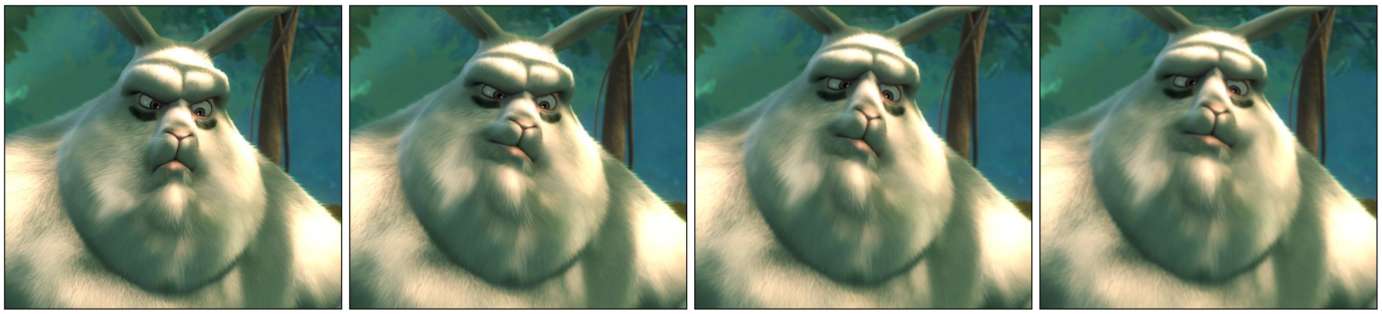
Regularized Kelvinlet Inversion for Real-Time Image Deformation and Video Time Warping

Guilherme G. Haetinger
Instituto de Informática – UFRGS
Porto Alegre, RS, Brazil
gghaetinger@inf.ufrgs.br

Eduardo S. L. Gastal
Instituto de Informática – UFRGS
Porto Alegre, RS, Brazil
eslgastal@inf.ufrgs.br



Reference image | Edit #1: Mouth (left side) | Edit #2: Eyebrows | Edit #3: Mouth (right side)

Fig. 1: Demonstration of our method's results. A series of smooth deformations applied to the reference image (far left), generating artificial pictures that could pass as rendered shots of the animation. Reference image ©2008, Blender Foundation.

***Abstract*—We present a novel approach for image deformation and video time warping. Our technique involves the inversion of the nonlinear regularized Kelvinlet equations, leading to higher-quality results and time/space efficiency compared to naive solutions. Inversion is performed by a per-pixel optimization process, being inherently parallel and achieving real-time performance in Full HD videos (over 300 fps). We demonstrate our method on a variety of images and videos, in addition to discussing some important technical and theoretical details.**

I. Introduction

The production of visually appealing images and videos is a critical factor for their consumption. In particular, the animation and movie industry have shown to be a much larger market than previously imagined, mostly because of amazing special effects that were not thought possible a few years ago. The development of new and improved image and video processing algorithms plays an important role in this context.

Our work targets both the fields of Image deformation and Video time warping, with the goal of creating visually appealing artificial results. Image deformation techniques create new images by "deforming the pixels" of existing images [1], [2], [3]. We propose a new real-time algorithm that creates smooth pictures with sub-pixel accurate deformations, as illustrated in Fig. 1. It is based on the elastic patterns designed into the Kelvinlet sculpting brush [4]. Our solution is trivially parallelizable using the graphics hardware, and can take advantage of the built-in mipmapping capabilities for antialiasing (required due to the resampling that occurs during deformation [5]), generating smooth images and high-quality results. Video time warping is a less-explored area, which can be seen as an extension of image deformation, from 2D space to 3D space-time. We use the concept of the space-time video volume and demonstrate the use of our technique for smooth, real-time temporal deformations (time warping).

The contributions of our work include:

- An inversion method for the nonlinear Kelvinlet equations, applied to image deformation and video time warping through backward mapping. This generates higher-quality results when compared to forward mapping;
- An efficient image and video warping algorithm that is inherently parallel and easy to implement. It is able to process Full HD videos at over 300 frames per second through our nonlinear per-pixel optimization process;
- A procedure for altering the Kelvinlet deformation field in order to preserve the image or video's rectangular boundary, while still generating smooth deformations;
- A discussion of important technical details for generating high-quality antialiased images, as well as for detecting and handling non-invertible deformations. We also discuss the relationship between temporal and spatial aliasing.

II. Related Works

A. Image Deformation

Image deformation has numerous applications, ranging from photo editing to content-aware image resizing [1]. Kauffman *et al.* [3] consider a content-aware method for image warping. They use an image meshing algorithm that respects the geometry of its objects. The same method is also extendable to adaptive meshing for video sequences, enabling the deformation to persist throughout the frames. However, their algorithm depends on object detection in the images. The energy-based deformation proposed by Karni *et al.* [2] uses energy minimization to avoid distorted results. This method uses multiple weight points to generate a structured deformation, much like the "warp tools" from modern photo editing software [6].

The regularized Kelvinlet brush [4] describes a volume-preserving force field that may be used for deformations. In their paper, De Goes and James are mostly focused on "sculpting" 3D polygonal meshes, where the displacements can be directly applied to the vertices through a *forward-mapping* implementation. This approach, however, is not ideal for processing images and videos, for the reasons presented in Section IV-A. As such, our work extends the Kelvinlet equations to a *backward-mapping* approach [5]. We discuss the Kelvinlet brush in more detail in Section III.

B. Video Time Warping

The relationship between videos and 3D volumes is not new to the Computer Graphics community [7], [8], [9], [10]. As pointed out by Rav-Acha *et al.* [9], the space-time volume used by our algorithm (to depict and modify time) was introduced as the *epipolar volume* by Bolles in 1987 [11], and it has since been adopted by many researchers for video processing and analysis [7], [12].

Many works make use of time warping for achieving specific artistic or technical results, without explicitly focusing on how the actual warping itself is performed. For example, Rav-Acha *et al.* [9] elaborate a time warping framework based on *evolving time fronts*, which are smooth space-time surfaces that traverse the video volume, creating new frames. They achieve interesting results with this idea, such as altering the winner in a swimming competition video by delaying (or advancing) each competitor in time. While they mostly focus on warping along the time dimension, they also mention the possibility of warping along time while also deforming in space (which our technique supports natively). Their work does not discuss the underlying algorithm used to warp each video frame, but one of the figures in their paper seems to indicate that they use a polygonal mesh and a forward-mapping implementation.

Other works with similar ideas include that of Cohen *et al.* [8], which presents a non-photorealistic rendering tool for videos. It explores the notion of time and space questioned by paintings of the Cubist and Futurist art movements. More recently, Solteszova *et al.* [10] target time warping for scientific visualizations. Operating on top of video streams instead of static files, *Memento* is their tool for real-time video interaction and exploration. It is applicable in diverse scenarios with a good user interface and useful results for monitoring and comparing video segments through time. Different from our approach, they only demonstrate warpings along time, and do not explore spatial deformations combined with time warping.

The aforementioned methods differ from ours in that they do not propose new algorithms for the actual warping procedure, and instead focus on applications of warping. We discuss this topic further in Section IV-A.

Relation to 3-D volume rendering: one of the difficulties involved in video time warping is the generation and "playback" of the resulting warped video frames. While seemingly related to Volume Rendering techniques [13], [7], [14], it is important to note that these are not directly applicable to rendering time-warped videos. This occurs because volume rendering methods treat the volume as a semi-transparent material that can emit, transmit, and absorb light, *i.e.* is intended to be rendered all at once, whereas the space-time video volume is meant to be rendered "slice by slice", *i.e.*, frame by frame.

Other related topics include the generation of regular speed videos from high-frame-rate inputs. Zhou *et al.* [15] process videos in order to detect movement. This way, they are able to lower the frame-rate by using temporally-variant filters to enhance the display of salient motions, which can be seen as a 1D global warping function applied uniformly to all pixels. Their work is orthogonal to ours and can be used together with our warping technique.

III. Background on the Regularized Kelvinlet

The Regularized Kelvinlets [4] are elastostatic deformation methods based on the fundamental solution of linear elasticity, also known as Kelvin's state. In this work, we focus on the application of the grab brush solution [4]. This method defines a displacement vector $K(p)$ for each point p in the domain (in our case, each pixel in a 2D image or 3D video volume), such that the resulting vector field has a controllable degree of volume preservation.

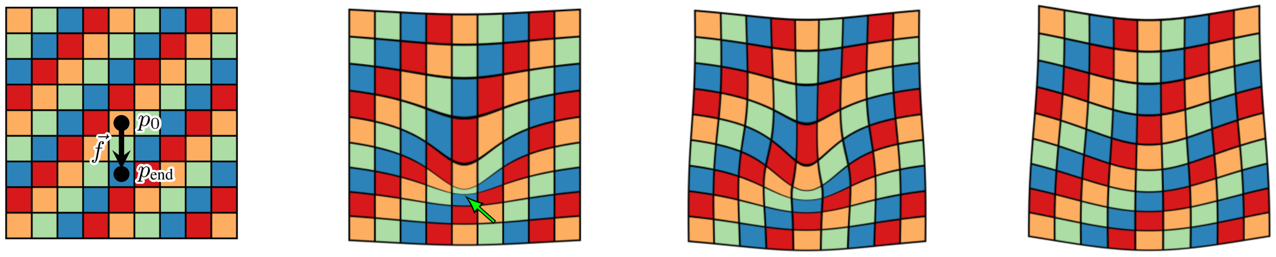
The Kelvinlet equations can be better understood from a simple usage example. Imagine a user that clicks on a point at position p_0 in an image (called the deformation pivot), and drags it to a new position p_{end} (Fig. 2a). This defines a displacement force at p_0 to be the vector $\vec{f} := p_{\text{end}} - p_0$. The force vectors for all other points in the domain are then given by [4]:

$$K(p) = c\,\varepsilon\,U(p - p_0)\,\vec{f}, \tag{1}$$

where

$$U(\vec{r}) = \frac{(a-b)}{r_\varepsilon} I + \frac{b}{r_\varepsilon^3}\vec{r}\,\vec{r}^\top + \frac{a}{2}\frac{\varepsilon^2}{r_\varepsilon^3} I, \tag{2}$$
$$\text{with } r_\varepsilon = \sqrt{\|\vec{r}\|^2 + \varepsilon^2}.$$

The values $a = \frac{1}{4\pi}$, $b = \frac{a}{4(1-\nu)}$, and $c = \frac{2}{3a-2b}$ are constants that depend on the Poisson ratio ν, which controls the volume-preservation properties of the Kelvinlet field [4]. The parameter ε controls the "radius" of the deformation effect, *i.e.*, how much the displacement at p_0 affects its neighborhood. The effects of ν and ε are illustrated in Fig. 2. For all results shown in the paper and supplementary materials, we use $\nu = 0.4$, which generates good results in practice. The values for ε, the pivot p_0 and the force $\vec{f}$ are provided in each figure's caption. Finally, we note that in Eq. (2), I denotes the identity matrix, $\|\cdot\|$ denotes the Euclidean norm, and $\cdot^\top$ denotes a transpose. Also observe that this equation is valid for both 2D and 3D deformations.



p_0
$\vec{f}$
p_{end}
($\nu = -\infty, \ \varepsilon = 50$)
($\nu = 0.4, \ \varepsilon = 50$)
($\nu = 0.4, \ \varepsilon = 130$)
(a) Reference image
(b) Low local volume preservation
(c) High local volume preservation
(d) Larger influence radius

Fig. 2: Demonstration of the influence of the Poisson ratio (ν) and the radius (ε) on the deformation result. (a) Reference image and deformation parameters: the pivot p_0 is dragged by a user to a new position p_{end}, defining a deformation force $\vec{f}$. (b) Deformations with $\nu \to -\infty$ do not preserve the *local* volumes, as shown by the severe compression indicated by the green arrow. (c) Deformations with $\nu \approx 0.5$ preserve local volumes more consistently across the domain. For this to be possible, the image must "bulge outwards." (Section IV-C discusses a method to preserve the rectangular image boundary, which was not applied here for illustrative purposes.) (d) Deformations with a larger influence radius ε generate less-concentrated results.

IV. Our Kelvinlet Inversion Method for Image and Video Warping

As opposed to the 3D sculpting brush described by De Goes and James [4], the deformation method proposed here uses an inverse transform. We now explain the inversion process, starting with a discussion on forward vs backward mapping (Section IV-A). All equations that follow are applicable to both image deformation (2D) and video time warping (3D domain).

A. *Forward vs Backward-Mapping Implementations*

The force field K from Eq. (1) translates each pixel p in the input image g to a new position $q = T(p) = p + K(p)$ in the output warped image g_w. A naive **forward-mapping** implementation for generating all output pixels works as follows:

Algorithm 1 Image deformation with forward mapping

```
1: For each input pixel $p$ in the input image $g$:
2:     Compute the output pixel location $q = T(p)$;
3:     Set the output pixel color $g_w(q) = g(p)$.
```

This solution has important difficulties [16]. For instance, note that most input pixels p are mapped to non-integer output locations q, and also that neighboring input pixels may be mapped to *non*-neighboring output pixels. As such, a post-deformation interpolation is required to fill the gaps generated by the input pixel's displacements [5]. One way to do this in 2D is by associating each pixel with a vertex in a triangle mesh, and then performing interpolation between the displaced pixels through triangle rasterization [17] (higher-order interpolations are also possible [5]). This process becomes significantly more complicated in 3D, where each triangle is replaced by a 3D tetrahedron. In this scenario, rasterization becomes a bottleneck, achieving hardly interactive rates even for small volumes [18]

A generally better solution is to use an inverse transformation, due to the fact that it does not require the post-deformation interpolation between pixels [5], [16]. Therefore, our method uses T^{-1} to map locations from the *output* image g_w to locations on the *input* image g (thus, in the reverse direction). This results in the following **backward-mapping** implementation:

Algorithm 2 Image deformation with backward mapping

```
1: For each output pixel location $q$ in the output image $g_w$:
2:     Compute the input pixel location $p = T^{-1}(q)$;
3:     Set the output pixel color $g_w(q) = g(p)$.
```

This solution addresses all of the difficulties discussed above. It does not require any post-deformation interpolation since the for-loop traverses all *output* pixel locations q. Furthermore, the result for each q is independent of the other pixels and thus all output pixels can be computed in parallel. The only interpolation required is in the input domain, when sampling the input image g at location p, to obtain the color $g(p)$. But since the input domain defines a regular 2D or 3D pixel grid, this interpolation is a trivial operation, that can be performed by standard bilinear or higher-order kernels. In a GPU implementation, bilinear interpolation is directly supported by the hardware. One can also take advantage of mipmapping for antialiasing (Section V-A).

The backward-mapping solution requires the computation of T^{-1}. As such, T must be invertible (a bijection), and one must be able to compute $T^{-1}(q)$ for each q. Unfortunately, in the case of the Kelvinlet field (Eq. (1)), there is no closed-form solution for its inverse. In the next section we show how a nonlinear optimization technique can be used to compute T^{-1} efficiently, and Section V-B discusses how to handle situations where T is not invertible. Fig. 3 illustrates the relationship between T and T^{-1}.

B. *Inverting the Kelvinlet Deformation using Gauss-Newton*

Given a particular pixel location q, we wish to compute $p = T^{-1}(q)$ (line 2 of Algorithm 2). Our insight is that this problem can be modeled as a nonlinear optimization:

$$T^{-1}(q) = \arg\min_{p'} \ \left\| T(p') - q \right\|^2 . \tag{3}$$

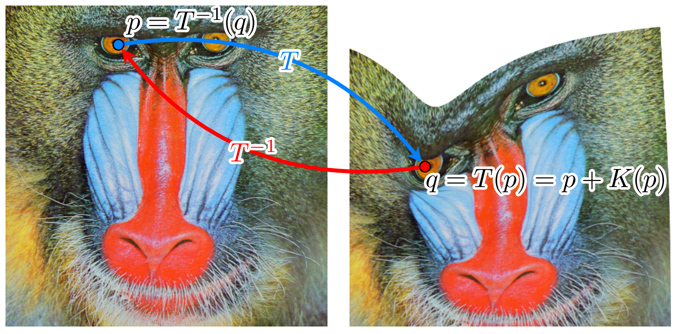


Fig. 3: Relationship between T and T^{-1}. The deformation $T(p) = p + K(p)$ maps pixels from the input image to new positions in the output deformed image, using the Kelvinlet field K. Our method computes the output image by "looking up" colors in the input domain using T^{-1}.

In other words, the point p' which minimizes the functional on the right-hand-side of Eq. (3) is our solution $p = T^{-1}(q)$.

We solve Eq. (3) using iterative Gauss-Newton [19], which is applicable in this situation since K (Eq. (1)), and thus T, is continuously differentiable (the basic idea is to approximate the right-hand-side of Eq. (3) with its first-order Taylor expansion [19]). The algorithm works as follows: starting from an initial guess p'_1, a series of improved solutions $p'_2, p'_3, \ldots$ are obtained from the update step $p'_{k+1} = p'_k + \delta_k$, where

$$\delta_k = \arg\min_{\delta} \; \left\| p'_k + \delta + K(p'_k) + J(p'_k)\,\delta - q \right\|^2 . \tag{4}$$

In the above equation, $J(p'_k)$ is the Jacobian matrix of the operator K, computed at p'_k. In the case of image deformations, J is a 2×2 matrix, and in the case of video time warping, J is a 3×3 matrix. We include the closed-form expressions for the Jacobians in our supplementary materials.

The right-hand-side of Eq. (4) is a quadratic functional on δ, whose minimum is obtained by differentiating and equating to zero. This results in a closed-form linear solution for δ_k:

$$\delta_k = A_k^{-1} \mu\, b_k , \tag{5}$$

where

$$A_k = J(p'_k)^\top J(p'_k) + 2J(p'_k)^\top + I, \tag{6}$$
$$b_k = -\left(J(p'_k)^\top + I\right) \xi_k, \tag{7}$$
$$\xi_k = p'_k + K(p'_k) - q. \tag{8}$$

We set the Gauss-Newton damping parameter $\mu = 0.5$ and stop the iterations when the residual norm falls below the threshold 10^{-1} (in pixel units): $\|\xi_k\| < 10^{-1}$. This permits a maximum error in $T^{-1}(q)$ of one-tenth the size of a pixel, which is visually imperceptible. For each pixel q we start with the initial guess $p'_1 = q$, which is a sensible choice since the majority of the pixels in an image are generally not affected by a particular deformation (*i.e.*, $K(p) \approx \vec{0}$ for most pixels, and thus $p = T^{-1}(q) \approx q$). Fig. 4 shows the typical number of iterations required for convergence.

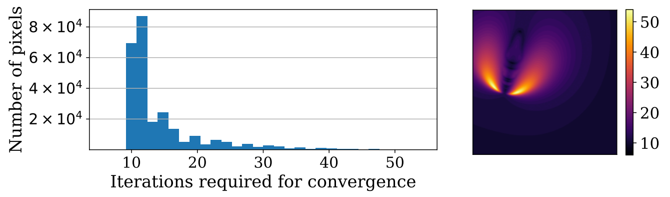


Fig. 4: (Left) Histogram of the number of iterations required for convergence of the Gauss-Newton iterations. Most pixels require around 10 to 20 iterations for the residual norm to fall below the threshold of 10^{-1}. (Right) Per-pixel visualization of the number of iterations required for convergence. Note how pixels that are closer to the deformation pivot require more iterations. Results computed for the deformation in Fig. 3.

C. *Fixed Domain Boundary*

Different from 3D sculpting of polygonal meshes [4], when deforming an image or video one is generally interested in preserving its rectangular boundary positions. Thus, we propose an additional term that smoothly decreases the deformation field's strength as one gets close to the boundary (Fig. 5b). This is done by using a control curve (Eq. (9)), which dictates how much of the original force will be used at a given pixel, depending on its distance from the boundary:

$$\beta_\circ(p) = \sin\left(\frac{\pi}{2}\frac{\min(D_\circ(p), \sigma)}{\sigma}\right). \tag{9}$$

In the equation above, σ represents the radius along the boundary where the force falloff starts to take place. Note that $\beta_\circ(p) \in [0,1], \forall p$, smoothly varies from 0 at the boundary to 1 at σ-units from the boundary. Furthermore, $\circ \in \{x, y, t\}$ stands for a particular dimension, and $D_\circ(p)$ computes the distance of pixel p from the boundary along that particular dimension. For example, in an image of size $W \times H$, with one-based pixel indexing:

$$\begin{aligned} D_x(p) &= \min(p_x - 1, W - p_x), \quad \text{and} \\ D_y(p) &= \min(p_y - 1, H - p_y). \end{aligned} \tag{10}$$

We use $\sigma = 50$ pixels for the examples shown in the paper where a fixed boundary was employed. The modified force field K_β is then given by:

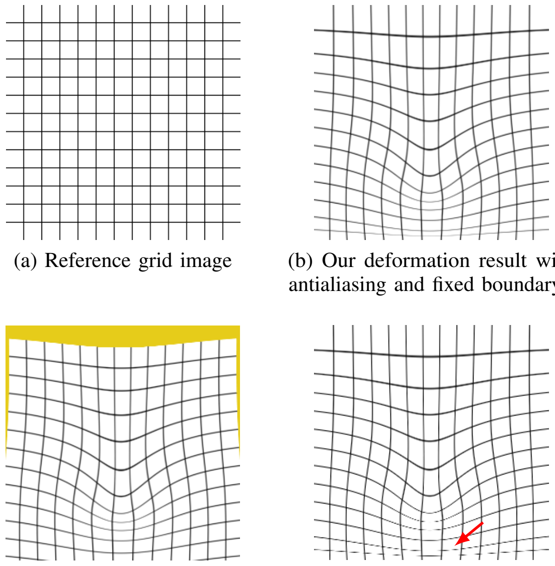
$$K_\beta(p) = \begin{bmatrix} \beta_x(p) & \\ & \beta_y(p) \end{bmatrix} K(p). \tag{11}$$

When processing 3D video volumes (time warping), the entry β_t is added in a third row and column to the matrix above.

Since these operations modify the deformation field, the Jacobian matrix J in Eq. (4) must also be updated. This is done by applying the differentiation product rule to Eq. (11), resulting in a new Jacobian J_β:

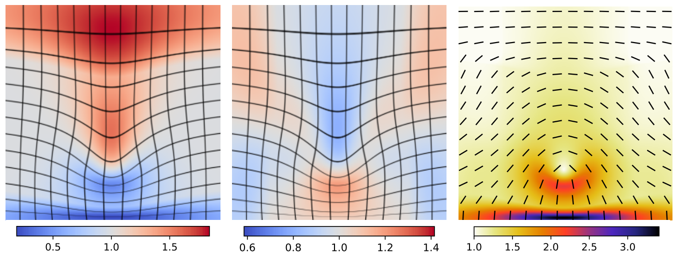
$$J_\beta = \begin{bmatrix} \beta_x & \\ & \beta_y \end{bmatrix} J + \begin{bmatrix} \frac{\partial \beta_x}{\partial x} & \\ & \frac{\partial \beta_y}{\partial y} \end{bmatrix} \begin{bmatrix} K_x & \\ & K_y \end{bmatrix}. \tag{12}$$

The new matrix $J_\beta(p'_k)$, evaluated at p'_k, must be used in the Gauss-Newton update steps in Eqs. (5) to (8).



(a) Reference grid image (b) Our deformation result with antialiasing and fixed boundary (c) Result *without* fixed boundary (d) Result *without* antialiasing

Fig. 5: (a) A reference image with fine details. (b) Our antialiased deformation with a fixed boundary results in a high-quality and smoothly-deformed image. (c) Without a fixed boundary, out-of-bounds pixels with undefined colors “leak” into the image (shown in yellow). (d) Without proper antialiasing, the fine lines from the reference grid result in “jagged” rendering artifacts (red arrow).



(a) $\varphi_{\vec{y}}$ (b) $\varphi_{\vec{x}}$ (c) Mipmap levels

Fig. 6: (a) Visualization of the *vertical* contractions (pixels with $\varphi_{\vec{y}}(p) < 1$, in shades of blue) and expansions (pixels with $\varphi_{\vec{x}}(p) > 1$, in shades of red), for the deformation in Fig. 5b. (b) A similar visualization for the *horizontal* contractions and expansions. (c) Illustration showing the major axes of spatial contraction $\vec{v}_1$ (black lines), and the corresponding mipmap pyramid levels computed from $\varphi_{\vec{v}_1}$ (colormap).

V. Implementation details

We now describe some relevant implementation details of our image and video warping methods.

A. Antialiasing through Anisotropic Filtering

The sample-rate of the output warped image, relative to the input domain, varies locally as a funcion of the deformation. This means that prefiltering must be performed when generating the output pixels (Fig. 5b), to avoid aliasing artifacts (Fig. 5d).

The deformation's Jacobian matrix $J(p)$, computed at pixel p, encodes the local expansion and contraction of the space around p [16]. It can thus be used to identify the proper antialiasing kernels. More precisely, the quadratic form

$$\varphi_{\vec{v}}(p) = 1 + \frac{\vec{v}^{\top} J(p)\, \vec{v}}{\|\vec{v}\|^2} \tag{13}$$

describes the local contraction (if $\varphi_{\vec{v}}(p) \in [0, 1)$) or expansion (if $\varphi_{\vec{v}}(p) \in [1, \infty)$) that occurs at pixel p in the direction $\vec{v}$. This concept is illustrated in Fig. 6(a–b) for the vertical ($\vec{v} = \vec{y}$) and horizontal ($\vec{v} = \vec{x}$) directions. The direction $\vec{v}_1$ that minimizes $\varphi_{\vec{v}}(p)$ representes the major axis of spatial contraction at p, and can thus be used to define oriented anisotropic antialiasing kernels (Fig. 6c). $\vec{v}_1$ is the eigenvector associated with the smallest eigenvalue of $J + J^{\top}$.

In our implementation, we employ a mipmap pyramid for fast antialiasing. We perform anisotropic filtering using multiple samples along the major axis $\vec{v}_1$, while selecting the mipmap level based on the contraction along the minor axis $\vec{v}_2$ [16], which is associated with the largest eigenvalue. This procedure is supported in hardware through the OpenGL extension `GL_ARB_texture_filter_anisotropic`.

B. Detecting Non-Invertible Deformations

If $\varphi_{\vec{v}_1}(p) < 0$ then the image is being *locally folded-over itself*, meaning that T is not invertible and the warped image is not properly defined. This often occurs if the deformation is too strong. Our implementation emits a warning to the user in such situations, and automatically dampens the deformation force by the smallest amount required to guarantee invertibility. This is done by replacing $T(p)$ with $T_\alpha(p) = p + \alpha K(p)$ where the scaling $\alpha \in (0, 1)$ is defined as

$$\alpha = \min_p -\frac{1-\epsilon}{\varphi_{\vec{v}_1}(p) - 1}. \tag{14}$$

$\epsilon = 10^{-2}$ is a small constant used to avoid numerical issues.

VI. Results and Discussion

We implemented our image deformation method in *Julia*, a just-in-time compiled programming language that focuses on efficiency for scientific computing. Our implementation is multithreaded and generates all output pixels in parallel, achieving high-performance deformations. For example, the execution time for the deformation in Fig. 7b (with bilinear prefiltering) is 0.042 seconds on a 6-core 3.2 GHz CPU.

For video time warping, we implemented our method in OpenGL to take advantage of the GPU's high processing power. This allows us to execute the deformation processing and video display in real time (over 300 fps for Full HD 1080p video on an RTX 2070S). We load the video into a 3D texture in GPU memory, and for each (x, y) fragment rendered in the current frame at time t, we find the corresponding point $T^{-1}(x, y, t)$ in the volume to sample its color. The Gauss-Newton iterations are computed in parallel for all pixels in the current frame, during video playback, using a GLSL shader. We solve the small 3×3 linear systems from Eq. (5) using GLSL's `inverse()` instruction, which is provided in the hardware.

All image deformation results shown in the paper were generated with our Julia implementation, while the video time warping results were generated with our OpenGL implementation. Now, we present results achieved with both methods (see our supplementary materials for the complete video sequences).

A. Image Deformation Results (2D)

Fig. 7 compares the result of our deformation method to the available alternatives from Krita, an open source image editing software [6]. Our method achieves better results when compared to Krita's warp and liquify tools (see discussion in the figure's caption). Fig. 8 shows a similar comparison on a synthetic image, showing that our approach preserves local volume, even while using our fixed boundary constraint. Krita's tool results in a strong shape deformation in the thick black line visible in Fig. 8a, and our result shows the line with minimal volume size change and with smooth, antialiased displacement (notice the aliasing artifacts in Krita's result). Our fixed border constraint also shows to be more rigid (Fig. 8b), as we see small portions of background in Fig. 8c. As such, our technique is able to generate novel and high-quality deformation results, allowing for a wider range of artistic possibilities.

Fig. 9 shows that our *backward-mapping* Kelvinlet warping method works great with fractal images. The fact that it does not need post-deformation interpolation allows us to deform these images with continuous precision, meaning that it maps the output pixel q to the perfect color value of the displaced point, by evaluating the fractal at $T^{-1}(q)$. In contrast, a *forward-mapping* Kelvinlet warping (implemented using the commonly-used triangle mesh technique) results in unwanted blurriness in highly displaced areas. Since the fractal defines a continuous image (broadband signal), all the results shown in Fig. 9 were antialiased through supersampling.

B. Video Time Warping Results (3D)

Fig. 10 illustrates the idea of time warping in a simple synthetic video sequence, and Fig. 11 shows how time warping can be applied to a steady zoom-out video sequence for a real use case: to increase the shot's emphasis on a given object (in this case, the flower).

Fig. 12 shows the results of time warping on a Full HD 1920×1080 video of someone pouring water into a cup, used to delay the cup's filling. As we apply a deformation that pushes back the water from filling the middle of the cup, it is visible, by looking at the top of the water level, that the cup is being filled above the deformation point. This is the result we want, since the transition from the top water level to the unfilled gap generated by the displacement is exactly as smooth as the transition between water and air on the original video Therefore, we are able to achieve smooth blending between objects in a scene through time. This result was generated at ~325 frames per second on an RTX 2070 Super GPU.

1) Temporal aliasing: Time sampling is inherently different from spatial sampling [20]. While our visual system is severely attuned to spatial discontinuities, object motion can be spaced throughout larger pixel intervals (temporal discontinuities) and still be perceived as continuous movement. That is why, depending on the frame-rate and shot steadiness, one can find jagged edges (temporal aliasing) when slicing a video through the time axis (Fig. 13(a-b)). Hence, when we combine different frames into a new frame through a warping of the time axis, temporal aliasing is transformed into spatial aliasing, as seen in Fig. 13c. This is a problem that occurs with any time-warping technique, and it can be fixed by capturing the video with a higher frame-rate or by applying a temporal antialiasing filter (Fig. 13d). This transforms aliasing into motion blur [21].

VII. Conclusions and Future Work

We have presented an algorithm for the inversion of the Regularized Kelvinlet equations, enabling its use for image deformation and video time warping through backward mapping. Inversion is performed by a nonlinear per-pixel optimization process (Gauss-Newton), allowing for fast execution time. Our solution is inherently parallel and achieves real-time performance in Full HD videos, computing deformations at over 300 fps on modern hardware.

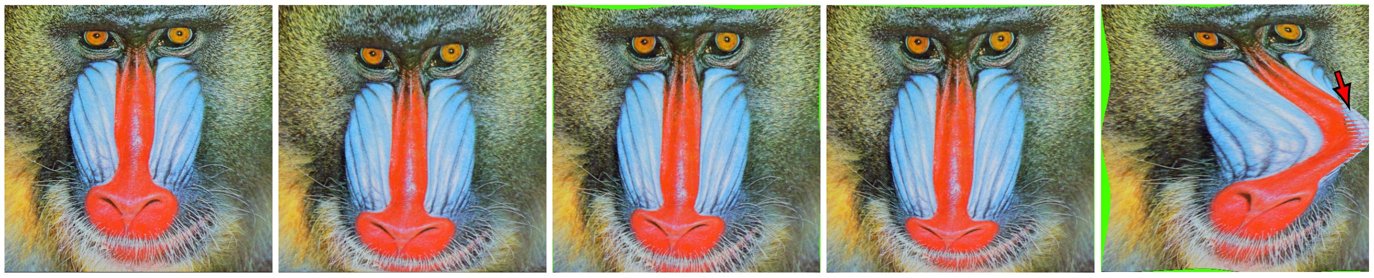
We showed how to preserve the image or video's rectangular boundary through a modification of the deformation field, and also how to detect and handle non-invertibility. Furthermore, we discussed the procedures for proper antialiasing of the resulting images, based on local measures of spatial contraction. We also discussed the relationship between temporal and spatial aliasing, providing some insights into the limits of time warping.

Our technique has a number of applications. Image deformation can be used for interactive photo editing, creating smooth curves and providing a realistic feeling of control over a picture, *i.e.*, "warping as if we were touching it". The 3D time warping brush enables the creation of movie reels that change the previously established sequence of frames. Possibilities of future work include the exploration of a user interface for real-time video interaction, as well as extending our solution to other types of deformation fields.

Acknowledgements. We would like to thank the anonymous reviewers for their comments. This work was sponsored by CNPq-Brazil (436932/2018-0), and financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

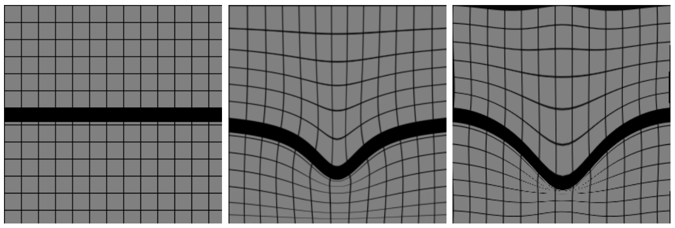
References

[1] S. Avidan and A. Shamir, "Seam carving for content-aware image resizing," in *ACM SIGGRAPH 2007 papers*, 2007, pp. 10–es.
[2] Z. Karni, D. Freedman, and C. Gotsman, "Energy-Based Image Deformation," *Computer Graphics Forum*, vol. 28, no. 5, 2009.
[3] P. Kaufmann, O. Wang, A. Sorkine-Hornung, O. Sorkine-Hornung, A. Smolic, and M. Gross, "Finite Element Image Warping," *Computer Graphics Forum*, vol. 32, no. 2pt1, pp. 31–39, 2013.
[4] F. De Goes and D. L. James, "Regularized kelvinlets: sculpting brushes based on fundamental solutions of elasticity," *ACM Transactions on Graphics*, vol. 36, no. 4, pp. 1–11, Jul. 2017.
[5] G. Wolberg, *Digital Image Warping*. IEEE computer society press Los Alamitos, CA, 1990, vol. 10662.
[6] K. Foundation, "Krita," 2020. [Online]. Available: https://krita.org/en/
[7] S. Frey, "Spatio-Temporal Contours from Deep Volume Raycasting," *Computer Graphics Forum*, vol. 37, no. 3, pp. 513–524, 2018.
[8] A. W. Klein, P.-P. J. Sloan, A. Finkelstein, and M. F. Cohen, "Stylized video cubes," in *SIGGRAPH/EG Symp. on Comput. Animation*, 2002.



(a) Original image (512×512) (b) Our deformation result (c) Krita warp tool (d) Krita liquify tool (e) Krita warp tool artifacts

Fig. 7: Comparison between our inverse Kelvinlet image deformation method and the Krita tools. (b) Our method produces a high-quality deformation result that is not possible to reproduce with Krita. It is able to smoothly translate the eyes and nose of the mandrill downward, while preserving the image's rectangular boundary. (c) Krita's warp tool requires two user interactions and introduces boundary artifacts (green pixels). (d) Krita's liquify tool produces a slighty better result for boundaries, but compresses the nose too much and does not move the eyes. (e) Krita's warp tool generates artifacts if the deformation is too strong (red arrow), since it uses a forward-mapping polygonal mesh for warping. Kelvinlet params.: $p_0 = (256, 256), \vec{f} = (0, -90), \varepsilon = 100$.



(a) Original image (b) Our result (c) Krita warp result

Fig. 8: Distortion comparison between our inverse Kelvinlet warping method and the Krita deformation tool. Kelvinlet deformation parameters: $p_0 = (128, 128), \vec{f} = (0, -70), \varepsilon = 30$.

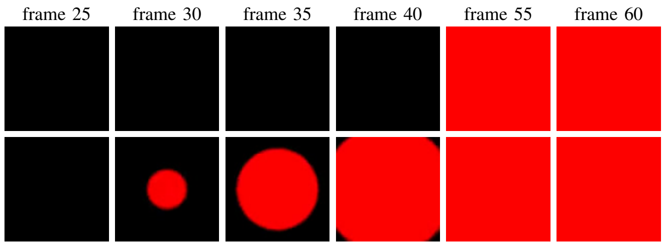
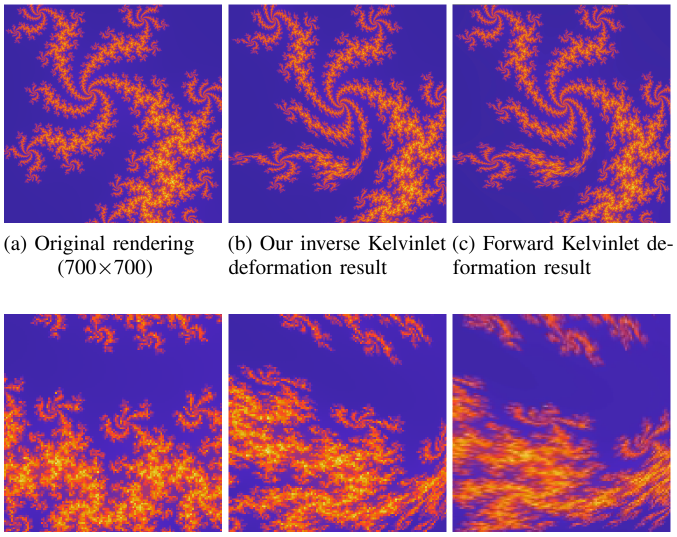


Fig. 10: (Top row) Input video consisting of an instantaneous transition from black to red at frame 52. (Bottom row) Smooth transition obtained using our time warping method to bring the red values forward in time. Warping parameters: $p_0 = (50, 50, 50), \vec{f} = (0, 0, -30), \varepsilon = 50$. Video size $50 \times 50 \times 100$.



(a) Original rendering (700×700) (b) Our inverse Kelvinlet deformation result (c) Forward Kelvinlet deformation result

(d) Zoom of (a) (e) Zoom of (b) (f) Zoom of (c)

Fig. 9: Deformation of continuous fractal images. (a,d) Original supersampled rendering. (b,e) Our *backward-mapping* method results in a high-quality deformed fractal. (c,f) A *forward-mapping* implementation results in a blurred image. Deformation parameters: $p_0 = (200, 400), \vec{f} = (200, -100), \varepsilon = 100$.

[9] A. Rav-Acha, Y. Pritch, D. Lischinski, and S. Peleg, "Evolving Time Fronts: Spatio-Temporal Video Warping," *Proc. 32nd Int. Conf. Comput. Graph. Interactive Tech*, p. 8, 2005.

[10] V. Solteszova, N. N. Smit, S. Stoppel, R. Grüner, and S. Bruckner, "Memento: Localized Time-Warping for Spatio-Temporal Selection," *Computer Graphics Forum*, vol. 39, no. 1, pp. 231–243, 2020.

[11] R. C. Bolles, H. H. Baker, and D. H. Marimont, "Epipolar-plane image analysis: An approach to determining structure from motion," *International Journal of Computer Vision*, vol. 1, no. 1, pp. 7–55, 1987.

[12] B. Bach, C. Shi, N. Heulot, T. Madhyastha, T. Grabowski, and P. Dragicevic, "Time Curves: Folding Time to Visualize Patterns of Temporal Evolution in Data," *IEEE TVCG*, vol. 22, no. 1, 2016.

[13] B. Wylie, K. Moreland, L. Fisk, and P. Crossno, "Tetrahedral projection using vertex shaders," in *Symp. on Vol. Vis. and Graph.*, 2002, pp. 7–12.

[14] C. T. Silva, J. L. D. Comba, S. P. Callahan, and F. F. Bernardon, "A Survey of GPU-Based Volume Rendering of Unstructured Grids," *Revista de informática teórica e aplicada*, no. 2, pp. 9–29, 2005.

[15] F. Zhou, S. B. Kang, and M. F. Cohen, "Time-Mapping Using Space-Time Saliency," in *CVPR*, Jun. 2014, pp. 3358–3365.

[16] R. Szeliski, *Computer Vision*. Springer London, 2011.

[17] P. S. Heckbert, "Fund. of texture mapping and image warping," 1989.

[18] J. Gascon, J. M. Espadero, A. G. Perez, R. Torres, and M. A. Otaduy, "Fast deformation of volume data using tetrahedral mesh rasterization," in *SIGGRAPH/EG Symp. on Comput. Animation*, 2013, pp. 181–185.

[19] J. Solomon, *Numerical algorithms: methods for computer vision, machine learning, and graphics*. CRC Press, Taylor & Francis Group, 2015.

[20] D. J. Finlay and P. C. Dodwell, "Speed of apparent motion and the wagon-wheel effect," *Perception & Psychophysics*, vol. 41, no. 1, 1987.

[21] J. Korein and N. Badler, "Temporal anti-aliasing in computer generated animation," in *SIGGRAPH*, Jul. 1983, pp. 377–388.

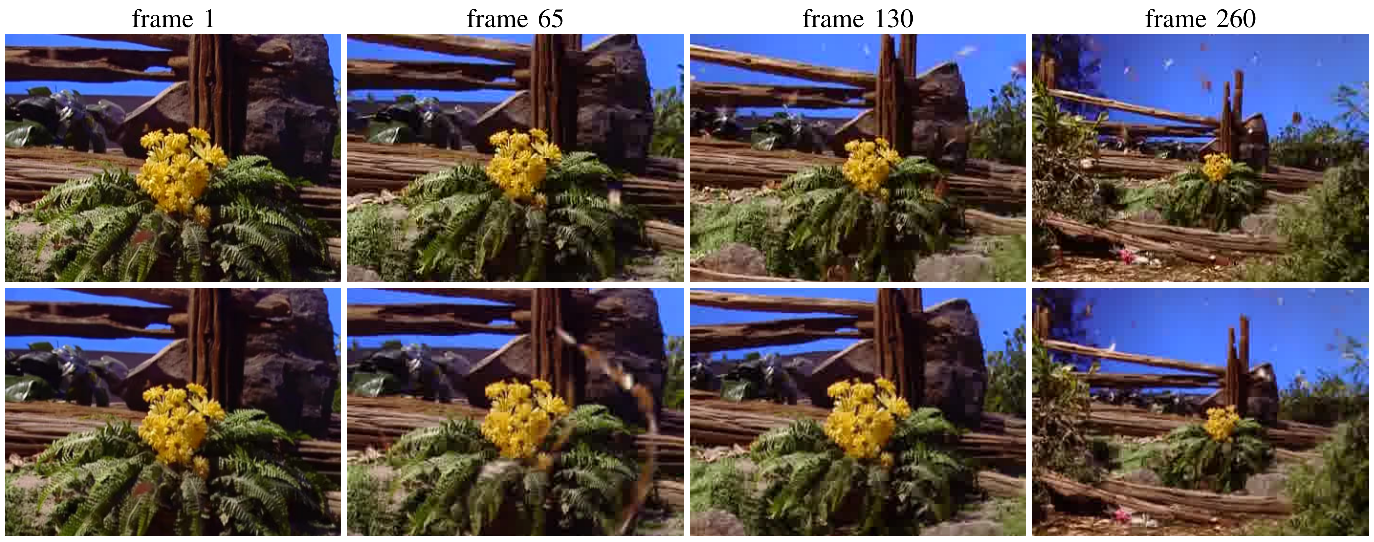


Fig. 11: (Top row) Input video of a flower with a zooming-out camera ($352 \times 288 \times 260$). (Bottom row) Time warped video, maintaining a zoomed-in flower size throughout most of the sequence, and concentrating the zoom-out at the last moments. Warping parameters: $p_0 = (170, 130, 20), \vec{f} = (0, 0, 100), \varepsilon = 80$. Input video sequence 'tempete' from Xiph.org.

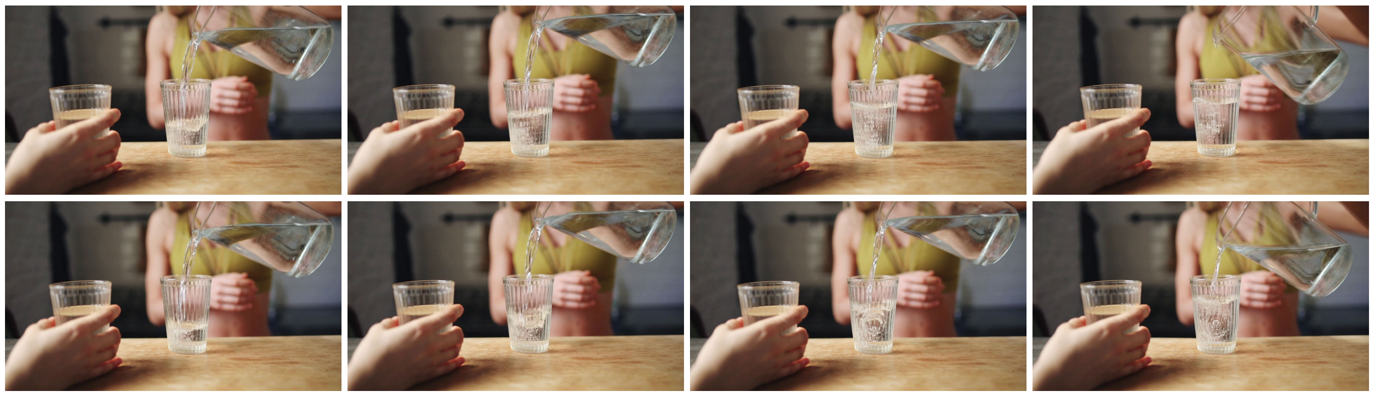
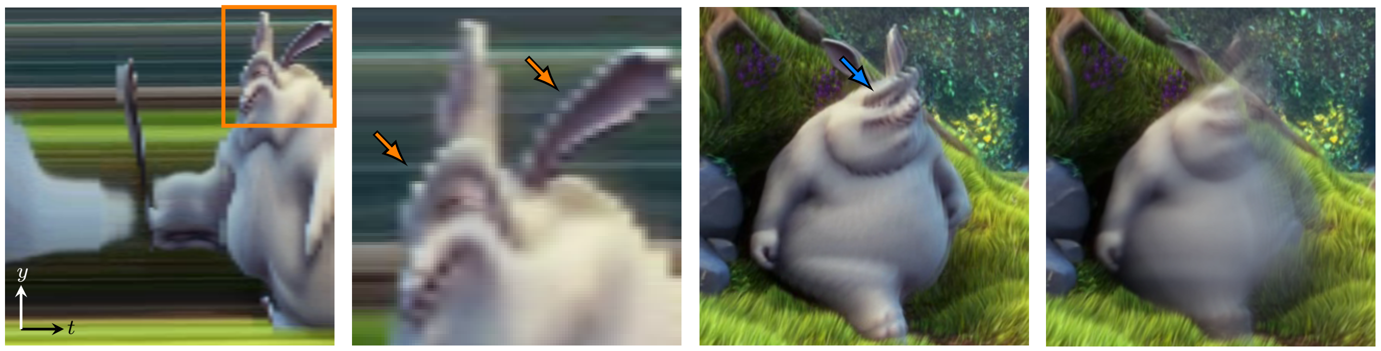


Fig. 12: (Top row) Unmodified video of someone pouring water into a cup ($1920 \times 1080 \times 118$). (Bottom row) Same video warped to delay the cup's filling. (See our supplementary materials for the full video sequences). Deformation parameters: $p_0 = (1050, 700, 10), \vec{f} = (0, 0, 60), \varepsilon = 60$. Input video sequence by Polina Tankilevitch on Pexels.com.



(a) Time slice of the original video

(b) Zoom of (a): temporal aliasing

(c) Time-warped video: temporal aliasing transformed into spatial aliasing

(d) Time-warped video with antialiasing along the time axis

Fig. 13: (a) Time slice along the (y, t) plane of a video sequence containing object motion. (b) Zoom of (a), showing the presence of jagged edges, *i.e.*, temporal aliasing (orange arrows). (c) Time-warped video sequence, showing how temporal aliasing has been transformed into spatial aliasing (blue arrow). (d) Applying an antialiasing filter along the time axis removes aliasing artifacts but introduces motion blur. Deformation parameters: $p_0 = (100, 100, 20), \vec{f} = (0, 0, 100), \varepsilon = 60$.