Sketch Based Image Deformation

Mathias Eitz Olga Sorkine Marc Alexa

TU Berlin
Email: {eitz,sorkine,marc}@cs.tu-berlin.de

Abstract

We present an image editing tool that allows to deform and composite image regions using an intuitive sketch-based interface. Users simply draw the outline of the source image region and sketch a new boundary shape onto the location where this region is to be pasted. The deformation optimizes a shape distortion energy, and we use Poisson cloning for subsequent compositing. Since the correspondence between the source and target boundary curves is not known a priori, we propose an alternating optimization process that interleaves minimization of the deformation energy w.r.t. the image region interior and the mapping between the boundary curves, thus automatically determining the boundary correspondence and deforming the image at the same time. Thanks to the particular design of the deformation energy, its gradient can be efficiently computed, making the entire image editing framework interactive and convenient for practical use.

Figure 1: The fish is deformed by applying a single sketch and seamlessly placed in an underwater scene.

1 Introduction

Gradient domain approaches have been successfully used for image manipulation. Preserving the gradients of the image *domain* leads to image deformation techniques [2, 5, 6, 10], which allow rotating and possibly scaling the image, but minimize deformation. Preserving the gradients of the image *function* (i.e., the colors) can be used to seamlessly paste a region of an image into another image, a technique called Poisson image editing [9]. Choosing the right region to be copied and pasted is not always easy, and several ways of using optimization to identify the region in the source and target images have been proposed [1, 7, 8].

Our idea is to allow deformation of the image region in this process. This would be an important improvement for several of the automatic approaches, as the source and target boundary curves could be different in shape. The shape can also be controlled by the user, who would simply sketch the boundary curves of the image part to be copied in the source image and the corresponding region in the target image. This simple and intuitive user interface is what we advocate in this work.

Starting with the two boundary curves, we try to minimize the deformation of the corresponding image domain. For this purpose, we define a rotation-invariant deformation energy (Section 3), essentially following the ideas of the gradient domain image deformation techniques mentioned above. Note that we are not assuming that the mapping between the two boundary curves is known. Instead, minimizing the deformation energy determines both, the mapping of the boundary curves onto each other, and the mapping of the region enclosed by the

boundary curves. We solve this minimization problem by interleaving the optimization of the interior region and optimization of the boundary curve mapping. In Section 4 we explain how this can be done by computing the gradient of the energy function with respect to the boundary vertices and then projecting these gradients onto the tangents of the boundary curve.

Using a least squares approach for enforcing the mapping of the boundary curves allows varying the degree of precision with which the deformation follows the boundary conditions, to enable gesturing of quick and crude sketches, as well as carefully drawn, precise curves. In the results section we show how this leads to a very easy to control image editing tool.

2 Framework

The user selects a region of the source image by drawing a simple closed curve. We represent this image region by a quad mesh $\mathcal{S} = \{V, E\}$ of a certain user-defined resolution, typically coarser than the original pixel grid in order to save computation time, similarly to [5, 10]. We will denote the original vertex positions of $\mathcal{S}$ by $\mathbf{v} \in \mathbb{R}^{2n}$, a $2n$-vector containing the coordinates of the n vertices (the n x-coordinates followed by the n y-coordinates). We refer to the position of the i-th vertex as $\mathbf{v}_i \in \mathbb{R}^2$ and the set of neighbors of vertex i is $\mathcal{N}(i)$.

The user then draws another simple closed curve into the target image to indicate placement and deformation of the selected image region. Depending on the input device, the sampling rate, and the expertise of the user, it might be necessary to filter out some noise in the sketched curve. After connecting the pixel midpoints with line segments, we apply the Douglas-Peucker polyline simplification algorithm [4] using a small ϵ. The result of this procedure is a piecewise linear target boundary curve, which we denote as γ.

The goal is to find displaced vertex positions $\mathbf{v'} \in \mathbb{R}^{2n}$ of the quad mesh that

1. minimize the distance of boundary vertices to γ and
2. minimize a deformation energy, which is invariant to rotation and isotropic scale.

In the following section we will briefly explain how we define the energy based on the results of gradient domain mesh deformation techniques [11].

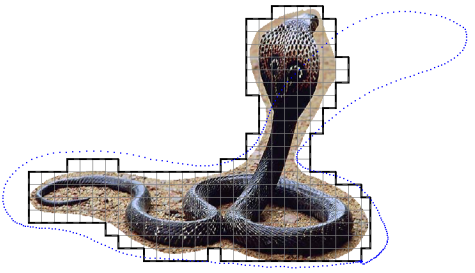


Figure 2: The user deforms an image region simply by drawing the desired new boundary curve γ, here shown as blue dots. The image region is represented by a quad mesh $\mathcal{S}$, in this case using a very coarse resolution (each quad is 20×20 pixels).

3 Deformation energy

It is common to measure deformation as a function of first and second derivatives of the mapping [3, 12]. The Laplacian of a mesh can be represented per vertex as

$$\boldsymbol{\delta}_i = \mathbf{L}(\mathbf{v}_i) = \mathbf{v}_i - (1/|\mathcal{N}(i)|) \sum_{j \in \mathcal{N}(i)} \mathbf{v}_j. \quad (1)$$

The mapping $\mathbf{v} \rightarrow \mathbf{v'}$ is a pure translation if all $\boldsymbol{\delta}_i$ remain unchanged. It is an isotropic scaling if the $\boldsymbol{\delta}_i$ are all scaled by the same factor. And it is a rotation if all $\boldsymbol{\delta}_i$ are rotated by the same angle. To measure deformation, we estimate isotropic scale and rotation locally (i.e., a similarity transformation), and compare the original $\boldsymbol{\delta}_i$ to the deformed ones.

A similarity transformation in 2D has the form

$$\mathbf{T}_i = \mathbf{T}_i(s, w) = \begin{pmatrix} s & w \\ -w & s \end{pmatrix}.$$

It can be estimated locally by considering each vertex and its neighbors, and then fitting the best similarity transformation in the least squares sense

$$\mathbf{T}_i(\mathbf{v'}) = \arg\min_{s,w} \sum_{j \in \{i\} \cup \mathcal{N}(i)} \|\mathbf{T}_i \mathbf{v}_j - \mathbf{v'}_j\|^2.$$

Note that $\mathbf{T}_i$ is a linear function in the $\mathbf{v'}$.

We use these local similarity transformations to transform the original Laplacians $\boldsymbol{\delta}_i$ and then compare them to the ones of the deformed mesh. The

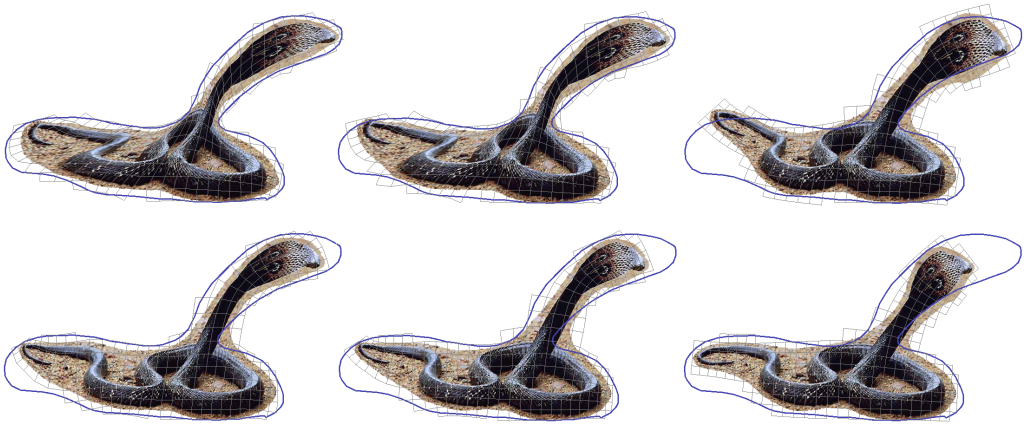


Figure 3: Comparison of deformation results using different handle weights ω (from left to right: $\omega = 1.0$, $\omega = 0.5$, $\omega = 0.1$). The top row shows results of the initial deformation while the bottom row shows the same results after applying our deformation minimization algorithm. The undeformed image region is shown in Figure 2.

squared difference of these vectors will be the deformation energy:

$$E_d(\mathbf{v}') = \sum_{i=1}^{n} \|\mathbf{T}_i(\mathbf{v}')\boldsymbol{\delta}_i - \mathbf{L}(\mathbf{v}'_i)\|^2 \qquad (2)$$

This is a quadratic expression in $\mathbf{v}'$. So it can be written as

$$\begin{aligned} E_d(\mathbf{v}') &= \|\mathbf{A}_d\mathbf{v}'\|^2 \\ &= \langle \mathbf{A}_d\mathbf{v}', \mathbf{A}_d\mathbf{v}' \rangle \\ &= \mathbf{v}'^{\mathsf{T}}\mathbf{A}_d^{\mathsf{T}}\mathbf{A}_d\mathbf{v}'. \end{aligned} \qquad (3)$$

It could be minimized by setting its partial derivatives to zero, leading to the equation $\mathbf{A}_d\mathbf{v}' = 0$. This means $\mathbf{v}' = 0$ is among the minimum energy states of $E_d(\mathbf{v}')$. The additional constraint that the boundary vertices of the mesh have to be close to the curve γ will yield non-zero solutions to the energy minimization.

4 Constrained energy minimization

We wish to find a deformation that minimizes the deformation energy so that all boundary vertices of the image region are close to the sketched curve γ. Assume we knew the target positions $\mathbf{h} \in \mathbb{R}^{2c}$ for the boundary vertices (again, $\mathbf{h}$ consists of the c concatenated x-coordinates and then the y-coordinates, and we denote the positions as $\mathbf{h}_i, i \in \mathcal{C}$). Then we could define an additional quadratic energy term

$$\begin{aligned} E_b(\mathbf{v}', \mathbf{h}) &= \sum_{i \in \mathcal{C}} \|\mathbf{v}'_i - \mathbf{h}_i\|^2 = \|\mathbf{A}_h\mathbf{v}' - \mathbf{h}\|^2 \\ &= \mathbf{v}'^{\mathsf{T}}\mathbf{A}_h^{\mathsf{T}}\mathbf{A}_h\mathbf{v}' - 2\,\mathbf{v}'^{\mathsf{T}}\mathbf{A}_h^{\mathsf{T}}\mathbf{h} + \mathbf{h}^{\mathsf{T}}\mathbf{h}, \end{aligned}$$

where $\mathbf{A}_h \in \mathbb{R}^{2c \times 2n}$ is a matrix that extracts the vector of constrained vertices out of $\mathbf{v}'$. Assume w.l.o.g. that $\mathcal{C} = \{1, \ldots, c\}$, then $\mathbf{A}_h$ has the simple form

$$\mathbf{A}_h = \begin{pmatrix} \mathbf{I}_{c\times c} & \mathbf{0}_{c\times(n-c)} & \mathbf{0}_{c\times n} \\ \mathbf{0}_{c\times n} & \mathbf{I}_{c\times c} & \mathbf{0}_{c\times(n-c)} \end{pmatrix}. \qquad (4)$$

Any weighted combination of the two energy terms

$$E_c(\mathbf{v}', \mathbf{h}) = E_d(\mathbf{v}') + \omega^2 E_b(\mathbf{v}', \mathbf{h}) \qquad (5)$$

is again a quadratic energy in $\mathbf{v}'$, i.e.

$$\begin{aligned} E_c(\mathbf{v}', \mathbf{h}) = \; & \mathbf{v}'^{\mathsf{T}}\left(\mathbf{A}_d^{\mathsf{T}}\mathbf{A}_d + \omega^2\mathbf{A}_h^{\mathsf{T}}\mathbf{A}_h\right)\mathbf{v}' - \\ & - 2\,\mathbf{v}'^{\mathsf{T}}\omega^2\mathbf{A}_h^{\mathsf{T}}\mathbf{h} + \omega^2\mathbf{h}^{\mathsf{T}}\mathbf{h}. \end{aligned} \qquad (6)$$

Now the resulting linear system defining the minimum energy state is nonhomogeneous, resulting in

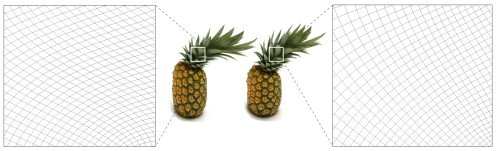


Figure 4: Comparison of an unoptimized (left) with an optimized mesh (right). The unoptimized mesh shows strong shearing artifacts which results in distortions of the image. After applying our energy minimization algorithm the resulting mesh is much more regular and thus visual distortions in the image are minimized. See Figure 6 for a higher resolution image.

a unique solution when more than two vertices are constrained and $\omega > 0$.

Of course, we don't have target positions $\mathbf{h}$ for the handle vertices a priori. All we know is that they are supposed to be on γ. So our approach to minimizing the energy is to compute target positions on the curve γ that minimize $E_d(\mathbf{v}')$, i.e., the positions are defined as

$$\arg\min_{\mathbf{h}\in\gamma} E_d(\mathbf{v}'). \tag{7}$$

The minimization is done iteratively, using a gradient descent approach. We first compute a rough initial mapping of the boundary vertices onto the user-sketch, i.e., matching the source boundary and the target sketch based on arc-lengths. We then take a small step into the negative direction of the gradients of the deformation energy with respect to the boundary vertices in order to find new handle positions resulting in smaller deformation energy and thus less distortion of the mesh. Since the handle positions are constrained to lie on the user-sketched boundary, we reproject the new handle positions onto the sketch and iterate the algorithm until a minimum is reached.

The deformation resulting from the rough arc-lengths based mapping is what we call "initial deformation" and compare our optimized results against.

4.1 Gradient on the boundary

Knowing the positions of the boundary vertices $\mathbf{h}$, we can compute $\mathbf{v}'$ as the minimum energy state of $E_c(\mathbf{v}', \mathbf{h})$ by taking the gradient of the energy w.r.t. $\mathbf{v}'$. From Eqn. (6) follows:

$$\frac{\partial E_c}{\partial \mathbf{v}'} = 2(\mathbf{A}_d^\mathsf{T}\mathbf{A}_d + \omega^2\mathbf{A}_h^\mathsf{T}\mathbf{A}_h)\mathbf{v}' - 2\omega^2\mathbf{A}_h^\mathsf{T}\mathbf{h}.$$

Setting the gradient to zero, we obtain

$$\mathbf{v}'(\mathbf{h}) = \omega^2(\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)^{-1}\mathbf{A}_h^\mathsf{T}\mathbf{h}, \tag{8}$$

where $\mathbf{A}_c \in \mathbb{R}^{(2n+2c)\times(2n)}$ is the combined rectangular matrix

$$\mathbf{A}_c = \begin{pmatrix} \mathbf{A}_d \\ \omega\mathbf{A}_h \end{pmatrix},$$

such that $\mathbf{A}_d^\mathsf{T}\mathbf{A}_d + \omega^2\mathbf{A}_h^\mathsf{T}\mathbf{A}_h = \mathbf{A}_c^\mathsf{T}\mathbf{A}_c$.

We proceed to compute the gradient $\mathbf{g}$ of the deformation energy functional with respect to the handles $\mathbf{h}$: $\mathbf{g} = \nabla E_d(\mathbf{v}'(\mathbf{h}))$. According to the chain rule,

$$\mathbf{g} = \nabla E_d(\mathbf{v}'(\mathbf{h})) = \left(\frac{\partial \mathbf{v}'}{\partial \mathbf{h}}\right)^\mathsf{T} \nabla E_d(\mathbf{v}').$$

Recall that the Jacobian matrix $\left(\frac{\partial \mathbf{v}'}{\partial \mathbf{h}}\right)$ is the matrix of partial derivatives:

$$\left(\frac{\partial \mathbf{v}'}{\partial \mathbf{h}}\right) = \begin{pmatrix} \frac{\partial v_1'}{\partial h_1} & \cdots & \frac{\partial v_1'}{\partial h_{2c}} \\ \vdots & \ddots & \vdots \\ \frac{\partial v_{2n}'}{\partial h_1} & \cdots & \frac{\partial v_{2n}'}{\partial h_{2c}} \end{pmatrix}.$$

Direct computation of the Jacobian is inefficient: from Eqn. (8) one can see that it requires explicitly having the inverse matrix $(\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)^{-1}$, which is expensive and would necessitate prohibitive storage, because the inverse matrix is generally not sparse. We describe how we circumvent this problem in Section 4.3.

4.2 Efficient gradient computation

Since the gradient descent algorithm requires several iterations until a minimum is reached, we are interested in computing $\mathbf{g}$ as efficiently as possible. The first part needed for the computation of $\mathbf{g}$ is $\nabla E_d(\mathbf{v}')$:

$$\nabla E_d(\mathbf{v}') = 2\left(\mathbf{A}_d^\mathsf{T}\mathbf{A}_d\right)\mathbf{v}'. \tag{9}$$

We can use the fact that $\mathbf{v}'$ is the solution of Laplacian mesh editing, which can be accelerated by computing the Cholesky factorization of the system

matrix $\mathbf{A}_c^\mathsf{T}\mathbf{A}_c$. Since this matrix does not change during our iterations, we can reuse this factorization; computing $\mathbf{v'}$ for a different right hand side then only requires a (fast) backsubstitution step.

4.3 Efficient Jacobian computation

From Eqn. (8) we can directly obtain the Jacobian of $\mathbf{v'}$, since $\mathbf{v'}$ is a linear function of $\mathbf{h}$:

$$\frac{\partial \mathbf{v'}}{\partial \mathbf{h}} = \omega^2 (\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)^{-1}\mathbf{A}_h^\mathsf{T}.$$

According to the structure of $\mathbf{A}_h$ (Eqn. (4)), it is easy to see that the Jacobian simply consists of the relevant columns of $(\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)^{-1}$. Let us denote the individual columns of $(\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)^{-1}$ by $\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_{2n}$. Then

$$\left(\frac{\partial \mathbf{v'}}{\partial \mathbf{h}}\right) = \omega^2(\mathbf{m}_1, \cdots, \mathbf{m}_c, \mathbf{m}_{n+1}, \cdots, \mathbf{m}_{n+c}).$$

To avoid explicitly computing the inverse matrix and then taking some of its columns, we observe what happens if we actually use *all* the columns. By "pretending" that the Jacobian actually equals $\omega^2(\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)^{-1}$, we obtain the following "overcomplete gradient":

$$\begin{aligned} \tilde{\mathbf{g}} &= (\omega^2(\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)^{-1})^\mathsf{T}(2(\mathbf{A}_d^\mathsf{T}\mathbf{A}_d)\mathbf{v'}) \quad (10) \\ &= 2\omega^2(\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)^{-1}\mathbf{A}_d^\mathsf{T}\mathbf{A}_d\mathbf{v'}. \quad (11) \end{aligned}$$

(Here we used the fact that since $\mathbf{A}_c^\mathsf{T}\mathbf{A}_c$ is symmetric, its inverse is also symmetric.) We can easily obtain $\tilde{\mathbf{g}}$ as the solution of the following linear system:

$$(\mathbf{A}_c^\mathsf{T}\mathbf{A}_c)\tilde{\mathbf{g}} = 2\omega^2(\mathbf{A}_d^\mathsf{T}\mathbf{A}_d)\mathbf{v'}.$$

This system is quite efficient to solve: it only requires one more back-substitution, since the prefactorization of the system matrix $\mathbf{A}_c^\mathsf{T}\mathbf{A}_c$ is already given.

Finally, we observe that since the real gradient $\mathbf{g}$ only has $2c$ rows, and in order to compute it we need the $2c$ columns of the inverse matrix, we can actually obtain $\mathbf{g}$ from $\tilde{\mathbf{g}}$ simply by erasing the unnecessary rows and only keeping the rows $1, \ldots, c$, $n+1, \ldots, n+c$.

4.4 Reprojecting gradients

To minimize deformation energy we apply the gradient descent algorithm. Given the gradient $\mathbf{g}$ of the deformation energy with respect to the handle vertices $\mathbf{h}$, we compute new handle positions $\mathbf{h'}$ by taking a small step into the opposite direction of the gradient of $\mathbf{h}$:

$$\mathbf{h'} = \mathbf{h} - \lambda\mathbf{g}.$$

Recall that we require the boundary vertices $\mathbf{h}$ to lie on the user-sketched curve γ in order for the deformed shape $\mathcal{S}'$ to follow the user-sketch. The new handle positions $\mathbf{h'}$ that result from an optimization iteration generally do not lie on γ anymore. To fix this, we reproject the handles $\mathbf{h'}$ onto γ.

We have implemented the reprojection of $\mathbf{h'}$ onto γ as a simple orthogonal projection; i.e., for each handle vertex $\mathbf{v}'_c, c \in \mathcal{C}$, we find the nearest point on γ. We denote the vector of reprojected handle vertices by $\mathbf{h'_{proj}}$. The reprojected handle $\mathbf{h'_{proj}}$ is then used to compute the new energy gradient, and the algorithm is iterated until convergence. We found that approximately 50 iterations combined with a stepsize of $\lambda = 0.5$ yielded good visual results in all cases and thus used those values in all our examples .

5 Deforming and cloning

To finally compute the deformed image from the deformed quad mesh we apply texture mapping with bilinear interpolation, which can be quickly performed by graphics hardware. Lastly, the resulting image is composited onto the target background image with Poisson cloning as illustrated in Figure 1. For completeness, in the following we briefly describe the compositing algorithm.

The final image is a result of an optimization process that minimizes the squared difference between the gradients of the unknown region in the target image and the given gradients of the pasted image for target boundary conditions. This results in a solution that has gradients similar to the original image while the boundary constraints ensure that the resulting colors match the given boundary colors.

The optimization amounts to solving the Poisson equation on the pasted image domain, to reconstruct the three color channels. Using sparse linear solver libraries, e.g. TAUCS [13], we can do this very efficiently by factoring the Laplace matrix once and solving by backsubstitution. This even allows interactively moving the pasted region around the target image, since only the right-hand side of the system

changes, and thus the cloning can be recomputed very quickly.

6 Results

We have successfully applied the proposed method to various images and show comparison results between original, initially deformed and optimized images in Figure 6 and Figure 7. As can be seen in the resulting optimized images, visual distortions in the deformed image are minimized by our approach, while still following the user's sketch.

In Figure 8 we show the deformation energy gradients with respect to the handle vertices. The length of the gradient vectors is exaggerated by a factor of 10 for illustration purposes. Note that in the initially deformed mesh, the gradient vectors have large components tangential to the boundary curve, indicating that mesh deformation could be reduced by moving vertices along the boundary. As the minimization procedure progresses, tangential components become smaller and the procedure converges in all gradients being orthogonal to the sketch. Figure 4 shows another example of an optimized mesh. Applying our energy minimization algorithm removes the strong shearing artifacts clearly noticeable in the initially deformed mesh. The result is a mesh with a structure very close to the original mesh, leading to smaller visual distortion in the deformed image.

The performance of our approach depends on the size of the involved linear system, i.e. on the size of the underlying mesh that is to be deformed. See Table 1 for a comparison of execution time against various mesh sizes, taken from the example in Figure 5. While using coarse meshes provides interactive feedback within fractions of a second, higher resolutions take significantly longer to compute but result in less distortions in the final deformed image. Note that only the minimization operation (last column in Table 1) has to be executed when redrawing a sketch, all other parts are precomputed.

We show a comparison of the influence of varying handle weights on the deformation results in Figure 3. Note that exactly following the user-sketch using higher handle weights results in stronger deformations in the initial deformation result. Our proposed deformation minimization removes those distortions while still following the user-drawn outline.

vertices	init	fact.	∇E_d	min.
300	0.047	0.234	0.172	0.344
916	0.141	0.765	1.141	1.078
3491	0.438	2.562	4.204	4.39

Table 1: Performance data measured in seconds on an Intel Core 2 Duo 2.4 Ghz with 2 GBytes RAM. From left to right: Number of vertices in the underlying quad mesh, time to create the system matrix $\mathbf{A}_c$, time needed to compute a sparse factorization of $\mathbf{A}_c^\mathsf{T}\mathbf{A}_c$, time needed for computing the $\mathbf{A}_d^\mathsf{T}\mathbf{A}_d$ part of the gradient of the deformation energy and time needed to execute 50 iterations of our minimization algorithm in order to compute the optimized image.

7 Discussion

We have presented a fast and robust optimization technique that implements sketch based image deformation resulting in high quality deformation results without visible distortions.

Our technique is interactive up to medium mesh resolutions and provides quick feedback when redrawing a boundary since the most expensive computations (factoring the system matrix and computing $\mathbf{A}^\mathsf{T}\mathbf{A}$) only have to be done once. It is also easy to implement since computing the gradient with respect to the handle vertices, as shown in Eqn. (11), reduces to simply solving two linear systems.

The constrained energy minimum is usually not unique – depending on the initial boundary matching the gradient descent procedure converges to the closest local minimum. Also, as the deformation energy is invariant under rotations, near circular boundaries could lead to ambivalent minimum energy states. In practice, we have not observed this problem. In any case, the user could add information on corresponding points on the boundary to resolve ambiguities.

Our current implementation requires the user to resketch the whole boundary for each modification, which can be cumbersome. Future work may thus include boundary manipulation techniques, allowing the user to edit small parts of the boundary instead of doing a complete resketch. This may be achieved by using intuitive curve editing interfaces [6].

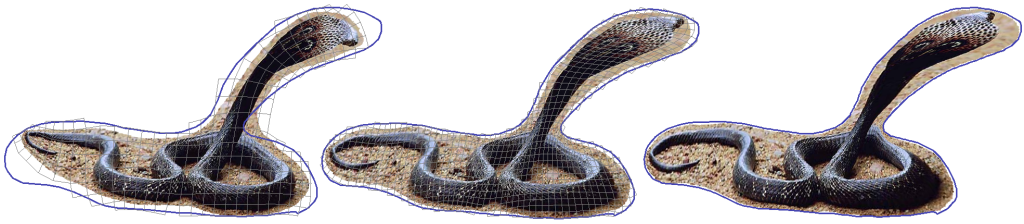
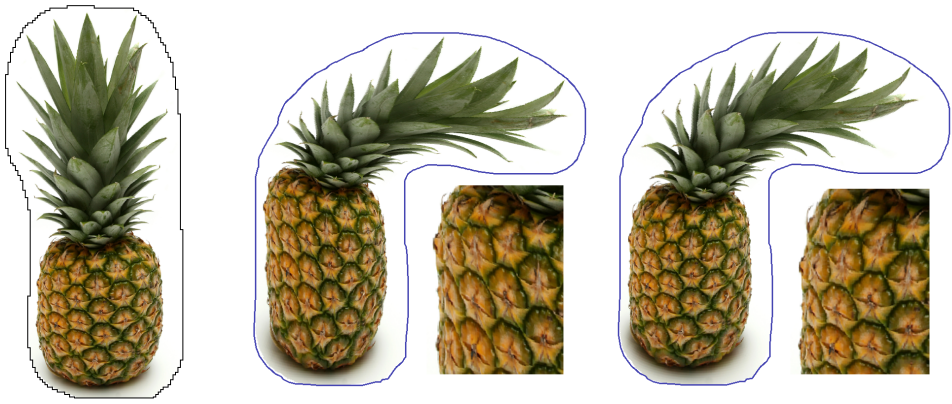
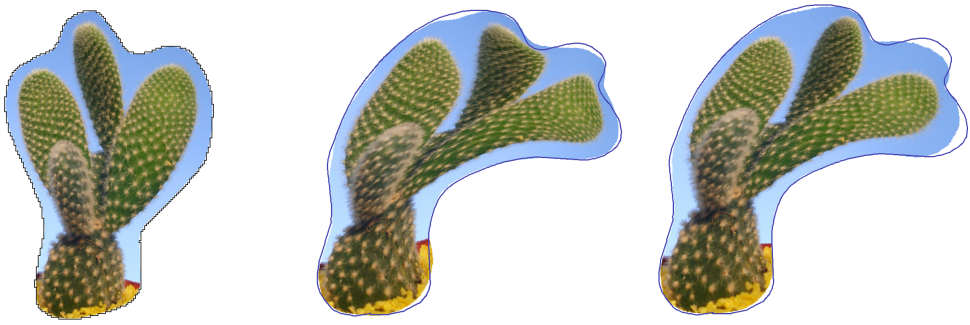


Figure 5: Deformation results using various mesh resolutions. From left to right: Using quad sizes of 20×20, 10×10 and 5×5 pixels, with constant handle weights $\omega = 1.0$.



(a) Original (b) Initial Deformation (c) Energy Minimized

Figure 6: Bending a pineapple. The result of the initial deformation (b) shows strong distortions. After minimizing the deformation energy, visible distortions are eliminated (c).



(a) Original (b) Initial Deformation (c) Energy Minimized

Figure 7: Bending the top of a cactus to the right. Again, the original, undeformed image is shown in (a), the initially deformed image in (b) and the optimized image after applying 50 iterations of our deformation minimization algorithm is shown in (c).

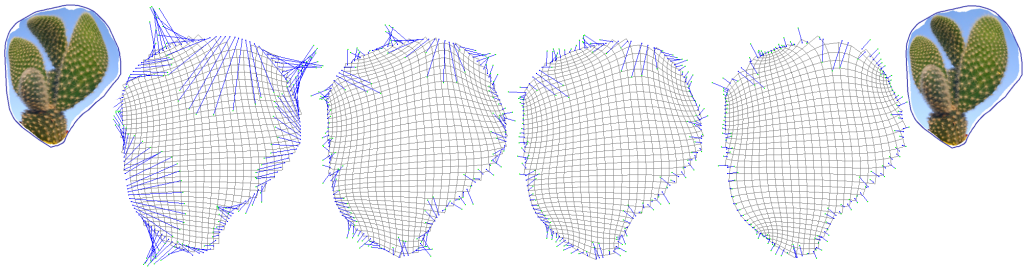


Figure 8: Development of a deformed mesh and its associated gradients of the handle vertices (blue lines) during the optimization process. The image on the top left corresponds to the leftmost, unoptimized mesh while the image on the top right corresponds to the optimized mesh on the right after applying 50 iterations of our optimization algorithm. Gradient vectors are scaled by a factor of 10 for illustration purposes.

Acknowledgements

We wish to thank Andrew Nealen for fruitful discussions and the anonymous reviewers for their valuable comments. This work was supported in part by the Alexander von Humboldt Foundation.

References

[1] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. *ACM Trans. Graph.*, pages 294–302, 2004.

[2] Marc Alexa, Daniel Cohen-Or, and David Levin. As-rigid-as-possible shape interpolation. In *Proceedings of ACM SIGGRAPH*, pages 157–164, 2000.

[3] Mario Botsch and Olga Sorkine. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics*, 2007. To appear.

[4] D.H. Douglas and T.K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[5] Ran Gal, Olga Sorkine, and Daniel Cohen-Or. Feature-aware texturing. In *Proceedings of Eurographics Symposium on Rendering*, pages 297–303, 2006.

[6] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. As-rigid-as-possible shape manipulation. *ACM Trans. Graph.*, 24(3):1134–1141, 2005.

[7] J. Jia, J. Sun, C.K. Tang, and H.Y. Shum. Drag-and-drop pasting. *ACM Trans. Graph.*, 25(3):631–637, 2006.

[8] Yin Li, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. Lazy snapping. *ACM Trans. Graph.*, pages 303–308, 2004.

[9] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Trans. Graph.*, 22(3):313–318, 2003.

[10] S. Schaefer, T. McPhail, and J. Warren. Image deformation using moving least squares. *ACM Trans. Graph.*, 25(3):533–540, 2006.

[11] Olga Sorkine, Yaron Lipman, Daniel Cohen-Or, Marc Alexa, Christian Rössl, and Hans-Peter Seidel. Laplacian surface editing. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 179–188. ACM Press, 2004.

[12] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *Proceedings of ACM SIGGRAPH*, pages 205–214, 1987.

[13] Sivan Toledo. TAUCS*: A Library of Sparse Linear Solvers, version 2.2*. Tel-Aviv University, Available online at `http://www.tau.ac.il/~stoledo/taucs/`, September 2003.