Interactive Moving Least-Squares Volume Deformation

Eduardo Tejada[1], Alvaro Cuno[3], Siegfried Hodri[2], João P. Gois[4], Antonio Oliveira[3], and Thomas Ertl[2]

[1] Universidad Católica San Pablo, Quinta Vivanco s/n, Arequipa, Peru
[2] Universität Stuttgart, Universitätsstr. 38, 70569 Stuttgart, Germany
[3] Universidade Federal do Rio de Janeiro, Cid. Universitária, Rio de Janeiro, Brazil
[4] Universidade de São Paulo, Av. Trab. Sãocarlense 400, São Carlos, SP, Brazil

Abstract. We study how to perform interactive volume deformation using the moving least squared (MLS) optimization method. Non-linear polynomials and weight manipulation are introduced into a MLS context for achieving bending and cuts operations, which are key features in a volume exploration tool. For that, recent results on meshless and non-physically-based MLS deformation are used. Additionally, an interactive hardware-accelerated implementation applicable to structured and unstructured grids is presented. Our implementation is specially designed for supporting known hardware-accelerated volume rendering methods.

Key words: Volume deformation, moving least-squares, graphics hardware, interactive visualization

1 Introduction

Volume deformation is used for a wide range of applications. While physically-based methods yield a plausible deformation, in applications such as volume registration and volume exploration the interest focuses on providing a fast and easy to manipulate means for volume inspection. Deformation algorithms for different types of grids based on a combination of atomic transformations, such as scale, twist, squeeze, taper or bend have been proposed. Although these transformations are easy to use separately, a combination of them to generate complex deformations is a difficult problem. Moreover, most of them are not suitable for direct manipulation by the user even when they need few parameters.

A meshless approach was proposed by Müller *et al.* [1] based on the idea of performing the registration of two point sets, where a polar decomposition of a square matrix is used. Interactive deformation of volumetric data is also addressed in the work by Chen *et al.* [2] and by Westermann *et al.* [3] by using free-form deformations. An hyperpatch with 64 control points is used to deform the enclosing space and thus the object. Gibson [4] proposed the 3D-Chainmail algorithm, which is based on the propagation of the deformation at the vertices of the grid, where the deformation of a vertex is based on the deformation of its neighbors.

XXXIV Conferencia Latinoamericana de Informática (CLEI 2008), pp. 813-822, 2008.

The term as-rigid-as-possible was introduced by Alexa *et al.* [5] to describe a deformation for which the scaling and shearing are minimal. This yields a natural and plausible deformation, as has been shown by Schaefer *et al.* [6] and Igarashi *et al.* [7] who presented deformation algorithms for bi-dimensional images. The advantages of the method by Schaefer *et al.* are that, by using the moving least-squares (MLS) approximation method, large linear systems are avoided and a closed formula for the deformation is obtained. An efficient extension of this method to three dimensions was proposed by Cuno *et al.* [8], which is the basis of the work presented in this paper, due to its advantages compared to methods based on orthogonal matrices [9, 10] and on quaternions [11].

Contributions. In summary, our contributions are: (1) a detailed derivation of closed formulas for linear and nonlinear deformation functions in 3D. (2) The inclusion of nonlinear polynomial transformations in the set of MLS deformations originally proposed by Schaefer and extended to three dimensions by Cuno (the original set comprises only affine, similarity and rigid transformations). Nonlinear polynomial transformations are able to provide deformations, such as bending, that cannot be modeled with linear transformations. Orthogonal polynomials are used to define a MLS polynomial approximation free of systems of equations. (3) It is shown how cuts can be introduced in MLS deformations. (4) A GPU implementation showing interactive frame rates is described.

2 Affine, Similarity and Rigid Deformations

Moving least-squares deformation uses control points to let the user manipulate the volume. Given a set of N control points $\{\mathbf{p}_i\}$ and their deformed positions $\{\mathbf{q}_i\}$, a function $\mathcal{M}f_{DMLS}$ that maps any point $\mathbf{u}$ in the undeformed volume to a point $\mathbf{v}$ in the deformed volume is approximated. The function $\mathcal{M}f_{DMLS}$ is a continuous interpolating function; *i.e.*, $\mathcal{M}f_{DMLS}(\mathbf{p}_i) = \mathbf{q}_i$, which holds that if $\mathbf{p}_i = \mathbf{q}_i$ then $\mathcal{M}f_{DMLS}(\mathbf{p}_i) = \mathbf{p}_i$; $i = 1 \cdots N$. By using MLS to find $\mathcal{M}f_{DMLS}$, Schaefer computes a different transformation $l_{\mathbf{u}}$ for $\mathbf{u}$ by solving

$$\min \sum_{i=1}^{N} \omega_i (l_{\mathbf{u}}(\mathbf{p}_i) - \mathbf{q}_i)^2 \ , \tag{1}$$

where $\omega_i = 1/(\mathbf{p}_i - \mathbf{u})^{2\alpha}$, $\alpha = h^2$, and h is the fill size. The function $\mathcal{M}f_{DMLS}$ that minimizes this expression provides a forward mapping. However, in order to better fit the deformation method to rendering algorithms such as ray-casting, the aim is to find the backwards transformation $l_{\mathbf{v}}^{-1}$ that maps each deformed position $\mathbf{v}$ to its position $\mathbf{u}$ in the undeformed volume. This is accomplished by solving

$$\min \sum_{i=1}^{N} \omega_i' (\mathbf{p}_i - l_{\mathbf{v}}^{-1}(\mathbf{q}_i))^2 \ , \tag{2}$$

where $\omega_i' = 1/(\mathbf{q}_i - \mathbf{v})^{2\alpha}$. Thus, the deformation is redefined as $\mathcal{M}f_{DMLS}(\mathbf{v}) = l_{\mathbf{v}}^{-1}(\mathbf{v})$. Since $l_{\mathbf{v}}^{-1}$ is an affine transformation, it can be written as

$$l_{\mathbf{v}}^{-1}(\mathbf{x}) = \mathbf{x}\mathbf{M} + \mathbf{t} \ , \tag{3}$$

where $\mathbf{M}$ and $\mathbf{t}$ describe a rotation and a translation respectively. As done by Schaefer, $\mathbf{t}$ can be written as $\mathbf{t} = \mathbf{p}_* - \mathbf{q}_*\mathbf{M}$, where $\mathbf{p}_* = \sum_{i=1}^{N} \omega_i' \mathbf{p}_i / \sum_{i=1}^{N} \omega_i'$ and $\mathbf{q}_* = \sum_{i=1}^{N} \omega_i' \mathbf{q}_i / \sum_{i=1}^{N} \omega_i'$. Therefore, it is possible to rewrite Equation 3 as

$$l_{\mathbf{v}}^{-1}(\mathbf{x}) = (\mathbf{x} - \mathbf{q}_*)\mathbf{M} + \mathbf{p}_* \ , \tag{4}$$

and the MLS problem from Equation 2 becomes

$$\min \sum_{i=1}^{N} \omega_i' |\hat{\mathbf{q}}_i \mathbf{M} - \hat{\mathbf{p}}_i|^2 \ . \tag{5}$$

where $\hat{\mathbf{p}}_i = \mathbf{p}_i - \mathbf{p}_*$ and $\hat{\mathbf{q}}_i = \mathbf{q}_i - \mathbf{q}_*$. The transformation matrix $\mathbf{M}$ determines the behavior of the deformation. In the following, close formylas for the backward mapping versions of the affine, rigid and similarity three-dimensional deformations are derived.

Affine deformations. If no restriction is imposed on $\mathbf{M}$ in Equation 5, the solution is an affine transformation that can be obtained by deriving the equation:

$$\frac{\partial \sum_{i=1}^{N} \omega_i' |\hat{\mathbf{q}}_i \mathbf{M} - \hat{\mathbf{p}}_i|}{\partial \mathbf{M}} = \frac{\partial \sum_{i=1}^{N} \omega_i' (\hat{\mathbf{q}}_i \mathbf{M} - \hat{\mathbf{p}}_i)(\hat{\mathbf{q}}_i \mathbf{M} - \hat{\mathbf{p}}_i)^T}{\partial \mathbf{M}} \tag{6}$$

$$= \sum_{i=1}^{N} \omega_i' \left(\hat{\mathbf{q}}_i^T \hat{\mathbf{q}}_i\right) \mathbf{M} - 2 \sum_{i=1}^{N} \omega_i' \hat{\mathbf{q}}_i^T \hat{\mathbf{p}}_i \ . \tag{7}$$

Thus, the transformation matrix is $\mathbf{M} = \left(\sum_{i=1}^{N} \omega_i' \hat{\mathbf{q}}_i^T \hat{\mathbf{q}}_i\right)^{-1} \sum_{i=1}^{N} \omega_i' \hat{\mathbf{q}}_i^T \hat{\mathbf{p}}_i$.

Rigid deformations. As done by Schaefer, to obtain a rigid deformation it is necessary to restrict $\mathbf{M}$ to a rotation matrix; *i.e.*, $\mathbf{M} \in SO_3(\mathbb{R})$. With this restriction, the optimization problem becomes $\min_{\mathbf{M} \in SO_3(\mathbb{R})} \sum_{i=1}^{N} \omega_i' |\hat{\mathbf{q}}_i \mathbf{M} - \hat{\mathbf{p}}_i|^2$. We can rewrite this equation as

$$\min_{\mathbf{M} \in SO_3(\mathbb{R})} \left(-2 \sum_{i=1}^{N} \omega_i' \hat{\mathbf{q}}_i \mathbf{M} \hat{\mathbf{p}}_i^T + \sum_{i=1}^{N} \omega_i' \hat{\mathbf{p}}_i \hat{\mathbf{p}}_i^T + \sum_{i=1}^{N} \omega_i' \hat{\mathbf{q}}_i \mathbf{M} \mathbf{M}^T \hat{\mathbf{q}}_i^T \right) \ . \tag{8}$$

Since $\mathbf{M}\mathbf{M}^T = \mathbf{I}$ the last two terms are constant and can be disregarded. Since the first term is negative, the problem can be rewritten as

$$\max_{\mathbf{M} \in SO_3(\mathbb{R})} \sum_{i=1}^{N} \omega_i' \hat{\mathbf{q}}_i \mathbf{M} \hat{\mathbf{p}}_i^T \ . \tag{9}$$

Three-dimensional rotation. In order to find a rotation matrix that solves Equation 9 it would be necessary to find nine unknowns. Therefore, a representation that uses less variables, specifically a rotation axis $\mathbf{e}$ and a rotation angle α is used. The rotation matrix depends on $\mathbf{e}$ and α as

$$\mathbf{M} = \mathbf{e}^T \mathbf{e} + \cos(\alpha)(\mathbf{I} - \mathbf{e}^T \mathbf{e}) + \sin(\alpha)[\mathbf{e}]_\times . \tag{10}$$

By replacing this in Equation 9, we obtain

$$\max_{\|e\|=1,\cos(\alpha)^2+\sin(\alpha)^2=1} \left(\mathbf{e}\mathbf{C}\mathbf{e}^T + \cos(\alpha)(s - \mathbf{e}\mathbf{C}\mathbf{e}^T\right) + \sin(\alpha)\mathbf{k}\mathbf{e}^T) \ , \tag{11}$$

where $\mathbf{C} = \sum_{i=1}^{N} \omega_i' \hat{\mathbf{p}}_i^T \hat{\mathbf{q}}_i$, $s = \text{trace}(\mathbf{C})$ and $\mathbf{k} = \sum_{i=1}^{N} \omega_i' \hat{\mathbf{p}}_i \times \hat{\mathbf{q}}_i$. The first restriction in the maximization problem given by Equation 11 makes the vector $\mathbf{e}$ be normalized. The second restriction is used to solve Equation 11 for $\sin(\alpha)$ and $\cos(\alpha)$. For this, we use the Lagrange function

$$\begin{aligned} L(\mathbf{e}, \sin(\alpha), \cos(\alpha); \lambda_1, \lambda_2) = \ & \mathbf{e}\mathbf{C}\mathbf{e}^T + \cos(\alpha)(s - \mathbf{e}\mathbf{C}\mathbf{e}^T) + \sin(\alpha)\mathbf{k}\mathbf{e}^T + \\ & \lambda_1(1 - \|\mathbf{e}\|) + \lambda_2(1 - \cos(\alpha)^2 - \sin(\alpha)^2) \ , \end{aligned} \tag{12}$$

with Lagrange multipliers λ_1 and λ_2. The stationary points of L are the roots of the first partial derivatives of L with respect to $\mathbf{e}$, $\cos(\alpha)$ and $\sin(\alpha)$:

$$(1 - \cos(\alpha))\mathbf{e}(\mathbf{C} + \mathbf{C}^T) + \sin(\alpha)\mathbf{k} = \lambda_1 \mathbf{e} \tag{13}$$

$$s - \mathbf{e}\mathbf{C}\mathbf{e}^T = 2\lambda_2 \cos(\alpha) \tag{14}$$

$$\mathbf{k}\mathbf{e}^T = 2\lambda_2 \sin(\alpha) \ . \tag{15}$$

From Equation 15, $\sin(\alpha) = \mathbf{k}\mathbf{e}^T/(2\lambda_2)$, which replaced in Equation 13 gives

$$\mathbf{e}(\mathbf{C} + \mathbf{C}^T) + \frac{1}{2\lambda_2(1 - \cos(\alpha))}\mathbf{k}\mathbf{e}^T\mathbf{k} = \frac{\lambda_1}{1 - \cos(\alpha)}\mathbf{e} \ . \tag{16}$$

By letting

$$\mathbf{N} = \mathbf{C} + \mathbf{C}^T, \quad a = \frac{1}{2\lambda_2(1 - \cos(\alpha))}, \quad \lambda = \frac{\lambda_1}{1 - \cos(\alpha)} \ , \tag{17}$$

it is possible to rewrite Equation 16 as

$$\mathbf{e}(\mathbf{N} + a\mathbf{k}^T\mathbf{k}) = \lambda \mathbf{e} \ , \tag{18}$$

which means that the rotation axis $\mathbf{e}$ is an eigenvector of the matrix $(\mathbf{N} + a\mathbf{k}^T\mathbf{k})$ corresponding to the eigenvalue λ. Since λ is a root of the characteristic polynomial $P(\lambda)$ of the matrix $(\mathbf{N} + a\mathbf{k}^T\mathbf{k})$, using the Frobenius norm $\|\cdot\|_F$ and the approximation $\det(\mathbf{I} + \mathbf{A}) \approx 1 + \text{trace}(\mathbf{A})$, we have

$$\begin{aligned} P(\lambda) \approx & -\det(\mathbf{N})(1 + \mathbf{k}\mathbf{N}^{-1}\mathbf{k}^T a) + \lambda^3 - \lambda^2 \left[\text{trace}(\mathbf{N}) + a\mathbf{k}\mathbf{k}^T\right] + \\ & \lambda \left[\frac{1}{2}(\text{trace}(\mathbf{N})^2 - \|\mathbf{N}\|_F^2) + a(\mathbf{k}\mathbf{k}^T\text{trace}(\mathbf{N}) - \mathbf{k}\mathbf{N}\mathbf{k}^T)\right] \ . \end{aligned} \tag{19}$$

Here, a is unknown since it depends on λ_2. It is possible to show that a and λ are related by multiplying $\mathbf{e}^T$ to both sides of Equation 13 and using Equations 14 and 15 to obtain $2(1 - \cos\alpha)(s - 2\lambda_2 \cos\alpha) + 2\lambda_2 \sin\alpha \sin\alpha = \lambda_1 \mathbf{e}\mathbf{e}^T$. Since $\mathbf{e}\mathbf{e}^T = 1$, using Equation 17, we obtain

$$a = 1/(\lambda - 2s) \ . \tag{20}$$

Thus, the equation $P(\lambda) = 0$ becomes

$$0 = \det(\mathbf{N})(2s - \mathbf{k}\mathbf{N}^{-1}\mathbf{k}^T) + \lambda\left[4\left(\|\mathbf{C}\|_F^2 - s^2\right)s - 2\mathbf{k}\mathbf{C}\mathbf{k}^T - \det(\mathbf{N})\right] + \lambda^2\left[6s^2 - 2\|\mathbf{C}\|_F^2\right] + \lambda^4 - \lambda^3 4s \ . \quad (21)$$

By replacing $y = \lambda - s$, we obtain the depressed quartic function

$$0 = y^4 - 2\|\mathbf{C}\|_F^2 y^2 - 8\det(\mathbf{C})y + \det(\mathbf{N})(2s - \mathbf{k}\mathbf{N}^{-1}\mathbf{k}^T) - 8\det(\mathbf{C})s + 2\|\mathbf{C}\|_F^2 s^2 - s^4 \ , \quad (22)$$

which can be solved using the method by Ferrari. Since Equation 9 is a maximization problem, the largest real root r_{max} of this function is the solution.

Determining the rotation axis and angle. Given the maximal root of the characteristic polynomial, by substituting $y = \lambda - s$ and using Equation 20, the rotation axis $\mathbf{e}$ can be obtained as an eigenvector of $(\mathbf{N} + a\mathbf{k}^T\mathbf{k})$ corresponding to the now known eigenvalue λ. For small deformations, this approach can be problematic since a can be very large. By substituting $\varpi = -a\mathbf{k}\mathbf{e}^T$ and $\mathbf{u} = \frac{\mathbf{e}}{\varpi}$ it is possible to rewrite Equation 18 as $\mathbf{u}(\mathbf{N} - \lambda\mathbf{I}) = \mathbf{k}$. With this, $\mathbf{e}$ is obtained by normalizing $\mathbf{u}$, which is in turn obtained by means of a matrix inversion. To obtain the rotation angle α, the optimization problem given by Equation 11 is solved with respect to $\sin(\alpha)$ and $\cos(\alpha)$. From Equations 17 and 20 we have $\lambda - 2s = 2\lambda_2(1 - \cos(\alpha))$. By adding Equation 14 we obtain, for λ_2, $2\lambda_2 = \lambda - s - \mathbf{e}\mathbf{C}\mathbf{e}^T$. Therefore, from Equations 14 and 15 we obtain $\cos(\alpha) = (s - \mathbf{e}\mathbf{C}\mathbf{e}^T)/(\lambda - s - \mathbf{e}\mathbf{C}\mathbf{e}^T)$ and $sin(\alpha) = (\mathbf{k}\mathbf{e}^T)(\lambda - s - \mathbf{e}\mathbf{C}\mathbf{e}^T)$.

Similarity deformations. A generalization of rigid deformations are the similarity deformations, which consist of a rotation and a uniform scaling. By introducing a scaling factor $\mu_s \in \mathbb{R}$ into Equation 2, a new minimization problem can be stated as $\min_{\mathbf{M}\in SO_3(\mathbb{R})} \sum_{i=1}^N \omega_i' |\mu_s \hat{\mathbf{q}}_i \mathbf{M} - \hat{\mathbf{p}}_i|$. This can also be restated as a maximization problem as $\max_{\mathbf{M}\in SO_3(\mathbb{R})} 2\mu_s \sum_{i=1}^N \omega_i' \hat{\mathbf{q}}_i \mathbf{M} \hat{\mathbf{p}}_i^T - \mu_s^2 \sum_{i=1}^N \omega_i' \hat{\mathbf{q}}_i \hat{\mathbf{q}}_i^T$. By assembling the Lagrange functions and deriving with respect to μ_s, we obtain the optimality condition $\sum_{i=1}^N \omega_i' \hat{\mathbf{q}}_i \mathbf{M} \hat{\mathbf{p}}_i^T - \mu_s \sum_{i=1}^N \omega_i' \hat{\mathbf{q}}_i \hat{\mathbf{q}}_i^T = 0$. Since the optimality condition of Equation 11 states that solving $\sum_{i=1}^N \omega_i' \hat{\mathbf{q}}_i \mathbf{M} \hat{\mathbf{p}}_i^T = r_{max}$ suffices, we obtain $\mu_s = r_{max}/(\sum_{i=1}^N \omega_i' \hat{\mathbf{q}}_i \hat{\mathbf{q}}_i^T)$.

3 Nonlinear Polynomial Deformations

As mentioned before, nonlinear polynomial transformations are able to provide deformations, such as bending, that cannot be modeled with linear transformations. Nonlinear MLS deformations can be easily obtained by solving Equation 2 with the function $l_{\mathbf{v}}^{-1}(\mathbf{x})$ being a polynomial of arbitrary degree. Let $\Psi = \{\psi_1, \ldots, \psi_M\}$, where ψ_j are basis functions (polynomials in this case) and $\mathcal{F} = \{\mathbf{p}_1, \ldots, \mathbf{p}_N\} \subsetneq \mathbb{R}$. Let also $\Psi_j = [\psi_j(\mathbf{q}_1), \ldots, \psi_j(\mathbf{q}_N)]$, $\Gamma = [\mathbf{p}_1, \ldots, \mathbf{p}_N]$ and define the inner product $\langle , \rangle_{\omega'} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+$ as a weighted sum

$$\langle \xi, \eta \rangle_{\omega'} = \sum_{i=1}^N \xi_i \eta_i \omega_i' \ . \quad (23)$$

Then, the new minimization problem can be formulated as

$$\min \sum_{i=1}^{N} \omega_i'(\mathbf{p}_i - \varrho_{\mathbf{v}}^{-1}(\mathbf{q}_i))^2 \ , \tag{24}$$

where $\varrho_{\mathbf{v}}^{-1}(\mathbf{v}) = \sum_{j=1}^{M} c_j(\mathbf{v})\psi_j(\mathbf{v})$ and c_j are the unknown coefficients to be found. Note that Ψ must not contain constant terms since the translation $\mathbf{t}$ is not directly computed (see Equation 4). These coefficients can be obtained by solving the corresponding normal equation given by

$$\left\{ \sum_{j=1}^{M} \langle \Psi_k, \Psi_j \rangle_{\omega'} c_j = \langle \Gamma, \Psi_k \rangle_{\omega'};\ k = 1, \ldots, M \ . \right. \tag{25}$$

Solving this problem would mean having to invert; for instance, for a complete quadratic polynomial, a 10×10-matrix and for a cubic polynomial a 20×20-matrix. The interest here lies on providing a closed formulation of the polynomial transformation of arbitrary degree. This can be achieved by means of multivariate orthogonal polynomials.

Thus, a set Ψ is defined, as before using some orthogonalization process, such that the inner product satisfies $\langle \Psi_i, \Psi_j \rangle_{\omega'} = \kappa_{ij}\delta_{ij}$, where δ_{ij} is the Kronecker delta, System 25 becomes a linear system where the coefficient matrix is diagonal. Thus, the approximation is given by the sum

$$\varrho_{\mathbf{v}}^{-1}(\mathbf{x}) = \sum_{j=1}^{M} \psi_j(\mathbf{x}) \langle \Gamma, \Psi_j \rangle_{\omega'} / \langle \Psi_j, \Psi_j \rangle_{\omega'} \ . \tag{26}$$

The mapping $l_{\mathbf{v}}^{-1}$ is then given by $l_{\mathbf{v}}^{-1} = \varrho_{\mathbf{v}}^{-1}(\mathbf{x} - \mathbf{q}_*) + \mathbf{p}_*$.

4 GPU-based MLS Displacement Map Computation

To perform hardware-assisted volume rendering of the deformed volume, we followed an approach based on computing a displacement map $\mathcal{D}$ covering the domain of the volume, which provides the map $l_{\mathbf{v}}^{-1}$. The displacement map is stored in a three-dimensional floating point texture and linear interpolation is used to obtain the displacement vector at any point in the domain. Thus, in a GPU-based volume renderer, it suffices to fetch the displacement vector corresponding to the current position on the ray from the displacement texture. This displacement map is calculated on the GPU to accelerated the process and avoid data transfers between CPU and GPU.

To fill the displacement texture, we use `framebuffer objects`. By rendering the slices of the 3D texture each fragment represents a voxel in the displacement texture. Given the position of the fragment in object space, the shader calculates the undeformed position as detailed previously. To perform this computation, the deformed and undeformed positions of the control points $\mathbf{q}_i$ and $\mathbf{p}_i$;

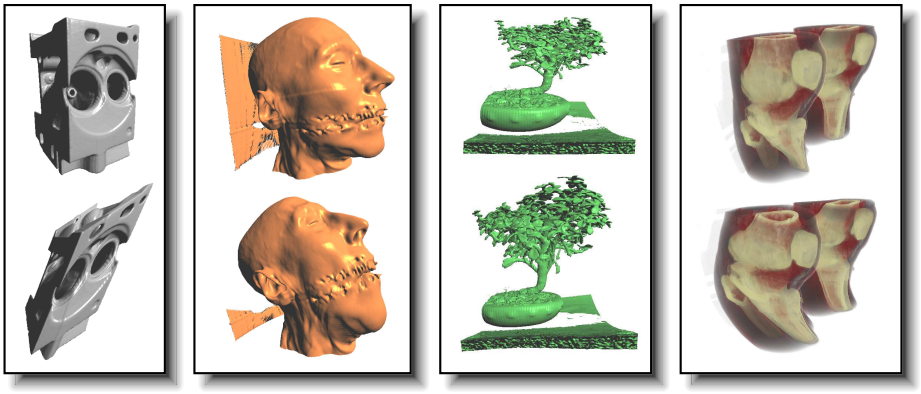

Fig. 1. Examples of affine (left), rigid (center-left), similarity (center-right), and nonlinear polynomial (right) MLS deformations. The bending effect shown in the nonlinear deformation is achieved by moving a single control point (out of 15).

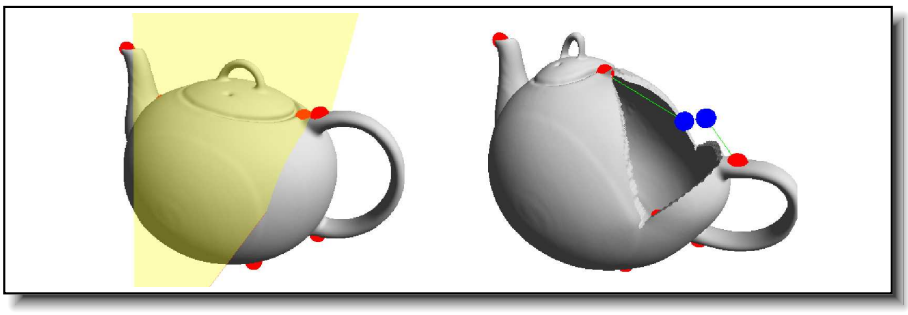

Fig. 2. Example of a cutting plane in combination with a rigid MLS deformation.

$i = 1, \cdots, N$, respectively, are required. Since N varies a constant equal to N is added dynamically as the first line of the shader code. This constant is used throughout the code to define the arrays holding the deformed and undeformed positions. Then, the shader is compiled in run-time each time the number of control points changes. The implementation of the fragment shader that calculates the displacement vector is straightforward, one consideration must be pointed out for nonlinear deformations. However, implementing the Gramm-Schmidt orthogonalization requires Shader Model 4 due to the large number of temporal variables needed.

Introducing cuts into MLS deformations. To introduce cuts as exploration tool, the GPU implementation of the deformation algorithms described above is extended. A cut can be realized by using a two-dimensional cutting element; *e.g.*, a plane, that divides a portion of the volume in two pieces. Without loosing generality, the following discussion will use half-planes.

In Figure 2 it is possible to see two features of cuts with MLS deformations: the manipulation points do not influence the portion of the volume on the other

Table 1. Processing time for the CPU and GPU implementations of the deformation methods (in seconds) for displacement maps of different sizes.

	Deformation algorithm (CPU/GPU)			
Size	affine	rigid	similarity	nonlinear (degree 2)
$64 \times 64 \times 64$	0.633/0.004	1.125/0.008	1.133/0.008	9.27/0.102
$128 \times 128 \times 128$	4.868/0.015	8.779/0.045	8.976/0.045	117.18/0.46
$256 \times 256 \times 256$	38.55/0.104	69.97/0.325	70.55/0.325	793.00/0.73

side of the cut and void regions are created when deforming the volume. To achieve this, a new weight function depending on the position of the point to be deformed relative to the cutting plane is defined as

$$\omega_i' = d(\mathbf{v}, \mathbf{q}_i)|\mathbf{q}_i - \mathbf{v}|^{-2\alpha} \ , \tag{27}$$

where d is a damping function that establishes how much influence a displaced control point $\mathbf{q}_i$ has on a voxel $\mathbf{v}$. If $\mathbf{q}_i$ and $\mathbf{v}$ are not occluded one from the other by the cutting plane, then $d(\mathbf{v}, \mathbf{q}_i) = 1$. The function d tends asymptotically to 0 with the size of the coverage. The damping function used in the tests was

$$d(\mathbf{v}, \mathbf{q}_i) = [(|\mathbf{q}_i - \mathbf{v}|)/(|\mathbf{q}_i - \mathbf{v}| + \phi(\mathbf{v}, \mathbf{q}_i))]^{\nu} \ , \tag{28}$$

where ϕ returns the length of the shortest indirect path from $\mathbf{q}_i$ to $\mathbf{v}$ that does not cross the cutting plane and the factor $\nu > 1$ is defined by the user. The choice of the damping function d influences directly how the cut will look like. If d is discontinuous, undesired sharp edges are obtained. Also, ϕ must be strictly monotonically decreasing with respect to the occlusion between $\mathbf{v}$ and $\mathbf{q}_i$. To create void regions, a test checking if the deformed and the undeformed positions of $\mathbf{v}$ are occluded one from the other by the cutting plane is performed, in which case the deformed position is not taken into account for the rendering.

To visualize the surface of a cut; *e.g.*, during isosurface rendering, the alpha channel of the displacement map is used to store the value 1 if the deformed position of the voxel belongs to a void region. During rendering, the current position on the ray is discarded if the interpolated value of the alpha channel is greater than 0.5.

Deforming normal vectors. Some volume rendering pre-compute gradient information to accelerate the computations. After a deformation, the pre-computed gradient vectors must be corrected. This can be done using an adaptation of the method proposed by Barr [12], who describes the transformation of a normal vector for a forward mapping F as $\mathbf{n_q} = \det \mathbf{J}_F \left(\mathbf{J}_F^{-}1\right)^T \mathbf{n_p}$, where $\mathbf{J}_F$ is the Jacobi matrix of F, $\mathbf{n_p}$ is the normal vector at the undeformed point $\mathbf{p}$ and $\mathbf{n_q}$ is the unknown normal vector at the deformed point $\mathbf{q}$. Let $\mathbf{n_q} = (\det \mathbf{J}_G)^{-1} (\mathbf{J}_G)^T \mathbf{n_p}$, where $\mathbf{J}_G$ is the Jacobi matrix of the backward mapping G. It is easy to see that for the Jacobi matrix $\mathbf{J}_D$ of the displacement map, $\mathbf{J}_G = \mathbf{I} + \mathbf{J}_D$, where $\mathbf{I}$ is the identity matrix. Thus,

$$\mathbf{n_q} = (\det(\mathbf{I} + \mathbf{J}_D))^{-1} (\mathbf{I} + \mathbf{J}_D)^T \mathbf{n_p} \ . \tag{29}$$

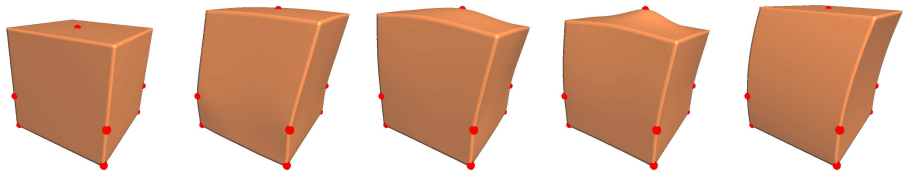

Fig. 3. Visual comparison of the different MLS deformation methods. From left to right: original model, affine deformation, similarity deformation, rigid deformation and nonlinear polynomial deformation of degree 2.

If the vectors are normalized, $1/\det(\mathbf{I}+\mathbf{J}_D)$ can be disregarded. $\mathbf{J}_D$ can be computed from the displacement texture by means of finite differences. To accelerate the rendering, the matrix $(\mathbf{I}+\mathbf{J}_D)^T$ can be pre-computed and stored in an additional texture. The gradient vector at the cut boundaries can be easily calculated as the gradient of the values in the alpha channel of the displacement texture.

5 Results and Discussion

In Figure 1 it is possible to see the characteristic behavior of each type of deformation. The effect obtained with affine these deformations seems very unnatural compared to the obtained with the rigid deformation. The similarity deformation shown in the figure, on the other hand, depicts clearly the expected scaling. A visual comparison of the deformations is shown in Figure 3. The affine transformation scales the cube, while the similarity transformation increases the size of the top of the cube. The rigid transformation avoids the scaling but the bump on the top is very pronounced as expected. The nonlinear deformation (degree 2) shows how bending can be achieved by moving only one manipulation point. This can also be seen in Figure 1. In Figure 2, a cut obtained with a rigid deformation is depicted. The use of manipulation points allows the user to easily change the appearance and nature of the cut.

Table 1 shows the performance of the deformation algorithms on the CPU and the GPU for different displacement map sizes and number of control points. The tests were performed on a standard PC equipped with a 2GHz processor, 1GB RAM and an NVidia GeForce 8800 Ultra. It is important to note that due to the use of backward mapping, no pre-processing can be applied to accelerate the computation of the displacement map as it was done by Schaefer *et al.* [13]. As expected, it is possible to see that the processing time is proportional to the size of the displacement texture. The computation of the displacement map is not interactive for medium and large resolutions. On the other hand, the GPU is well suited for this problem as shown in the table, where it can be seen that we can achieve a performance increase of a factor of 100 compared to the CPU implementation.

6 Conclusion

While physically-based deformations are suitable for simulation tasks and visualization problems where realism is important, MLS deformations are more suitable for exploratory tasks due to the ease of manipulation, for instance, in illustrative, educational and entertainment software. One task that is difficult to accomplish with both physically-based and non-physically-based deformations is performing cuts. In the case of MLS deformations this is easily accomplished by modifying the weighting function as was shown above. Also, the use of graphic hardware to accelerate the computations aid in attaining the interactivity desired. By using features of modern graphics processors, it is possible to perform even complicated computations such as the orthogonalization of the polynomial basis. However, it is necessary to extend our implementation to include other manipulation tools besides cutting planes. Also, the artifacts produced by the linear interpolation of the indicators of void regions must be addressed.

References

1. Müller, M., Heidelberger, B., Teschner, M., Gross, M.: Meshless deformations based on shape matching. ACM Transactions on Graphics **24**(3) 471–478 (2005)
2. Chen, Y., Zhu, Q., Kaufman, A., Muraki, S.: Physically-based animation of volumetric objects. In: Proceedings of the Computer Animation. 154–160, IEEE Computer Society (1998)
3. Westermann, R., Rezk-Salama, C.: Real-time volume deformations. Computer Graphics Forum **20**(3) 443–451 (2001)
4. Gibson, S.F.: 3D chainmail: a fast algorithm for deforming volumetric objects. In: Proceedings of the Symposium on Interactive 3D Graphics. 149–ff., ACM (1997)
5. Alexa, M., Cohen-Or, D., Levin, D.: As-rigid-as-possible shape interpolation. In: Proceedings of the ACM SIGGRAPH. 157–164, ACM Press/Addison-Wesley Publishing Co. (2000)
6. Schaefer, S., McPhail, T., Warren, J.: Image deformation using moving least squares. ACM Transactions on Graphics **25**(3) 533–540 (2006)
7. Igarashi, T., Moscovich, T., Hughes, J.F.: As-rigid-as-possible shape manipulation. In: Proceedings of the ACM SIGGRAPH. 1134–1142, ACM Press (2005)
8. Cuno, A., Esperança, C., Oliveira, A., Roma, P.: 3D as-rigid-as-possible deformations using MLS. In: Proceedings of Computer Graphics International. 115–122 (2007)
9. Horn, B.: Closed-form solution of absolute orientation using orthonormal matrices. Journal of the Optical Society of America **5**(7) 1127–1135 (1987)
10. Arun, K.S., Huang, T.S., Blostein, S.D.: Least-squares fitting of two 3-d point sets. IEEE Transactions on Pattern Analysis and Machine Intelligence **9**(5) 698–700 (1987)
11. Horn, B.: Closed-form solution of absolute orientation using unit quaternions. Journal of the Optical Society of America **5**(4) 629–642 (1987)
12. Barr, A.H.: Global and local deformations of solid primitives. In: Proceedings of the ACM SIGGRAPH. 21–30, ACM Press (1984)
13. Schaefer, S., McPhail, T., Warren, J.: Image deformation using moving least squares. In: Proceedings of ACM SIGGRAPH. 533–540, ACM Press (2006)