A Volumetric Approach to Interactive Shape Editing

Carsten Stoll, Edilson de Aguiar, Christian Theobalt, Hans-Peter Seidel

MPI–I–2007–4–004 June 2007

Authors' Addresses

Carsten Stoll
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany

Edilson de Aguiar
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany

Christian Theobalt
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany

Hans-Peter Seidel
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany

Acknowledgements

This work was supported in part by AIM@SHAPE, a Network of Excellence project (506766) within EU's Sixth Framework Programme and by EC within FP6 under Grant 511568 with the acronym 3DTV.

The Raptor model is provided courtesy of SenSable Technologies inc. by the AIM@SHAPE Shape Repository. The Armadillo and Dragon models are provided courtesy of the Stanford Computer Graphics Laboratory. The optical motion capture data used for animating the models are provided courtesy of Eyes Japan Co. (www.mocapdata.com).

Abstract

We present a novel approach to real-time shape editing that produces physically plausible deformations using an efficient and easy-to-implement volumetric approach. Our algorithm alternates between a linear tetrahedral Laplacian deformation step and a differential update in which rotational transformations are approximated. By means of this iterative process we can achieve non-linear deformation results while having to solve only linear equation systems. The differential update step relies on estimating the rotational component of the deformation relative to the rest pose. This makes the method very stable as the shape can be reverted to its rest pose even after extreme deformations. Only a few point handles or area handles imposing an orientation are needed to achieve high quality deformations, which makes the approach intuitive to use. We show that our technique is well suited for interactive shape manipulation and also provides an elegant way to animate models with captured motion data.

Keywords

Interactive shape editing, Tetrahedral meshes

Contents

1 Introduction

In recent years, interactive shape deformation and editing has been a very active field of research. The goal is the development of algorithms that enable shape deformation in an intuitive and, more importantly, natural looking way under a given set of constraints. This usually means that the deformation of the given shape must be calculated in a physically plausible manner, i.e. it must satisfy the expectations the user has due to his experience with deforming objects in the real world. Correct physical simulation requires setting up and minimizing complex non-linear energies, which is computationally expensive and thus often only reasonable in non-interactive applications. Since users will usually not be happy with the first deformation they produce and will iteratively modify the deformation constraints until they are satisfied with the results, offline methods are not feasible for shape editing. To enable immediate feedback to the user and reach interactive editing performance it is necessary to use simpler and easy-to-compute deformation techniques that still behave plausibly.

Lately, a popular way to address this issue has been to employ simplified thin-plate and thin-shell representations which effectively model a shape as hollow object. Deformations are computed by representing an object as a triangle mesh and minimizing the stretching and bending energies on its surface under a given set of constraints. Many recent papers suggest to use the linearized discrete surface Laplacian as a base for their deformation techniques [29, 19, 34, 18, 35] or to directly minimize the non-linear energies as efficiently as possible [6, 25, 13].

While volumetric shape representations such as tetrahedral meshes or similar structures have been widely used in physical simulation [21], there are only a few shape editing methods that rely on this solid object representation [36, 6]. The benefit of a volumetric deformation framework is that it better prevents unintuitive shape transformations, such as local self-intersections of opposing surfaces. Furthermore, it enables distance preservation not only on an object's surface, but also throughout its interior, which makes the

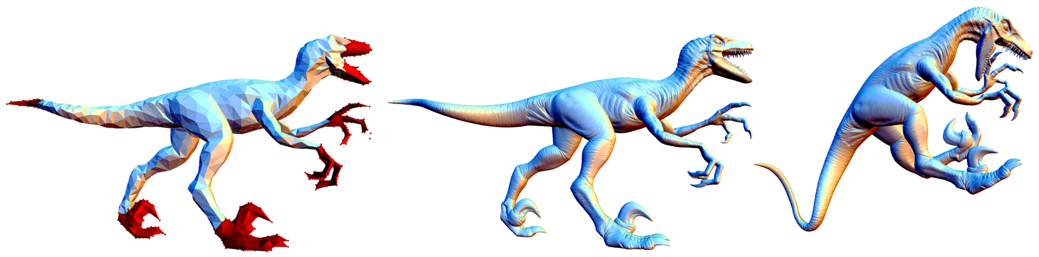

Figure 1.1: Example of a deformation of a raptor created with our method. From left to right: Reduced tetrahedral model with handles in red, original high resolution input model, deformed input model.

deformations resistant to changes in volume and cross-sectional areas.

Non-linear approaches can compute large shape deformations in one step. However, during an interactive editing session the user typically grabs a handle and moves it along a continuous path to its target position. In consequence, if the deformations can be computed quickly, only small changes to the constraints are necessary at every frame that is rendered.

We capitalize on these observations and propose a novel interactive shape editing approach that approximates non-linear deformation effects in real-time by iteratively solving linear problems, Sect. 2.3. Our method relies on a tetrahedral mesh representation of the input shape that can easily be generated for triangle-mesh objects. The proposed algorithm is highly robust and provides immediate visual feedback to the user controlling the deformation. During each pose update step, a first version of the transformed model is reconstructed by means of a linear tetrahedral Laplacian deformation. Since this first step is unable to influence the rotational components of the tetrahedra, a second step analyzes the output of the Laplacian deformation and updates the differential coordinates to compensate for the rotational invariance. Two optional steps improve the behavior of our method under rotational constraints given by the user and allow for the specification of rigid areas on the object.

In case of very large input triangle meshes, we employ decimated tetrahedral representations to maintain real-time frame rates. The detailed mesh in its target pose can be quickly generated from the deformed tetrahedral mesh by means of an efficient and stable free-form deformation approach, Sect. 2.4.

As shown in Sect. 3, our algorithm enables even unexperienced users to quickly generate plausible edited shapes. We also show on a variety of examples that the flexibility in constraint placement inside and outside the object makes the method well-suited for creating realistic mesh animations from captured real-world motion data.

1.1 Related work

Shape editing is an active field of research in computer graphics, and consequently a variety of different strategies were proposed to attack the problem. Early methods like free-form deformation [24] enable high-quality shape modeling, but typically fail to reproduce physically plausible transformation results if only a handful of manipulation constraints are used.

Physical plausibility, however, is a highly-desirable property, as it leads to a deformation behavior of the edited objects that a user is familiar with from real-world experience. Therefore, it has recently become very popular to model deformation by minimizing bending and stretching energies, which leads to thin-shell and thin-plate representations. For the sake of efficiency these non-linear energies are often linearized, for example as linear Laplacian or Poisson systems. A comprehensive survey of linear Laplacian techniques can be found in [28]. Most of these methods suffer from the problem that it is not possible to couple translations and rotations in a linear way. Therefore, most methods which use translational constraints exhibit insensitivity to rotations [29, 19], whereas methods relying on rotational constraints are insensitive to translations [34, 18, 35]. A method for deformation with varying stiffness has been propsed by [23]. Several methods have been proposed which try to remove this linear dependency by using multi-step approaches, like multi-scale decomposition [16, 4, 7], or skeleton based techniques [33].

Other approaches propose to solve the computationally expensive non-linear equations directly. By this means the above insensitivity problems can be prevented, but the price to be paid is often that interactive deformation is only feasible on models of reduced complexity. As an example, [13] use a subspace solver to reduce the dimension of the involved optimization problem. [25] proposes an non-linear differential coordinate setup, while [6] minimizes bending and stretching energies using a coupled shell of prisms. Instead of trying to solve the non-linear equations directly, we approximate non-linear deformation behavior in real-time by solving a sequence of linear equation systems.

Our shape editing approach also targets similar applications like previous work on animation transfer between meshes [30], as well as the algorithm to interpolate target shape poses from an example database presented in [10].

All of the linear methods that we mentioned so far rely on a surface mesh representation, i.e. they work on a triangulated manifold. This kind of representation may lead to local self intersections and shrinking effects in an edited object, as opposing surfaces are not constrained to retain their distance. To prevent such artifacts, we design our method around a volumetric scene representation. Our tetrahedral mesh representation is similar to the one proposed by [36] who suggest to use a volumetric graph structure as basis for a linear Laplacian deformation. While their method exhibits much better volume preservation properties than most surface-based algorithms, it suffers from the same translational/rotational insensitivity. In contrast, our algorithm handles rotations and translations faithfully. Furthermore, our unified tetrahedral Laplacian setting is a lot less complex than their hybrid setting which uses a triangle-based Laplacian for the surface and a graph-based quadratic programming formulation for computing interior weights.

A linear approach to deformation with guaranteed volume-preservation was proposed in [32]. Although their vector-field-based framework enables the definition of advanced implicit deformation tools, it is not well-suited for handle-based shape modification in the manner necessary in our context.

Volumetric representations have been widely-used in physics-based simulation approaches. However, their focus lies on accurate reproduction of the dynamic behavior of objects with known physical material parameters under the influence of forces or moments. Mueller et al. [21] use a rotational update computation similar to ours to mimic non-linear deformations during simulation of elastic objects. In contrast, we don't require a full-blown physics-simulation framework, and couple a clever rotation update step with a fast volumetric Laplacian deformation to obtain plausible edited shapes at real-time frame rates.

Our approach is also related to the work by Alexa et al. [1] and Igarashi et al. [14], who try to interpolate respectively edit two-dimensional shapes in such a way that they behave locally as rigid as possible. Both algorithms optimize local elements in such a way that transformations induced by their deformations are as close as possible to pure rotations. Our method achieves a similar behavior for 3D tetrahedral elements.

The iterative differential update technique is related to [2], who iteratively rescale the deformed geometries differential coordinate to its original length, which may lead to accumulating tangential drift if applied to the original geometry, making it necessary to work on the dual mesh to avoid these problems. Our iterative update process avoids this by working on the tetrahedra and rebuilding differential coordinates similar to gradient based editing methods, making it stable even under extreme conditions.

2 Technique

2.1 Overview

The input to our algorithm is a (usually high resolution) triangle mesh to be deformed, denoted by $\mathcal{M}_{input} = (\mathbf{V}_{input}, \mathbf{T}_{input})$, which consists of a set of vertices $\mathbf{V}_{input}$ and triangles $\mathbf{T}_{input}$. From this we derive a tetrahedral mesh $\mathcal{T} = (\mathbf{V}_{tet}, \mathbf{T}_{tet})$ with vertices $\mathbf{V}_{tet}$ and tetrahedra $\mathbf{T}_{tet}$, which is created by performing a quadric error decimation on $\mathcal{M}_{input}$ [11] and then building a face-constrained Delaunay tetrahedralization [27].

The first step of our pipeline after loading the meshes is the precalculation of a coefficient matrix $\mathbf{B}$ containing linear combination coefficients of the vertices of $\mathcal{M}_{input}$ with respect to the tetrahedral mesh. Subsequently, the user specifies deformation handles and rigid regions on $\mathcal{T}$.

Our approach augments tetrahedral Laplacian editing (Sect. 2.2) to plausibly approximate non-linear deformation behavior by solving only linear equation systems. The core of our algorithm is an iteration of purely linear deformation, rotation estimation and interpolation, and differential coordinate update, Sect. 2.3. This loop runs permanently in the background enabling real-time visual feedback to the user manipulating the constraints. Once the user is satisfied with the deformation of the tetrahedral mesh, we reconstruct $\mathcal{M}_{input}$ in the final pose (see Sect. 2.4) using the coefficients in $\mathbf{B}$, and thereby yield a natural looking deformation of our original input.

2.2 Tetrahedral Laplacian formulation

Our deformation technique is based on Laplacian editing. However, unlike most of the related papers we are not working on a triangulated manifold but rather build our system based on a tetrahedral mesh volume. Because of this we need to set up the Laplacian equations in a slightly different manner.

Given a tetrahedral mesh $\mathcal{T} = (\mathbf{V}_{tet}, \mathbf{T}_{tet})$ with n vertices $\mathbf{V}_{tet} = \{\mathbf{v}_1 \ldots \mathbf{v}_n\}$

and m tetrahedrons $\mathbf{T}_{tet} = \{\mathbf{t}_1 \ldots \mathbf{t}_m\}$ we want to build a linear Laplacian system and find the mesh's differential coordinates δ as

$$\mathbf{L}\mathbf{x} = \delta \ . \tag{2.1}$$

For deformation, this system is to be solved under the influence of a set of given constraints. The matrix $\mathbf{L}$ is the discrete Laplace operator on our tetrahedral mesh, and $\mathbf{x}$ are the coordinates of the mesh's vertices. The Laplace operator is defined as the divergence of the gradient of a scalar field. In our setting, it can be constructed by calculating a gradient operator matrix $\mathbf{G}_j$ for each tetrahedron $\mathbf{t}_j$ which contains the gradients of the tetrahedron's shape functions ϕ_i

$$\begin{aligned} \mathbf{G}_j &= (\bigtriangledown\phi_1, \ldots, \bigtriangledown\phi_4) && (2.2) \\ &= \begin{pmatrix} (\mathbf{p}_1 - \mathbf{p}_4)^T \\ (\mathbf{p}_2 - \mathbf{p}_4)^T \\ (\mathbf{p}_3 - \mathbf{p}_4)^T \end{pmatrix}^{-1} \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{pmatrix} && (2.3) \end{aligned}$$

Here $\mathbf{p}_i$ are the four corner vertices of the tetrahedron. The matrices $\mathbf{G}_j$ can be combined into a large $4m \times n$ gradient operator matrix $\mathbf{G}$, which can be used to construct the Laplacian matrix as

$$\mathbf{L} = \mathbf{G}^T\mathbf{D}\mathbf{G} \ , \tag{2.4}$$

where $\mathbf{D}$ is a $4m \times 4m$ diagonal matrix containing the tetrahedra's volumes, and $\mathbf{G}^T\mathbf{D}$ is the discrete divergence operator.

The differential coordinates δ for the initial mesh can now be calculated explicitly using Eq. (2.1) or, more importantly, based on the tetrahedra's gradients as in Poisson surface editing [34] as

$$\delta = \mathbf{G}^T\mathbf{D}\mathbf{g} \ . \tag{2.5}$$

Here, $\mathbf{g}$ is the set of tetrahedron gradients, each being calculated as $\mathbf{g}_j = \mathbf{G}_j\mathbf{p}_j$. More details on the construction of the Laplacian from gradients can be found in [7].

This construction allows us to apply a separate transformation $\mathbf{T}_j$ to each tetrahedron $\mathbf{t}_j$ (which, in a sense, "explodes" the mesh). We can then reconstruct it in its new target configuration by using the transformed gradients $\mathbf{g}'$ on the right hand side during calculation of the differential coordinates.

2.3 Tetrahedral deformation technique

Our real-time mesh deformation method operates on a tetrahedral version $\mathcal{T}$ of the input model. The deformation approach itself comprises of several

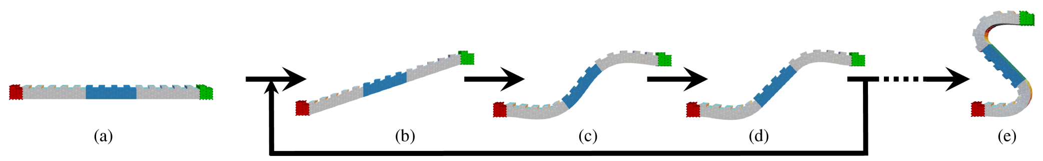

Figure 2.1: Overview of our tetrahedral mesh deformation pipeline - (a) shows the input model with handles in red and green and a rigid section marked in blue. (b),(c) and (d) show the output of the individual iterative processing steps, namely linear Laplacian deformation (b), rotation extraction/interpolation (c), and rigid section handling (d). (e) shows the final deformed model at the end of the user interaction.

algorithmic steps that are permanently iterated in the background while the user interacts with the model, Fig. 2.1. The individual steps of the method are as follows (processing steps that are optional are printed in italics):

- Deformation setup, load $\mathcal{T}$, selection of constrained and *rigid* areas
- Iterative mesh deformation
  - Linear Laplacian deformation
  - Rotation extraction
  - *Rotation interpolation*
  - *Rigid region handling*
  - Differential coordinate update

In the following, we explain the individual steps in more detail.

2.3.1 Deformation setup

The input to our method is a tetrahedralized version $\mathcal{T}$ of the input triangle mesh. Triangle meshes of moderate size can be tetrahedralized such that the surface triangulation is preserved and thus a final pose transfer is not required. To interactively deform meshes of very high resolution, we apply a quadric error mesh simplification method to $\mathcal{M}_{input}$ and tetrahedralize the low resolution mesh afterwards. Unlike the algorithm proposed in [36], which relies on the creation of a regular grid structure inside the object, our approach produces plausible deformation even if $\mathcal{T}$ does not contain internal vertices and therefore contains very elongated tetrahedra. Imposing bounds on tetrahedral shape quality results in a more complex tessellation but does not lead to noticeable deformation improvements. Thus, a coarse tessellation is sufficient and inner vertices may only need to be inserted for topological reasons, which significantly reduces the computational complexity.

After all input data is properly set up, the user defines control handles by marking parts of $\mathcal{T}$. On the one hand, handles can be complete regions of the tetrahedral mesh. In this case, rotational constraints are automatically imposed as the orientation for the handle is fixed by the user. On the other hand, handles can be single vertices, which leaves the orientation of the surrounding tetrahedra free to be determined by the deformation method. Please note that it is also possible to define handles at arbitrary positions inside the object. This can be achieved by representing the handle point as a linear combination of the vertices of $\mathcal{T}$ as presented in Sect. 2.4. The latter method even allows us to add handles outside of the object but close to the surface. Using this we can, for instance, directly use raw marker positions from optical motion capture data as constraints.

Once all handles are set, the user can optionally specify rigid regions. In this step, sets of tetrahedra that ought to maintain a constant relative orientation can be marked. Conveniently, our system does not impose any constraints on the size, shape or topology of rigid regions. It is even possible to mark disconnected parts as belonging to the same rigid part. This allows us to quickly specify a kind of *pseudo*-skeleton without having to define joints or a bone hierarchy.

2.3.2 Iterative mesh deformation

Once the deformation setup is completed, user-guided mesh deformation can commence. While the user interacts with the handles a new deformed mesh is calculated for every frame rendered, i.e. the deformation algorithm continuously iterates in the background.

The first step in each iteration is a basic linear Laplacian deformation. Since this linear step does not properly handle rotations, we perform an analysis of the linear results to extract for each tetrahedron j an estimate of its rotational transformation $\mathbf{R}_j$, Sect. 2.3.2. If necessary, user-defined constraint rotations are interpolated across the mesh and added up to the previously extracted per-tetrahedron rotations $\mathbf{R}_j$. Subsequently, we make sure that all rigid mesh regions rotate in the same way by assigning to each rigid group of tetrahedrons its averaged rotational transformation component. Finally, the resulting $\mathbf{R}_j$ are used to construct an updated set of differential coordinates.

By iterating these steps, we achieve natural and plausible looking deformations of our object at real-time frame rates. In the following, we detail the individual deformation steps, commencing with the three ones that have to be performed in any case, and ending with the two ones that are only required if either rotational or rigidity constraints were specified.

Linear Laplacian deformation

The linear deformation follows the setting outlined in Sect. 2.2. The matrix $\mathbf{L}$ of Eq. (2.1) can be constructed following Eq. (2.4). Accordingly, the differential coordinates can be computed based on Eq. (2.5).

Handles can be factorized into the matrix $\mathbf{L}$ by eliminating the corresponding row and column in the matrix and incorporating the values into the right hand side δ. The deformation can then be generated by solving the reduced Eq. (2.1).

The resulting deformed mesh $\mathcal{T}'$ only looks natural for very small deformations (see Fig. 2.1), as the basic Laplacian setting does not induce any kind of rotation or scaling. The deformed mesh $\mathcal{T}'$ can be computed efficiently with the help of a Cholesky factorization of the left hand side matrix.

Rotation extraction

In the context of our iterative deformation scheme, we can plausibly approximate non-linear transformation components, such as local rotations, in a purely linear setting. We analyze the output of the linear Laplacian deformation in order to extract for each tetrahedron $\mathbf{t}_j$ an estimate of a rotational transformation. These per-tetrahedron rotations are eventually used to update the differential coordinates for the linear deformation.

To put this idea into practice, we compare the deformed mesh $\mathcal{T}'$ to the original mesh $\mathcal{T}$ in its rest pose. By comparing the positions of the vertices $\mathbf{v}_i$ of a single tetrahedron $\mathbf{t}_j$ in $\mathcal{T}$ to their new positions $\mathbf{v}'_i$ in $\mathcal{T}'$, we can calculate a 3×3 transformation matrix $\mathbf{T}_j$ as

$$\mathbf{T}_j = \begin{pmatrix} (\mathbf{v}_1 - \mathbf{v}_4)^T \\ (\mathbf{v}_2 - \mathbf{v}_4)^T \\ (\mathbf{v}_3 - \mathbf{v}_4)^T \end{pmatrix}^{-1} \begin{pmatrix} (\mathbf{v}'_1 - \mathbf{v}'_4)^T \\ (\mathbf{v}'_2 - \mathbf{v}'_4)^T \\ (\mathbf{v}'_3 - \mathbf{v}'_4)^T \end{pmatrix} \tag{2.6}$$

such that $(\mathbf{v}_i - \mathbf{v}_4)\mathbf{T}_j = (\mathbf{v}'_i - \mathbf{v}'_4)$ for $i = 1, \ldots, 3$.

Matrix $\mathbf{T}_j$ only consists of anisotropic scaling (stretching) and shearing components. As shown by Shoemake et al. [26], $\mathbf{T}_j$ can thus be factored into an othonormal rotation matrix $\mathbf{R}_j$ and a non-rotational part $\mathbf{S}_j$,

$$\mathbf{T}_j = \mathbf{R}_j \mathbf{S}_j \ . \tag{2.7}$$

Technically, the rotational component can be computed by iteratively averaging $\mathbf{T}_j$ with its inverse transpose as

$$\mathbf{H}_0 = \mathbf{T}_j \ , \ \mathbf{H}_{k+1} = \frac{1}{2}(\mathbf{H}_k + {\mathbf{H}_k}^{-T}) \ . \tag{2.8}$$

The decomposition iterates up to a step k after which $\mathbf{H}_k$ does not change anymore, and then sets $\mathbf{R}_j = \mathbf{H}_k$. In our experiments it was sufficient to perform two to three computation steps followed by a normalization of each matrix row to achieve a decent separation. Note that in our three-dimensional setting we also have to make sure that $\mathbf{R}_j$ does not contain a mirroring component to keep tetrahedra from inverting. To serve this purpose, we check the determinant and multiply the matrix by -1 if necessary.

Differential coordinate update

Now that we have calculated rotations $\mathbf{R}_j$ for each tetrahedron it is straightforward to use them to calculate a new set of differential coordinates. We apply the rotations to their respective tetrahedra $\mathbf{t}_j$ from the original mesh and calculate a new set of transformed tetrahedron gradients $\mathbf{g}_j$, from which the new δ can be calculated using Eq. (2.5). This is similar to the "exploding" of the mesh in [30].

Rotation interpolation

By iterating the above three steps linear Laplacian deformation, rotation extraction, and differential coordinate update, we can mimic non-linear behavior in our deformations. Repeating them also minimizes the amount of shear in the results.

While the system adapts very well to translational constraints, rotational constraints (especially twisting) may require a higher number of iterations to propagate through the mesh. One important observation is that the limit of the deformation under rotational constraints is usually very similar to the deformation that would have been generated by using a Poisson mesh editing technique where rotations are interpolated between the constraints (see Fig. 2.3.2). Thus, it is only natural to combine the two methods.

We decided to adapt the approach by Zayer et al. [35], which uses a harmonic interpolation of quaternions across the mesh, to our purpose. One major disadvantage of quaternion-based rotation interpolation is its inability to properly reproduce rotations by more than 180 degrees. We solve this problem by interpolating tetrahedron rotations that are relative to the current estimates $\mathbf{R}_j$ instead of absolute rotations with respect to the rest

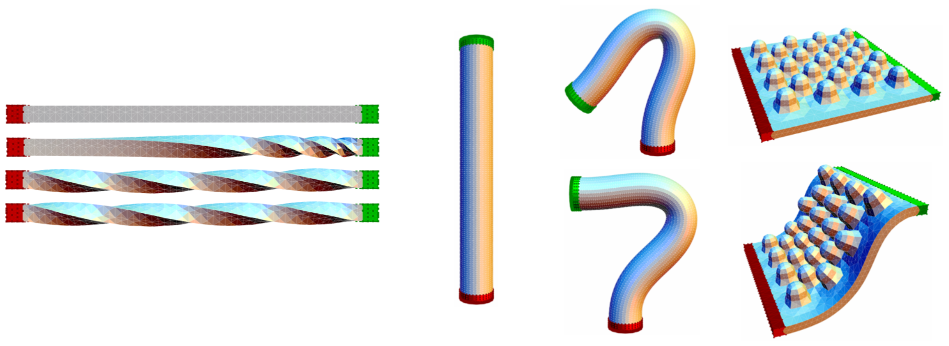
(a) (b)

Figure 2.2: (a) Behavior of our deformation technique when a 360° twist is applied to the green handle. From top to bottom: Original model of a bar; result after applying the twist without rotation interpolation; deformation without rotation interpolation converged after several seconds; instantaneous result *with* rotation interpolation. (b) Examples of deformations which are difficult for linear Laplacian methods. *Left*: A cylinder with deformations generating by applying a rotation of 135° (top), respectively 90° and a translation to the right (bottom). *Right*: A bumpy plane to which we apply a pure translation. Note the rotations of the bumps on the plane generated even though only a translational constraint has been applied.

state. These relative rotations are usually very subtle and can thus be easily interpolated with the harmonic interpolation scheme. Technically, the interpolated quaternions are simply added to the current frame's rotation matrices $\mathbf{R}_j$ before calculating the updated differential coordinates.

An interpolation of rotations is only necessary if the user actually rotates a handle and if that handle consists of one or more tetrahedra in which all vertices were selected. In any other case, the rotation of a handle can not be determined uniquely.

Rigid section handling

Adapting our method to handle rigid parts in the mesh is a straightforward task. For every rigid set of tetrahedra $\mathbf{T}_{rigid} \subseteq \mathbf{T}_{tet}$ we can calculate an average rotation from the $\mathbf{R}_l$ of all the $\mathbf{t}_l \in \mathbf{T}_{rigid}$. Each such $\mathbf{t}_l$ is then assigned this fixed average rotation. By this means, we remove the ability to bend from the set $\mathbf{T}_{rigid}$ while the averaging still allows the set to orient itself in such a way that the deformation's shear is minimized, Fig. 2.1. If our goal is not to generate completely rigid parts but rather to vary the stiffness of certain parts we can achieve this by scaling the gradients $\mathbf{g}_j$ and gradient operator matrices $\mathbf{G}_j$ of the respective elements when building the linear system from Eq. 2.1 and updating the differential coordinates in Sec. 2.3.2. Increasing the scaling factor increases the relative stiffness to the surrounding elements and vice versa.

2.4 Deforming high resolution meshes

High-resolution meshes $\mathcal{M}_{input}$, as generated by lase scanning or similar techniques, cannot be deformed at interactive rates using our method. Thus we need to work on a lower-resolution tetrahedral mesh version of the original input mesh, and afterwards transfer the final pose of $\mathcal{T}$ to obtain $\mathcal{M}_{input}$ in the target pose.

A fast and easy-to-implement approach to pose transfer generates for each vertex $\mathbf{v}_i$ of $\mathcal{M}_{input}$ a set of barycentric coordinates with respect to the closest tetrahedron of $\mathcal{T}$ in the rest pose. By applying those coordinates to the deformed tetrahedral mesh, the deformed fine mesh $\mathcal{M}'_{input}$ is reconstructed. Although this approach is fast it has several drawbacks, the main one being that the reconstruction is only piecewise linear and thus artefacts like dents and self intersections may appear in the reconstructed mesh.

Mean value coordinates [15] provide an alternative that circumvents these problems. Here, a set of coefficients $\mathbf{c}_i = \mathbf{c}_i^0 \ldots \mathbf{c}_i^m$ is computed for each vertex $\mathbf{v}_i$ such that ${\mathbf{c}_i}^T\mathbf{V}_{tet} = \mathbf{v}_i$, i.e. each point of the input mesh is represented as a linear combination of all the points of $\mathcal{T}$. Mean value coordinates are $\mathcal{C}^2$-smooth and enable plausible pose transfer to $\mathcal{M}_{input}$ as shown in a similar context in [13]. Unfortunately, they are very memory consuming and mesh reconstruction takes significantly longer.

[33] use displaced subdivision surfaces [17] to transfer their deformation from low resolution mesh to a high resolution input. But this method can only construct a remeshed version of the input mesh with the connectivity of the subdivision surface created from the low resolution version, and therefore it is not suitable for our purpose.

We propose a method that combines the advantages of the first two methods mentioned above. Our approach represents vertices of the input mesh as linear combinations of tetrahedra in the local neighborhood. By selecting the coefficients carefully, we can achieve a smooth deformation transfer with a method that is far less memory consuming and computationally more efficient than mean value coordinates.

To put this into practice, we find for each vertex $\mathbf{v}_i$ in $\mathcal{M}_{input}$ the set $\mathbf{T}_r(\mathbf{v}_i)$ of all tetrahedra that have a boundary face center $center(\mathbf{t}_j)$ which lies within a local spherical neighborhood of radius r (in all our cases r was set to 5% of the mesh's bounding box diagonal). Subsequently, we calculate the barycentric coordinate coefficients $\mathbf{c}_i(j)$ of the vertex with respect to all $\mathbf{t}_j \in \mathbf{T}_r(\mathbf{v}_i)$ and compute the combined coefficient vector $\mathbf{c}_i$ as

$$\mathbf{c}_i = \frac{\sum_{\mathbf{t}_j \in \mathbf{T}_r(\mathbf{v}_i)} \mathbf{c}_i(k)\phi(\mathbf{v}_i, \mathbf{t}_j)}{\sum_{\mathbf{t}_j \in \mathbf{T}_r(\mathbf{v}_i)} \phi(\mathbf{v}_i, \mathbf{t}_j)} \ . \tag{2.9}$$

$\phi(\mathbf{v}_i, \mathbf{t}_j)$ is the Wendland weighting function with respect to the distance of $\mathbf{v}_i$ to the boundary face center of tetrahedron $\mathbf{t}_j$

$$\phi(\mathbf{v}_i, \mathbf{t}_j) = \begin{cases} 0 & \text{if } d > r \\ (1 - \frac{d}{r})^4(\frac{4d}{r} + 1) & \text{if } d \leq r \end{cases} \tag{2.10}$$

$$\text{with } d = \|\mathbf{v}_i - center(\mathbf{t}_j)\| \ . \tag{2.11}$$

The coefficients for all vertices of $\mathcal{M}_{input}$ are combined into the previously mentioned matrix $\mathbf{B}$. Thanks to the smooth partition of unity definition and the local support of our parametrization, we can quickly compute a smooth and natural looking transformed input $\mathcal{M}'_{input}$ at moderate memory costs by calculating new vertex positions as

$$\mathcal{M}'_{input} = \mathcal{T}'\mathbf{B} \tag{2.12}$$

3 Results

We have tested our real-time editing technique for performing high-quality shape deformation on a variety of complex models, Sect. 3.1. Additionally, we demonstrate that our method is well-suited for generation of lifelike mesh animations from motion capture data, Sect. 3.2. Finally, we first discuss some of the basic geometric properties of our technique and highlight advantages over related approaches, Sect. 3.3.

3.1 Model editing

We have used our approach for interactive editing of several high resolution meshes, as shown in Figs. 1.1 and 3.1. Tab. 3.1 contains detailed information on the complexity of different models, as well as timing results for the individual processing steps in our approach. The accompanying video also shows a live capture of an editing session, illustrating that plausible deformations can be quickly achieved using easy-to-specify deformation handles. In our test cases, both large region handles (prescribing position and orientation) and point handles (prescribing only position) were used to create the various poses. Please note the natural intuitive behavior of the deformation, as well as the authentic look of the deformations also on the high-resolution meshes. Even when using only a few handles it is possible to easily create convincing results that look physically plausible. We would also like to point out that thanks to the tetrahedral setting there is no noticeable loss in volume even after very large deformations .

3.2 Animation using motion capture data

The flexibility in constraint handle positioning inside and outside a model, as well as the ability to create plausible deformations from moving point constraints only, enables us to quickly generate realistic animations of models

model	fig	input	tet	fact	trans	recons	fps
Bar	2.3.2	N/A	857 / 2289	0.039	N/A	N/A	>120
B. Bar	2.1	N/A	1440 / 4302	0.074	N/A	N/A	∼85
Plane	2.3.2	N/A	5066 / 16028	0.331	N/A	N/A	∼20
Cylinder	2.3.2	N/A	7235 / 23831	0.565	N/A	N/A	∼12
Man	3.2	39771	3474 / 12324	0.241	42.8	0.179	∼24
Dragon	3.1	100004	3021 / 9802	0.128	94.5	0.442	∼37
Raptor	1.1	175100	2512 / 9372	0.136	167.6	0.965	∼37
Armadillo	3.1	195948	1688 / 5373	0.084	96.0	0.484	∼57

Table 3.1: This table shows performance figures for our technique on an AMD X2 5000+. The columns from left to right contain: Model name, figure number in the paper, number of input mesh vertices, number of tet mesh vertices/tetrahedra; thereafter, timings are given for pre-factorization of the left-hand side matrix of Eq. (2.1), precalculation of pose transfer coefficients, deformation transfer to input mesh; the rightmost column gives frames per second during interactive editing of the tetrahedral mesh. All other timings are given in seconds.

using captured motion data. To demonstrate this, we mapped motion data acquired with a marker-based optical capturing system, as well as data captured with a marker-free motion capture algorithm [9] to high-quality laser scans of humans, Fig. 3.2, as well as to the Stanford Armadillo model. The accompanying video shows several examples in which the models perform complex motions, such as a cartwheel. Depending on the type of motion data, different kinds of deformation handles can be specified.

In the case of marker-based motion capture data, we can use the raw 3D marker trajectories as deformation constraints. As a preprocessing step, we align the model to the marker positions in the first frame by hand. After alignment, we generate an additional vertex at the marker positions and connect these markers to the model $\mathcal{T}$ by adding a set of invisible tetrahedra which connect the markers to the closest boundary faces. This allows us to use markers at arbitrary positions close to our model surface to deform the object.

By this means, we can bypass the labor-intensive and often error-prone reconstruction of a kinematic skeleton from the marker data and utilize the raw measurement output straight away. In case we are already given a template skeleton mesh (as for example the default motion template from 3D Studio) we can pre-align our model to it and use vertices of the skeleton as deformation handles by means of the same linear-combination approach.

The marker-less motion capture results come already in the form of a coarse moving kinematic body model comprising of individual triangle mesh segments. By specifying corresponding vertices on the template and the mesh $\mathcal{T}$, it becomes straightforward to map the motion onto a scanned model. The accompanying video shows an example sequence in which the dancing performance of an actor was captured and applied to a scan of himself.

If only few handles are used, we additionally mark a set of rigid groups on the model to be deformed, Fig. 3.2. The whole handle setup-process typically takes only 3-4 minutes for a single animation.

The final animation is computed as follows: For each frame we update the position of the handle constraints, run the iterative deformation for a number of cycles (typically 2-3 cycles are sufficient) and afterwards deform the input mesh using the technique from Sect. 2.4. In practice, we use the tetrahedral mesh model for real-time pre-visualization of the animation, but render the high-quality deformation in a batch process that takes between 0.1 and 0.5 s to write one frame (see Tab. 3.1).

Our results show that our method provides a fast and easy-to-use algorithmic alternative to create high-quality mesh animations from captured motion data. Realistic smooth surface deformations and subtle motion details are faithfully reproduced without having to rely on an intermediate skeleton representation with associated skinning weights.

3.3 Geometric properties

Our deformation technique has several distinct advantages over a purely linear deformation method. While the inability of a purely linear approach to handle rotations quickly leads to a deterioration of the deformed geometry, our iterative approach handles rotations faithfully lending a plausible appearance of the transformed model. Our results are also more pleasing than the ones obtained with a gradient-based deformation techniques, as we faithfully handle translations. By iteratively computing small pose updates in a clever linear framework we can produce similar results as state-of-the-art non-linear methods, such as [6].

Despite these advantageous properties, our tetrahedral method is not suitable for real-time deformation of very large triangle meshes. We thus deform them based on decimated tetrahedral representations. However, we would like to point out that this is not a principle disadvantage since it is our primary goal to modify the low-frequency components of a model which contain information about its pose. Working on a low resolution mesh is equivalent to performing a low-pass filtering of our geometry and adding the high frequency detail back after the low frequency components have been

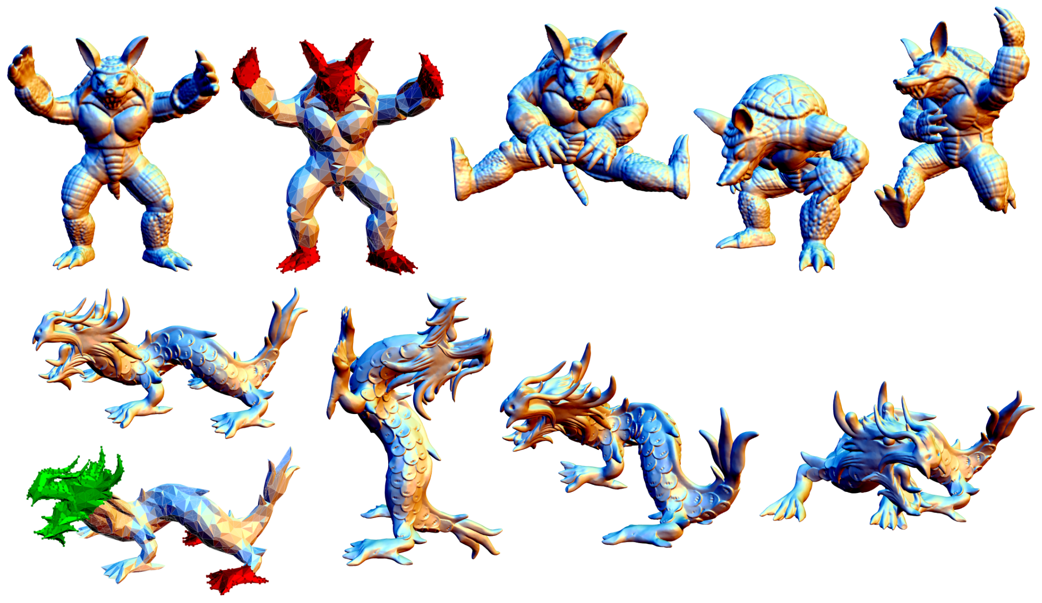

Figure 3.1: Results of an interactive editing session. Input models and their tetrahedral counterparts with handles on the left, three resulting poses on the right.

modified.

Our volumetric method has more favorable shape preservation properties than approaches working on triangulated manifolds. Due to the tetrahedral construction our approach aims at preserving distances between diametrically opposed vertices on the model's surface. Given an appropriately set up mesh, this can be regarded as a tendency to preserve cross-sectional areas. Although this does not imply global volume preservation, it comes close to a local volume preservation which is very useful when animating shapes where one wants, for instance, the thicknesses of limbs to be preserved. We'd also like to emphasize that our method does not impose strict shape quality requirements to the generated tetrahedra, and even meshes without interior vertices can be deformed in an intuitive way.

Another beneficial property of our deformation is its robustness even under extreme conditions. Even if we completely randomize the differential coordinates of an object constrained by a few handles we can recover the original shape after a number of iterations (see the accompanying video for an example). This means that it is virtually impossible to break the deformation process by careless constraint placement.

Our algorithm can also be interpreted as a form of extremely simple nonlinear optimization of elastic energy. Elastic deformation is characterized

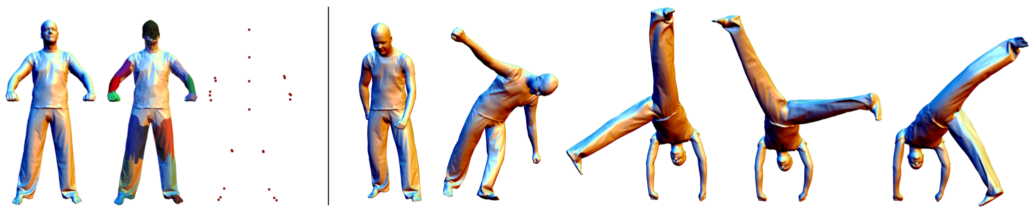

Figure 3.2: Animation of a scanned character using optical MoCap data. From left to right: Input model, reduced tetrahedral model with rigid regions marked roughly, 22 input handles, five frames out of a cartwheel sequence.

as a deformation which behaves as rigid as possible locally. Our iterative approach achieves this by iteratively removing as much non-rigid deformation from the tetrahedra as possible. This can also be observed in the results of our algorithm, as they behave very similar to elastic deformation simulations. A way to speed up convergence of this non-linear approach would be to apply multi-resolution techniques by generating a hierarchy of models.

In all our test cases and application scenarios we found that the deformation behaves intuitively and is easy to use even for unexperienced users. The real-time deformation process in conjunction with immediate visual feedback makes it convenient to obtain the envisioned results.

4 Conclusion

We presented a volumetric interactive shape editing technique that alternates between a linear tetrahedral Laplacian deformation and a differential update step. By this means non-linear deformation behavior can be mimicked without having to solve complex non-linear systems. Our method is highly stable and enables even unexperienced users to create convincing results using only few manipulation handles. Translational and rotational constraints can be specified flexibly at arbitrary locations inside and outside the object. This also allows us to use our algorithm for rapid generation of high-quality mesh animations from motion capture data.

In the future, we plan to apply our method to enhance the quality of model-based 3D video and to explore the possibility of using the approach in the context of non-rigid shape matching.

Bibliography

[1] M. Alexa, D. Cohen-Or, and D. Levin. As-rigid-as-possible shape interpolation. In *Proc. ACM SIGGRAPH*, pages 157–164, 2000.

[2] O. K.-C. Au, C.-L. Tai, L. Liu, and H. Fu. Dual laplacian editing for meshes. *IEEE TVCG*, 12(3):386–395, 2006.

[3] J. Barbic and D. L. James. Real-time subspace integration for st. venant-kirchhoff deformable models. In *Proc. ACM SIGGRAPH*, pages 982–990, 2005.

[4] M. Botsch and L. Kobbelt. Multiresolution surface representation based on displacement volumes. In *Proc. Eurographics*, pages 483–491, 2003.

[5] M. Botsch and L. Kobbelt. An intuitive framework for real-time freeform modeling. In *Proc. ACM SIGGRAPH*, pages 630–634, 2004.

[6] M. Botsch, M. Pauly, M. Gross, and L. Kobbelt. Primo: Coupled prisms for intuitive surface modeling. In *Proc. Symposium on Geometry Processing*, pages 11–20, 2006.

[7] M. Botsch, R. Sumner, M. Pauly, and M. Gross. Deformation transfer for detail-preserving surface editing. In *Proc. Vision, Modeling and Visualization*, pages 357–364, 2006.

[8] H. L. M. M. H.-P. S. C. Theobalt, N. Ahmed. Seeing people in different light: Joint shape, motion, and reflectance capture. *IEEE TVCG*, 2007.

[9] J. Carranza, C. Theobalt, M. Magnor, and H.-P. Seidel. Free-viewpoint video of human actors. *Proc. ACM SIGGRAPH*, pages 569–577, 2003.

[10] K. G. Der, R. W. Sumner, and J. Popovic. Inverse kinematics for reduced deformable models. In *Proc. ACM SIGGRAPH*, pages 1174–1179, 2006.

[11] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proc. ACM SIGGRAPH*, pages 209–216, 1997.

[12] N. Higham. Computing the polar decomposition - with applications. *SIAM J. Sci. and Stat. Comp.*, pages 1160–1174, 1986.

[13] J. Huang, X. Shi, X. Liu, K. Zhou, L.-Y. Wei, S.-H. Teng, H. Bao, B. Guo, and H.-Y. Shum. Subspace gradient domain mesh deformation. In *Proc. ACM SIGGRAPH*, pages 1126–1134, 2006.

[14] T. Igarashi, T. Moscovich, and J. F. Hughes. As-rigid-as-possible shape manipulation. In *Proc. ACM SIGGRAPH*, pages 1134–1141, 2005.

[15] T. Ju, S. Schaefer, and J. Warren. Mean value coordinates for closed triangular meshes. In *Proc. ACM SIGGRAPH*, pages 561–566, 2005.

[16] L. Kobbelt, J. Vorsatz, and H.-P. Seidel. Multiresolution hierarchies on unstructured triangle meshes. *Comput. Geom. Theory Appl.*, 14(1-3):5–24, 1999.

[17] A. Lee, H. Moreton, and H. Hoppe. Displaced subdivision surfaces. In *Proc. ACM SIGGRAPH*, pages 85–94, 2000.

[18] Y. Lipman, D. Cohen-Or, G. Ran, and D. Levin. Volume and shape preservation via moving frame manipulation. *ACM Trans. Graph.*, 26(1), 2007.

[19] Y. Lipman, O. Sorkine, D. Cohen-Or, D. Levin, C. Rössl, and H.-P. Seidel. Differential coordinates for interactive mesh editing. In *Proc. of Shape Modeling International*, pages 181–190, 2004.

[20] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or. Linear rotation-invariant coordinates for meshes. In *Proc. ACM SIGGRAPH*, pages 479–487, 2005.

[21] M. Mueller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler. Stable real-time deformations. In *Proc. of SCA '02*, pages 49–54, 2002.

[22] A. Nealen, O. Sorkine, M. Alexa, and D. Cohen-Or. A sketch-based interface for detail-preserving mesh editing. In *Proc. ACM SIGGRAPH*, pages 1142–1147, 2005.

[23] T. Popa, D. Julius, and A. Sheffer. Material-aware mesh deformations. In *Proc. Shape Modeling International*, page 22, 2006.

[24] T. Sederberg and R. Scott. Free-form deformation of solid geometric models. In *Proc. ACM SIGGRAPH*, pages 151–160, 1986.

[25] A. Sheffer and V. Kraevoy. Pyramid coordinates for morphing and deformation. In *Proc. 3D Data Processing, Visualization, and Transmission*, pages 68–75, 2004.

[26] K. Shoemake and T. Duff. Matrix animation and polar decomposition. In *Proc. of Graphics Interface*, pages 258–264, 1992.

[27] H. Si and K. Gaertner. Meshing piecewise linear complexes by constrained delaunay tetrahedralizations. In *Proc. International Meshing Roundtable*, pages 147–163, 2005.

[28] O. Sorkine. Differential representations for mesh processing. *Computer Graphics Forum*, 25(4):789–807, 2006.

[29] O. Sorkine, Y. Lipman, D. Cohen-Or, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian surface editing. In *Proc. Symposium on Geometry Processing*, pages 179–188, 2004.

[30] R. W. Sumner and J. Popovic. Deformation transfer for triangle meshes. In *Proc. ACM SIGGRAPH*, pages 399–405, 2004.

[31] R. W. Sumner, M. Zwicker, C. Gotsman, and J. Popovic. Mesh-based inverse kinematics. In *Proc. ACM SIGGRAPH*, pages 488–495, 2005.

[32] W. von Funck, H. Theisel, and H.-P. Seidel. Vector field based shape deformations. In *Proc. ACM SIGGRAPH*, pages 1118–1125, 2006.

[33] S. Yoshizawa, A. G. Belyaev, and H.-P. Seidel. Free-form skeleton-driven mesh deformations. In *Proc. Symposium on Solid modeling and applications*, pages 247–253, 2003.

[34] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum. Mesh editing with poisson-based gradient field manipulation. In *Proc. ACM SIGGRAPH*, pages 644–651, 2004.

[35] R. Zayer, C. Rössl, Z. Karni, and H.-P. Seidel. Harmonic guidance for surface deformation. In *Proc. Eurographics*, pages 601–609, 2005.

[36] K. Zhou, J. Huang, J. Snyder, X. Liu, H. Bao, B. Guo, and H.-Y. Shum. Large mesh deformation using the volumetric graph laplacian. In *Proc. ACM SIGGRAPH*, pages 496–503, 2005.

MPI-I-2006-1-007	H. Bast, I. Weber, C.W. Mortensen	Output-Sensitive Autocompletion Search
MPI-I-2006-1-006	M. Kerber	Division-Free Computation of Subresultants Using Bezout Matrices
MPI-I-2006-1-005	A. Eigenwillig, L. Kettner, N. Wolpert	Snap Rounding of Bzier Curves
MPI-I-2006-1-004	S. Funke, S. Laue, R. Naujoks, L. Zvi	Power Assignment Problems in Wireless Communication
MPI-I-2005-5-002	S. Siersdorfer, G. Weikum	Automated Retraining Methods for Document Classification and their Parameter Tuning
MPI-I-2005-4-006	C. Fuchs, M. Goesele, T. Chen, H. Seidel	An Emperical Model for Heterogeneous Translucent Objects
MPI-I-2005-4-005	G. Krawczyk, M. Goesele, H. Seidel	Photometric Calibration of High Dynamic Range Cameras
MPI-I-2005-4-005	M. Goesele	?
MPI-I-2005-4-004	C. Theobalt, N. Ahmed, E. De Aguiar, G. Ziegler, H. Lensch, M.A. Magnor, H. Seidel	Joint Motion and Reflectance Capture for Creating Relightable 3D Videos
MPI-I-2005-4-003	T. Langer, A.G. Belyaev, H. Seidel	Analysis and Design of Discrete Normals and Curvatures
MPI-I-2005-4-002	O. Schall, A. Belyaev, H. Seidel	Sparse Meshing of Uncertain and Noisy Surface Scattered Data
MPI-I-2005-4-001	M. Fuchs, V. Blanz, H. Lensch, H. Seidel	Reflectance from Images: A Model-Based Approach for Human Faces
MPI-I-2005-2-004	Y. Kazakov	A Framework of Refutational Theorem Proving for Saturation-Based Decision Procedures
MPI-I-2005-2-003	H.d. Nivelle	Using Resolution as a Decision Procedure
MPI-I-2005-2-002	P. Maier, W. Charatonik, L. Georgieva	Bounded Model Checking of Pointer Programs
MPI-I-2005-2-001	J. Hoffmann, C. Gomes, B. Selman	Bottleneck Behavior in CNF Formulas
MPI-I-2005-1-008	C. Gotsman, K. Kaligosi, K. Mehlhorn, D. Michail, E. Pyrga	Cycle Bases of Graphs and Sampled Manifolds
MPI-I-2005-1-007	I. Katriel, M. Kutz	A Faster Algorithm for Computing a Longest Common Increasing Subsequence
MPI-I-2005-1-003	S. Baswana, K. Telikepalli	Improved Algorithms for All-Pairs Approximate Shortest Paths in Weighted Graphs
MPI-I-2005-1-002	I. Katriel, M. Kutz, M. Skutella	Reachability Substitutes for Planar Digraphs
MPI-I-2005-1-001	D. Michail	Rank-Maximal through Maximum Weight Matchings
MPI-I-2004-NWG3-001	M. Magnor	Axisymmetric Reconstruction and 3D Visualization of Bipolar Planetary Nebulae
MPI-I-2004-NWG1-001	B. Blanchet	Automatic Proof of Strong Secrecy for Security Protocols
MPI-I-2004-5-001	S. Siersdorfer, S. Sizov, G. Weikum	Goal-oriented Methods and Meta Methods for Document Classification and their Parameter Tuning
MPI-I-2004-4-006	K. Dmitriev, V. Havran, H. Seidel	Faster Ray Tracing with SIMD Shaft Culling
MPI-I-2004-4-005	I.P. Ivrissimtzis, W.-. Jeong, S. Lee, Y.a. Lee, H.-. Seidel	Neural Meshes: Surface Reconstruction with a Learning Algorithm
MPI-I-2004-4-004	R. Zayer, C. Rssl, H. Seidel	r-Adaptive Parameterization of Surfaces
MPI-I-2004-4-003	Y. Ohtake, A. Belyaev, H. Seidel	3D Scattered Data Interpolation and Approximation with Multilevel Compactly Supported RBFs
MPI-I-2004-4-002	Y. Ohtake, A. Belyaev, H. Seidel	Quadric-Based Mesh Reconstruction from Scattered Data
MPI-I-2004-4-001	J. Haber, C. Schmitt, M. Koster, H. Seidel	Modeling Hair using a Wisp Hair Model
MPI-I-2004-2-007	S. Wagner	Summaries for While Programs with Recursion
MPI-I-2004-2-002	P. Maier	Intuitionistic LTL and a New Characterization of Safety and Liveness
MPI-I-2004-2-001	H. de Nivelle, Y. Kazakov	Resolution Decision Procedures for the Guarded Fragment with Transitive Guards