Eurographics Symposium on Geometry Processing (2007)
Alexander Belyaev, Michael Garland (Editors)

GPU-assisted Positive Mean Value Coordinates for Mesh Deformations

Yaron Lipman[1] and Johannes Kopf[2] and Daniel Cohen-Or[1] and David Levin[1]

[1]Tel Aviv University
[2]University of Konstanz

Abstract

*In this paper we introduce positive mean value coordinates (PMVC) for mesh deformation. Following the observations of Joshi et al. [JMD*07] we show the advantage of having* positive *coordinates. The control points of the deformation are the vertices of a "cage" enclosing the deformed mesh. To define positive mean value coordinates for a given vertex, the* visible portion *of the cage is integrated over a sphere.*
*Unlike MVC [JSW05], PMVC are computed numerically. We show how the PMVC integral can be efficiently computed with graphics hardware. While the properties of PMVC are similar to those of Harmonic coordinates [JMD*07], the setup time of the PMVC is only of a few seconds for typical meshes with 30K vertices. This speed-up renders the new coordinates practical and easy to use.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Line and Curve Generation

1. Introduction

In recent years surface manipulation has received a considerable amount of attention. The prominent recent progress has been in the development of surface-based techniques, where the deformations are applied on the surface itself. Less attention has been paid to free-form deformation (FFD) techniques which deform the space in which the shape is embedded. The work we present here is a space-deformation technique which builds upon the mean-value coordinates [Flo03, FKR05, JSW05, HF06]. Specifically, as we will describe in this paper, our work uses the key idea introduced by Joshi *et al.* [JMD*07], namely, positive coordinates, and render it practical and easy to use.

When comparing surface-based methods to space deformation methods there are few immediate conclusions. The main advantage of surface-based methods is the high quality of detail preservation. These are typically achieved using differential coordinates, which requires solving a large system (linear or non-linear) which is computational expensive. The dimension of such systems is proportional to the number of vertices in the surface representation. Moreover, these methods rely on differential operators which are applied on a mesh, and thus sensitive to the mesh quality and noise.

Space deformation techniques have less control on the shape detail preservation, and it is not clear what surface properties are maintained in such deformation. However, they are much simpler and require considerably less computational cost than surface-based methods. Since they deform the space around the mesh, rather than the mesh itself, they are inherently insensitive to the mesh quality. Due to their simplicity they are commonly used in commercial software such as Maya and 3D studio.

The control points of early space deformation methods are geometry-oblivious [SP86, Coq90], and thus the user control over the deformation of the embedded mesh is limited. The introduction of mean value coordinates (MVC) as a space deformation method led to much more control over the embedded mesh since the control points are geometry-aware. The control points are defined as the vertices of a control mesh, which can conveniently be called a "cage" [JMD*07]. The cage encloses the mesh in a loose manner, flexible enough so that many similar meshes can fit within the same cage. Yet, the cage is tight enough to provide a good control of the space in which the mesh is embedded.

The MVC posses several useful properties such as linear precision, interpolation and smoothness. However, as

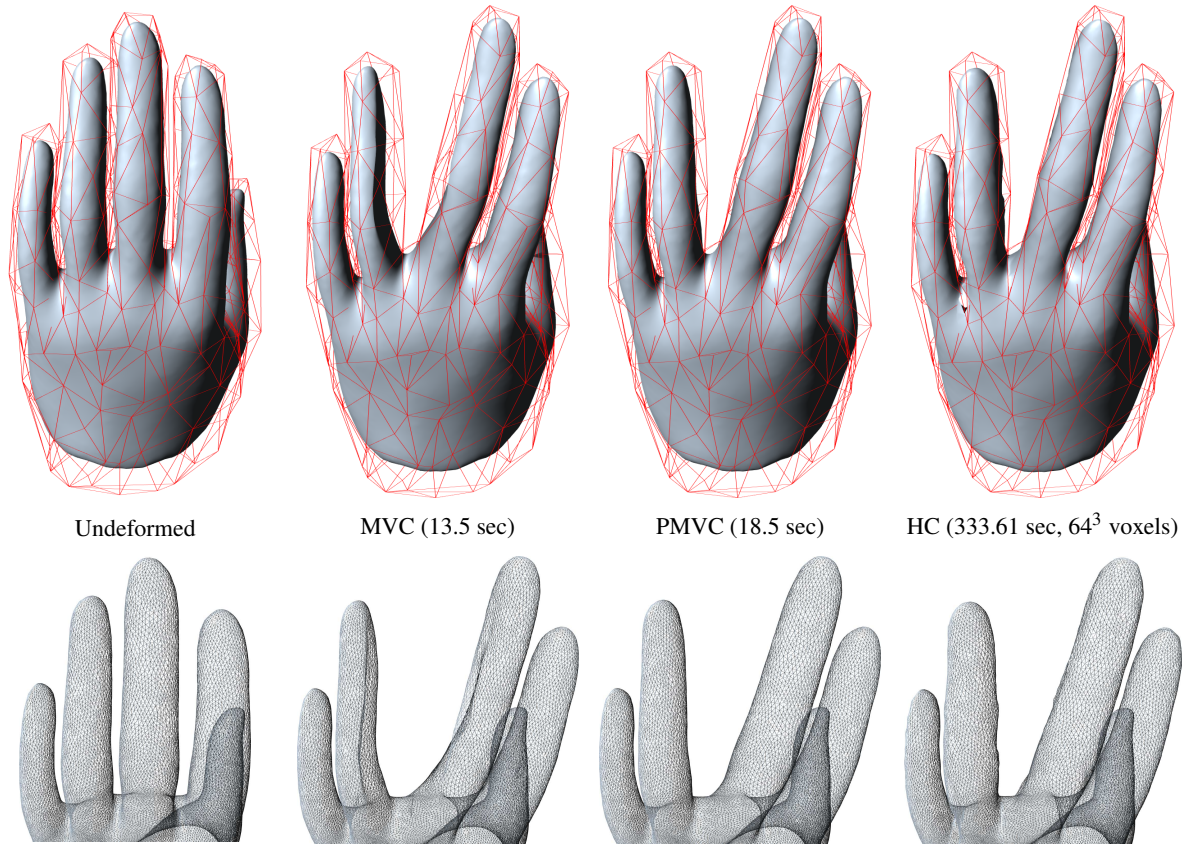


Figure 1: *Comparison between MVC, PMVC, and HC. The weakness of negative cordinates is clearly evident, while HC takes a long time to compute. The PMVC, in contrast, have no negative coordinates and are computed almost as quickly as the MVC.*

shown by Joshi *et al.* [JMD*07], the main flaw of MVC as a tool for surface deformation is that they are not necessarily positive on non-convex domains. This in turn leads to counter-intuitive deformation when the control mesh is non-convex. As illustrated in Figure 1, the negative coordinates of a remote branch of the cage distort the geometry in a non-intuitive unexpected way.

Joshi *et al.* [JMD*07] suggest using harmonic functions with Kronecker Delta-type boundary conditions to furnish the desired positive coordinates. hereafter we will refer to these coordinates with HC. Their solution is mathematically elegant and guarantee positiveness, however, rather expensive in practice; the coordinates requires solving the Laplace equation on the *whole* interior of a three dimensional domain (the interior of the cage). The solution of the Laplace equation is a non-local expensive process. The solution is practically calculated on a grid inside the control mesh, where the grid resolution is determined by an error criterion. In that case one has to balance accuracy with storage and computation; increasing the grid resolution by one level octuplicates (8X) the storage and leads to significantly slower computation.

In this work we introduce positive mean value coordinates (PMVC). Unlike the MVC, the modified coordinates are unconditionally positive, and require only a local computation. We demonstrate the advantage of positive coordinates in various examples of surface deformation (e.g., Figure 1). The PMVC are too involved to be computed analytically from their closed form formula, instead, we introduce a GPU-assisted technique to calculate numerically the coordinates of a given input mesh. As we will show, the computation of new coordinates, either as a result of a new cage or a new mesh, requires few seconds only.

2. Positive Mean Value Coordinates

Given a shape to be deformed, denote by C a coarse control mesh enclosing it. We will refer to C as a "cage" similar to Joshi *et al.* [JMD*07]. We denote by $V = \{v_i\}_{i \in I_V}$ the vertices of the cage C, where I_V is the corresponding index set. Similarly to the cited previous works, the goal is to define a set of functions $\lambda_i(x)$, $i \in V$ such that the operator

$$\mathscr{P}_f(x) := \sum_{i \in V} \lambda_i(x) f(v_i), \qquad (1)$$

and the functions $\lambda_i(x)$ satisfy the following properties:

1. *Affine invariance*: $\sum_i \lambda_i(x) = 1$.
2. *Linear reproduction*: $\mathscr{P}_f(x) = f(x)$ for all linear functions $f(x)$.
3. *Interpolation*: $\lim_{x\to c} \mathscr{P}_f(x) = f(c)$ where $c \in \partial C$ (the boundary of C).
4. *Smoothness*: $\lambda_i(x)$ are smooth.
5. *Positivity*: $\lambda_i(x) \geq 0$ for all i.

where all these properties should hold for all $x \in C$ (inside the cage). The first four properties were formulated in [JSW05], and the last property is introduced in [JMD*07].

The PMVC is defined as follows:

$$\mathscr{P}_f(x) = \frac{\int_{s\in S_x} f(\pi(s))w(x,s)d\sigma}{\int_{s\in S_x} w(x,s)d\sigma}, \tag{2}$$

where S_x is a unit sphere centered at x, and $\pi(s)$ is the first intersection of the line $\ell(t) := x + (s-x)t$ and C, for $t \geq 0$, and $w(x,s) = 1/\|x-\pi(s)\|$.

Note the interesting relation between MVC and PMVC; in PMVC the sphere is projected on the cage, while in the former the cage is projected onto the sphere.

Given $f(v_i), i \in I_V$, let $\psi_i(x)$ be a piecewise-linear function such that $\psi_i(v_{i'}) = \delta_{i,i'}$, where $\delta_{i,i'} = 1$ iff $i = i'$, and 0 else. Define the piecewise linear function $\sum_{i\in V} f_i\psi_i(x)$ on the boundary of C, and then the approximant is:

$$\mathscr{P}_f(x) = \sum_{i\in V} f_i \frac{\int_{s\in S_x} \psi_i(\pi(s))w(x,s)d\sigma}{\int_{s\in S_x} w(x,s)d\sigma}.$$

Then, we define

$$\lambda_i(x) := \frac{\int_{s\in S_x} \psi_i(\pi(s))w(x,s)d\sigma}{\int_{s\in S_x} w(x,s)d\sigma}, \tag{3}$$

and the interpolant can then be written in the form (1).

The $\lambda_i(x)$ are called *coordinate functions* for the PMVC interpolant. The coordinates are used for defining a transformation T from the interior of the cage C into $\mathbb{R}^d$ ($d = 2,3$):

$$T : \text{interior}(C) \to \mathbb{R}^d.$$

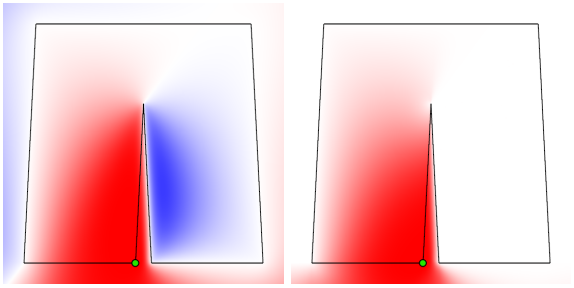
The transformation T is defined as follows: A point $p \in \text{interior}(C)$ can be written as the following affine combination due to the linear reproduction property

$$p = \sum_{i\in I_V} \lambda_i(p)v_i.$$

Then, given a deformed cage $\tilde{C}$ with vertices $\tilde{V} = \{\tilde{v}_i\}_{i\in i_V}$ the transformed position of p is defined as

$$T(p) := \sum_{i\in I_V} \lambda_i(p)\tilde{v}_i.$$

Therefore, the properties of the transformation T are derived from the properties of the coordinate functions $\lambda_i(x)$. Let us prove some of the properties listed above.



MVC PMVC

Figure 2: *A coordinate function $\lambda_i(x)$ is drawn for a non-convex polygon. The i-th vertex is marked with green point. The red color stands for positive values and the blue are negative values.*

The interpolation and linear reproduction properties can be understood by the argumentation of [JSW05], but for the sake of completeness we will lay it here also. First, the interpolation is due to the fact that $\frac{w(x,y)}{\int_{s\in S_x} w(x,s)d\sigma}$ is converging to the Delta function on the sphere as $x \to c$ for $c \in \partial C$. As for the linear reproduction property it results from the symmetry argument:

$$\int_{s\in S_x} \frac{x-\pi(s)}{\|x-\pi(s)\|} d\sigma = 0.$$

And therefore,

$$x = \frac{\int_{s\in S_x} \pi(s)w(x,s)d\sigma}{\int_{s\in S_x} w(x,s)d\sigma}.$$

That is, the coordinate functions are reproduced, and from the linearity of the operator (1) and the affine invariance the property results. As to the non-negativity property of the coordinate functions, this readily results from the fact that the coordinates $\lambda_i(x)$ are defined via an integration of a non-negative function over a sphere in Eq. (3). For example, see Figure 2 for comparison with MVC.

Smoothness One of the strong properties of the MVC is that they are smooth. The PMVC definition involves visibility consideration which incurs singularities across supporting planes of reflex vertices. The supporting planes partition the cage into regions within which the PMVC are smooth, while across the supporting planes smoothness is not guaranteed. An example of such scenario is shown in Figure 4, where the coordinate function associated with the 'spike' vertex is not smooth. However, as depicted in that figure, minor refinement of the cage alleviates this problem. In practice, we found that in most cases the result is plausible and smooth. This can be observed in Figure 3, where a shape of a checkerboard pattern that crosses a supporting line is smoothly bent. The example shows the effect of editing a single vertex and consequently of a single coordinate $\lambda_i(x)$.

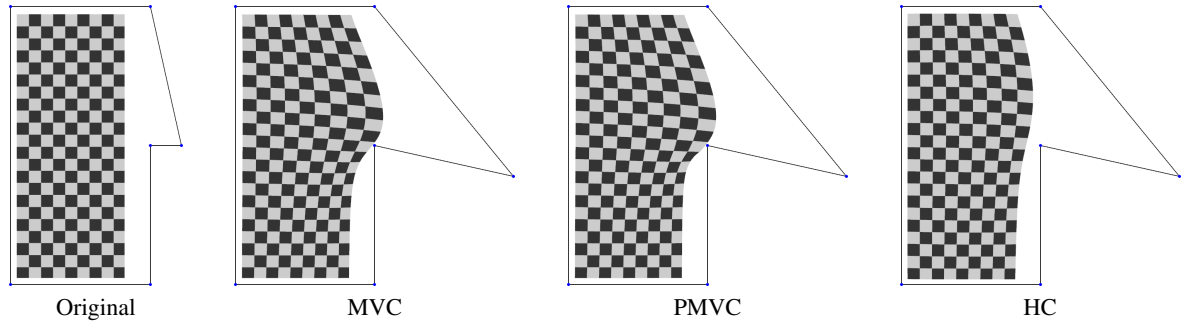


Figure 3: *Smoothness test across supporting visibility line. The influence of the coordinate function associated with the rightmost vertex is practically smooth across the supporting line; this is depicted by the deformation of a checkerboard shape (which cross the supporting line) when moving this vertex only.*

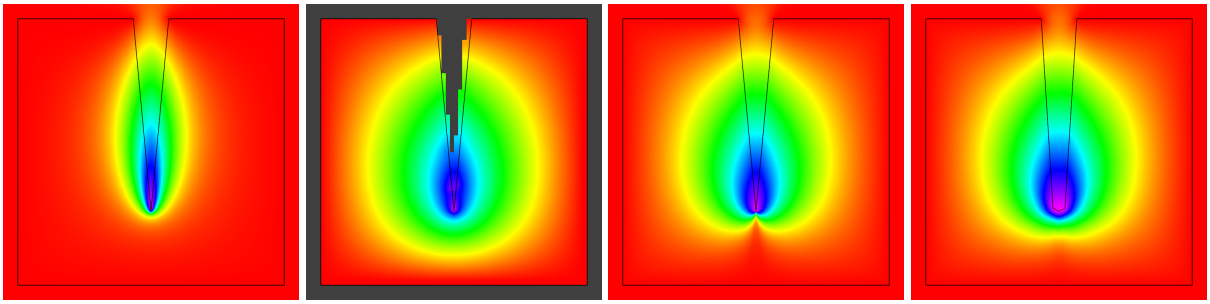


Figure 4: *In the vicinity of very large reflex angles the PMVC might be non smooth. This problem is alleviated by subdividing the acute angles to obtain a cage with a smaller maximum angle. In the rightmost image the sum of all three relevant coordinate functions are colored. Note that subdividing the cage has almost negligible impact on the PMVC performance (see Table 2).*

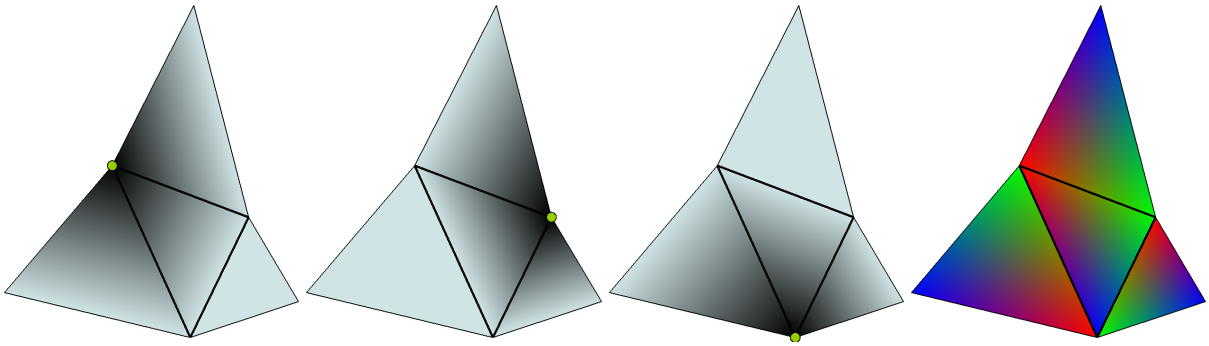
3. Calculating the coordinates

In this section we explain our method to compute the PMVC coordinates λ_i for a vertex v. The coordinates could theoretically be computed analytically with the MVC framework. However, this would require to re-mesh the cage *for every object vertex*, so that only the visible parts are included; triangles need to be cut along the visibility lines. This operation is very expensive, and renders this approach too slow to be useful in practice. Our approximate method, in contrast, computes the coordinates much faster using the GPU.

Equation 3 computes a coordinate λ_i by integrating the *visible parts* of the associated ψ_i function over a sphere. We compute this efficiently by rendering the ψ functions into cube maps, and then integrating (summing) the rendered values. The advantage of this method is that graphics hardware is designed to perform visibility computations very efficiently, and that *all coordinates* for a given vertex can be computed *simultaneously*. We start, however, by first explaining how to compute a *single* coordinate λ_i using our method. Later, we show how it can be accelerated by computing all coordinates simultaneously.

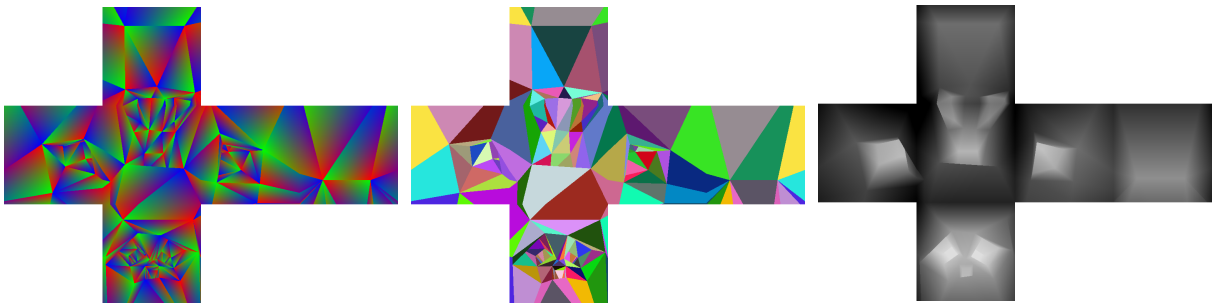
We set the colors $c_{i'}$ of the cage vertices, such that $c_{i'} = \delta_{i,i'}$ (i.e. $c_{i'} = \psi_i(v_{i'})$. In other words, vertex i is set to one, and all other vertices to zero, as shown in Figure 5(a–c). When rendering the cage, the built-in barycentric interpolation of the GPU computes automatically the correct values of ψ_i at each pixel. We start with rendering the cage as viewed from vertex v into a cube map. Then, we read back the colors and depth, and integrate the rendered values on the CPU. We give each cube map pixel a specific weight to account for the fact that we integrate over a cube rather than a sphere. The weight is the area of the cube map pixel projected on the unit sphere. This can be effectively computed as the sum of areas of the two spherical triangles that can be formed from the projected pixel vertices.

The GPU stores internally a transformed version of the true depth to achieve better accuracy for near objects. The exact depth can be computed as $d = z_a/(z - z_b)$, where z is the transformed depth from the GPU, $z_a = \frac{d_{far}}{d_{far} - d_{near}}$, $z_b = \frac{d_{near} d_{far}}{d_{near} - d_{far}}$, and d_{near} and d_{far} are the distances of the near and far clipping planes, respectively. To maximize depth accuracy, the near and far planes are set tightly around the in-



(a) (b) (c) (d)

Figure 5: *(a-c) ψ functions for three vertices of a cage mesh, (d) all ψ functions packed into a single rendering.*



Color Buffer Index Buffer Depth Buffer

Figure 6: *Render buffers read back by the algorithm. The color and index buffers are interleaved in a single buffer.*

terval of possible depths. The near plane is set slightly closer as the closest point-to-triangle distance of the object vertices, and the far plane is set slightly further away than the maximum distance between cage vertices.

The system so far explained requires rendering and integrating a cube map for each vertex and for each coordinate. We accelerate the computation using the following observation: each point on the surface of a triangular cage has at most three ψ functions that take on a non-zero value. Inside each cage triangle, these are the ψ_i corresponding to the triangle vertices. We leverage this fact by using a different color scheme for the cage that encodes all non-zero ψ functions inside each triangle. Then, we can compute all coordinates in a single sweep over the rendered data.

Each cage triangle is colored in the same way, with one vertex red, one green, one blue, i.e., the vertices have the colors $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$, respectively. This causes each vertex to influence exactly one of the RGB color channels (see Figure 5). The remaining 4^{th} color channel is constant over each triangle, and takes the triangle index number (see Figure 6). During integration, when examining a pixel, we use the triangle index in the 4^{th} color channel to look-up which cage vertices correspond to the RGB color channels. The respective λ values are then updated with the contributions of the color channels. Figure 6 shows the full cube map for a vertex in the Armadillo model. Reading back the render buffers after each draw operation would cause a significant overhead, because of CPU stalling when waiting for draw operations to finish. Therefore, we use a larger render buffer (e.g. 2048^2 pixels) that can hold many cube map faces. We render as many faces as possible into this buffer, and only once it is full we read it back for integration. Our computation scheme is detailed in pseudo-code in Algorithm 1.

4. Results and discussion

We have experimented with PMVC for mesh deformation, with various models and cages, and compared it with MVC and HC. The advantage of positive coordinates is manifested for example in shapes with close distance between branches of the cage, such as in the hand model in Figure 1 Figure 7 shows an example of well separated limbs cage. In that case, the negative coordinates have only little effect and the MVC and PMVC perform similarly. PMVC and HC generally yield similar visual results, except in cases where PMVC is not smooth (see Section 2), or HC has some numerical artifacts caused by an overly coarse grid, see for example Figure 1 and 4.

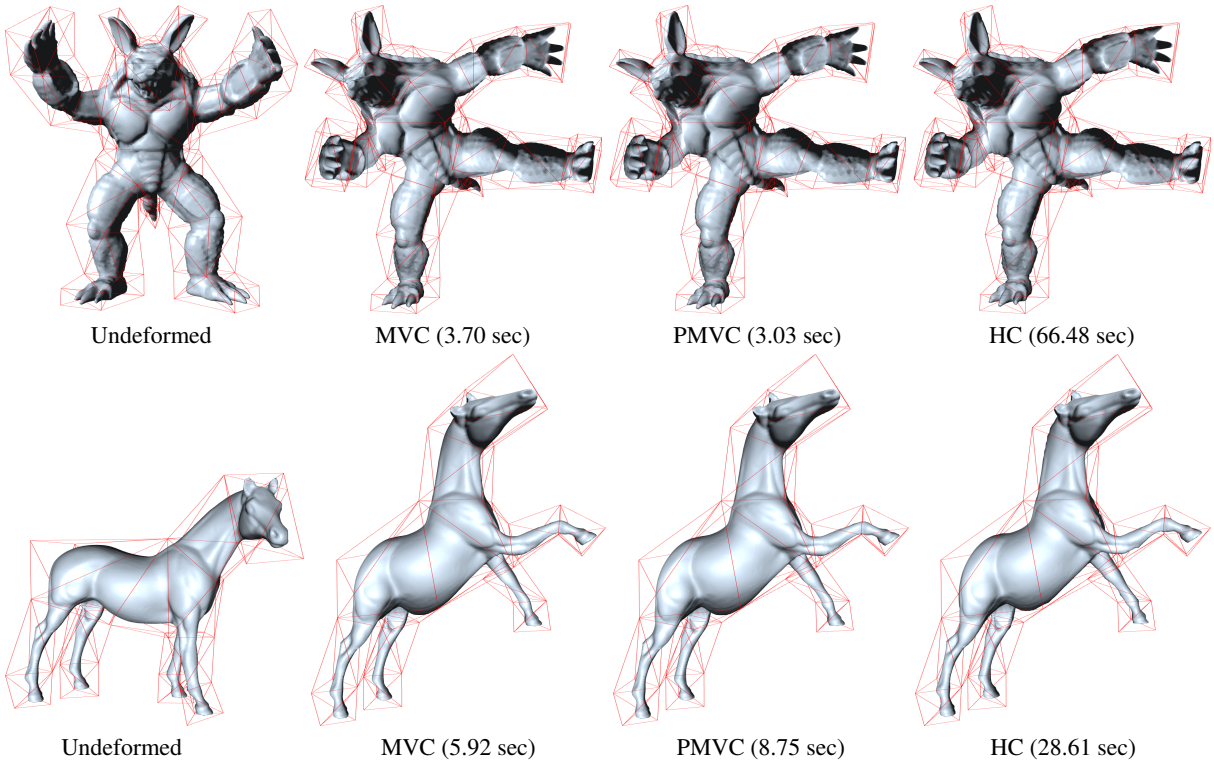


Figure 7: *Comparison between MVC, PMVC, and HC deformations of the Armadillo and Horse models. Note that on models like these, where the cage has well seperated limbs, the deformation quality is similar.*

The running times of PMVC are generally in the same order of magnitude as MVC, and about 10–100 times faster than HC, depending on mesh and cage complexity. The performance is roughly linear in the number of cube map pixels. We found that for most meshes 32^2 cube maps have fine enough resolution so that no quantization artifacts are noticeable. The hand model (Figure 1) was created with 64^2 cube maps. Table 1 provides detailed timings for various examples shown in this paper. The performance of PMVC is only loosely connected to cage complexity. Table 2 shows timings for a single object with increasingly more complex cages.

As to memory usage, it should be noted that PMVC, similar to MVC, computes the coordinates of each embedded shape point directly on-the-fly during setup.

5. Conclusions

We presented mean value coordinates that consider only the visible portion of the cage to guarantee, like harmonic coordinates, that the coordinate values are always positive. The key point is that the positive mean value coordinates can be computed fast by exploiting the readily available visibility computation of the GPU. Furthermore, as we showed, the GPU speed turns the computation practically insensitive to the cage resolution. This allows us to refine the cage and improve the quality of the interpolation without significant cost. Our method successfully brings the idea of positive coordinates to the point where it is truly a practical and useful tool for mesh deformation. We believe that more research can lead to even faster methods to compute local coordinates with assistance of the GPU.

Acknowledgements

We thank Scott Schaeffer for providing us with the cages and models of the Armadillo and horse, and Hongbo Fu and Kun Zhou for the hand model. We also thank Tao Ju for helpful and insightful comments. This work was supported in part by the Israel Science Foundation.

References

[Coq90] COQUILLART S.: Extended free-form deformation: a sculpturing tool for 3d geometric modeling. *Proceedings of SIGGRAPH '90* (1990), 187–196.

[FKR05] FLOATER M. S., KOS G., REIMERS M.: Mean value coordinates in 3d. *Computer Aided Geometric Design 22, 7* (2005), 623–631.

[Flo03] FLOATER M. S.: Mean value coordinates. *Computer Aided Geometric Design 20, 1* (2003), 19–27.

```
// Initialization
set all λ_{v,i} = 0
set all wsum_v = 0

// Rendering and integration
foreach vertex v do
    foreach cube map face f do
        render f to render buffers
        if render buffer full or last vertex-face pair then
            // Integration
            foreach render buffer pixel (x,y) do
                tri = getValue(x,y,3)
                w = getWeight(x,y)
                for c = 0 to 2 do
                    i = vertexList[tri*3+c]
                    λ_{v,i} += getValue(x,y,c)·w
                end
                wsum_v += w
            end
            clear render buffers
        end
    end
end

// Normalization
foreach vertex v do
    foreach cube map face f do
        λ_{v,i} /= wsum_v
    end
end
```

Algorithm 1: *Our algorithm to compute the PMVC coordinates.*

	Armadillo	**Horse**	**Hand**
Mesh Vertices	15,002	48,485	24,795
Cage Vertices	110	51	252
MVC	3.70s	5.92s	13.50s
PMVC 32^2	3.03s	8.75s	6.64s
PMVC 64^2	9.82s	30.64s	18.50s
HC 64^3	66.48s	28.61s	333.61s
HC 128^3	770.00s	305.93s	4413.66s

Table 1: *Performance comparison.*

Cage Vertices	**Cage Triangles**	**PMVC timing**
51	98	16.07s
102	200	16.32s
198	392	16.73s
402	800	17.44s
843	1682	18.70s
1523	3042	20.41s

Table 2: *The performance of PMVC is only loosely connected to the cage complexity. The table shows timings for the hand model (24,795 vertices) and spherical cage. The cage was subdivided into increasing numbers of triangles.*

[HF06] HORMANN K., FLOATER M. S.: Mean value coordinates for arbitrary planar polygons. *ACM Transactions on Graphics 25, 4* (2006), 1424–1441.

[JMD*07] JOSHI P., MEYER M., DEROSE T., GREEN B., SANOCKI T.: Harmonic coordinates for character articulation. *Transactions on Graphics 26, 3 (Proc. SIGGRAPH)* (2007).

[JSW05] JU T., SCHAEFER S., WARREN J.: Mean value coordinates for closed triangular meshes. vol. 24, 3 (Proc. SIGGRAPH), pp. 561–566.

[SP86] SEDERBERG T. W., PARRY S. R.: Free-form deformation of solid geometric models. *Proceedings of SIGGRAPH '86* (1986), 151–160.