*Cages: A multi-level, multi-cage based system for mesh deformation

Francisco González García, Teresa Paradinas, Narcís Coll and Gustavo Patow
University of Girona

Cage-based deformation has been one of the main approaches for mesh deformation in recent years, with a lot of interesting and active research. The main advantages of cage-based deformation techniques are their simplicity, relative flexibility and speed. However, to date there has been no widely accepted solution that provides both user control at different levels of detail and high quality deformations. We present *Cages (star-cages), a significant step forward with respect to traditional single-cage coordinate systems, and which allows the usage of multiple cages enclosing the model for easier manipulation while still preserving the smoothness of the mesh in the transitions between them. The proposed deformation scheme is extremely flexible and versatile, allowing the usage of heterogeneous sets of coordinates and different levels of deformation, ranging from a whole-model deformation to a very localized one. That locality allows faster evaluation and a reduced memory footprint, and as a result outperforms single-cage approaches in flexibility, speed and memory requirements for complex editing operations.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Modeling packages*

Additional Key Words and Phrases: Cage-based deformation, mesh deformation, coordinate systems

1. INTRODUCTION

Shape deformation in both in two and three dimensions plays a central role in computer graphics. Space deformation techniques especially cage-based methods as a practical means to manipulate 3D models [Floater et al. 2005] [Ju et al. 2005] [Joshi et al. 2007] [Lipman et al. 2008] [Weber et al. 2009], have received a lot of attention. A cage is a low polygon-count polyhedron, which typically has a similar shape to the enclosed object. The object points inside the cage are represented by affine sums of the cage elements (vertices or faces) multiplied by special precalculated weight functions called coordinates. The main advantages of these space deformation techniques are their simplicity, relative flexibility and speed on applying the deformation. Also, as each point is transformed independently, these techniques are indifferent to the surface representation and in general free of discretization errors.

However,to date there has been no widely accepted solution that provides both user control and high-quality deformations. It is commonly accepted that an ideal deformation system should allow user intervention when required, but automatically infer all the missing data. For instance, given a user-chosen set of constraints, the system should find the best deformed shape that satisfies those constraints. Several possible alternatives do exist, such as Mean Value Coordinates (MVC) [Floater et al. 2005], Harmonic Coordinates (HC) [Joshi et al. 2007] or Green Coordinates (GC) [Lipman et al. 2008]. All of them are characterized by the use of a single cage to compute the final deformation. This particularity presents some problems:

—**Locality**. Most current cage-based deformation approaches can be classified as global deformation methods because they are defined in terms of a single cage which affects *all* mesh vertices and which means they cannot produce local deformations.

—**Time and memory consumption**. In general, the global behavior of single cage-based techniques results in each point storing weights for all cage vertices, increasing both the consumption of memory and the number of evaluations.

—**Smoothness**. As we will later explain, all single cage-based methods have continuity problems on the cage boundaries, ranging from the lack of smoothness to the presence of discontinuities. This is one of the main reasons as to their use is currently limited to monolithic single cages.

—**Coordinate combination**. Each of the single cage-based methods uses different types of coordinates and as a result the deformations also differ (e.g. MVC and GC). The user has to decide to use one coordinate type or another for the whole model, depending on the desired results, and without the option of combining their strengths.

In order to avoid such problems, one way would be to use many cages instead of just one. As a consequence, they would be easier to create and manipulate. Each of these cages should use different coordinate types, which would increase the potential of different results over the final deformations. All these cages should be used at different levels of granularity to allow a more localized control over the final deformation, consuming only the resources needed for each cage in isolation.

In this paper we present *Cages (pronounced *star-cages*), a cage-based deformation method that involves a hierarchical set of cages where the leaf cages bound the object in a piecewise manner. Cage-coordinates can be individually defined for each leaf

Authors' addresses: gonzalezgarciafran@gmail.com, {teresap, coll, dagush}@ima.udg.edu

DOI 10.1145/XXXXXXX.YYYYYYY
http://doi.acm.org/10.1145/XXXXXXX.YYYYYYY

cage, and blended among neighboring cages to produce a smooth (class C^1) deformation, thus offering localized deformation control with swift computation. The hierarchy further allows deformation control to take place at multiple levels. In this sense, we can say that *Cages *complements* the existing techniques rather than *competing* with them. Hence, the rationale behind of its name: *Cages can accommodate *any* coordinate system inside a cage, and smoothly combine *any* number of cages to obtain a flexible and general deformation system at *any* level of detail.

The main contributions of the proposed technique are:

—An unlimited number of cages to smoothly deform a base mesh.

—This set of cages produces more localized deformations and as a consequence consumes less time and memory.

—This is the first system that allows the usage of heterogeneous sets of coordinates, and is able to define different coordinates for different cages and use them together in combination.

—The ability to produce multi-level deformations, where different cages are used to control different levels of detail in the deformation of a model.

All this together gives rise to an extremely versatile deformation approach, which is much more intuitive and user-friendly.

2. PREVIOUS WORK

Cage-based deformation methods, which drive the deformation via a control cage enclosing the model to be deformed, are considered to be one of the most important space deformation techniques. The first method based on three dimensional regular lattices was introduced by Sederberg and Parry [1986]. Later, this method was extended to handle general lattices [Coquillart 1990] and LOD management [Seo and Thalmann 2000]. In recent years, new deformation methods have been proposed based on the use of coordinates computed with respect to the vertices of a single enclosing cage. Floater and co-workers [Floater 2003] [Floater et al. 2005] [Ju et al. 2005] introduced Mean Value Coordinates (MVC) as a method for constructing an interpolant for closed triangular meshes, with a closed-form formulation which is able to reproduce linear functions. MVC are well defined both inside and outside the control mesh (C^∞ continuous) but they are only C^0 continuous across the cage faces. Later, Joshi et al. [2007] proposed Harmonic Coordinates (HC) for character articulation, which are positive and C^∞ continuous inside the cage, C^0 continuous on the boundary and have no definition outside the cage. Lipman et al. [2007] presented an alternative non-negative coordinate definition to MVC (PMVC). The coordinates are computed numerically by using a GPU-friendly approach. Later, Lipman et al. [2008] proposed a new shape-preserving space deformation approach called Green Coordinates (GC). The work, motivated by Green's third integral identity, produces conformal mappings, and extends naturally to quasi-conformal mappings in 3D by using both the vertex positions and face orientations of the cage. GC are C^∞ continuous inside and outside the cage but discontinuous at the boundary, although some restrictive extension mechanism can be applied. However, *Cages is a technique that provides smoothness to any of these coordinates across multiple cages, allowing their usage in combination.

Jacobson et. al [2011] proposed bounded biharmonic weights, a linear blending scheme that is able to produce smooth, intuitive and flexible deformations for 2D and 3D shapes using handles of different topology (points, bones and cages). Contrary to single cage-based approaches, they can naturally use partial cages to locally deform a mesh without any special restriction or consideration. However, this technique does not guarantee linear precision and needs transformations associated to each handle, which in this sense makes them more similar to skeletal deformation than to barycentric coordinates. In any case, this technique is fully compatible with *Cages allowing its fusion with classic cage-based techniques.

A similar hierarchical approach to the one we propose with our method was introduced in [Zheng et al. 2011]. There, the authors allowed the user to group a set of controllers to obtain a hierarchy to deform a mesh. Their approach is only applicable to man-made models and uses a small set of representative controllers. As the authors mention, they discarded cages as handlers given the difficulties at cage boundaries, and which is exactly what *Cages addresses.

Langer et al. [2008] developed a criteria for the construction of smooth maps, called Bézier maps, that are a piecewise homogeneous polynomial in generalized barycentric coordinates. To avoid discontinuities, they had to increase the number of control points and the order of the polynomials, thus increasing computational costs. In the work by Ben-Chen et al. [2009], the challenge was to find a harmonic map from a domain in such a way that it satisfies constraints specified by the user, and is detail-preserving and intuitive to control. Huang et al. [2009] presented a mesh deformation technique using modified barycentric coordinates with a tetrahedron control mesh that avoids first order discontinuities across the cage boundaries. Unlike them, we are more flexible in the nature of the cages we can use and require fewer resources. Finally, even though it does not use cages, Botsch et al. [2005] proposed a real-time freeform shape editing technique that allows user-defined modeling constraints to be posed directly on the surface.

Another GC-based technique to locally deform a mesh contained by an automatically generated umbrella-shaped cell was presented by Li et al. [2010]. Although their cage is local, they need to bind coordinates for all mesh vertices, thus increasing memory consumption. Ju et al. [2008] introduced skinning templates as a solution to share and reuse skinning behaviors for similar joints and similar characters. The skinning templates were implemented using cage-based deformations, and thus they can benefit from all the features of our approach. A hybrid approach that combines surface-based and cage-based deformations were presented by Borosan et al. [2010], but, as they note, it is not smooth at the boundary of the cage and meshes that are too coarse limit its effectiveness making their approach suitable only for local deformations. *Cages, instead, is able to provide smoothness to an arbitrary combination of different coordinates at different deformation levels.

Recently, Landreneau and Schaefer [Landreneau and Schaefer 2010] introduced a Poisson-based method to reduce the storage needs of the coordinates for animated meshes, aiming at making a coordinate system local while still using a global cage. This method has the advantage of allowing a reduced number of coordinate evaluation for each mesh vertex at the price of requiring the user to provide a set of initial mesh poses and only providing smoothness for deformations similar to the initial set. *Cages can naturally work in combination with this method and also benefits from the reduction it can achieve.

Cage-based deformations have been also applied to planar domains. One related work was introduced by Meng et al. [2009] who designed a method to keep the shape of images during the deformation of a region of interest, but continuity depends on the cage coordinates used (MVC, HC or GC). Later, Weber et al. [2009], generalized the concept of barycentric coordinates from real numbers to complex numbers, but this is only applicable to two dimensional shape deformations. Should be noted that our method can be applied to planar domains as well as 3D domains.

3. *CAGES

As has been previously explained, current cage-based deformation approaches use a single cage to deform a mesh. Some of these methods, such as MVC, PMVC and HC, express a point p inside a cage c as an affine combination of the cage vertices v by:

$$p = \sum_{v \in c} w_c(v,p)v \tag{1}$$

where $w_c(v,p)$ are the coordinate basis functions. Let us note that, while the use of the cage subindex c may seem redundant, its usefulness will become apparent in the following explanations. The natural way to define a deformation inside cage c of a point p is given by:

$$T_c(p) = \sum_{v \in c} w_c(v,p)v' \tag{2}$$

where v' are the deformed control cage vertices.

Instead, *Cages encloses a mesh in a connected set of controlling cages $\{c_0, c_1, \ldots, c_n\}$. These cages do not intersect and they use their own independent set of coordinates (See the left image in Figure 1 for an example). Let C be the union of them. For each of these cages, we can define a transformation $T_{c_i}(p)$. In general, the piecewise transformation defined on C by transformations $T_{c_i}(p)$ is at most C^0, e.g., it can generate first-order discontinuities across boundaries between adjacent cages. See Figures 2(b) and 2(d) where classic MVC/PMVC/HC and GC were used, respectively. In the case of GC, $T_{c_i}(p)$ would use both vertices and face normals of the cages in its definition, but the corresponding piecewise transformation would be discontinuous at the cage boundaries. The insets in the figures show a detail of these discontinuities.

Therefore, if we want to use a set of cages as the base of our deformation tool, instead of a single cage, it seems clear that we need to address this continuity problem. For that purpose we are going to define a *smooth transformation* S that will replace the classic $T_c(p)$ to deform the points. S will be piecewise defined on each cage c_i by transformations $S_{c_i}(p)$, so it is of class C^1 in the interior of C. Our proposal consists of defining the transformation $S_{c_i}(p)$ by blending the traditional transformation $T_{c_i}(p)$ defined in each cage c_i with a new transformation $J_{c_i}(p)$, called *join transformation*. $J_{c_i}(p)$ will be responsible for guaranteeing smooth transitions between neighboring cages and will behave similar to standard cage-based transformations.

More formally, we define the transformation $S_{c_i}(p)$ by:

$$S_{c_i}(p) = b_{c_i}(p)T_{c_i}(p) + (1 - b_{c_i}(p))J_{c_i}(p) \tag{3}$$

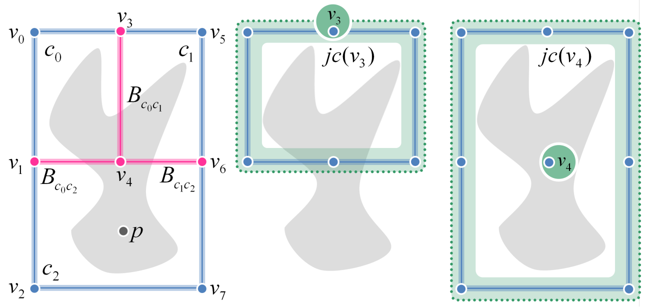

Fig. 1. Left: $c_0 = v_0v_1v_4v_3$, $c_1 = v_3v_4v_6v_5$, $c_2 = v_1v_2v_7v_6v_4$, $B_{c_0} = B_{c_0c_1} \cup B_{c_0c_2}$, $B_{c_1} = B_{c_0c_1} \cup B_{c_1c_2}$, $B_{c_2} = B_{c_0c_2} \cup B_{c_1c_2}$. Middle: Join cage generated by v_3. Right: Join cage generated by v_4.

where $b_{c_i}(p)$ is the *boundary weight function* (See Section 3.2), which is a class C^1 function in c_i that is equal to zero at any point p lying on the border between c_i and any adjacent cage and different to zero at the interior of c_i. In this manner, $J_{c_i}(p)$ or $T_{c_i}(p)$ will be fully applied on a mesh point p depending on its position in respect to any border of c_i. The fact that transformation $T_{c_i}(p)$ is the one used by previous single-cage approaches will allow the user to choose a different coordinate system for each individual cage in a way that suits his/her needs. Thus, in the following subsections we will explain the two elements needed to define the smooth transformation $S_{c_i}(p)$: the *join transformation* $J_{c_i}(p)$ and the *boundary weight function* $b_{c_i}(p)$.

3.1 Join transformation $J_{c_i}(p)$

First, let us introduce some definitions illustrated by the scheme shown in Figure 1. As mentioned before, *Cages uses a set of controlling cages. Given two cages c_i and c_j, we consider them adjacent if they share a set of faces. Then, given cage c_i let us denote its adjacent cages by $Adj(c_i)$ (e.g. $Adj(c_0) = \{c_1, c_2\}$ in Figure 1). For a cage $c_j \in Adj(c_i)$ let $B_{c_ic_j} = c_i \cap c_j$ be the border between c_i and c_j (e.g. $B_{c_0c_1} = c_0 \cap c_1$) and let the *boundary* of c_i, noted B_{c_i}, be the union of all borders $B_{c_ic_j}$ (e.g. $B_{c_0} = B_{c_0c_1} \cup B_{c_0c_2}$).

Our goal is to define a transformation $J_{c_i}(p)$ that can smoothly cross boundaries between adjacent cages while verifying two important constraints: First, as $J_{c_i}(p)$ will be responsible for glue cage-based deformations it must behave similar to standard cage-based methods to produce fair deformations. Second, to provide a high degree of locality, $J_{c_i}(p)$ has to take into account only the local information concerning the boundaries between cages. As we will explain during this section, transformation $J_{c_i}(p)$ verifies both restrictions: First, $J_{c_i}(p)$ is built by using standard coordinates (MVC/GC), which allows not only a way to control the behavior of the blending region, but also a way to ensure the fairness of the deformations. Second, $J_{c_i}(p)$ keeps transformations as local as possible to the boundaries between cages, by relying only on their local information, which is their vertices.

So, given a boundary $B_{c_ic_j}$ between cages c_i and c_j, we define as *boundary vertices* the set of vertices that belong to that boundary $B_{c_ic_j}$. As can bee seen in Figure 1, any number of cages can meet at a *boundary vertex*. Thus, let us define the *join cage* of a *boundary vertex* v, denoted by $jc(v)$, as the union of all the cages incident at v. Figure 1 shows a set of control cages

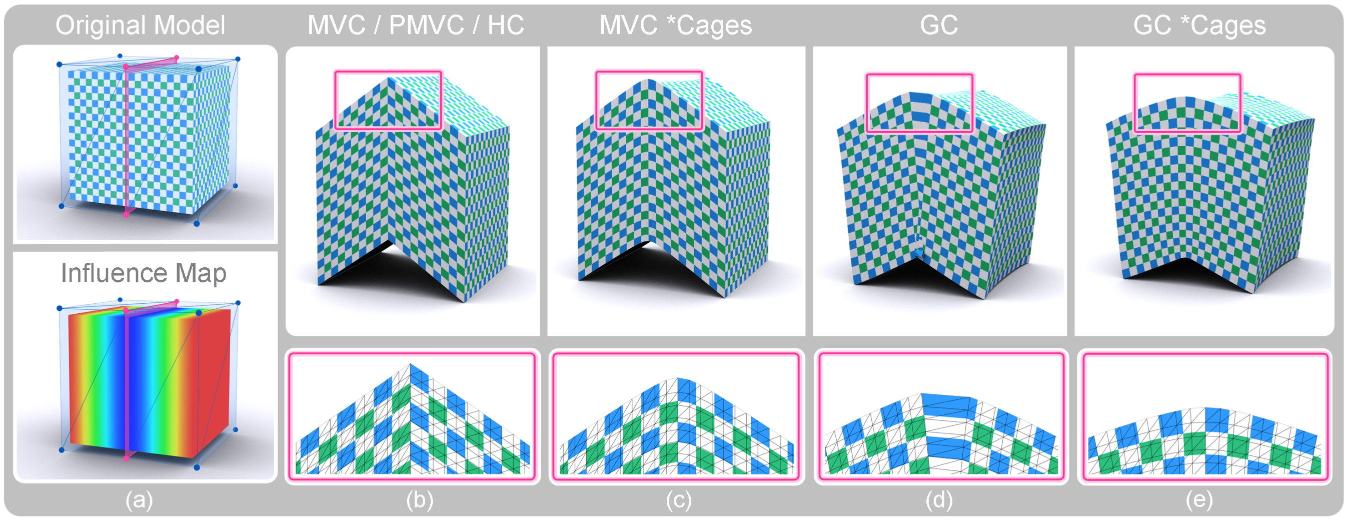

Fig. 2. A comparison between piecewise deformations. (a) The original model enclosed in two cages and its influence map. (b) MVC/PMVC/HC deformation. (c) *Cages with MVC deformation, both for $J_{c_i}(p)$ and $T_{c_i}(p)$. (d) GC deformation. (e) *Cages with GC deformation, both for $J_{c_i}(p)$ and $T_{c_i}(p)$. The second row shows close-up views of the deformed model. Notice that only (c) and (e) are C^1.

(left), the join cage of the vertex v_3 (middle), and the join cage of the vertex v_4 (right). As can bee seen, a *boundary vertex* may or may not belong to its associated join cage. For the former we will call them *non-interior vertices*, while for the later we will call them *interior vertices*. Relying on the concept of *join cage*, we will define, vertex-wise for each *boundary vertex* v, a smooth and local transformation $L_v(p)$. Then, for a cage c_i, the final *join transformation* will be defined as a blending of all the smooth transformations $L_v(p)$ related to the vertices of its boundary B_{c_i}.

Thus, in our scheme, the transformation $J_{c_i}(p)$ in c_i is defined by:

$$J_{c_i}(p) = \sum_{v \in B_{c_i}} W(v,p) L_v(p) \tag{4}$$

where weights $W(v,p)$ are normalized to 1. As an example, in Figure 1, note that $J_{c_2}(p) = W(v_1,p)L_{v_1}(p) + W(v_4,p)L_{v_4}(p) + W(v_6,p)L_{v_6}(p)$ and $J_{c_0}(p) = W(v_1,p)L_{v_1}(p) + W(v_3,p)L_{v_3}(p) + W(v_4,p)L_{v_4}(p)$.

The weight function $W(v,p)$ will be responsible for telling us how much of the transformation $L_v(p)$ from each boundary vertex v is blended. If we are at v itself, then transformation $L_v(p)$ will be fully applied and its contribution will smoothly decrease as we move away. $W(v,p)$ will completely vanish at the other boundary vertices of B_{c_i}. Let us define the weight function $W(v,p)$ by:

$$W(v,p) = \Omega(v,p) I(v,p) \tag{5}$$

where $\Omega(v,p)$ and $I(v,p)$ are two smooth functions. The first, called *vertex influence function*, will provide us with a way to express the influence of vertex v in its join cage, while the second, called *interior vertices influence function*, will take into account the influence of *interior vertices*, if any, on the rest of boundary vertices (*interior* or *non-interior*). Both are explained below.

Vertex influence function $\Omega(v,p)$. This function is a bell-shaped function defined over the join cage $jc(v)$ of vertex v, which allows us to specify a smooth region of incidence of vertex v on its join cage $jc(v)$. This function has to satisfy the following properties: be non-negative, be smooth, be equal to one at v and equal to zero on any face of $jc(v)$ not incident to v. As an example, in Figure 1, $\Omega(v_3,p)$ should be equal to one at v_3 and equal to zero on all the faces of $jc(v_3)$ except the two containing $v3$. One way to define $\Omega(v,p)$ could be based on Gaussian functions. However, the need to keep the transformations local has led us to define it by the use of a weight measure with respect to the faces of the join cage (a sort of distance). So, let t be a face of $F(jc(v),v)$, which is the set of faces of $jc(v)$ not incident to v. Thus, we define $\Omega(v,p)$ as a product of normalized and smoothed distances to that set of faces as follows:

$$\Omega(v,p) = \prod_{t \in F(jc(v),v)} f_1\left(\frac{d_{jc(v)}(t,p)}{d_{jc(v)}(t,v)}\right) \tag{6}$$

where f_h is a *smoothing* function parameterized with parameter $h \in (0,1]$, satisfying $f_h(0) = f_h'(0) = f_h'(1) = 0$, $f_h(x) = 1$ for $x \geq h$, and $f_h'(x) \geq 0$. h is used to contract or expand f in the range $(0,1]$. Here, for Ω, we set $h = 1$, but its full meaning will become apparent in Section 3.2. In our implementation we have tested several functions that fulfill those properties: $f_h(x) = \frac{1}{2}\sin(\pi(\frac{x}{h} - \frac{1}{2})) + \frac{1}{2}$, $f_h(x) = -2(\frac{x}{h})^3 + 3(\frac{x}{h})^2$ and $f_h(x) = -8(\frac{x}{h})^5 + 20(\frac{x}{h})^4 - 18(\frac{x}{h})^3 + 7(\frac{x}{h})^2$, all defined for $x \in [0,h]$, and $f_h(x) = 1$ for $x \in [h,1]$. The results obtained were similar in all cases without observing noticeable differences. The images in the paper have been generated with the first function.

The distance function $d_c(t,p)$ specified in cage c is defined by:

$$d_c(t,p) = 1 - \sum_{u \in V(t)} w_c(u,p) \tag{7}$$

where $V(t)$ are the vertices of face t and $w_c(u,p)$ are the coordinate basis functions (MVC/HC) used in cage c. In the case that c is not convex, weights $w_c(u,p)$ have to be computed by HC to prevent negative coordinate values. Otherwise

MVC can be used. Observe that function $d_c(t,p)$ is of class C^∞ inside cage c and C^0 in the boundary of c. It is also equal to one on the faces of c non adjacent to t, and it is equal to zero on t.

Properties for $\Omega(v,p)$:

Notice that function $\Omega(v,p)$ possesses the required conditions and also satisfies:

—$\partial_p\Omega(v,v) = 0$, where ∂_p is the directional derivative respect to p

—$\partial_p\Omega(v,p) = 0$, for any point p lying on any face of $jc(v)$ not incident to v

—$\Omega(v,p) < 1$, for any point $p \neq v$

Interior vertices influence function $I(v,p)$. Given a boundary vertex v (*interior* or *non-interior*), function $I(v,p)$ is responsible for introducing the influence of the rest of boundary vertices classified as *interior vertices* over v (e.g. $v4$ over v_3 in Figure 1). As we have explained, given the fact that function $\Omega(v,p)$ gives a way to determine the influence of a vertex v, we rely on it to define $I(v,p)$ as:

$$I(v,p) = \prod_{u \in Int(jc(v))-\{v\}} f_{h_{c_i}}\left(\frac{1-\Omega(u,p)}{1-\Omega(u,v)}\right) \tag{8}$$

being $Int(jc(v))$ the set of interior vertices of $jc(v)$. Note that if the set of vertices $Int(jc(v)) - \{v\}$ is void, this means that there are no interior vertices, so $I(v,p)$ is equal to one and thus $W(v,p)$ will be influenced only by the smooth function $\Omega(v,p)$. This means two things: first, the influence of vertex v is not affected by other interior vertices and second, the vertex v is only influenced by the rest of boundary vertices (*non-interior*) of $jc(v)$ which are taken into account in the first smoothing function $\Omega(v,p)$.

As an example, in Figure 3 we illustrate the weight function $W(v,p)$ on several different borders between two neighboring cages. In Figures 3(a) and 3(b), the border has only *non-interior vertices*, while on the rest of images a couple of *interior vertices* appear. In Figure 3(a) we show the weight $W(v_1,p)$ for vertex v_1 and in Figure 3(b) the weight $W(v_4,p)$ for the vertex v_4. Observe the smoothness of the weights and, in the later case, the lack of negative coordinates as we use HC. In Figure 3(c) we show the weight $W(v_6,p)$ of the *interior vertex* v_6 and in Figure 3(d) we show the weight of vertex v_1 but now with the influence of the *interior vertex* v_6. Observe the difference with the same weight of Figure 3(a). Finally, in Figures 3(e) and 3(f) we show the mutual influence of two *interior-vertices*.

Properties for $W(v,p)$:

The weight $W(v,p)$ is a non-negative function of class C^1 in $jc(v)$ satisfying the following properties:

—$W(v,v) = 1$ and $\partial_p W(v,v) = 0$

—$W(v,p) = 0$ and $\partial_p W(v,p) = 0$, for any point p lying on any face of $jc(v)$ not incident to v

—$W(v,u) = 0$ and $\partial_p W(v,u) = 0$, for any vertex $u \in Int(jc(v)) - \{v\}$

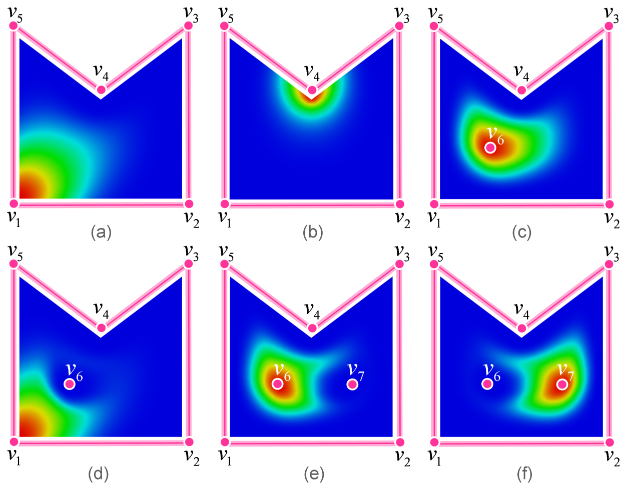

Fig. 3. Variation of weight $W(v,p)$ on different borders between two cages.

Now that we have set the weight $W(v,p)$, we still lack the definition of the transformation $L_v(p)$ to have the join transformation $J_{c_i}(p)$ completely specified. Thus, let us define transformation $L_v(p)$ differently depending on whether v is *boundary vertex* classified as an *interior* or a *non-interior* vertex of $jc(v)$:

—v **is a *non-interior* vertex of** $jc(v)$. Transformation $L_v(p)$ is defined by a smooth arbitrary transformation $T_{jc(v)}(p)$ (defined with MVC/HC/GC) in $jc(v)$ determined by the deformed vertices of $jc(v)$ by:

$$L_v(p) = T_{jc(v)}(p) \tag{9}$$

—v **is an *interior* vertex of** $jc(v)$. Transformation $L_v(p)$ is defined by a smooth arbitrary transformation $T_{jc(v)}(p)$ (MVC/HC/GC) in $jc(v)$, plus a smooth gradation of the transformation that moves $T_{jc(v)}(v)$ to v', which is the deformed cage vertex v. That is:

$$L_v(p) = T_{jc(v)}(p) + W(v,p)(v' - T_{jc(v)}(v)) \tag{10}$$

Note that as we get closer to the interior vertex v, the second term of the sum will increase, and thus the transformation $L_v(p)$ will be altered by adding the influence of vertex v. Otherwise, as we get far away from the interior vertex v, the transformation of point p in $jc(v)$, that is $T_{jc(v)}(p)$, will be fully applied with no influence of v.

Properties for $L_v(p)$:

Observe that transformation $L_v(p)$ verifies the following properties:

—$L_v(v) = v'$ being v' the deformed cage vertex.

—$L_v(u) = u'$ for $u \in V(jc(v))$, where u' is the deformed cage vertex u and $V(jc(v))$ are the vertices of cage $jc(v)$.

—If $T_{jc(v)}(p)$ and v' are given by a linear function then $v' = T_{jc(v)}(v)$ and $L_v(p)$ also is given by a linear function.

Now that we have explained in detail the *join transformation* $J_{c_i}(p)$ through its two components, the weight $W(v,p)$ and the

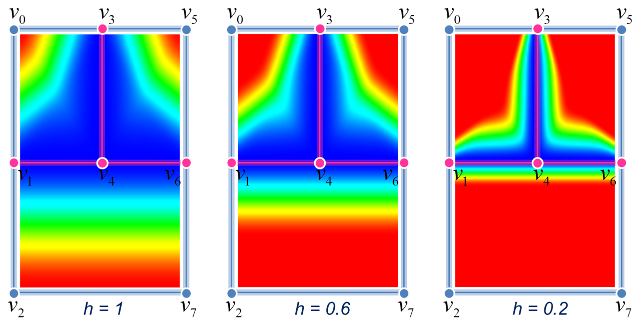

Fig. 5. Influence map variation on the scheme shown in Figure 1.

local transformations related to the boundary vertices $L_v(p)$, we can guarantee the following property with respect to its smoothness:

Properties for $J_{c_i}(p)$:

Observe that for a point p on a boundary $B_{c_ic_j}$ between cages c_i and c_j we have:

$$J_{c_i}(p) = \sum_{v \in B_{c_ic_j}} W(v,p)L_v(p)$$

which guarantees:

—$J_{c_i}(p) = J_{c_j}(p)$, $\partial_p J_{c_i}(p) = \partial_p J_{c_j}(p)$, and consequently smooth transitions between cages are obtained.

Note that $J_{c_i}(p)$ is not defined on the faces that do not have any boundary vertex (e.g face consisting of vertex v_2 and vertex v_7 of cage c_2 in Figure 1). As we will show in the following subsection, this is not problematic because transformation $J_{c_i}(p)$ will never be applied there, as the boundary weight function $b_{c_i}(p)$ will be equal to one and thus, $J_{c_i}(p)$ will not be applied.

3.2 Boundary weight function $b_{c_i}(p)$

In this subsection we specify the weight function $b_{c_i}(p)$ involved in the *smooth transformation* $S_{c_i}(p)$. This function is responsible for determining where a point p is located respect to the boundaries of a given cage c_i, so more formally we can say $b_{c_i}(p)$ needs to be equal to 0 when $p \in B_{c_i}$ (fully apply the *join transformation* $J_{c_i}(p)$) and needs to be equal to 1 on the faces of cage c_i that are not incident to any vertex of B_{c_i} (fully apply the own cage transformation $T_{c_i}(p)$, e.g face created by vertices v_2 and v_7 in Figure 1).

Thus, let us define in a cage c_i the weight function $b_{c_i}(p)$ as a product of weights with respect to any border $B_{c_ic_j}$ as follows:

$$b_{c_i}(p) = f_{h_{c_i}}\left(\prod_{c_j \in Adj(c_i)} (1 - \sum_{v \in B_{c_ic_j}} w_{c_i}(v,p))\right) \tag{11}$$

where $f_{h_{c_i}}$ is the smoothing function defined in Section 3.1. If $B_{c_i} = \emptyset$, i.e., when c_i does not have any neighboring cage, $b_{c_i}(p)$ is set to be 1. As an example, in Figure 1 the distance $b_{c_2}(p)$ of cage c_2 will be 0 when p belongs to the boundary of the cage ($B_{c_2} = B_{c_0c_2} \cap B_{c_1c_2}$) and to 1 on the face generated by vertices v_2 and v_7.

Properties for $b_{c_i}(p)$:

The weight function $b_{c_i}(p)$ verifies these properties:

—$\partial_p b_{c_i}(p) = 0$, for any point p satisfying $b_{c_i}(p) = 1$,
—$\partial_p b_{c_i}(p) = 0$, for any point p satisfying $b_{c_i}(p) = 0$, that is, lying on any face of B_{c_i}.

As can be seen in Formula (3), the weight $b_{c_i}(p)$ is a measure of the influence of the transformation $T_{c_i}(p)$ in $S_{c_i}(p)$, and can be adjusted by changing the parameter h_{c_i}. To visualize the effect when altering h_{c_i} we use an influence map, where the model is painted in blue-red gradation according to the distance $b_{c_i}(p)$. As an example, in Figure 5 we visualize three different influence maps for the scheme shown in Figure 1. Also, in Figure 4 we show the chinchilla model enclosed in 9 cages. We have used GC for the ears and MVC for the rest of the cages, as well as the join transformations. At the right, results obtained from three different values of h_{c_i} corresponding to the left ear cage, are shown.

3.3 Smooth Transformation $S_{c_i}(p)$

Up to now, we have specified all the components that are part of the piecewise smooth transformation $S_{c_i}(p)$ defined in Formula (3). In Figures 2(c) and 2(e) we illustrate the effects of the proposed smooth transformation. In Figure 2(c) a deformation has been performed using MVC to compute both, the cage transformations $T_{c_i}(p)$ and the *join transformation* $J_{c_i}(p)$, while in Figure 2(e) GC are used for both transformations.

Properties for $S_{c_i}(p)$:

The transformation $S_{c_i}(p)$ defined in a cage c_i satisfies the required continuity conditions for any point p on $B_{c_ic_j}$:

—$S_{c_i}(p) = J_{c_i}(p) = J_{c_j}(p) = S_{c_j}(p)$
—$\partial_p S_{c_i}(p) = \partial_p J_{c_i}(p) = \partial_p J_{c_j}(p) = \partial_p S_{c_j}(p)$

Now we want to show that the transformation S of the whole system of cages C preserves all the good properties of the standard cage-based techniques while satisfying the required continuity conditions. One important aspect is that transformation S inherits properties of transformations $T_{c_i}(p)$ and $L_v(p)$ (used to create $J_{c_i}(p)$) such as linear reproduction. The only requirement for the coordinate system used to compute these transformations is that it must be defined inside the respective cage as MVC/HC and GC. So, if A is an arbitrary linear function, we need to show that $S_{c_i}(p) = A(p)$ in case that $T_{c_i}(p) = A(p)$ and $L_v(p) = A(p)$ for all join cages. This can be easily demonstrated from the partition of unity property of the weight functions:

$$S_{c_i}(p) = b_{c_i}(p)T_{c_i}(p) + (1 - b_{c_i}(p)) \sum_{v \in B_{c_i}} W(v,p)L_v(p) =$$
$$= b_{c_i}(p)A(p) + (1 - b_{c_i}(p)) \sum_{v \in B_{c_i}} W(v,p)A(p) =$$
$$= b_{c_i}(p)A(p) + (1 - b_{c_i}(p))A(p) = A(p)\,.$$

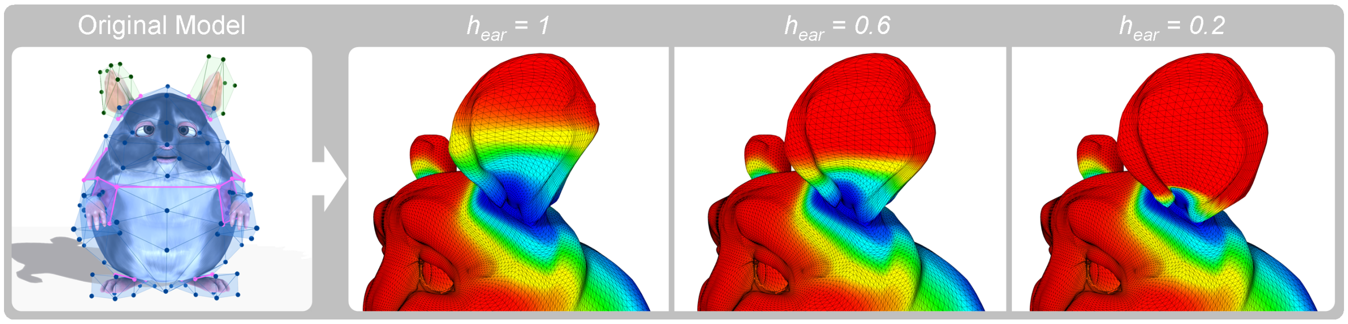

Fig. 4. Influence map variation on the chinchilla model. Left: Original model and initial cages. Right: Results obtained by using different h_{c_i} values for the left ear cage. Red and blue regions mean transformations $T_{c_i}(p)$ and $J_{c_i}(p)$ respectively are fully applied.

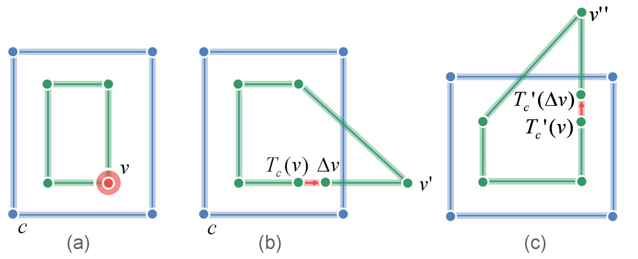

Fig. 6. Multi-level deformation for coordinates not defined outside the cage. (a) Initial cages. (b) Direct cage vertex movement. (c) Parent transformation.

Moreover, if transformations $T_{c_i}(p)$ perform boundary interpolation, transformation S performs boundary interpolation on faces not adjacent to any boundary between cages. For a point on this kind of faces all weights with respect to vertices of B_{c_i} are equal to 0. Consequently, the distance $b_{c_i}(p)$ with respect to B_{c_i} is equal to 1 and, then, $S_{c_i}(p) = T_{c_i}(p)$, which means that the *smooth transformation* in c_i is equal to the transformation (MVC/GC/HC) defined for cage c_i.

3.4 Multi-level deformations

We can use *Cages to build a multi-level system which gives flexibility, versatility, interactivity and control over the deformations to be applied to a part of the model. In our scheme, upper-level cages can own an arbitrary set of vertices of lower-level cages, the only restriction being that cages must have a hierarchical relationship (e.g. a *Directed Acyclic Graph* or a tree) and that a given cage vertex cannot be controlled by more than one parent cage. In general, two kinds of cages can be distinguished: *leaf-cages* that directly control the mesh and satisfy the conditions enumerated in Section 3, and the *internal-cages* that control cage vertices of lower-level cages and do not directly affect the mesh, so they can intersect and smoothness does not need to be enforced for them.

Our multi-level system relies on a simple yet effective observation: When a cage in the multi-level system changes, the effects of this change should only be propagated downwards but not upwards in the hierarchy. This means that, when a vertex v of a cage is changed, the positions of all the vertices in the containing and neighboring cages should be updated as usual, but the parent cage c containing v would not be affected. However, if the parent cage c changes later on, v should be updated accordingly. Here, we cannot directly transform v, as it has a different position than the one used when computing its coordinates (binding time) with respect to c, so the coordinates for v must be recalculated, and then the process continues as usual.

If coordinates that are defined everywhere are used, such as MVC, then the above implementation works as described. However, in the case of coordinates *not* being defined outside (e.g. HC), special measures should be taken. We propose an easy but effective solution without the need of any cage recomputation. We can express any new position for v as $v' = T_c(v) + U(v)$ with $U(v)$ being the user-generated displacement, and $T_c(v)$ the transformation of v with respect to the parent cage c. We can express $U(v) = \lambda \cdot \Delta U(v)$, where λ is an adequate multiplicative factor and $\Delta U(v)$ is a displacement small enough to satisfy that point $\Delta v = T_c(v) + \Delta U(v)$ is within cage c at its current position. Now, if cage c undergoes another transformation T_c' that converts $T_c(v)$ into $T_c'(v)$ and Δv into $T_c'(\Delta v)$, we will update the current position of v by $v" = T_c'(v) + \lambda(T_c'(\Delta v) - T_c'(v))$ (see Figure 6).

4. RESULTS AND DISCUSSION

Throughout the paper, we have used the following coloring code: Blue cages use MVC, red ones use HC and the green ones use GC. We have also drawn the boundaries between cages in pink. The implementation of *Cages has been carried out using the Ogre3D engine on a Quad Core Duo (2.83GHz) with 4GB of RAM. In the interests of fairness, we have implemented MVC/HC and GC in our unoptimized CPU-based system, carefully following the pseudocodes provided in those papers.

To show that the deformations produced by *Cages result in deformations with a level of quality on par with common single cage-based approaches, in Figure 7 we present some comparisons. The prism model (206312 triangles) is enclosed using four cages (20 vertices) by *Cages and the union of cages by the single cage-based methods. The deformation is obtained by twisting the prism (each cage is rotated by $\pi/2$ with respect to the previous one). Note the large similarity between the results obtained by MVC and GC (Figures 7(a) and 7(d)) and *Cages (Figures 7(b) and 7(e)), even at the boundaries between cages where the new *Join Transformation* is fully applied. In the figure, both cage ($T_{c_i}(p)$) and *join transformations* ($J_{c_i}(p)$) have been computed with the same coordinate systems (MVC/GC). The corresponding difference maps are shown in Figures 7(c) and Figure 7(f). The maximum and RMS difference values shown are computed with

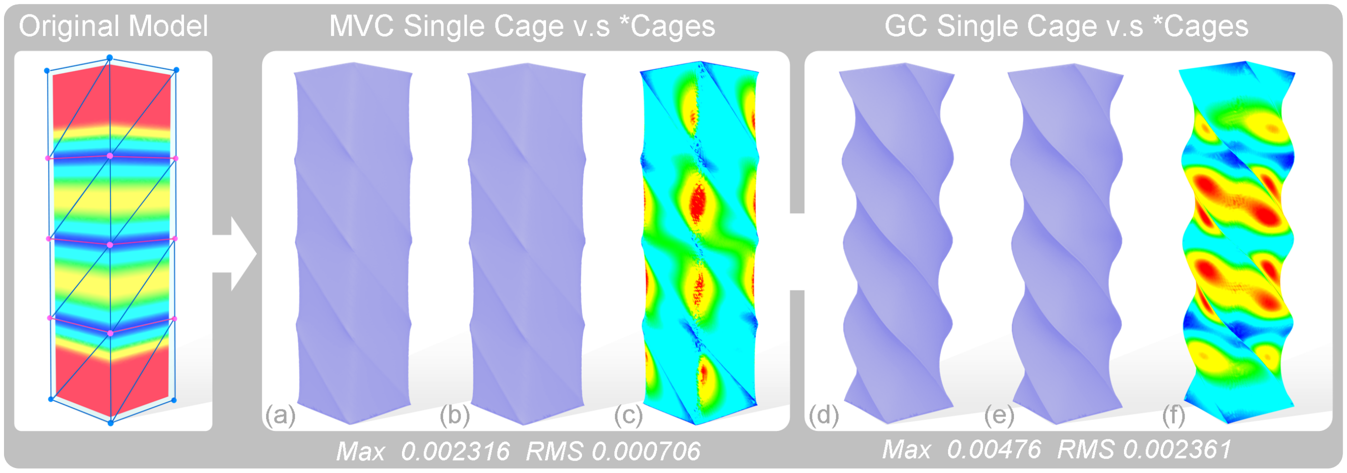
Original Model
MVC Single Cage v.s *Cages
GC Single Cage v.s *Cages
(a)
(b)
(c)
(d)
(e)
(f)
Max 0.002316 RMS 0.000706
Max 0.00476 RMS 0.002361

Fig. 7. Twisting a prism using MVC (left) and GC (right). See the similarity between a single cage (a,d) and *Cages (b,e). In (c,f) we can see the corresponding similarity maps. Red means lower similarity and blue means higher similarity.

respect to the bounding box of the model.

One important feature that *Cages has, in contrast to previous single cage-based methods, is the degree of locality obtained in the deformations. As a way of illustrating that important contribution, in Figure 8 we compare the locality of the deformations achieved by *Cages and single cage-based methods. For the former we have enclosed the train model (60052 triangles) in 4 cages (28 vertices), while for the latter we have used their union as a global cage for MVC and GC. As can be seen in the figure, single cage approaches cannot perform very localized deformations as all mesh vertices are deformed with respect to all cage vertices. For instance, if the cage vertices of the top of the central wagon are moved, the single cage version of MVC or GC deforms the head and tail wagons too, while *Cages only affects the center wagon, as one would expect. Also, notice the unsatisfactory deformations produced by MVC in Figure 8(a) at the sides of the wagon roofs due to the presence of negative coordinates produced by non-convex cages (the joining part between wagons). Let us note that, even though some coordinates like HC and GC present some kind of local control over the deformation, they have some issues we should discuss:

—HC uses *Interior Cages* to reduce the coordinate influence at the expense of building an extra interior cage, which leads to more memory and time consumption during the deformation. Moreover, one needs to be careful when creating such cages, because if the mesh goes through them, discontinuities will appear.

—In the case of GC, we could use the so called *Partial Cages* as a tool to provide local control. As discontinuities would arise at cage boundaries, the authors propose smoothing the deformation by extending the local deformation performed inside a partial cage to the rest of the mesh through some selected faces. This operation is not always possible and results in more computation time.

It is well known that not only each of the existing coordinates produces different deformation results, but also that they have different properties and computational resource needs (e.g. GC produces more time consuming deformations as they take into account the faces of the cages, and HC needs much more time to compute coordinates for each vertex than MVC). Thus, the combination of different coordinate types in a unique framework is a useful feature that allows users to freely choose between coordinates and, as a result, they are able to produce different deformations with the same cage configuration. Furthermore, coordinate selection allows us to benefit from their good properties and concrete computational resources needs depending on the situation:

—As can be seen in Figure 9, *Cages is able to smoothly combine different coordinate types between different cages. Here, the butterfly model (61366 triangles) has been enclosed in 6 cages (114 vertices): two for each wing, one for the lower body, and another one for the head. At the top right, we can observe the combination of GC for the top wings and MVC for the bottom ones. Full MVC and GC deformation using *Cages can be seen at the bottom row. Observe the difference between the deformations even when they use the same cage configuration, and how *Cages smoothly glues them.

—*Cages also allows us to benefit from the good properties of each coordinate type depending on the situation. In Figure 10 we show an example in which we can see the deformations produced by a single-cage with MVC on the hand model. As it is well-known, MVC has problems with concave cages because they result in negative coordinates (second column in the figure). By default, *Cages reduces this problem (third column) to regions near the boundaries when MVC is used to compute the *join transformations*. Here, the negativity still does not completely disappear because even when each single cage is convex, the resulting join cages are still concave. That is, cage transformations $T_{c_i}(p)$ do not have negative coordinates but *join transformations* $J_{c_i}(p)$ continue to be affected by the MVC negative behavior. However, the fact that *Cages is able to combine different coordinates, allows us to completely solve the problem (4th column) by using HC to compute only the *join transformations* when concave join cages are detected. Although this situation is not solved in a direct way by our approach, the combination of coordinates localize the usage of HC to only the places where it is needed, and so consuming less preprocessing time and memory to produce a deformation free of negative coordinates.

In situations where a combination of coordinates is used, an important aspect is how fair the resulting deformation is, especially in regions near the boundaries, which show a transition between different coordinate types. Given the nature of the propose blend-

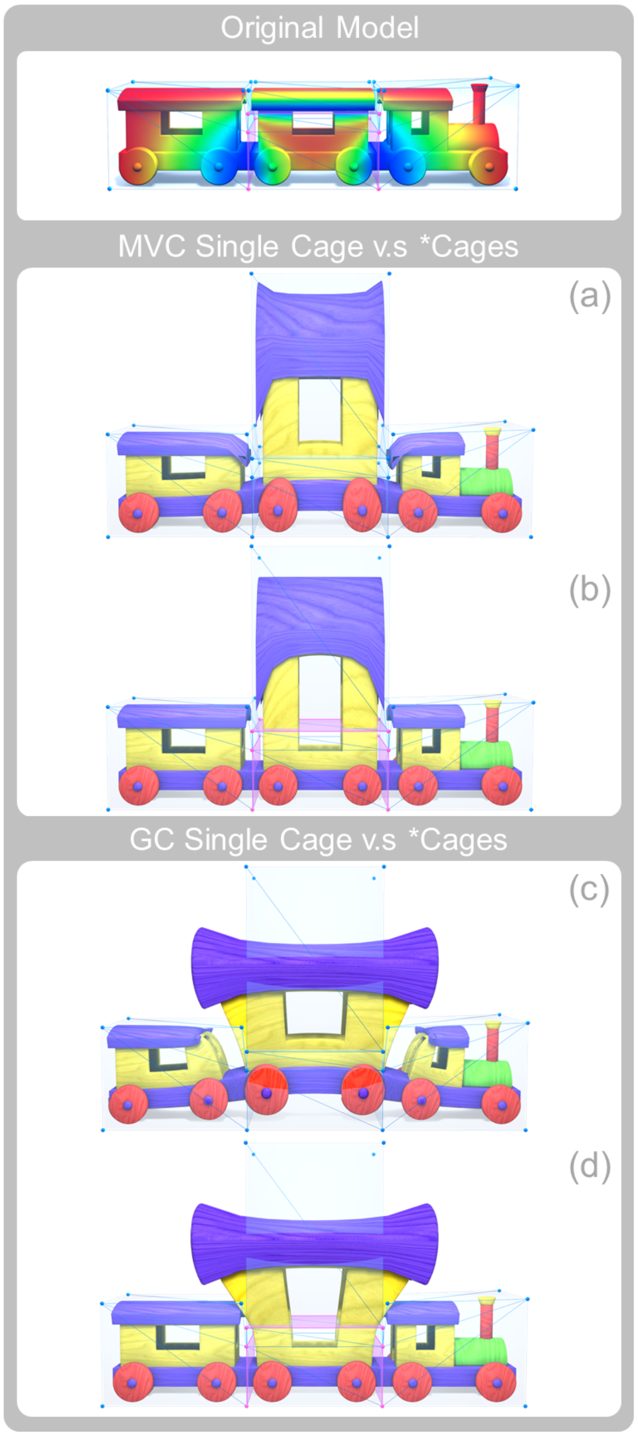

Fig. 8. Locality: Comparison of single cage approaches and *Cages. At top, the cages and the influence map on the train model. (a) single cage MVC, (b) *Cages whith MVC, (c) single cage GC, (d) *Cages with GC.

ing scheme, we can guarantee that the resulting deformations will produce correct results, as we smoothly shift from one deformation type to another. As an example, in Figure 11 we show results of the fairness of the deformations produced by our approach when

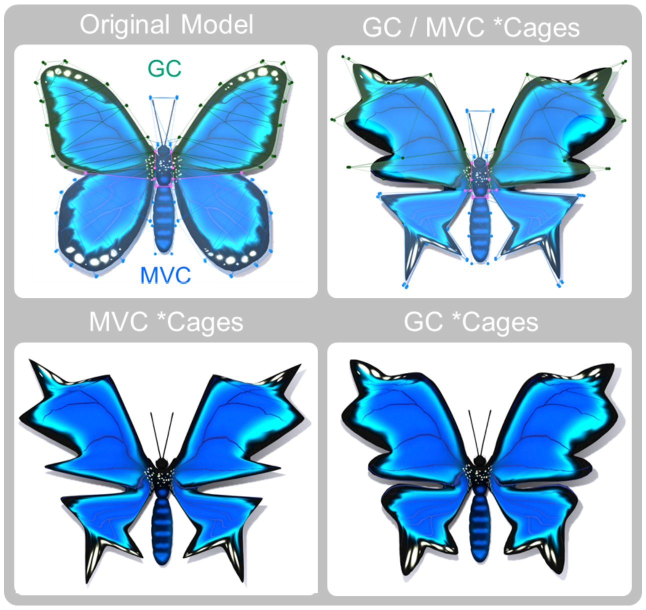

Fig. 9. Combined deformations on the butterfly model. The top row shows the cages and a *Cages-combined GC/MVC deformation. The bottom row shows two different deformations using MVC and GC with *Cages.

different coordinates are combined. In the top row, we can see the bumpy surface and its influence map at binding time. Three cages have been used: left and middle cages use MVC, while the right cage has GC. The join transformations applied between cages are of type MVC. In the bottom row we show a deformation using single cage approaches with MVC (left) and GC(middle), as well as the result of applying *Cages. See how the bumps in the model follow the faces of the cage when GC are used (see Figure 11, middle). On the other hand, if we perform the same deformation using MVC, the bumpy details get stretched (see Figure 11, left). This behavior can also be observed in the deformation produced by *Cages as well as the fairness in the deformation over the transition between cages of different coordinate types (see the insets in the bottom row in Figure 11). Observe how the GC deformation shifts to the MVC one.

*Cages is able to handle any number of cages meeting at a boundary cage vertex. Figure 12 shows a deformation obtained from a flower model (21903 triangles) enclosed in 13 cages (88 vertices) using different coordinate types. Here it is important to observe the correct behavior of the method even when cage vertices with more than two incident cages exist.

Two different multi-level deformations (see Section 3.4) can be seen in Figures 13 and 14. On one hand, in Figure 13, the squirrel head model (19552 triangles) has been enclosed by four leaf-cages (66 vertices): teeth, face, left and right ear. There are also two internal-cages (16 vertices), which are colored in grey: The global ears cage, which encloses some vertices of the left and right ear cages, and the head cage, which encloses all unbinded vertices of the previous cages (see Figure 13(a)). The sequence of deformations in the figure is as follows: First the teeth have been deformed in Figure 13(b), second the ears in 13(c), then the entire head in 13(d) and finally the face in 13(e). On the other hand, we have also applied our multi-level approach over the whole squirrel model (37588 triangles), as can be seen on the right of

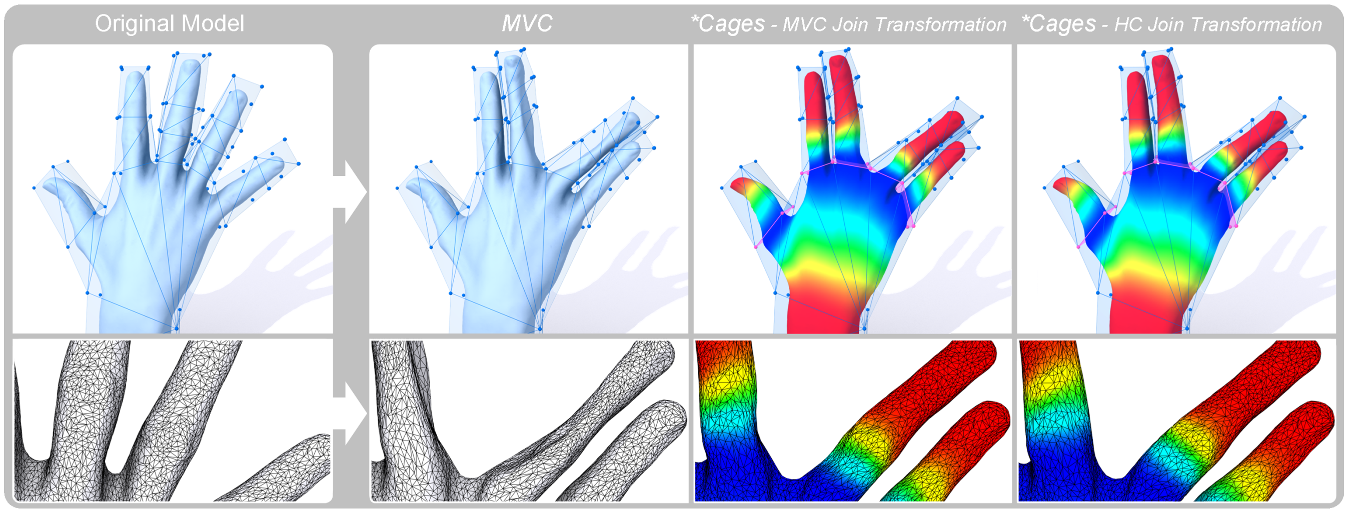
Original Model
MVC
*Cages - MVC Join Transformation
*Cages - HC Join Transformation

Fig. 10. Deformations on the hand model. Columns from left to right: Defined cages (72 vertices), deformation using MVC (observe the effect of negative coordinates of non-convex cages), and *Cages using MVC (third column) and HC (fourth column) as join transformations, respectively.

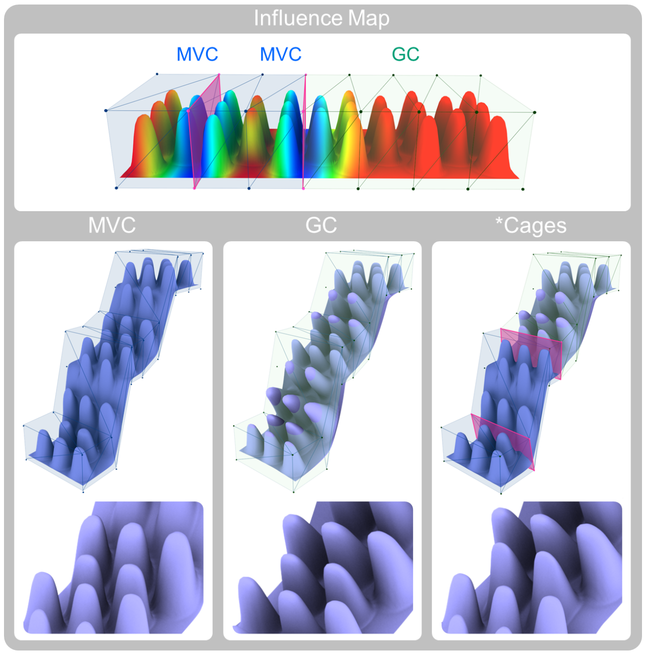

Fig. 11. Fairness of the resulting MVC (left), GC (middle) and MVC/GC with *Cages (right) deformations. The top row shows the influence map and the cages at binding time.

Figure 14. The model has been enclosed in 12 *leaf-cages* (132 vertices) and 3 *internal-cages* (24 vertices), shown at the top-left side of the image. Also, observe the usage of different coordinates for different cages in addition to the deformations produced at different levels of detail. As can be seen, we are able to perform deformations at different levels of detail without recomputing any coordinate for the mesh vertices, which is something that none of the existing single-cage-based approaches allows. As an example, imagine we have a *partial cage* with HC for the ear of the squirrel model in Figure 13 and a single cage containing the whole model. One could compute coordinates for each cage, then deform them using the *partial cage* and then the single cage. What happens here is that, when deforming using the later, the coordinates won't be valid as the vertices affected by the partial cage have seen modified their binding positions. So, we must recompute coordinates each time we go from the usage of partial (local) to single (global) cages and vice versa. In contrast, *Cages can change the level of detail over the deformation without the need for any coordinate recomputation for mesh vertices, as they are controlled by the leaf-cages.

As has been explained in Section 3, *Cages naturally supports the presence of *interior cage vertices*. In Figure 15 (left) we show a ”Easter Egg” model enclosed in a grid of twelve cages. This set of cages generates two interior vertices, one at the top and the other at the bottom of the ”surprise” mesh. As a way to demonstrate the good behavior of deformations when these kinds of vertices are involved, we have generated two different deformations: the first deformation has been obtained by moving up the top interior vertex and leaving the rest of cage vertices stationary (Figure 15(a)), while the second deformation has been achieved by moving the top row of cage vertices up and leaving the rest in their original positions (Figure 15(b)).

Given the multi-cage nature of *Cages it allows the user to naturally reduce the number of weights stored for each mesh vertex and, as a consequence, to obtain faster evaluations. The computational and memory costs become optimal when the number of cages used is high and they have a small adjacency degree, that is, when one cage is connected to a reduced number of neighboring cages, as in Figure 16. The small adjacency degree gives as a result a small number of *join transformations* involved in the final *smooth transformation* of every single cage, and consequently faster and lower memory-consuming deformations. Despite this, it is important to mention that the user can control the influence of *join transformations* inside a cage by adjusting the influence map parameter h_{c_i}. In Table I we show the memory and time requirements depending on the influence of the parameter h_{c_i} in the boundary weight function $b_{c_i}(p)$ (see Section 3.2). We

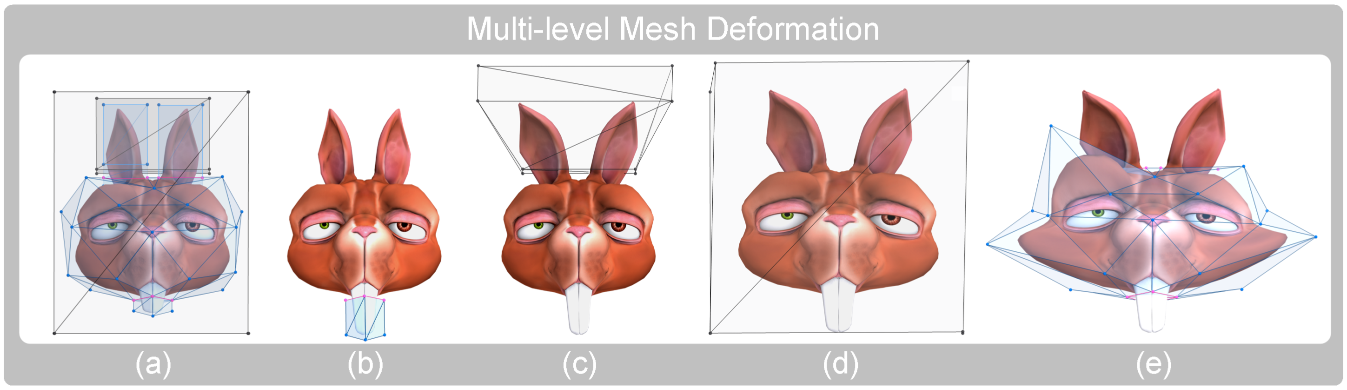

Fig. 13. Multi-level deformation of the squirrel model. (a) Multi-level cages. (b) Leaf deformation: teeth cage. (c) Internal deformation: ears' cage. (d) Internal deformation: head cage. (e) Leaf deformation: face cage.

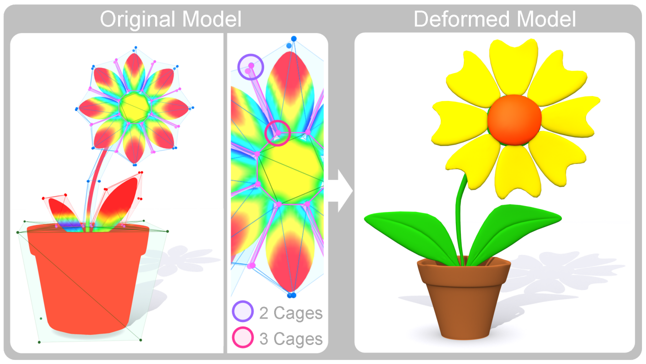

Fig. 12. Multiple cages meeting at a cage vertex. Left: Original model with 13 cages using different coordinates. Middle: close view. Right: Deformation with *Cages.

compare the results obtained for three different h_{c_i} values for the set of 9 cages used (134 vertices) in the chinchilla model (140126 triangles), both for MVC and GC. In these cases, cage and *join transformations* have been computed with the same coordinate types. Observe that the memory usage and the computational cost are nearly proportional to h_{c_i}. This is because, as the h_{c_i} values decrease, transformations $T_{c_i}(p)$ are fully applied on more mesh vertices, and for these the *join transformations* $J_{c_i}(p)$ do not need to be stored and computed. As can bee seen, the parameter h_{c_i} has a drastic impact on *Cages requirements, but using an insufficient value for h_{c_i} could introduce visible non-smooth transitions in extreme deformation conditions. Also, it should be noted that the value for h_{c_i} can be set in an easy and independent manner for each border, for each cage or for the whole model. In Figure 4 we have used the second approach, while for all the tables we have used a single h_{c_i} value for the entire model to make comparisons fairer. In our system, the user is provided with a simple slider to control this parameter independently for each selected cage.

In Figure 16 we show three different deformations applied over the Sintel model (66845 triangles), which has been used in real film production. Note that the model has 15 leaf-cages (193 vertices), as can be seen on the left part of the image. Different coordinates have been used for the different cages to obtain the deformations shown. Note the good results obtained by our approach even when several types of coordinates are used in such complex deformations.

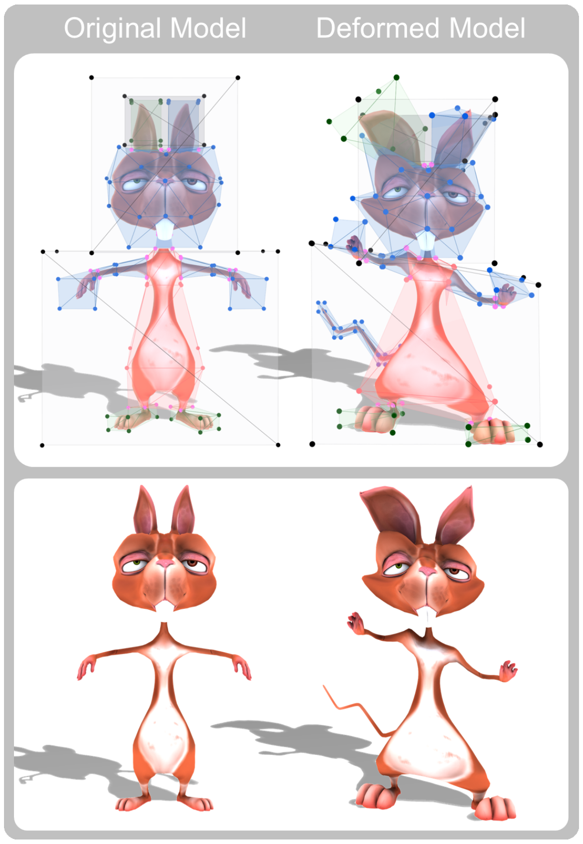

Fig. 14. Deformation of the squirrel model using *Cages. Left: The model and its multi-level cages at binding time. Right: Composition of a pose.

In Table II we compare *Cages with MVC and GC on the Sintel model of Figure 16 and on the squirrel model (12 leaf-cages with 132 vertices and 3 internal-cages with 24 vertices) of Figure 14. Observe that *Cages consumes less than half the memory for the squirrel model and 4 times less than the memory for the

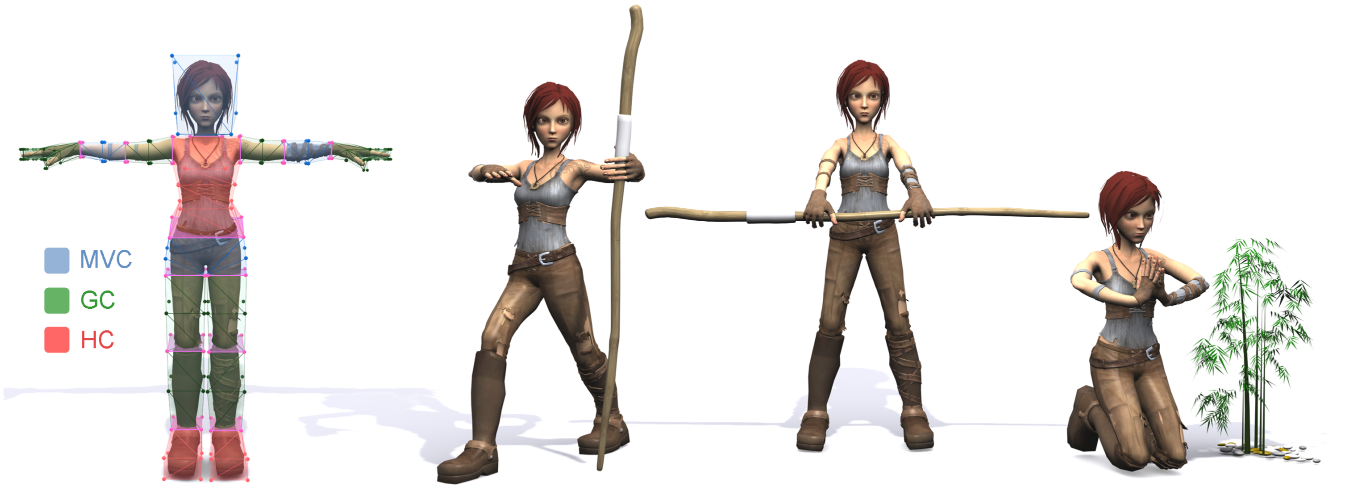

Fig. 16. Deformations of the Sintel model (66845 triangles) using *Cages. Left: Cages at binding time with different coordinates (Blue - MVC, Green - GC, Red - HC, and pink cage boundaries). Right: Composition of different poses.

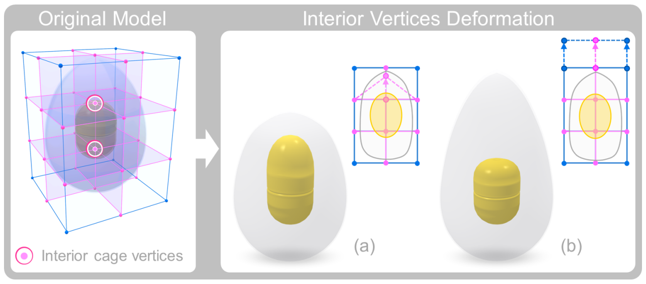

Fig. 15. Deformation involving interior points of the "Easter Egg" model using *Cages. Left: The model and the grid of cages at binding time. Highlighted vertices are interior points. Right: Composition of two different deformations.

Sintel model (column 2). The total time required for the preprocess is shown in column 3, specifying the amount of time dedicated to compute the coordinates with respect to the parent cages. Also, *Cages takes much less time to compute cage coordinates because each of the cages used are simpler and smaller than a whole single cage. The rest of the time is needed to compute join cages and the coordinates with respect to them. In the case of using GC, *Cages requires even less preprocessing time because of the nature of their computations [Lipman et al. 2008]. The deformation times (column 4) are the averages of the times needed for the deformation of a cage vertex. Observe that our approach is significantly faster for both models, where we achieve between 3 and 5 times the speed of MVC, and between 7 and 18 times that of GC.

We would like to emphasize that, even our code is unoptimized and CPU-based, *Cages allows for a more GPU-friendly implementation than single cage-based approaches do, as it has a much lower number of weights to store for each mesh vertex. Moreover, unlike the technique presented by Landreneau and Schaefer [2010], we don't need to be constrained by having to create new deformations that must be similar to an initial range of predetermined poses to be able to reduce memory and time consumption. Instead, we give the user the freedom to perform any type of deformation while also keeping the memory and time requirements small, as well. Let us note that *Cages is fully compatible with the work by Landreneau and Schaefer [2010], and our computational requirements could be reduced even more if used together: Their compression could be used for both cage and join transformations. The latter case would benefit *Cages the most, as join transformations are more computationally demanding to evaluate than regular cage transformations.

*Cages is not related with the modeling of cages themselves. As the examples throughout the paper have shown we use a set of individual cages, the union of which result in a single cage for the entire model. This has been done as a way to make comparisons to previous single cage-based approaches fairer. With *Cages we don't need to create the whole set of cages that are equivalent to a single cage. For instance, if we want to deform only the head of the Sintel model shown in Figure 16, we are not required to build all the cages shown there, we only need to model the ones needed to make this task simpler. Moreover, the modeling of cages used to deform a small region is usually easier and faster and so, the use of many cages to deform a mesh can result in a more user-friendly element for the cage-modeling phase.

As a space deformation approach, *Cages can be used in the same domains as previous methods. For instance, the lowest-level cages of our hierarchy could be deformed by a simple skeleton, as Ju at al. [2008] did. Thanks to the local behavior of our approach, we could provide a finer degree of control over the skeleton and, as a result, a smoother final animation. *Cages also can be used to perform deformations in 2D, as long as the cages satisfy the requirements described in Section 3. *Cages is a cage-based method that can be also integrated with other deformation techniques that uses other types of handles, as the one proposed by Jacobson et al. [2011]. For instance, our approach can be used in a certain region of the model to perform local and hierarchical deformations with MVC/HC or GC. Then, on the rest of the model, the bounded biharmonic weights could be used with point and

Table I. Memory and time requirements for the chinchilla model using different h_{c_i} values for *Cages with MVC and GC.

Chinchilla	**Memory**	**Preprocess**		**Deform**
	(MB)	Cage Coord.	Total	(sec)
MVC *Cages				
h_{c_i} = 1.0	45.45	3.8045	29.4625	0.3308
h_{c_i} = 0.6	29.75	3.7958	17.3009	0.1409
h_{c_i} = 0.2	14.05	3.8034	9.5280	0.0662
GC *Cages				
h_{c_i} = 1.0	131.27	10.1461	134.8266	1.0725
h_{c_i} = 0.6	84.35	10.3721	73.5973	0.4460
h_{c_i} = 0.2	37.43	10.2712	49.1273	0.1888

Table II. Memory and time requirements: comparison between *Cages (h = 0.5) and single cage-based methods.

Model	**Memory**	**Preprocess** (sec)		**Deform**
	(MB)	Cage Coord.	Total	(sec)
Squirrel				
MVC	25.55	7.4815	7.4815	0.0904
MVC *Cages	11.26	3.4861	6.6197	0.0372
GC	63.30	39.7976	39.7976	0.7095
GC *Cages	30.39	8.3214	28.9178	0.0969
Sintel				
MVC	52.52	20.1988	20.1988	0.2105
MVC *Cages	13.54	8.3469	13.7599	0.0427
GC	155.41	107.4353	107.4353	1.8824
GC *Cages	34.94	16.3395	47.9066	0.1074

bone handlers. *Cages would be responsible for smoothly gluing the deformations provided by cage-based methods (MVC/HC/GC) with those produced with the technique of Jacobson et al. [2011], thus avoiding the discontinuities that would appear through cage boundaries.

Acknowledgements

We would like to thank the anonymous reviewers for their insightful comments and suggestions. This work was partially funded with grant TIN2010-20590-C02-02 from the Ministerio de Ciencia e Innovacin.

REFERENCES

BEN-CHEN, M., WEBER, O., AND GOTSMAN, C. 2009. Variational harmonic maps for space deformation. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*. ACM, New York, NY, USA, 1–11.

BOROSAN, P., HOWARD, R., ZHANG, S., AND NEALEN, A. 2010. Hybrid mesh editing. In *Proc. of Eurographics 2010 (short papers)*.

BOTSCH, M. AND KOBBELT, L. 2005. Real-time shape editing using radial basis functions. In *Computer Graphics Forum*. 611–621.

COQUILLART, S. 1990. Extended free-form deformation: a sculpturing tool for 3d geometric modeling. *SIGGRAPH Comput. Graph. 24*, 187–196.

FLOATER, M. S. 2003. Mean value coordinates. *Comput. Aided Geom. Des. 20,* 1, 19–27.

FLOATER, M. S., KÓS, G., AND REIMERS, M. 2005. Mean value coordinates in 3d. *Computer Aided Geometric Design 22,* 7, 623–631.

HUANG, J., CHEN, L., LIU, X., AND BAO, H. 2009. Efficient mesh deformation using tetrahedron control mesh. *Comput. Aided Geom. Des. 26,* 6, 617–626.

JACOBSON, A., BARAN, I., POPOVIĆ, J., AND SORKINE, O. 2011. Bounded biharmonic weights for real-time deformation. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH) 30,* 4, 78:1–78:8.

JOSHI, P., MEYER, M., DEROSE, T., GREEN, B., AND SANOCKI, T. 2007. Harmonic coordinates for character articulation. *ACM Trans. Graph. 26,* 3, 71.

JU, T., SCHAEFER, S., AND WARREN, J. D. 2005. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph. 24,* 3, 561–566.

JU, T., ZHOU, Q.-Y., VAN DE PANNE, M., COHEN-OR, D., AND NEUMANN, U. 2008. Reusable skinning templates using cage-based deformations. *ACM Trans. Graph. 27,* 5 (Dec.), 122:1–122:10.

LANDRENEAU, E. AND SCHAEFER, S. 2010. Poisson-based weight reduction of animated meshes. *Comput. Graph. Forum 29,* 6, 1945–1954.

LANGER, T., BELYAEV, A., AND SEIDEL, H.-P. 2008. Mean value bézier maps. In *GMP'08: Proceedings of the 5th international conference on Advances in geometric modeling and processing*. Springer-Verlag, Berlin, Heidelberg, 231–243.

LI, Z., LEVIN, D., DENG, Z., LIU, D., AND LUO, X. 2010. Cage-free local deformations using green coordinates. *Vis. Comput. 26,* 6-8, 1027–1036.

LIPMAN, Y., KOPF, J., COHEN-OR, D., AND LEVIN, D. 2007. Gpu-assisted positive mean value coordinates for mesh deformations. In *SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 117–123.

LIPMAN, Y., LEVIN, D., AND COHEN-OR, D. 2008. Green coordinates. *ACM Trans. Graph. 27,* 3 (Aug.), 78:1–78:10.

MENG, W., SHENG, B., WANG, S., SUN, H., AND WU, E. 2009. Interactive image deformation using cage coordinates on gpu. In *VRCAI '09: Proceedings of the 8th International Conference on Virtual Reality Continuum and its Applications in Industry*. ACM, New York, NY, USA, 119–126.

SEDERBERG, T. W. AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph. 20*, 151–160.

SEO, H. AND THALMANN, N. M. 2000. Lod management on animating face models. In *Proceedings of the IEEE Virtual Reality 2000 Conference*. VR '00. IEEE Computer Society, Washington, DC, USA, 161–.

WEBER, O., BEN-CHEN, M., AND GOTSMAN, C. 2009. Complex barycentric coordinates with applications to planar shape deformation. *Computer Graphics Forum 28,* 2, 587–597.

ZHENG, Y., FU, H., COHEN-OR, D., AU, O. K.-C., AND TAI, C.-L. 2011. Component-wise controllers for structure-preserving shape manipulation. In *Computer Graphics Forum (In Proc. of Eurographics 2011)*. Vol. 30. to appear.

Received September 2008; accepted March 2009