only 50 sample points each. Sampling artifacts can be observed in the head area of the Bunny and neck of the Dragon.

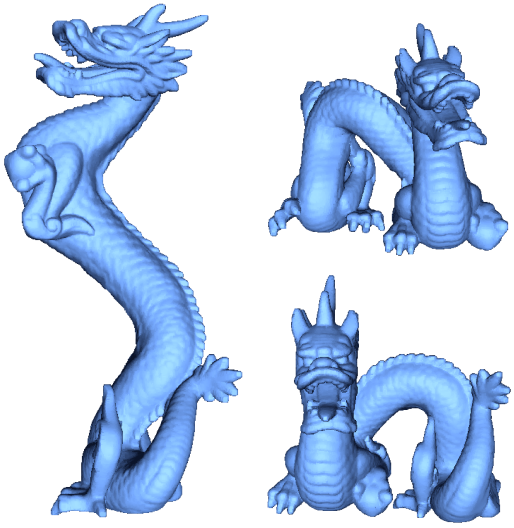

Figure 5: Dragon model after rotation and translation of the head. Session time including subsampling and precomputation for each result: 1 minute.

5 CONCLUSIONS AND FUTURE WORK

We have presented a novel method to interactively edit large point clouds. Using a non-linear deformation model allows to produce physically plausible deformations even for large modifications. The multiresolution approach faithfully handles the coarse scale deformations while the model details are preserved. Our parallel implementation provides the speed necessary for real-time editing scenarios, shortening the time required to produce a desired result. For models which are larger than the available main memory, our method could be extended with a further sampling level, for which the upsampling step would be performed offline. Our method does not require any manual preconditioning steps to create a control mesh, and is thus suited for direct editing of arbitrary point clouds.

For the future we plan to incorporate the deformation features into a model reconstruction framework which can work on a number of scanner-acquired time-varying point clouds.

REFERENCES

[AOW+08] B. Adams, M. Ovsjanikov, M. Wand, H. Seidel, and L. Guibas. Meshless modeling of deformable shapes and their motion. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 77–86, Dublin, Ireland, 2008. ACM/Eurographics, Eurographics Association.

[BPGK06] M. Botsch, M. Pauly, M. Gross, and L. Kobbelt. Primo: coupled prisms for intuitive surface modeling. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 11–20. Eurographics Association, 2006.

[BPWG07] M. Botsch, M. Pauly, M. Wicke, and M. Gross. Adaptive space deformations based on rigid cells. *Computer Graphics Forum*, 26(3):339–347, 2007.

[BSS07] T. Boubekeur, O. Sorkine, and C. Schlick. Simod: Making freeform deformation size-insensitive. In *IEEE/Eurographics Symposium on Point-Based Graphics 2007*, September 2007.

[DBP08] F. Duranleau, P. Beaudoin, and P. Poulin. Multiresolution point-set surfaces. In *GI '08: Proceedings of graphics interface 2008*, pages 211–218, Toronto, Ont., Canada, Canada, 2008. Canadian Information Processing Society.

[DEG+99] J. Demmel, S. Eisenstat, J. Gilbert, X. Li, and J. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Analysis and Applications*, 20(3):720–755, 1999.

[HSL+06] J. Huang, X. Shi, X. Liu, K. Zhou, L. Wei, S. Teng, H. Bao, B. Guo, and H. Shum. Subspace gradient domain mesh deformation. *ACM Trans. Graph.*, 25(3):1126–1134, 2006.

[MFXP08] Y. Miao, J. Feng, C. Xiao, and Q. Peng. High frequency geometric detail manipulation and editing for point-sampled surfaces. *Visual Computer*, 24(2):125–138, 2008.

[MKN+04] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa. Point based animation of elastic, plastic and melting objects. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 141–151, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[PDK07] N. Paries, P. Degener, and R. Klein. Simple and efficient mesh editing with consistent local frames. Technical Report CG-2007-3, Universität Bonn, July 2007.

[PGK02] M. Pauly, M. Gross, and L. Kobbelt. Efficient simplification of point-sampled surfaces. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 163–170, Washington, DC, USA, 2002. IEEE Computer Society.

[PKG06] M. Pauly, L. Kobbelt, and M. Gross. Point-based multiscale surface representation. *ACM Trans. Graph.*, 25(2):177–193, 2006.

[PKKG03] M. Pauly, R. Keiser, L. Kobbelt, and M. Gross. Shape modeling with point-sampled geometry. *ACM Trans. Graph.*, 22(3):641–650, 2003.

[SA07] O. Sorkine and M. Alexa. As-rigid-as-possible surface modeling. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 109–116, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[SB09] O. Sorkine and M. Botsch. Tutorial: Interactive shape modeling and deformation. In *Eurographics*, 2009.

[WBB+07] M. Wand, A. Berner, M. Bokeloh, A. Fleck, M. Hoffmann, P. Jenke, B. Maier, D. Staneker, and A. Schilling. Interactive editing of large point clouds. In Baoquan Chen, Matthias Zwicker, Mario Botsch, and Renato Pajarola, editors, *Symposium on Point-Based Graphics 2007 : Eurographics / IEEE VGTC Symposium Proceedings*, pages 37–46, Prague, Czech Republik, 2007. Eurographics Association.

[WSG05] M. Wicke, D. Steinemann, and M. Gross. Efficient animation of point-sampled thin shells. *Computer Graphics Forum*, 24(3):667–676, 2005.

[XZY+07] W. Xu, K. Zhou, Y. Yu, Q. Tan, Q. Peng, and B. Guo. Gradient domain editing of deforming mesh sequences. *ACM Trans. Graph.*, 26(3):84, 2007.