ISSN 1213-6972 Journal of WSCG http://www.wscg.eu Vol.28, No.1-2, 2020

The Effects of Different Triangulation Techniques for Cage Based Image Deformation Using Generalized Barycentric Coordinates

Ákos Tóth
University of Debrecen, Doctoral School of Informatics; Faculty of Informatics, University of Debrecen
4028 Debrecen, Hungary
toth.akos@inf.unideb.hu

Roland Kunkli
Faculty of Informatics, University of Debrecen
4028 Debrecen, Hungary
kunkli.roland@inf.unideb.hu

ABSTRACT

In computer graphics, the generalized barycentric coordinates (GBC) are often used for image deformation. To manipulate an input image using a cage based image deformation method, we usually have to consider a source polygon with a triangulation; but defining the triangulation of the source polygon is not a trivial task in most cases. In this paper, a uniform and a non-uniform triangulation technique—which can be the basis of the cage based image deformation—are introduced and compared. Moreover, different texture filtering methods are tried, producing various deformation results. Experimental results and demonstrative pictures show the behaviors of the triangulation methods.

Keywords

generalized barycentric coordinates, image deformation, triangulation

1 INTRODUCTION

Barycentric coordinates are frequently used to represent a point inside a polygon as the weighted sum of its vertices. In the last years, many generalizations (e.g., harmonic coordinates [Jos07], mean value coordinates [Ju05], local coordinates [Zha14], or blended coordinates [Ani17a]) and techniques [Ani16] have appeared with a different set of properties. However, most of them satisfy the *linear reproduction property*; therefore, the barycentric coordinates are often used for different interpolation tasks, e.g., shading, mesh parameterization, and shape [Cas18] or image deformation [Hor06].

To use the generalized barycentric coordinates [Flo15, Hor17, Nie13] for cage based image deformation, we usually have to create a triangulation of the source polygon, which envelops the input image to be deformed. However, this mentioned triangulation is not always self-evident because it can affect the quality of the deformation [Ani17b]. Nowadays, the cage based image deformation algorithms are usually implemented on the GPU and operate with a uniform triangulation (e.g., Delaunay) [Web09]. Although they can create a smooth deformation with several thousands of interior vertices in most cases, we can decrease the number of the triangles to save computation time and cost by using a non-uniform triangulation (see Figure 1).

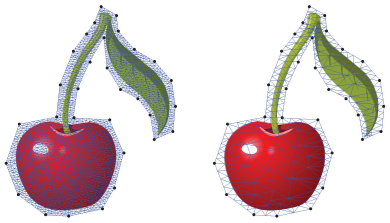

Figure 1: Input image with source polygon using (left) a uniform and (right) a non-uniform triangulation technique. The black dots mark the source polygon defined by the user manually.

Therefore our goal was to examine and compare the two different triangulation techniques in the aspect of the applied cage based image deformation methods. In our comparison, we take into account the input image, the source polygon—which is often determined by the user—, and the used coordinate method. Moreover, we investigate different texture filtering algorithms to improve the quality of the deformation results.

In the next section, we give a short overview of the image deformation methods based on the generalized barycentric coordinates. Then, in Section 3, we discuss the different triangulation techniques. In Section 4, we introduce our non-uniform triangulation algorithm in detail. We show the way how the quality of the deformation results can be improved in Section 5, while in the last two sections, we present our results and future work as well.

2 IMAGE DEFORMATION BASED ON GENERALIZED BARYCENTRIC COORDINATES

As we recalled in Section 1, one of the main application areas of GBC is image deformation.

We can deform an input image I, which is enveloped by a source polygon P with vertices $\mathbf{v}_i$. The source polygon has a triangulation T with vertices $\mathbf{t}_j$. After relocating the source polygon P to P' with vertices $\mathbf{v}'_i$, the new positions $\mathbf{t}'_j$ of the vertices of the triangulation can be computed by the following interpolation function:

$$\mathbf{t}'_j = \sum_{i=1}^{n} b_i(\mathbf{t}_j)\mathbf{v}'_i, \tag{1}$$

where $b_i(\mathbf{t}_j)$ are the barycentric coordinates of the vertex $\mathbf{t}_j$ of the triangulation T respect to the vertex $\mathbf{v}_i$, while n is the number of the vertices of the source polygon P.

The generalized barycentric coordinates $b_i(\mathbf{v})$ of a point $\mathbf{v}$ inside a polygon can be computed by the equations below:

$$\mathbf{v} = \frac{\sum_{i=1}^{n} w_i(\mathbf{v})\mathbf{v}_i}{\sum_{j=1}^{n} w_j(\mathbf{v})}, \tag{2}$$

$$b_i(\mathbf{v}) = \frac{w_i(\mathbf{v})}{\sum_{j=1}^{n} w_j(\mathbf{v})}, \tag{3}$$

where $w_i(\mathbf{v})$ are the homogeneous coordinates. The above mentioned barycentric coordinates are usually computed in a precomputation step before the deformation.

Image deformation based on GBC is widely used in computer graphics because of the straightforward and real-time computation of barycentric coordinates [Ska08]. However, we have to notice that the deformed image depends on the initial and the deformed source polygons, the used coordinate method [Ani19], and the given triangulation as well.

3 GENERATING THE 2D MESH

As we have mentioned in Section 2, defining a triangulation is a crucial task of image deformation based on GBC. After the user marks the desired initial source polygon of the input image (see Figure 1) by defining its vertices manually using a graphical user interface, we have to create a 2D triangular mesh. To create that, we have to partition a given region into simplices which satisfy different criteria. In our case, the triangulation domain of the region is marked by the initial source polygon, and we use shape and size criteria. The shape criterion is an upper bound $\boldsymbol{B}$ on the circumradius-to-shortest edge length ratio, while the size criterion is an upper bound $\boldsymbol{S}$ on the length of the longest edge of triangles.

We used Shewchuk's algorithm [She02]—which is based on the Delaunay refinement method—to produce 2D meshes for the given domain. The algorithm starts with a constrained Delaunay triangulation and inserts new vertices until it satisfies the criteria.

Definition 3.1. A triangulation T is a *Delaunay triangulation* if there exists a circle C for each edge e of T with the following properties:

- the endpoints of e are on the boundary of C, and
- the circle C does not contain another vertex of T in its interior.

Definition 3.2. Let G be a planar straight-line graph (PSLG). A triangulation T of G is a *constrained triangulation* if it contains all edges of G as a part of the triangulation. The edges are called *constrained edges*.

Definition 3.3. A *constrained Delaunay triangulation* (CDT) [Che89] of G is a *constrained triangulation* of the vertices of G, which is as close to the *Delaunay triangulation* as possible.

As we can see in the definitions above, we can construct a uniform triangulated 2D mesh using the Delaunay refinement algorithm if we define G from the edges of the source polygon.

A non-uniform triangulated 2D mesh can be constructed if we define further segments or vertices as constraints in G from the triangulation domain and set a proper criterion.

The modification of shape and size criteria is an excellent way to increase or decrease the number of vertices of the mesh. By default, $\boldsymbol{B} = \sqrt{2}$, which guarantees that the Delaunay refinement algorithm will terminate, but as we increase it, the resolution

of the triangulation will be changed. In the same way, the adjustment of $\boldsymbol{S}$ will affect on the constructed mesh (see Figure 2).

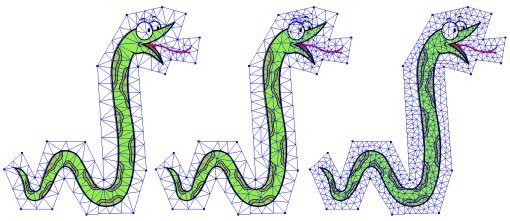

Figure 2: Generated meshes for a given initial source polygon with different parameters. The shape and size criteria are (left) $\boldsymbol{B} = 0.125$; $\boldsymbol{S} = 0.2$, (middle) $\boldsymbol{B} = 0.25$; $\boldsymbol{S} = 0.2$, and (right) $\boldsymbol{B} = 0.125$; $\boldsymbol{S} = 0.05$.

4 OUR NON-UNIFORM TRIANGULATION METHOD FOR CAGE BASED IMAGE DEFORMATION

In the following, we introduce our method, which is able to define an adaptive non-uniform triangulation where the triangles are placed with respect to the input image and the effect of the used coordinate method. Therefore, we can preserve the smoothness of the contour curve of the input image after the deformation, and we can decrease the number of triangles of the triangulation, which results in less computation time. Moreover, the position of the source polygon does not affect the quality of the deformation result.

The inputs of our algorithm are the input shape I and the source polygon P that is often marked by the user manually using a graphical user interface.

4.1 Extracting the contour

After the user defined the source polygon, the input image has to be converted to a binary one, on which a threshold or canny edge detection has to be applied in order to use a contour detection technique, e.g., the method of Suzuki et al. [Suz85]. The input image has to be separable from the background, or it needs to have a coherent black contour, so the contour detection method can work successfully.

In the following part of the paper, we refer to the mentioned contour as $C = \{(x_t, y_t)\}_{t=1}^{N}$, where N is the number of points on the boundary.

4.2 Calculation of curvature

Our algorithm calculates the curvature κ_t of the contour curve of the input shape. The curvature defines the rate of change of the unit tangent vector at a given point, and it can be computed as

$$\kappa(t) = \frac{||\boldsymbol{r}'(t) \times \boldsymbol{r}''(t)||}{||\boldsymbol{r}'(t)^3||}. \tag{4}$$

However, in the discrete world, the contour curve is represented as a chain of segments that are built from a set of points; therefore, we have to associate curvature with vertices. The discrete curvature $\widehat{\kappa}_t$ of the contour curve in vertex $\mathbf{c}_t \in C$ is the change between segments (meeting at $\mathbf{c}_t$) in tangent direction:

$$\widehat{\kappa}_t = \angle((\mathbf{c}_t - (\mathbf{c}_{t-1} - \mathbf{c}_t)), \mathbf{c}_t, \mathbf{c}_{t+1}) = \theta_t, \tag{5}$$

where t is a position in the contour (see Figure 3). The curvature values are higher where the change between segments are high, and it is constant if the chain of segments is almost flat (see Figure 4). Using the curvature values, we can decide in which parts of the curve we have to use a high density of sampled points to follow the shape of the contour curve properly.

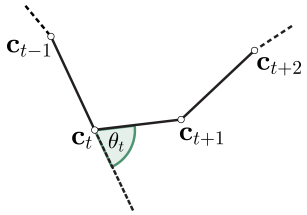

Figure 3: Notations for calculating the discrete curvature.

We notice that our algorithm allows us to use different step sizes for discrete curvature calculation; thus, the noise of the contour curve can be reduced.

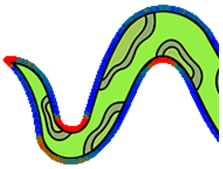

Figure 4: Visualized contour curve curvature. Those parts of the curve where the curvature is high are marked by red.

4.3 Calculation of the effect of the deformation

As we mentioned previously, the deformation results depend on the position of the initial source polygon and the used coordinate method as well (see Figure 5). Therefore, our algorithm calculates the effect $F(\mathbf{c}_t)$ of the nearest vertex of the source polygon P in vertex $\mathbf{c}_t \in C$ by

$$F(\mathbf{c}_t) = \frac{w(\mathbf{c}_t)}{\sum_{k=1}^{n} w_k(\mathbf{c}_t)}. \tag{6}$$

With these values, we can decide which parts of the input image will be deformed better. The effect of deformation is higher in those parts of the input, where the initial source polygon—marked by the user—is close to the image, or the normalized barycentric coordinates values are close to 1.

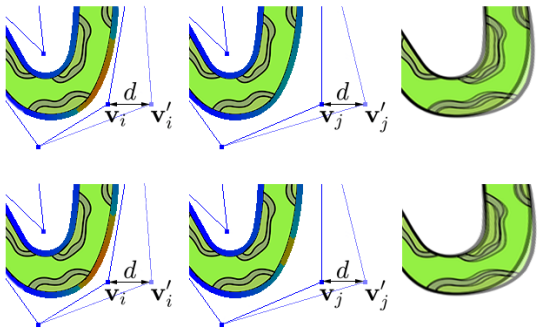

Figure 5: The effect of deformation with different initial source polygons using (top) mean value and (bottom) maximum entropy coordinate methods. The higher the effect of the deformation, the redder the contour. We made deformations with both source polygons, we translated vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ to $\mathbf{v}'_i$ and $\mathbf{v}'_j$ with the same distance d. The comparisons of the deformation results can be seen in the third column. They show us that the deformation depends on the position of the source polygon as well.

4.4 Sampling of the contour curve

An essential step of our algorithm is to define those points of the contour curve, which will be the basis of the non-uniform triangulation. Therefore, we have to sample the contour C with respect to the previously computed $\widehat{\kappa}_t$ and $F(\mathbf{c}_t)$ by

$$r = \begin{cases} r_1 & \text{if } \widehat{\kappa}_t > \Delta\widehat{\kappa} \text{ and } F(\mathbf{c}_t) > \Delta F \\ r_2 & \text{otherwise} \end{cases} \tag{7}$$

$$C' = \{(x_t, y_t) \in C \,|\, t \pmod r = 0\}, \tag{8}$$

where r is the sampling ratio, and $\Delta\widehat{\kappa}$ and ΔF are lower bounds.

Sampling methods for freeform curves [Pag18] are usually used to select sample points from the curve according to some criterion (e.g., curvature, arc length, parameterization, or complexity). Uniform sampling methods for curves are the most popular, but, unfortunately, they are not precise enough in some cases. Other techniques are the adaptive approaches (also called non-uniform sampling), which lead to increased sampled density on those parts of the curve where, e.g., the curvature is high. These techniques give us more efficient approximations.

We use the adaptive technique to determine points to be sampled from the contour curve C with respect to the previously computed curvature and the effect of deformation values (see Figure 6). Our goal is to increase the density of the sampling on those parts of the curve where the curvature and the effect are higher while we want to set the sampling ratio to minimal in other areas. Naturally, other properties can be examined as well.

In our algorithm, we set r_1 to 4, r_2 to 20, ΔF to 0.7, and $\Delta\widehat{\kappa}$ is exactly the average curvature of the contour curve. In that way, we sample every fourth point of the curve if the curvature is greater than the average and the effect of the deformation is greater than 0.7. Otherwise, we sample every twentieth point. We notice that these values highly depend on the resolution of the input image. We worked with full HD images, but if we want to use larger images, we have to increase the values r_1 and r_2.

In addition to all of this, the r_1, r_2, $\Delta\widehat{\kappa}$, and ΔF values in Equation 7 are modifiable freely; thus, the resolution of the approximation can be increased and decreased.

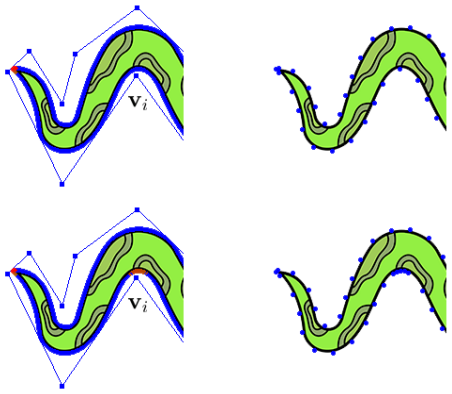

Figure 6: The sampling of the contour curve with the same initial source polygon using (top) mean value and (bottom) metric coordinates. The contour is colored with respect to the curvature and the effect of deformation. We increased the sampled density near to vertex $\mathbf{v}_i$ (bottom) because the metric coordinates has a higher effect on the input image than the mean value.

After the sampling, we have to consider the sampled points as the constraints of a constrained Delaunay triangulation; thus, a non-uniform triangulated 2D mesh can be constructed as we discussed in Section 3. The edges of the generated triangulation follow the contour curve of the input shape. Moreover, in those parts of the triangulation where

the curvature and the effect of the source polygon are higher, there are more triangles, while in other areas, our method minimizes the number of triangles.

4.4.1 Holes in the mesh

Another advantage of the non-uniform triangulation over the uniform that it can handle holes in the generated mesh. The user has the opportunity to define holes—which are simple polygons H_i—in the interior of the initial source polygon. In that case, the algorithm uses the edges of H_i as constraint edges in the constrained Delaunay triangulation.

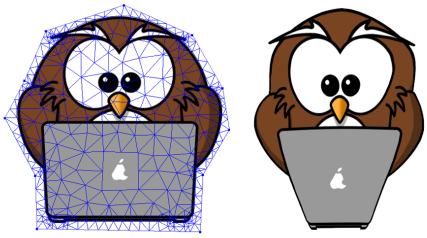

Figure 7: Generated mesh with a hole for an input image and its deformation result.

This solution can be very useful if we want to deform images with logos, texts, or parts which should not be damaged (see Figure 7).

4.5 The step by step process of our non-uniform triangulation algorithm

We can summarize our method in the following steps:

1. Consider the input shape I, the source polygon P, and the hole polygons H_i.
2. Extract the contour $C = \{(x_t, y_t)\}_{t=1}^{N}$ of the input.
3. Calculate the curvature $\widehat{\kappa}_t$ of the contour curve.
4. Calculate the effect $F(\mathbf{c}_t)$ of the nearest vertex of the source polygon P in vertex $\mathbf{c}_t \in C$.
5. Sample the contour curve C respect to $\widehat{\kappa}_t$ and $F(\mathbf{c}_t)$.
6. Generate a 2D mesh obtained from a constraint Delaunay triangulation using C' and H_i.

5 IMPROVING THE QUALITY OF THE DEFORMATION

The image deformation applications based on the barycentric coordinates usually use the OpenGL library to render the deformation. Therefore, as we discussed in Section 2, we have to triangulate the initial source polygon, then we have to apply the input image to it as a texture. In order to draw the texture, OpenGL executes a sampling operation, when the texture coordinates of the vertices are mapped to the texture image. The result of the sampling are texels, which are pixels in the texture image. These texels often contain color information; thus, the final color of the corresponding pixel can be defined. However, it can happen that a computed texture coordinate will not match a texel exactly. In these cases, OpenGL has to figure out which texel corresponds to the texture coordinate. The method, when OpenGL decides the final texel value, is called texture filtering. During the deformation, we usually do many transformations with the vertices of the initial source polygon; therefore, we have to choose well the texture filtering algorithm. The most used filtering methods are the `GL_NEAREST` and `GL_LINEAR`, but these cannot produce good image quality in most cases, e.g., if we stretch the image to be deformed. The `GL_NEAREST` interpolation selects the closest pixel to the texture coordinates; therefore, the resulting images will be blocky. The `GL_LINEAR` method interpolates the closest four pixels, and it creates smoother images than the previous interpolation technique, but staircase effect may appear at the edges of the image.

Using the OpenGL Shading Language (GLSL), we have the opportunity to write shaders on the GPU, where we can use our own filtering methods. In order to increase the quality of the deformation results, we implemented a generalized bicubic interpolation method [Pra13] by the following equation:

$$F(p',q') = \sum_{m=-1}^{2} \sum_{n=-1}^{2} F(p+m, q+n) \cdot R_C\{(m-a)\} R_C\{-(n-b)\}, \tag{9}$$

where the pixel $F(p,q)$ is the nearest to the pixel to be interpolated, a and b are distances between the previously mentioned pixels, while $R_C(x)$ is a bicubic interpolation function such as triangle, cubic B-spline, or Catmull-Rom interpolation.

While the `GL_LINEAR` method uses the nearest four pixels to define the color of the intermediate pixel, the bicubic interpolation uses the nearest sixteen pixels (see Figure 8). Using a texture filtering method based on a bicubic interpolation function, the quality of the deformation can be further improved, as we can see in Figure 9.

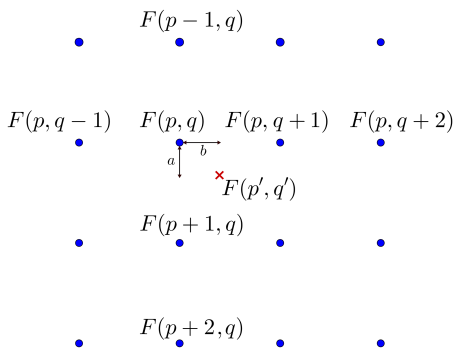

Figure 8: Notations for the bicubic interpolation.

Figure 9: From left to right, top to bottom: deformation result is rendered by `GL_NEAREST`, `GL_LINEAR`, triangle, B-spline, bell, and Catmull-Rom texture filtering methods.

6 RESULTS

6.1 Implementation

We implemented the prototype of the uniform and our non-uniform triangulation algorithms in C++ on an Intel i7 6700K CPU at 4.0 GHz with 16 GB memory. In the case of non-uniform triangulation, to extract the contour of the input, we used the method of Suzuki et al. [Suz85], which is implemented in the OpenCV library [Bra00]. In order to produce a 2D mesh for a source polygon using the extracted contour points, we have to build a constrained Delaunay triangulation then we have to use the Delaunay refinement algorithm on it. In the case of uniform triangulation, the mentioned algorithms can be useful as well. These methods can be found in the CGAL library [The19]. The implementation of the coordinate methods was based on the CGAL library and the Ph.D. thesis of Anisimov [Ani17b].

6.2 Validation and comparison

We tried the methods on many input images, and we compared the deformation results using uniform and non-uniform triangulations (see Figure 10). We used different coordinate methods for the deformations: mean value, maximum entropy, metric, Poisson, and blended coordinates. We can say that the uniform and the non-uniform triangulations works well with every coordinate method.

The introduced non-uniform triangulation method only takes care of the outer contour of the input images. However, in those situations when we want to deform, e.g., cartoon figures—which interiors are not so detailed—, images with big homogeneous regions, it produces smooth deformation results. Furthermore, it can preserve the boundary of the image after the deformation. Moreover, if the graphical resources of the used platform or device are limited, or we want to save computation cost by reducing the number of triangles, the non-uniform method is usable as well.

Although, if we can use a GPU implementation of the cage based image deformation methods, and can increase the number of the triangles up to several thousands, the uniform triangulation can produce smooth enough deformation results as well.

To validate the algorithms, we also created a pixel-based deformation (see the middle column of Figure 10) for the inputs. It can be used as a reference resulting image because we computed the barycentric coordinates of all the pixels of the input image in the precomputation step, then we computed new positions of them in the deformation phase. In the case of large deformations, it can produce holes because there are regions which are not covered by deformed pixels, but using a filtering algorithm, the result will be fully contiguous without holes.

We used the same deformation polygon P' to get three resulting images based on the uniform triangulation, the non-uniform triangulation, and the pixel-based one. For the comparison, we measured the computation times of the triangulations and SSIM (Structural Similarity Index) [Wan04] values. The SSIM value is frequently used to define the similarity of two images $\mathbf{X}$ and $\mathbf{Y}$ with the following formula:

$$\mathrm{SSIM}(\mathbf{x},\mathbf{y}) = [l(\mathbf{x},\mathbf{y})]^{\alpha} \cdot [c(\mathbf{x},\mathbf{y})]^{\beta} \cdot [s(\mathbf{x},\mathbf{y})]^{\gamma}, \quad (10)$$

where $\mathbf{x}$ and $\mathbf{y}$ are square windows of $\mathbf{X}$ and $\mathbf{Y}$ (located at the same spatial position). The $l(\mathbf{x},\mathbf{y})$, $c(\mathbf{x},\mathbf{y})$, and $s(\mathbf{x},\mathbf{y})$ are luminance, contrast, and structure comparison functions, respectively. $\alpha > 0$, $\beta > 0$, and $\gamma > 0$ are parameters which define the relative importance of the three comparison functions. The SSIM value can be between 0 and 1. The value 1 means that the two images are similar, while the value 0 means that they are very different. Table 1 summarizes the computation times of the triangulation on CPU and the SSIM values between the two triangulations on different input images and the number of triangles. In the case of the *Snake* input image, the two triangulations have almost the same number of triangles, and the same computation times, while the non-uniform solution increases the SSIM value. In the

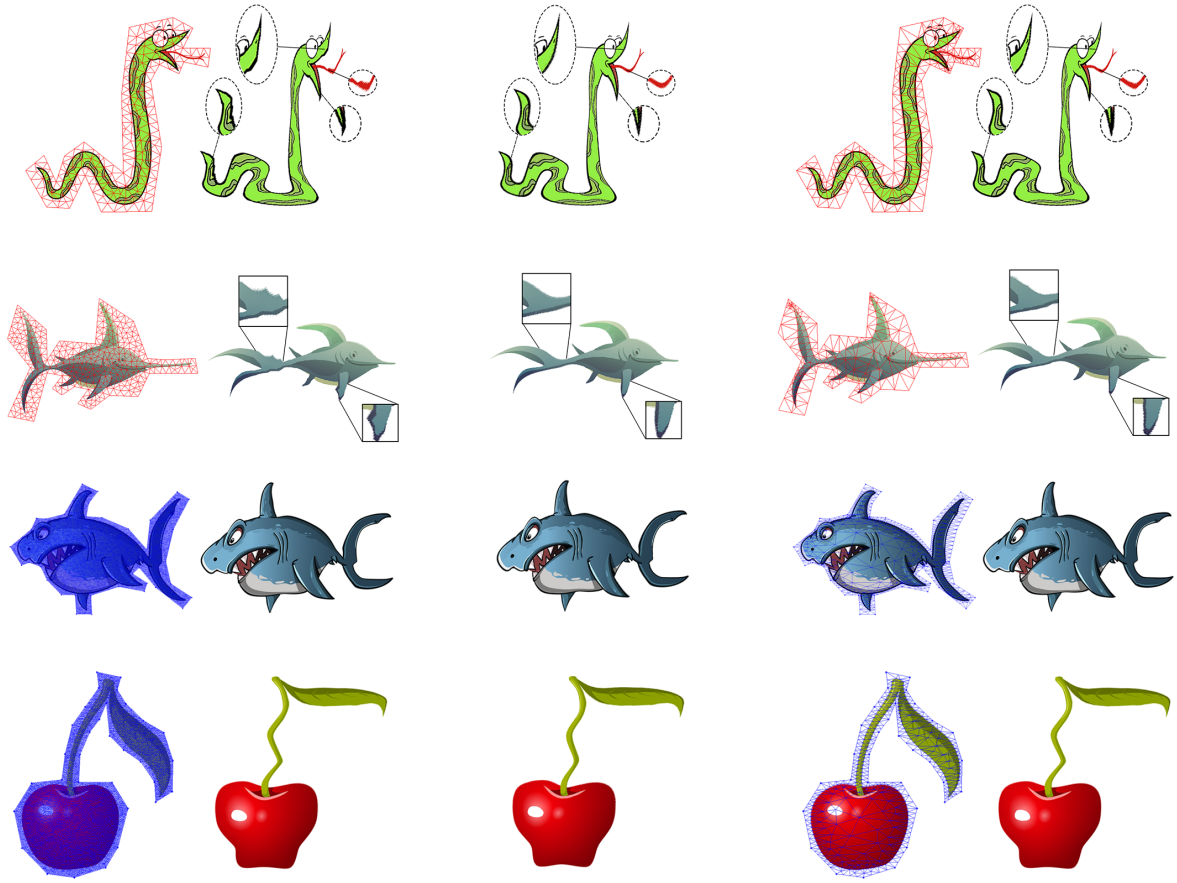

Figure 10: Input images with uniform triangulation on the left column and with non-uniform triangulation on the right column, and their deformation results using mean value coordinates, while the reference resulting images are in the middle.

	Snake		Swordfish		Shark		Cherry	
	u	nu	u	nu	u	nu	u	nu
Triangles	340	335	638	360	29926	1342	16185	715
SSIM	0.956	0.959	0.978	0.979	0.939	0.938	0.984	0.984
Computation time (s) of the triangulation	0.001	0.001	0.003	0.001	1.898	0.008	0.563	0.003

Table 1: The comparison of the uniform (u) and the non-uniform (nu) triangulations.

case of the *Swordfish*, the non-uniform method uses much fewer triangles than the uniform triangulation, so it decreases the computation time and increases the SSIM value. In the case of the *Shark* and the *Cherry*, the uniform method uses ≈30k and ≈16k triangles; thus, the computation times are increased, while the two types of triangulations have almost the same SSIM value. More comparison figures and values can be seen in our supplementary material.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we have shown how the different triangulation techniques can be used for cage based image deformation using the generalized barycentric coordinates. Our non-uniform triangulation method was introduced in detail as well. The algorithm takes into consideration the position of the source polygon and the used coordinate method; moreover, it can handle holes in the interior of the polygon. The main difference from the earlier systems that the edges of the non-uniform triangulation follow the contour of the input shape, therefore it can preserve the smoothness of the curves of the input. We compared the uniform and the non-uniform triangulation methods as well. The non-uniform one can decrease the computation time by minimizing the number of triangles of the triangulation on those parts where the effect of the defor-

mation is lower. The uniform triangulation method can produce smooth deformation results by an increased number of triangles. The quality of the deformation can be increased by using different texture filtering methods. Based on our results, we can say that the non-uniform triangulation is more efficient in cases when we work with fewer triangles, and it can produce more accurate deformation results than the methods using the uniform one. Furthermore, we have to notice that the deformation depends on the deformed source polygon as well because the barycentric coordinates are computed with respect to the vertices of the source polygon, which are not changed after the precomputation.

In our non-uniform triangulation, the vertices on the boundary of the hole polygon do not change positions in the deformation phase, and it can lead to deformation errors around the hole in some cases. In order to solve this problem, further investigation is required in the future.

8 ACKNOWLEDGEMENT

This work was supported by the construction EFOP-3.6.3-VEKOP-16-2017-00002. The project was supported by the European Union, co-financed by the European Social Fund.

9 REFERENCES

[Ani16] Anisimov, D., Deng, C., and Hormann, K. Subdividing Barycentric Coordinates, Comput. Aided Geom. Des. 43 pp.172–185, 2016.

[Ani17a] Anisimov, D., Panozzo, D., and Hormann, K. Blended barycentric coordinates, Comput. Aided Geom. Des. 52–53 pp.205–216, 2017.

[Ani17b] Anisimov, D. Analysis and New Constructions of Generalized Barycentric Coordinates in 2D, Ph.D. thesis, Universitá della Svizzera italiana, 2017.

[Ani19] Anisimov, D., Hormann, K., and Schneider, T. Behaviour of exponential three-point coordinates at the vertices of convex polygons, J. Comput. Appl. Math. 350 pp.114–129, 2019.

[Bra00] Bradski, G. The OpenCV Library, Dr. Dobb's Journal of Software Tools 2000.

[Cas18] Casti, S., et al. CageLab: an Interactive Tool for Cage-Based Deformations. In *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference 2018*, pp.65–74. Eurographics, 2018.

[Che89] Chew, L. P. Constrained Delaunay Triangulations, Algorithmica 4 pp.97–108, 1989.

[Flo15] Floater, M. S. Generalized barycentric coordinates and applications, Acta Numer. 24 pp.161–214, 2015.

[Hor06] Hormann, K., and Floater, M. S. Mean Value Coordinates for Arbitrary Planar Polygons, ACM Trans. Graph. 25 No.4 pp.1424–1441, 2006.

[Hor17] Hormann, K., and Sukumar, N. (eds.). Generalized Barycentric Coordinates in Computer Graphics and Computational Mechanics. CRC Press, 2017.

[Jos07] Joshi, P., et al. Harmonic Coordinates for Character Articulation, ACM Trans. Graph. 26 No.3 pp.71:1–71:9, 2007.

[Ju05] Ju, T., Schaefer, S., and Warren, J. Mean Value Coordinates for Closed Triangular Meshes, ACM Trans. Graph. 24 No.3 pp.561–566, 2005.

[Nie13] Nieto, J. R., and Susín, A. Cage Based Deformations: A Survey. In González Hidalgo, M., Mir Torres, A., and Varona Gómez, J. (eds.) *Deformation Models: Tracking, Animation and Applications*, pp.75–99. Springer, Dordrecht, 2013.

[Pag18] Pagani, L., and Scott, P. J. Curvature based sampling of curves and surfaces, Comput. Aided Geom. Des. 59 pp.32–48, 2018.

[Pra13] Pratt, W. K. Introduction to Digital Image Processing, CRC Press, 2013.

[She02] Shewchuk, J. R. Delaunay refinement algorithms for triangular mesh generation, Comput. Geom. 22 No.1-3 pp.21–74, 2002.

[Ska08] Skala, V. Barycentric coordinates computation in homogeneous coordinates, Comput. & Graph. 32 No.1 pp.120–127, 2008.

[Suz85] Suzuki, S., and Abe, K. Topological Structural Analysis of Digitized Binary Images by Border Following, Comput. Vis. Graph. Image Process. 30 No.1 pp.32–46 1985.

[The19] The CGAL Project. CGAL User and Reference Manual. CGAL Editorial Board, 4.14 edition, 2019.

[Wan04] Wang, Z., et al. Image Quality Assessment: From Error Visibility to Structural Similarity, IEEE Trans. Image Process. 13 No.4 pp.600–612, 2004.

[Web09] Weber, O., Ben-Chen, M., and Gotsman, C. Complex Barycentric Coordinates with Applications to Planar Shape Deformation. Comput. Graph. Forum 28 No.2 pp.587–597, 2009.

[Zha14] Zhang, J., et al. Local Barycentric Coordinates, ACM Trans. Graph. 33 No.6 Article 188. 2014.