Multicage Image Deformation On GPU

Weiliang Meng[1*] Xiaopeng Zhang[1†] Weiming Dong[1‡] Jean-Claude Paul[2§]

[1]LIAMA - NLPR, Institute of Automation, CAS, China
[2]INRIA, France

Figure 1: *Image deformation based on the multicage using GPU in realtime. The multicage has* 4 *polygons with* 68 *vertices(ignoring the image bounding box), two of which are embedded in one. (a) is the original image, we set the ROI by the multicage in (b) and deform the image in (c). (d) is as the same as (c) with the multicage removed for better observation. Region* 1 *and* 2 *marked in blue in (b) are enclosed by polygons which will preserve the features when the peripheral polygons deformed. Region* 3 *and* 4 *marked in blue in (c) are the deformed results of region* 1 *and* 2*, with tiny automatical adjustments of the feature by our algorithm. The " two lotus" image size is* $1024 * 768$.

Abstract

As a linear blending method, cage-based deformation is widely used in various applications of image and geometry processing. In most cases especially in the interactive mode, deformation based on embedded cages does not work well as some of the coefficients are not continual and make the deformation discontinuous, which means existing "spring up" phenomenon. However, it's common for us to deform the ROI(Region of Interest) while keeping local part untouched or with only small adjustments. In this paper, we design a scheme to solve the above problem. A multicage can be generated manually or automatically, and the image deformation can be adjusted intelligently according to the local cage shape to preserve important details. On the other hand, we don't need to care about the pixels' position relative to the multicage. All the pixels go through the same process, and this will save a lot of time. We also design a packing method for cage coordinates to pack all the necessary coefficents into one texture. A vertex shader can be used to accelerate the deformation process, leading to realtime deformation even for large images.

CR Categories: Numerical Analysis [G.1.1]: Interpolation—Interpolation formulas; Information Interfaces and Presentation [H.5.1]: Multimedia Information Systems—Animations; Computer Graphics [I.3.3]: Picture/Image Generation—Display algorithms; Computer Applications [J.6]: computer-aided engineering—Computer-aided design.

Keywords: image deformation, GPU, cage, coordinates

1 Introduction

Image deformation has a lot of methods to be used, in which linear blending and corresponding variants are the most practical as the high speed deformation. For a typical linear blending method, the point on the object is transformed by a linear combination of affine transformations. The user only needs to construct a few handles and then manipulate them to control the shape. Free-form deformation belongs to linear blending methods, but the regular structure restriction makes the control of concave objects complicated. Although skeleton-based deformations can provide natural control for object with rigid limbs, they are less convenient for flexible regions. Cage-based interactive space deformation is booming as it can deform a significant portion of the object, leading to easy bulging and thinning of ROI(Region Of Interest). The cage can be constructed interactively or manually in advance, and the object vertices are represented as linearly combinations of cage vertices (may be also with edge or face normals). The weights of the combination can be computed before the deformation and associate with the object vertices, which can be called 'binding' process. During the *pose time*, i.e. the period that a user manipulates the cage vertices to deform the object, the weights are fixed and using the cage vertices' position to generate the deformed object.

Unfortunately, most cage-based method doesn't deal with the embedding cases very well, especially for the interactive deformation. Focusing on the situation in 2D case as shown in Fig.1, where the region is split by some embedded polygons(i.e. 2D cage) which we called 'multicage', when we move the cage vertices interactively,

*e-mail:weiliang.meng@ia.ac.cn
†e-mail:xiaopeng.zhang@ia.ac.cn
‡e-mail:weiming.dong@ia.ac.cn
§email:paul@inria.fr

VRCAI 2011, Hong Kong, China, December 11 – 12, 2011.
© 2011 ACM 978-1-4503-1060-4/11/0012 $10.00

we hope all the region will deform smoothly as long as the moving is reasonable(no overmuch changes that leading to overlap or cage intersection). Every isolated part should not go beyond its boundary to make sure the deformations are cage-aware, or else the cage can't have a leading meaning for the deformation. For ideal state, the cage should also be generated interactively to satisfy the special needs of the users.

In this paper, we study the case and propose a method to keep each isolated region details smoothly during the pose time. We also provide a packing method for the weights in order to warping the large image in realtime on GPU. Previous acceleration methods don't pack the weights, leading to many textures to be set during the preprocess. As the number of textures has limits for most display card, this confines the cage vertices' numbers. Using our method, the weights are packed into a regular single texture and can be addressed quickly during the rendering time. The multicage can have hundreds of vertices for medium sized images, and dozens of vertices for large images, which mainly depend on the memory.

Our contribution can be listed as follows:

- A novel scheme for warping images based on embedded cages. The deformation is cage-aware conforming to the embedded cages. All the pixels will go through the same process, having no bearing on their positions to the multicage.
- A new packing method for cage coordinates to deform images on GPU in real time.

Our technique can be applied to intelligent image warping and animation. The content of the image will be adjusted intuitively and automatically. GPU is used for generating realtime intuitive cage-based deformation. Based on our method, any possible image warping result can be easily achieved by using different multicages on the deformed image repeatedly.

2 Related work

Plenty of methods can compute high-quality shape-preserving deformations based on the selected handles, having the form of points [Bookstein 1989], lines [Beier and Neely 1992] or bones[Weber et al. 2007], and polygon grids [MacCracken and Joy 1996]. Users modify the positions and orientations of handles interactively to achieve an intuitive deformation. The handles can be on the surface of the targets [Igarashi et al. 2005; Botsch et al. 2006; Sorkine and Alexa 2007; Botsch and Sorkine 2008], or can be extended to other off-surface handles[Botsch et al. 2007]. The deformation is heavily rely on optimization at pose time, and most mentioned methods above are non-linear which recede the efficiency and make them too slow for deforming high-resolution images or objects.

Using a weighted blend of handle transformation, the computation can be fast at pose time. Schaefer et al. [2006] use linear combination of Moving Least Squares (MLS) for image warping, and the deformation time is linear proportional to the number of sample grid vertices. Weng et al. [2008] deform images on GPU for real-time performance based on the sketch, whose selection may be troublesome for users.

Cage-based methods can also be seen as a handle-deformation technique, in which the handles are the cage vertices. This is essentially a kind of linear blend skinning methods[Magnenat-Thalmann et al. 1988], where the handle(cage vertex) transformations are restricted to be translations. The core for cage-based method is how to choose the weights in order to make the deformation smooth. Many feasible ways have been proposed including Mean Value Coordinates(MVC)[Floater 2003; Hormann and Floater 2006; Ju et al. 2005; Floater et al. 2005; Lipman et al. 2007], Harmonic Coordinates(HC) [Derose and Meyer 2006; Joshi et al. 2007], Green Coordinates(GC) [Lipman et al. 2008], and complex barycentric coordinates(CBC) and its variants [Weber et al. 2009; Ben-Chen et al. 2009]. Jacobson et al. [2011] develop linear blending weights that produce smooth and intuitive deformations using many kinds of handles including points, bones and cages. However, all the above methods don't display interactive image deformation for the embedded cages case in their work, as this is a troublesome problem, which will be solved by this paper.

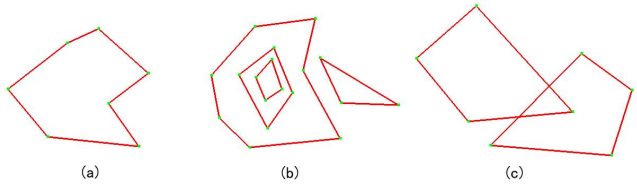

Figure 2: *2D Multicage demonstration. (a) a single polygon, (b) 4 polygons with embedding and neighboring, (c) 2 polygons with intersect edges. (a) and (b) are multicages while (c) is not.*

Once the weights for cages are obtained, they will keep fixed during the whole deformation. Real time deformation can be achieved based on GPU, which is used for general purpose computation[Luebke et al. 2004; Göddeke 2005]. Meng et al.[2009] design a framework for the implementation of cage-based image deformation method on GPU. Their method are limited by the number of cage vertices because of needing many unpacked textures.

Our work solves the embedded cages image deformation, and remove the limited number for cage vertices on GPU. Section 3 gives the concrete process for embedded cage image deformation, and section 4 shows the packing algorithm.

3 Multicage Deformation

In this section, we first give the definition of 'multicage' for the image, and then show the pipeline to compute the coefficients of the pixels relative to the multicage.

3.1 Definition of Multicage in 2D

A cage is a low polygon-count polyhedron that has a similar shape to the enclosed object. The genus of the cage can be nonzero, meaning that the cage can have a "hole". Multi-cage is a series of independent closed polygons in 2D with disjoint edges, i.e. the edges of any two cages have no intersection. The polygons may be embedded each other, or be neighbors, and can be concave(Fig.2). The main difference between the cage and the multicage is that many cages can construct a multicage, and we treat each closed polygons(not the cage) independently rather than as a whole entity.

Most cage-based image deformation use only one cage to deform. This is practical for the images with simple background. During the deformation, contents in the cage are only affected by the enclosed cage. However, the following facts during the image deformation process can't be ignored: all the pixels may have special meaning with each other in the image especially for those with complicated contents, therefore deforming any region of the image will affect all the other pixels, some of which may have merely suffered undetectable changes as the distances from the ROI to the pixels are far. On the other hand, there are some special regions which we may want to keep untouched or with only a little modification despite

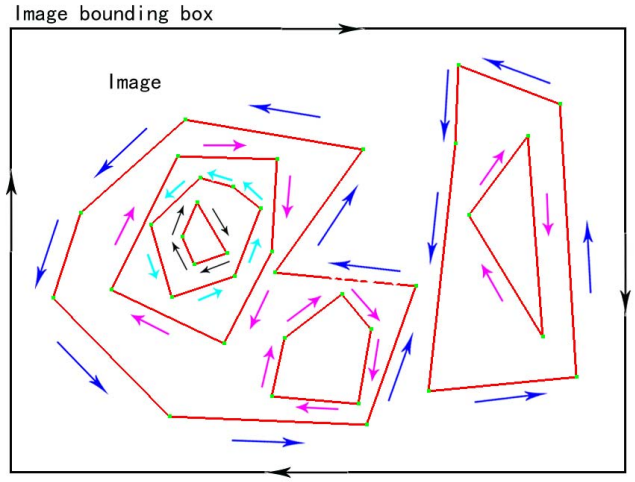

Figure 3: *The direction of polygons in the multicage.*

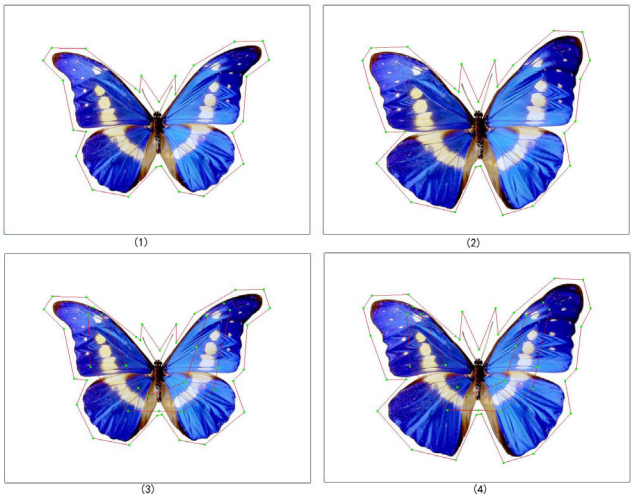

Figure 4: *"Butterfly" deformation comparison. (1) and (2) are the original image and result image repectively using only one polygon for deformation, while (3) and (4) using the multicage. The exterior polygon in (3) is the same as in (1) and deformed to the same state as shown in (2) and (4). We can see that the region in the interior polygon are kept well in (4). The image size is* $800 * 600$.

the fact that the deformed region is close, as these special regions have some features that we want to keep.

The situation mentioned above can be dealt with based on the multicage which is generated interactively, and no explicit deformed information needs to be prescribed by the user. The generation process for a multicage is as follows: for embedded polygons, the outermost layer should be anti-clockwise. The second outermost layer should be clockwise, the third should be anti-clockwise, and so on(Fig.3). This pattern guarantees the signs of the cage coefficients for the simply connected regions, which in turn makes the deformation smooth, or else the "spring up" will appear during the deformation.

3.2 Computing the coefficients of multicage vertices

In order to deforming the image, we transformed the image into a mesh, with each pixel represents one mesh vertices, and four square pixels adjacent with each other are connected as two triangles. During the warping, the vertices' colors are fixed as the originals, with only positions changed. The color of each face is interpolated using the vertices' colors and the process is achieved automatically by hardware.

For smooth deformation, each pixel will be represented as a linear combination of the multicage vertices. For a multicage with m vertices, the weights ω_j for one pixel $\mathbf{p}$ must satisfy the following equation:

$$argmin_{\omega_j, j=1,\cdots,m} \sum_{j=1}^{m} \int_{\Omega} \|\Delta\omega_j\|^2 dV \tag{1}$$

subject to:

$$\omega_j|_{H_k} = \delta_{jk} \tag{2}$$

$$\omega_j|_F \ is \ linear \ \ \forall F \in F_C \tag{3}$$

$$\sum_{j=1}^{m} \omega_j(\mathbf{p}) = 1 \forall \mathbf{p} \in \Omega \tag{4}$$

where F_C is the set of all multicage faces (i.e. edges of the polygons in 2D case). H_k is the $k-$th handle(multicage vertex), δ_{jk} is Kronecker's delta, and Ω denotes the domain enclosed by the given shape(i.e. the image plane in the 2D case).

The mesh vertices' position can be linear combination of the multicage vertices' position, and the coefficients can be computed in various ways [Floater 2003; Ju et al. 2005; Joshi et al. 2007; Lipman et al. 2007; Lipman et al. 2008; Weber et al. 2009]. We use the concept "cage coordinates"[Meng et al. 2009] or "CC" for short to represent all the available coefficients for the cage-based deformation.

Once the multicage is given, traditional methods will try to classify the pixels into three types for CC computation: **IN_CAGE**, **ON_CAGE**, and **OUT_CAGE**. Polygon scan conversion algorithm can be used for the classification, and the process is more complicated in the embedded cases.

However, for those cage coordinates that are infinitely differentiable in and out of cages, the position of the pixel relative to the multicage don't need to be recognized. Considering the case that a point is in the plane with a multicage, different multicage vertices will have different impacts depending on the shape and position of the polygons. As long as their impacts are all smooth, the combination of them will still be smooth. What we need to do is to make sure that the sum of coefficients should satisfy Equ.4. Furthermore, in order to keep the image to be rectangle, we use image bounding box as constraints as in [Meng et al. 2009], which is a clockwise polygon belonging to the multicage(the black line in Fig.3).

The description of the computation flow is given by Algorithm 1. The computation of coefficients of x relative to p depends on the different choices for weights. The weights should satisfy the condition that C^1 at the multicage vertices and C^∞ everywhere else. Not all the cage-based methods satisfy the conditions: MVC and HC are good while GC and CBC are not, as when using GC or CBC for deformation, the ROI may be out of the polygons which indicates that they are not continuous when across cage edges. Readers can refer [Hormann and Floater 2006; Derose and Meyer 2006; Lipman et al. 2008; Weber et al. 2009] for the above coordinates respectively to get the concrete computation process. When all the weights are

Algorithm 1 Computing CC base on the multicage

Associate each pixel x in the image with an array CC_x of length c.
/* c is the number of multicage vertices*/.
$base = 0$;
for all polygon p with i vertices in the multicage **do**
  for all pixel x in the image **do**
    Compute the coefficients of x relative to p to fill $CC_x[base, base + i - 1]$.
  end for
  $base = base + i$;
end for
for all pixel x in the image **do**
  Normalize CC_x to make $\sum_{j=0}^{c} CC_x[j] = 1$.
end for

obtained, we can use them for deformation.

Using the scheme we given above, the region in the closed cage(may be with non-zero genus) will not change out of the cage when only the peripheral polygons deformed, which in turn keeps local details well(Fig.4), as long as the peripheral deformed polygons don't suffer overlap warping which is unreasonable. Moreover, most cage-based methods require the coefficients should be non-negative:

$$0 \leq \omega_j(\mathbf{p}) \leq 1, j = 1, \cdots, m, \forall \mathbf{p} \in \Omega \tag{5}$$

as negative weights lead to unintuitive handle influences. In our case, we found negative weights may lead to meaningful results. In Fig.5, the outer "big" polygon in Fig.5.(d) is the same as in Fig.5.(a). In Fig.5.(b) and Fig.5.(e), the "big" polygon is deformed to the same position, mainly in the left part. The coefficients of the pixels in the "small" polygon is negative relative to the "big" polygon of the multicage in Fig.5.(d), and the deformation will move the pixel to the opposite position of the "large" polygon, as shown in the blue circle. Fig.5.(c) and Fig.5.(f) are the deformed results which have removed the deformed polygons for better observation. We can see that the petal in the "small" polygon are preserved better in multicage in Fig.5.(f) as our expection than the ones in Fig.5.(c) which using only one cage for deformation.

4 Cage Coordinates Packing

The coefficients of the cage for each pixels keep fixed during the whole deformation process once they are computed based on the initial multicage. In order to accelerate the deformation, we can send them to GPU as textures. Currently, float type are extendedly supported and no special treat need to be made even though the coefficients are less than 0. [Meng et al. 2009] gives a workflow for acceleration on GPU, but need too many textures. If the number of cage vertices is more than 64, then we need 16 or more textures which may not be supported by most display cards as this exceeds the limit of the hardware.

Our deformation acceleration is based on the framework of [Meng et al. 2009], with packing the CC into one texture to remove the confine. In order to make the texture as square as possible to save the graphic memory, the texture size is defined as follows:

For an image I with $m \times n (m > n)$ pixels, if the number of multicage vertices is c, then the width w of the texture is:

$$w = \lceil (\sqrt{\lceil c/4 \rceil * \lfloor m/n \rfloor}) \rceil * m \tag{6}$$

Figure 6: *The texture size is* $5120 * 3072$. *The last* 2 *blocks are grey as they are empty. Note that the* 0 *value corresponding to the "grey" color after the linear transformation.*

and the height h of the texture is:

$$h = \lceil (\lceil c/4 \rceil / \sqrt{\lceil c/4 \rceil * \lfloor m/n \rfloor}) \rceil * n \tag{7}$$

Here,$\lceil . \rceil$ and $\lfloor . \rfloor$ are the *ceil* function and the $floor$ function respectively. The upper two equations are quite suitable for those images with $m > n$. If $m < n$, we just exchange m and n, also w and h correspondingly. In a nutshell, the texture is composed of matrix blocks, and each block has the same size with the original image size.

After we transmit this texture into GPU, the addressing in the texture for a pixel of the original image have the following rules: for a pixel with (x, y) in the image with $m * n$ size, the coefficients on the texture are the all the RGBA values with coordinates$(x + k * m, y + l * n)$, in which k, l are integers,$k = 0, 1, \cdots, l = 0, 1, \cdots$, and $x + k * m <= w$, $y + l * n <= h$, w and h are give in Equ. (6) and Equ. (7). Each channel represents one coefficient value which is corresponding to a multicage vertex. For the reason that some values of the coefficients may be less than 0, we can't display the packing texture directly by image. But as all the coefficients are between -1 and 1, we can make a simply linear transformation to generate the image for intuitive observation.

Fig.6 shows the packed cage coordinates texture. Using the above packing method, the last few blocks may be empty(i.e. all the RGBA channel of the blocks are zero), but this will have no influence for the computation on GPU, as GPU will not address this part during the deformation computation.

As the multicage is generated interactively, the vertex shader program will be generated automatically based on the multicage in the subsequent process before the deformation. Different multicages define different vertex shader programs, and the vertex shader program will not be changed unless a new multicage is used.

5 Implementation

All the tests are on the PC which has an Intel Core i5 $2.67GHz$ CPU, together with an $nVidiaGTX470$ GPU and 4G of RAM.

The generation of multicage is mainly based on the user's selection, which allows more flexible control for the deformation. Using the multicage and our algorithm, we can finish some meaningful image warping process(Fig.7). We can keep the local details while deforming the peripheral region(Fig.8), or we can make the local deformation affects less area of the peripheral region(Fig.9).

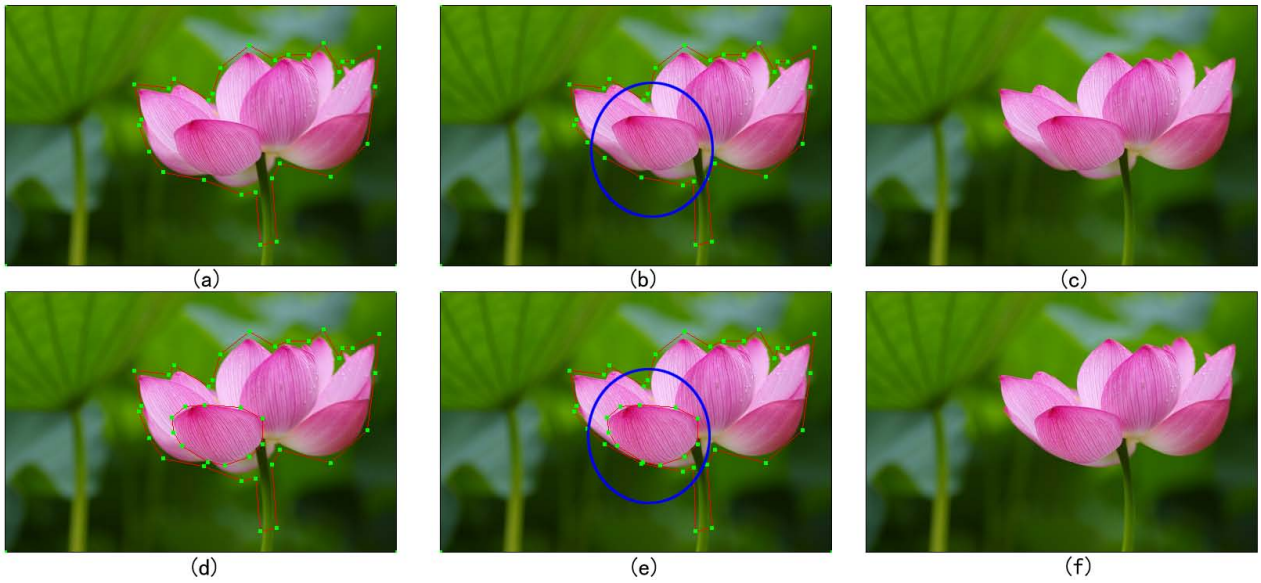

Figure 5: *(a), (b) and (c) use one cage for deformation, while (d), (e) and (f) use a multicage for deformation. (a) and (d) are the initial state. The region in the blue circle is the mainly deformed part which can be detected. The petal in the blue circle are preserved better in (f) than in (c). The image size is* $500 * 333$.

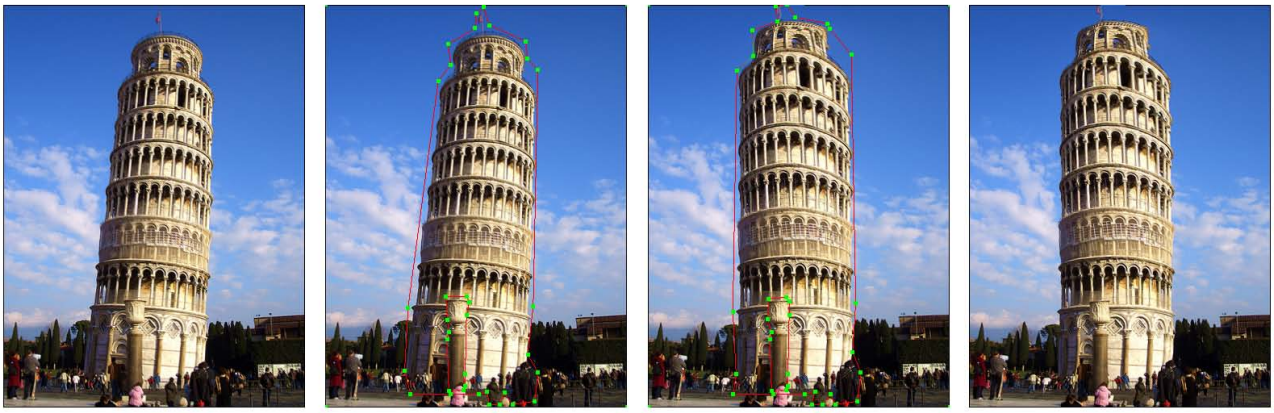

Figure 7: *Rectified "Pisa" tower. The left is the original and the right one is the result. The middle two images show the selected multicage and its deformation respectively. Note that the lamp post before the tower is kept well, which is still as straight as the original. The image size is* $701 * 525$.

Table 1: *Statistics of our tests.*

Image	Image size	Multicage vertex number	Use GPU	
			Preprocessing time	Updating time
two lotus	$1024 * 768$	68	$9.034s$	less than $0.001s$
butterfly	$800 * 600$	44	$3.478s$	less than $0.001s$
lotus	$500 * 333$	40	$1.147s$	less than $0.001s$
tower	$375 * 500$	35	$1.135s$	less than $0.001s$
head	$701 * 525$	29	$1.465s$	less than $0.001s$
heart-reflector	$1000 * 701$	41	$3.864s$	less than $0.001s$

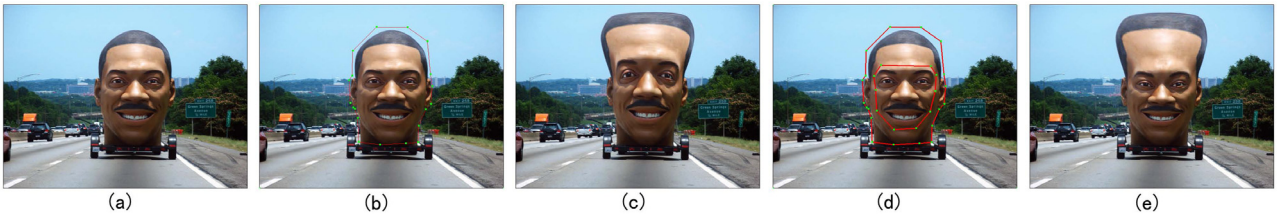

Figure 8: *Deform the "head" on a car. (a) is the original, (b) and (c) show a single cage position and its deformed result based on the cage. Here we don't show the final cage for better observation. A new polygon is added in (d) to construct a multicage, compared to (b). Now we deform the head by moving the 'big polygon' to the same position as in (c), we get the result as (e). We can see that the face is kept better after we use the multicage. The image size is* $701 * 525$.

Table 1 shows the statistics of our tests. As different cage coordinates will lead to the different computation time of CC, we don't give statistics time for CC computation. From the table, we can see that the time for preprocessing on GPU is proportional to the vertices numbers of the multicage and the image size, but this will not exceed 10 seconds in our examples. The interactive deformation is real time after the preprocessing.

For the packing method, we found that [Meng et al. 2009] method can't work for the cases when the number of multicage vertices is over 60, which means the image 'two lotus' deformation can't be executed as it has 68 multicage vertices. The limit number is dependent on the display card, and may be smaller for some low performance ones, while our GPU acceleration method doesn't have such limit based on our packing pattern. We test the case with hundreds of multicage vertices for deforming the image 'lotus', and our method can still work normally.

6 Conclusion and Future Work

Aiming to solve the embedded cage image deformation that can't be properly dealt with by the previous methods, we successfully introduce a new scheme that using multicage cage which is generated interactively to deform images on GPU in real time. A new packing method for cage coordinates is designed to allow more multicage vertices to be used for image deformation on GPU. The experiments verify the effects of our algorithm, which means image deformation based on cages can generate more available results than previous methods.

On the other hand, the cage generation is still a little bothering for dozens of clicking. We should generate the multicage based on image segmentation or edge detection, and automatically simplify the rough boundaries as the multicage. We can simply adjust the multicage according to our needs, and this will improve user experience for the image deformation.

We can set multicage vertices' moving function to create animation as in [Meng et al. 2009](see the accompany video), and in the future, we should seek for the technique that sets the multicage's moving automatically according to the feature extracted from the image or the positions of the embedded polygons, leading to more naturally image deforming results and less setting work. We should also look for how to use the multicage deformation as constraints in image resizing or other image processing applications.

Acknowledgements

We thank the following Flickr (http://www.flickr.com/) members and other websites for making their images available through creative common rights:seri* (lotus), Hayley Grimes (head), Stuck in Customs (heart-reflector), http://fdsysl.5d6d.com(two lotus), http://www.hncts.cn(tower) and http://www.sucai.com(butterfly).

This work is supported by National Natural Science Foundation of China($No.$60872120, 60902078, 61172104), Beijing Natural Science Foundation(Content-Aware Image Synthesis and Its Applications, $No.$4112061), French System@tic Paris-Region(CSDL Project) and ANR-NSFC($No.$60911130368).

References

BEIER, T., AND NEELY, S. 1992. Feature-based image metamorphosis. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 35–42.

BEN-CHEN, M., WEBER, O., AND GOTSMAN, C. 2009. Variational harmonic maps for space deformation. *ACM Trans. Graph. 28*, 3, 1–11.

BOOKSTEIN, F. L. 1989. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Trans. Pattern Anal. Mach. Intell. 11*, 6, 567–585.

BOTSCH, M., AND SORKINE, O. 2008. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics*, 213–230.

BOTSCH, M., PAULY, M., GROSS, M., AND KOBBELT, L. 2006. Primo: coupled prisms for intuitive surface modeling. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, Eurographics Association, 11–20.

BOTSCH, M., PAULY, M., WICKE, M., AND GROSS, M. 2007. Adaptive space deformations based on rigid cells. In *Computer Graphics Forum*, vol. 26, Wiley Online Library, 339–347.

DEROSE, T., AND MEYER, M. 2006. Harmonic coordinates. Tech. rep., Pixar Animation Studios.

FLOATER, M. S., KÓS, G., AND REIMERS, M. 2005. Mean value coordinates in 3d. *Comput. Aided Geom. Des. 22*, 7, 623–631.

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

GÖDDEKE, D. 2005. Gpgpu–basic math tutorial. Tech. rep., Nov.

HORMANN, K., AND FLOATER, M. S. 2006. Mean value coordinates for arbitrary planar polygons. *ACM Trans. Graph. 25*, 4, 1424–1441.

IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. F. 2005. As-rigid-as-possible shape manipulation. In *SIGGRAPH '05: ACM*

Figure 9: *Deform the view in the "heart-reflector". (1) is the original image, (2) and (3) are the single cage selection and its corresponding deformation, (4) and (5) are the multicage selection and its corresponding deformation. Only a bigger polygon is added in (4) compared to (2). (6) is the difference image which is generated by subtract (5) from (3), and black color means no difference. We can see that the boundary of the "heart" is kept better after we use the multicage. The image size is* $1000 * 701$.

SIGGRAPH 2005 Papers, ACM, New York, NY, USA, 1134–1141.

JACOBSON, A., BARAN, I., POPOVIC, J., AND SORKINE, O. 2011. Bounded biharmonic weights for real-time deformation. In *SIGGRAPH '11: ACM SIGGRAPH 2011 Papers*.

JOSHI, P., MEYER, M., DEROSE, T., GREEN, B., AND SANOCKI, T. 2007. Harmonic coordinates for character articulation. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, ACM, New York, NY, USA, 71.

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

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. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, ACM, New York, NY, USA, 1–10.

LUEBKE, D., HARRIS, M., KRÜGER, J., PURCELL, T., GOVINDARAJU, N., BUCK, I., WOOLLEY, C., AND LEFOHN, A. 2004. Gpgpu: general purpose computation on graphics hardware. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes*, ACM, New York, NY, USA, 33.

MACCRACKEN, R., AND JOY, K. I. 1996. Free-form deformations with lattices of arbitrary topology. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 181–188.

MAGNENAT-THALMANN, N., LAPERRIRE, R., THALMANN, D., ET AL. 1988. Joint-dependent local deformations for hand animation and object grasping. In *In Proceedings on Graphics interface88*, Citeseer.

MENG, W., SHENG, B., WANG, S., SUN, H., AND WU, E. 2009. Interactive image deformation using cage coordinates on gpu. In *Virtual Reality Continuum and its Applications in Industry*, 119–126.

SCHAEFER, S., MCPHAIL, T., AND WARREN, J. 2006. Image deformation using moving least squares. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, ACM, New York, NY, USA, 533–540.

SORKINE, O., AND ALEXA, M. 2007. As-rigid-as-possible surface modeling. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, Eurographics Association, 109–116.

WEBER, O., SORKINE, O., LIPMAN, Y., AND GOTSMAN, C. 2007. Context-aware skeletal shape deformation. *Computer Graphics Forum (Proceedings of Eurographics) 26*, 3.

WEBER, O., BEN-CHEN, M., AND GOTSMAN, C. 2009. Complex barycentric coordinates with applications to planar shape deformation. *Computer Graphics Forum (Proceedings of Eurographics) 28*, 2.

WENG, Y., SHI, X., BAO, H., AND ZHANG, J. 2008. Sketching MLS image deformations on the GPU. *Computer Graphics Forum 27*, 7, 1789–1796.