Volume-preserving deformation using generalized barycentric coordinates

M.Àngels Cerveró Àlvar Vinacua Pere Brunet

macervero@lsi.upc.edu alvar@lsi.upc.edu pere@lsi.upc.edu

Departament de Llenguatges i Sistemes Informàtics LSI

Universitat Politècnica de Catalunya

Campus Nord

08034 Barcelona

Abstract

The *cage-based deformation* of a 3D object through *generalized barycentric coordinates* is a simple, efficient, effective and hence widely used shape manipulation scheme. Editing vertices of the polyhedral cage induces a smooth space deformation of its interior; the vertices thus become control handles of the final deformation. However, in some application fields, as medicine, constrained volume-preserving deformations are required.

In this paper, we present a solution that takes advantage of the potential of the deformations based on *generalized barycentric coordinates* while adding the constraint of keeping a volume constant. An implementation of the proposed scheme is presented and discussed. A measure of local stress of the deformed volume is also proposed.

1 Introduction

The deformation of geometric models has been extensively studied for a number of different applications. Seminal works in the area include [16] and [7]. Another approach to deformation, extensively used in the field of character animation, operates through an association with an abstract skeleton.

Here we focus on methods based on a *space deformation*, and more specifically, on those that specify that deformation by enclosing the area of interest inside a (sometimes convex) polyhedron, transferring the deformations that the user prescribes for that polyhedron to a region of space, usually inside the polyhedron, in a smooth way. This transference takes place by means of a coordinate system defined in the spirit of *barycentric coordinates* for simplexes, albeit overdetermined, since the polyhedron is usually not a simplex. These *cage-based deformation* schemes have been proved useful in the deformation and animation of mesh models, where the number of free controls is drastically reduced, while retaining an intuitive interface and sufficient flexibility.

Motivated by problems where arbitrary deformations are not acceptable, because certain physical constraints need to be enforced, we discuss here the problem of constraining these cage-based methods in order to preserve the volume enclosed by a mesh modeling the boundary of a solid and deformed through the cage and an appropriate *barycentric coordinate* system. Our implementation of this technique is based on the *Mean-Value Coordinates* introduced in [9], but the results are applicable to other *barycentric coordinate* schemes that use only the positions of the cage's vertices.

In the following section, we discuss more in detail the previous work, and the main properties of *barycentric coordinates* studied therein. We also present some prior work focused on preserving the volume of deformed objects. We then proceed in Section 3 to discuss the volume-preservation constraint and how it limits the deformations of the cage itself, one vertex at a time. Then Section 4 discusses our approach to measure the local stress produced, and finally, before presenting our conclusions, Section 5 gives details on our implementation and the results obtained with several test models.

2 Previous work

Barycentric Coordinates, as proposed by Möbius in 1827, are a very elegant and simple way to define the position of points $\mathbf{p}$ in simplex domains (triangles in 2D, tetrahedra in 3D, etc.).

In recent years, a number of authors have proposed extensions of this scheme in the case of more general polygons in (2D) and polyhedra (in 3D). Coordinates of this kind are called *Generalized Barycentric Coordinates.* In [15, 9, 10, 12], the position of any point $\mathbf{p}$ can be computed as a convex combination of the vertices $\mathbf{v}^c$ of the polygon (or polyhedron) with the coordinates of $\mathbf{p}$ as weights:

$$\mathbf{p} = \sum_{i \in I_{V^c}} \mathbf{v}_i^c \alpha_i(\mathbf{p}), \qquad (1)$$

where I_{V^c} are the indices of the vertices of the cage.

Green Coordinates [14] complement the Equation 1 by also including a linear combination of the normals of the polyhedron faces. This results on a double set of coordinates:

$$\mathbf{p} = \sum_{i \in I_{V^c}} \mathbf{v}_i^c \alpha_i(\mathbf{p}) + \sum_{j \in I_{F^c}} \vec{\mathbf{n}}_j^c \beta j(\mathbf{p}), \qquad (2)$$

where I_{F^c} are the indices of the faces of the cage.

Generalized Barycentric Coordinates satisfy the two well-known properties of the standard *Barycentric Coordinates*:

- Constant precision (reproduction of unity):

$$\sum_{i \in I_{V^c}} \alpha_i(\mathbf{p}) = 1 \quad \forall \mathbf{p}$$

- Linear precision (reproduction of the identity):

$$\mathbf{p} = \sum_{i \in I_{V^c}} \mathbf{v}_i^c \alpha_i(\mathbf{p}) \quad \forall \mathbf{p}$$

In 2002, Meyer et al. developed the *Generalized barycentric coordinates* [15], a 2D system that can be used over any convex polygon (not only triangles). A year later, Floater presented the *Mean-Value Coordinates* [9], a 2D system that improves Meyer's because it can be used not only over convex polygons but also in the kernel of concave ones. These coordinates were generalized to 3D polyhedral domains in 2005 by Floater et al. [10].

In 2007, Meyer et al. introduced the *Harmonic coordinates* [12], based on the solutions to the Laplace's equation (harmonic functions). They work on both convex and concave polygons and polyhedra. This scheme has the disadvantage, however, of not having a closed form solution. The boundary conditions and the solutions must be defined for every particular case.

Based on the Green's function, Lipman et al. developed the *Green coordinates* [14] in 2008. These coordinates can be used over convex and concave polygons and polyhedra and, moreover, being based on harmonic functions, produce a conformal mapping in 2D and a quasi-conformal one in 3D.

All of these schemes are usable for *cage-based object deformation.* In this case, the object to be deformed is located into a polyhedron (or polygon) which in this context is called cage. The vertices of the cage will be used as control points of the deformation (together with the face normals in the case of *Green Coordinates*). When the vertices of the cage are interactively edited, the object is deformed according to the *generalized barycentric coordinates* of its mesh vertices. The overall deformation works in two steps [6, 14] (see Figure 1):

1. Preprocess step (Initial conditions): the *generalized barycentric coordinates* with respect to the initial cage vertices are computed (and stored) for each mesh vertex of the object.

2. Deformation step: every time that the cage is modified, the mesh inside it is recalculated using the new cage and the coordinates computed in Step 1.

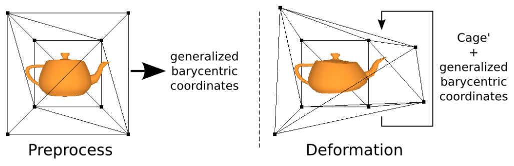

Figure 1: Cage-based deformation pipeline

The use of a particular *generalized barycentric coordinates* system is only affecting the initial conditions in Step 1. Coordinates are never recomputed during the real-time interaction in Step 2, because of the complexity involved in this recomputation. It is in Step 2 that properties such as the preservation of the volume of the initial object might be preserved during the interaction, and as a consequence, the constraints are independent of the *generalized barycentric coordinates* scheme being used.

Tables 1 and 2 present a comparison among the discussed *generalized barycentric coordinate* schemes. It can be observed that most systems support 3D domains. Guaranteeing a conformal mapping is not simple, but we have observed that this is not a fundamental requirement for the applications that will be presented in the next Sections.

On the other hand, there are some research articles that present deformation systems (not cage-based) that preserve the volume of the object being modified.

In 1997, Aubert et al. [3] described a volume-preserving free-form deformation technique based on the DOGME [4] model. The DOGME model deforms an object placing some constraint points on the desired locations and guaranteeing some constraints through an optimization parameter. The volume is preserved with an iterative algorithm that seeks the deformation that keeps volume constant and minimizes the norm of the DOGME optimization parameter.

Later, in 1999, Hirota et al. [11] developed another volume-preserving free-form deformation method. In this case, the goal is achieved through a constrained spring energy minimization (each control point of the deformation is connected to its neighbours in the control lattice through a spring). Every time the user moves one of the control points, the algorithm finds a new configuration for the lattice in such a way that the spring energy of the system is the minimum that keeps constant volume.

In 2006, von Funck et al. [17] developed a vector field based deformation. This method divides the deformation space into three subspaces: the *inner region*, where the deformation is guided by a constant or linear vector field, the *outer region*, where no deformation occurs, and the *intermediate region*, where the deformation is performed with a divergence-free and C^1 continuous vector field. Each time the user induces a modification, the system performs a path line integration and updates the vector field to find the new position of the 3D model.

Some authors (like [5] and [13]) discuss the local preservation of volume, which is not to be confused with the object of this work. Their aim is to preserve details on a surface as it evolves. For example, in [5], Botsch et al. presented a method to deform a 3D object keeping its local volume constant, based on multiresolution surfaces and displacement volumes. The technique consists in simplifying the initial detailed surface S_d into a smoother surface S and storing the details as volume displacements (the volume of the prism that every triangle on S_d projects over S). When S is deformed, obtaining as a result S', the details are added back with an iterative algorithm that reconstructs S'_d. This algorithm finds the deformed points $\mathbf{p}'_i$ of S'_d by minimizing the local error in the volumes of the prisms to which $\mathbf{p}'_i$ belongs.

3 Volume preservation

As previously mentioned, our goal is to induce *cage-based deformations* with volume preservation on a 3D triangular mesh. From the

Coordinates	Dimension		Kind of polygon/polyhedron		
	$\mathbb{R}^2$	$\mathbb{R}^3$	Simplex	General	
				Concave	Convex
Barycentric	X	X	X		
Generalized barycentric	X				X
Harmonic	X	X		X	X
Mean-Value	X			X	X
Mean-Value in 3D		X		X	X
Green	X	X		X	X

Table 1: Comparison between the coordinate systems studied.

Coordinates	Basic formula		(Quasi) Conformal mapping	
	pos (1)	pos & $\vec{\mathbf{n}}$(2)	Yes	No
Barycentric	X			X
Generalized barycentric	X			X
Harmonic	X			X
Mean-Value	X			X
Mean-Value in 3D	X			X
Green		X	X	

Table 2: Comparison between the coordinate systems studied.

analysis in Tables 1 and 2 we decided to use *3D Mean-Value Coordinates* because of their computational simplicity and acceptable generality. *Mean-Value Coordinates* use the scheme in Equation 1 to compute the coordinates of the mesh vertices $\mathbf{p}$ in terms of the cage vertices $\mathbf{v}^c$. The $\alpha(\mathbf{p})$ values for a general interior point $\mathbf{p}$ of the cage are computed as follows [9]:

$$\begin{aligned} \alpha_i(\mathbf{p}) &= \frac{w_i(\mathbf{p})}{\sum_{k \in I_{V^c}} w_k(\mathbf{p})} \qquad (3) \\ w_i(\mathbf{p}) &= \frac{1}{r_i} \sum_{\mathbf{v}_i^c \in T} \frac{\gamma_{jk} + \gamma_{ij} \vec{\mathbf{n}}_{ij} \cdot \vec{\mathbf{n}}_{jk} + \gamma_{ki} \vec{\mathbf{n}}_{ki} \cdot \vec{\mathbf{n}}_{jk}}{2 \vec{\mathbf{e}}_i \cdot \vec{\mathbf{n}}_{jk}}, \end{aligned}$$

where T are the triangular faces of the cage. As shown in Figure 2, the coordinates depend on the angles and normals of the lateral faces of the pyramids between the internal point $\mathbf{p}$ and each of the triangular faces of the cage.

Once we have introduced the basic components of our solution, in the next subsections we will present some tools to analyse the behaviour of the volume of the interior objects during the deformation process.

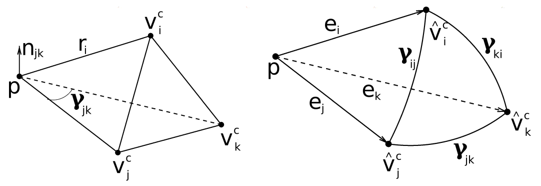

Figure 2: Mean-Value Coordinates 3D

3.1 Mesh volume and the Gauss Theorem

We need a computational tool to compute the volume of a closed 3D model from the information of its boundary surface (triangular mesh) S. Such a tool is the *Gauss Theorem* or *Divergence Theorem* [2], which states the following:

$$\iiint_V \nabla \cdot \mathbf{F} dV = \iint_S \mathbf{F} \cdot \vec{\mathbf{n}} dS, \qquad (4)$$

where $\mathbf{F}$, in the right part of the equation, is computed on all points on the surface S, and $\vec{\mathbf{n}}$ are the outward normals at these points. By taking advantage of the discrete mesh structure of S, the surface integral in Equation 4 can be computed in different ways. We use Gauss numerical integration [1] because of its known stability properties and absence of truncation errors:

$$\iint_S \mathbf{F} \cdot \vec{\mathbf{n}} dS = \frac{1}{3} \sum_{j=1}^{m} \vec{\mathbf{n}}_j \cdot \left(\sum_{i=1}^{n} \boldsymbol{\xi}_i \omega_i \right),$$

where m is the number of triangles, or faces, (t_j) in the mesh, n is the order of the method, ω_i are the Gauss weights, $\vec{\mathbf{n}}_j$ is the outward normal of t_j with $\|\vec{\mathbf{n}}_j\| = 2 \cdot Area(t_j)$ and $\boldsymbol{\xi}_i$ are the points over t_j defined by:

$$\boldsymbol{\xi}_i = (\mathbf{p}_j^0 h_i^0 + \mathbf{p}_j^1 h_i^1 + \mathbf{p}_j^2 h_i^2),$$

where $\mathbf{h}_i = (h_i^0, h_i^1, h_i^2)$ are the standard Gauss *barycentric coordinates* of the integration points in the face t_j [8] and $\{\mathbf{p}_j^k\}_{k=0..2}$ are the vertices of t_j.

Furthermore, we know that the vertices of the mesh are defined with the *Mean-Value Coordinates* by Equation 1. If we put this information all together we obtain:

$$\begin{aligned} V &= \frac{1}{3} \sum_{j=1}^{m} \vec{\mathbf{n}}_j \cdot \sum_{i=1}^{n} \boldsymbol{\xi}_i \omega_i \\ &= \frac{1}{3} \sum_{j=1}^{m} \vec{\mathbf{n}}_j \cdot \sum_{i=1}^{n} (\mathbf{p}_j^0 h_i^0 + \mathbf{p}_j^1 h_i^1 + \mathbf{p}_j^2 h_i^2) \omega_i \\ &= \frac{1}{3} \sum_{j=1}^{m} \vec{\mathbf{n}}_j \cdot \sum_{i=1}^{n} \left[\left(\sum_{k \in I_{V^c}} \mathbf{v}_k^c \alpha_k^0 \right) h_i^0 + \right. \\ &\left. + \left(\sum_{k \in I_{V^c}} \mathbf{v}_k^c \alpha_k^1 \right) h_i^1 + \left(\sum_{k \in I_{V^c}} \mathbf{v}_k^c \alpha_k^2 \right) h_i^2 \right] \omega_i \end{aligned} \qquad (5)$$

3.2 The Volume gradient

Equation 5 enables us to compute the gradient (∇) of the volume V when we perform an infinitesimal displacement of a single cage vertex $\mathbf{v}_q^c$. Using standard derivation techniques we obtain:

$$\frac{\partial V}{\partial v_{q_x}^c} = \frac{1}{3} \left(\sum_{j=1}^{m} \frac{\partial \vec{\mathbf{n}}_j}{\partial v_{q_x}^c} \cdot \sum_{i=1}^{n} \boldsymbol{\xi}_i \omega_i + \sum_{j=1}^{m} \vec{\mathbf{n}}_j \cdot \sum_{i=1}^{n} \frac{\partial \boldsymbol{\xi}_i}{\partial v_{q_x}^c} \omega_i \right)$$

$$\frac{\partial V}{\partial v_{q_y}^c} = \frac{1}{3} \left(\sum_{j=1}^{m} \frac{\partial \vec{\mathbf{n}}_j}{\partial v_{q_y}^c} \cdot \sum_{i=1}^{n} \boldsymbol{\xi}_i \omega_i + \sum_{j=1}^{m} \vec{\mathbf{n}}_j \cdot \sum_{i=1}^{n} \frac{\partial \boldsymbol{\xi}_i}{\partial v_{q_y}^c} \omega_i \right)$$

$$\frac{\partial V}{\partial v_{q_z}^c} = \frac{1}{3} \left(\sum_{j=1}^{m} \frac{\partial \vec{\mathbf{n}}_j}{\partial v_{q_z}^c} \cdot \sum_{i=1}^{n} \boldsymbol{\xi}_i \omega_i + \sum_{j=1}^{m} \vec{\mathbf{n}}_j \cdot \sum_{i=1}^{n} \frac{\partial \boldsymbol{\xi}_i}{\partial v_{q_z}^c} \omega_i \right)$$

$$\nabla V = \left(\frac{\partial}{\partial v_{q_x}^c}, \frac{\partial}{\partial v_{q_y}^c}, \frac{\partial}{\partial v_{q_z}^c} \right) V \qquad (6)$$

Of course, any infinitesimal displacement $\vec{\boldsymbol{\delta}}$ of $\mathbf{v}_q^c$ such that $\vec{\boldsymbol{\delta}} \cdot \nabla V = 0$ will preserve the volume enclosed by our mesh. We now turn to consider larger displacements.

3.3 Gradient's behaviour during the deformation

Let us note $\nabla V(\mathbf{v}_q^c)$ the gradient of the volume with respect to $\mathbf{v}_q^c$, Equation 6. It can be shown, after some algebraic manipulations, that $\nabla V(\mathbf{v}_q^c) = \nabla V(\mathbf{v}_q^c + \vec{\boldsymbol{\delta}})$ for any $\vec{\boldsymbol{\delta}}$ orthogonal to the gradient.

We can therefore conclude that for any cage vertex $\mathbf{v}_q^c$, both the volume enclosed by the deformed mesh and the volume gradient $\nabla(\mathbf{v}_q^c)$ remain constant. In other words, for every cage vertex $\mathbf{v}_q^c$ we have an associated plane containing the vertex and having a normal vector coinciding with the volume's gradient (see Figure 3), such that when we move this vertex $\mathbf{v}_q^c$ within this plane, the volume of the 3D mesh inside the cage remains constant.

The practical implementation of this algorithm is straightforward: when the user selects a cage vertex to be moved, we compute the volume's gradient and the corresponding control plane. The interface ensures volume

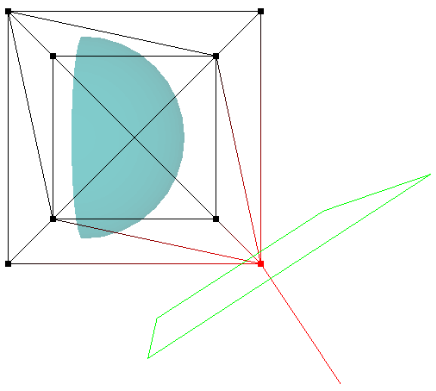

Figure 3: Cage with plane associated to a vertex

preservation by simply constraining the displacement of the selected vertex to its corresponding plane.

4 Local deformation stress

Given a point $\mathbf{r}$ inside the volume of the object to be modified, we want to study the distortion that the deformation causes around it. Let us define an arbitrarily small sphere centered on $\mathbf{r}$ and compute a unit vector $\vec{\mathbf{g}}$ in the direction of $\mathbf{r}_s - \mathbf{r}$ for each point $\mathbf{r}_s$ on its surface. Our goal is to study the modifications induced to this vector $\vec{\mathbf{g}}$ during the deformation (see Figure 4).

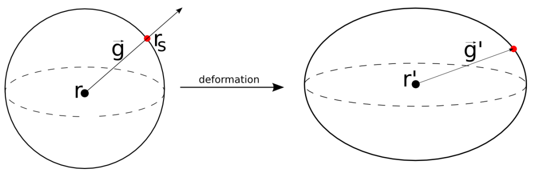

Figure 4: Deformation in the space around an $\mathbf{r}$ point

By computing directional derivatives in Equation 1 (using Equation 3 to calculate the coordinates), we can derive the linear expression that shows the modification of $\vec{\mathbf{g}}$

$$\vec{\mathbf{g}}' = \mathbf{M} \cdot \vec{\mathbf{g}},$$

where matrix $\mathbf{M}$ is:

$$\mathbf{M} = \sum_{i \in I_{V^c}} \left[\frac{\partial \alpha_i}{\partial x}, \frac{\partial \alpha_i}{\partial y}, \frac{\partial \alpha_i}{\partial z} \right] \cdot \mathbf{v}_i^{c'}$$

Let us now perform a Singular-Value Decomposition on $\mathbf{M}$ ($\mathbf{M} = \mathbf{U}^{-1}\mathbf{S}\mathbf{V}$). We obtain:

- $\mathbf{U}$ and $\mathbf{V}$: two orthogonal matrices that produce the same rotations on any $\vec{\mathbf{g}}$ and, therefore, introduce no local deformation.
- $\mathbf{S}$: an anisotropic scaling $\mathbf{S} = diag(s_1, s_2, s_3)$ which introduces a rotational stress.

We define the stretch (or rotational stress) in the point $\mathbf{r}$ as the maximum rotation produced by $\mathbf{S}$, once the isotropic transformations $\mathbf{U}$ and $\mathbf{V}$ have been removed:

$$\sigma = \max_g \left[\arccos \left(\frac{\vec{\mathbf{g}} \cdot \mathbf{S}\vec{\mathbf{g}}}{\|\mathbf{S}\vec{\mathbf{g}}\|} \right) \right]$$

After doing some simulations with numerous triplets of values (s_1, s_2, s_3) with $s_1 \geq s_2 \geq s_3$, we have observed that the rotational stress produced by the singular values depends on the maximum and minimum values between them, that is, s_1 and s_3 (see Figure 5).

Figure 6 shows the maximum angle produced in the anisotropic scaling depending on s_1 and s_3.

Moreover, we have approximated the angle function depending on s_1 and s_3 with a polynomial of degree 6 with an average absolute error of $0.44°$.

We plan to use this measure in order to perform deformations moving more than one vertex of the cage simultaneously. This means that the user will be able to move a set of vertices of the cage without any restriction while the algorithm will compute how to move some of the free remaining vertices in order to preserve the volume and trying to minimize the rotational stress produced in the witness points $\mathbf{r}$ inside the volume of the object.

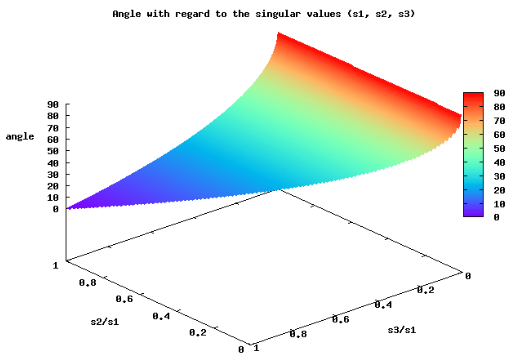

Figure 5: Maximum angle values with regard to (s_1, s_2, s_3)

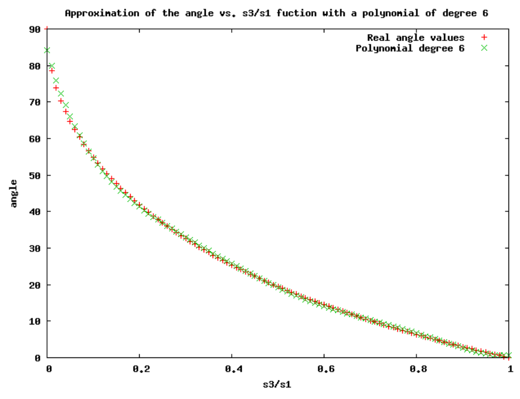

Figure 6: Approximation of the angle function with a polynomial of degree 6

5 Implementation and Results

In this section, we present the results obtained in applying our technique, which we have implemented partially in the GPU. The gradient of the volume with respect to a cage vertex is calculated in the CPU, because this operation has to be performed only once, when a control vertex is picked to be moved. For each new position of the cage's vertex, the deformation is computed in the GPU and the new point is refreshed with the result at an interactive rate.

5.1 Calculation process in GPU

According to the steps indicated in Section 2, we proceed as follows:

1. Preprocess step (Initial conditions): the computed *Mean-Value Coordinates* of the vertices of the mesh are stored in a texture array of floats of 32 bits in the CPU. To be more precise, if the cage has N vertices, we have N *Mean-Value Coordinates* for each vertex of the mesh. To store them, we need M RGB or RGBA textures, depending on if N can be divided by 3 or by 4 respectively. Once we know how many textures will be used, we have to store the *Mean-Value Coordinates* of the i-th vertex of the mesh in the i-th pixel of these textures. For example, imagine we have a cage with $N = 20$ vertices; this means that we need a texture array of $M = 5$ RGBA textures. Then, we store the *Mean-Value Coordinate* 1 to the *Mean-Value Coordinate* 4 of the i-th vertex of the mesh in the i-th pixel of the first texture, the *Mean-Value Coordinate* 5 to the *Mean-Value Coordinate* 8 of the same vertex in the i-th pixel of the second texture, and so on.

2. Deformation step: every time the cage is modified we compute the deformation in the GPU in a two-pass algorithm. First, a shader reads the textures created in the Preprocess step and, using Equation 1, calculates the new mesh. The new positions of the vertices are stored in another texture (see Figure 7) and the information is passed to the second step through a *Pixel Buffer Object*. More precisely, in order to get the new position of the i-th vertex, we multiply the first control vertex with the R component of the first texture, the second control vertex with the G component of the same texture, and so on. Then, we compute the sum of all this products and the result is stored in the i-th pixel of a new RGB texture of floats of 32 bits. This new texture is used to initialize the *Pixel Buffer Object* which will be used as a *Vertex Buffer Object* to

render the geometry in the second render-pass.

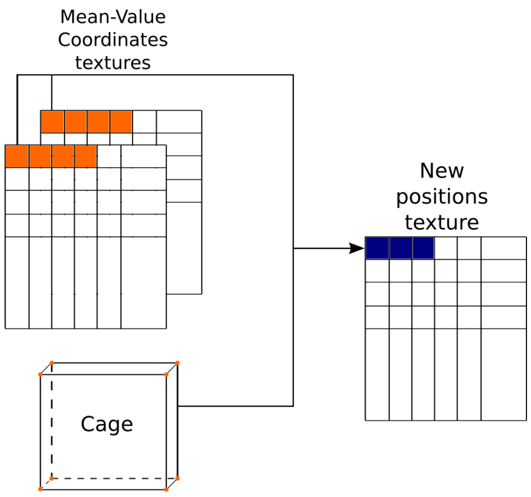

Figure 7: GPU shader

With this GPU-based implementation, we can deform medium-sized models in real time, as shown in Table 3. This table shows the time costs of our algorithm (the computation of the gradient in the CPU and the deformation in the GPU). It also presents the comparison between doing the deformation in the GPU or doing it in the CPU. We can observe that if we use the GPU implementation we can accelerate the deformation process between 50 or 60 times for the big models (Cylinder and Bust).

These tests have been made in an Intel Core2 Duo E8400 with an Nvidia GTX280.

6 Conclusions

We have introduced a technique to preserve the volume of 3D models in a *cage-based deformation* method. When the user selects a vertex of the cage to move, we compute a control surface such that constraining the vertex in to this surface is equivalent to constraining the volume to be constant (see Figure 8).

Moreover, we have demonstrated that, if we use a *generalized barycentric coordinate* system of the family of Equation 1, the control surface is found to be a constant plane through a vertex of the cage. Using this, we give a closed formula to calculate the gradient (the normal of this plane) in an easy and efficient way.

A measure of the local rotational stress has also been proposed. We are currently working in the development of efficient algorithms for *volume-preserving cage-based deformation* with a constraint of minimal stress in a number of predefined witness points.

We have also presented an implementation using the capabilities of GPUs in order to increase the performance of the computation of the deformation to achieve real time response for models of fairly large scale (see Table 3).

Acknowledgments

This project has been funded by *Comissionat per a Universitats i Recerca del Departament d'Innovació, Universitats i Empresa de la Generalitat de Catalunya i Fons Social Europeu* and by the project *TIN2007-67982-C02-01* of the Spanish Government.

We wish to also thank the reviewers for their helpful comments which have helped us improve this final revision.

References

[1] M. Abramowitz and I. Stegun. *Handbook of mathematical functions: with formulas, graphs and mathematical tables.* New York. John Wiley and Sons, 1964.

[2] T. Apostol. *Calculus, Volume II: Multi-Variable Calculus and Linear Algebra with Applications.* Wiley, 2nd Edition June 1984.

[3] F. Aubert and D. Bechmann. Volume-preserving space deformation. *Comput. Graph.*, 21(5):625–639, 1997.

[4] P. Borrel and D. Bechmann. Deformation of n-dimensional objects. In *SMA '91: Proceedings of the first ACM symposium on Solid modeling foundations and*

Model	Vertices	Triangles	Gradient time (ms)	Deformation time (ms)	
				GPU	CPU
Teapot	530	1024	1.98	0.18	7.43
Semi-sphere	2524	1054	2.03	0.37	3.38
Monkey	2868	968	1.86	0.19	3.78
Bunny	35947	69451	124.97	0.81	32.69
Cylinder	303504	266240	369.79	4.48	272.58
Bust	893866	1782976	2303.58	15.69	830.95

Table 3: Comparison between calculation times of the Gradient (CPU), the Deformation in GPU and the Deformation in CPU.

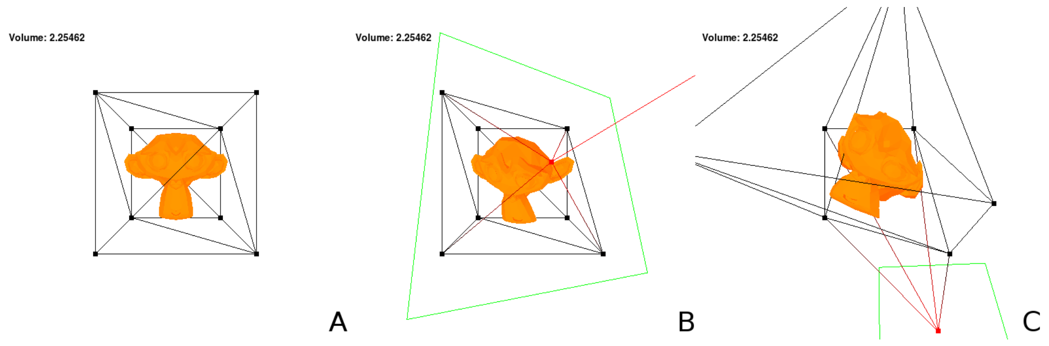

Figure 8: Deformation sequence

CAD/CAM applications, pages 351–369, New York, NY, USA, 1991. ACM.

[5] M. Botsch and L. Kobbelt. Multiresolution surface representation based on displacement volumes. *Computer Graphics Forum 22(3), Proc. Eurographics 2003*, 2003.

[6] D. Cohen-Or. Space deformations, surface deformations and the opportunities in-between. *J. Comput. Sci. Technol.*, 24(1):2–5, 2009.

[7] S. Coquillart. Extended free-form deformation: a sculpturing tool for 3d geometric modeling. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 187–196, New York, NY, USA, 1990. ACM.

[8] G. R. Cowper. Gaussian quadrature formulas for triangles. *International Journal for Numerical Methods in Engineering*, 7(3):405–408, 1973.

[9] M. S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27, March 2003.

[10] M. S. Floater, G. Kós, and M. Reimers. Mean value coordinates in 3d. *Comput. Aided Geom. Des.*, 22(7):623–631, 2005.

[11] G. Hirota, R. Maheshwari, and M. C. Lin. Fast volume-preserving free form deformation using multi-level optimization. In *SMA '99: Proceedings of the fifth ACM symposium on Solid modeling and applications*, pages 234–245, New York, NY, USA, 1999. ACM.

[12] P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki. Harmonic coordinates for character articulation. *ACM Trans. Graph.*, 26(3):71, 2007.

[13] Y. Lipman, D. Cohen-Or, R. Gal, and D. Levin. Volume and shape preservation via moving frame manipulation. *ACM Trans. Graph.*, 26(1):5, 2007.

[14] Y. Lipman, D. Levin, and D. Cohen-Or. Green coordinates. *ACM Trans. Graph.*, 27(3):1–10, 2008.

[15] M. Meyer, A. Barr, H. Lee, and M. Desbrun. Generalized barycentric coordinates on irregular polygons. *J. Graph. Tools*, 7(1):13–22, 2002.

[16] T. W. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph.*, 20(4):151–160, 1986.

[17] W. von Funck, H. Theisel, and H.-P. Seidel. Vector field based shape deformations. *ACM Trans. Graph.*, 25(3):1118–1125, 2006.