TexToons: Practical Texture Mapping for Hand-drawn Cartoon Animations

Daniel Sýkora[*] CTU in Prague, FEE
Mirela Ben-Chen Stanford University
Martin Čadík MPI Informatik
Brian Whited Maryann Simmons Walt Disney Animation Studios

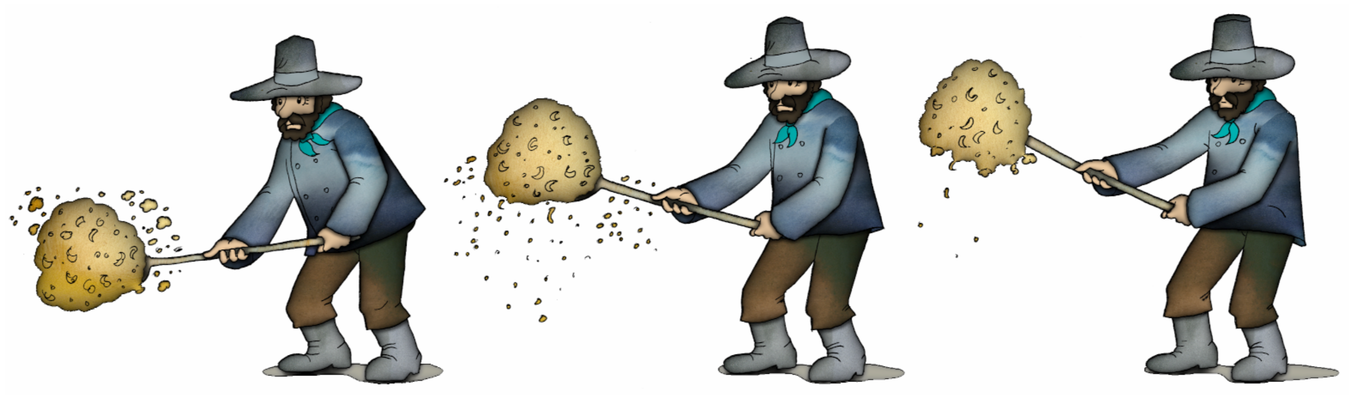
Figure 1: *An example of output from our system: texture-mapped hand-drawn cartoon enhanced by 3D-like effects.*

Abstract

We present a novel and practical texture mapping algorithm for hand-drawn cartoons that allows the production of visually rich animations with minimal user effort. Unlike previous techniques, our approach works entirely in the 2D domain and does not require the knowledge or creation of a 3D proxy model. Inspired by the fact that the human visual system tends to focus on the most salient features of a scene, which we observe for hand-drawn cartoons are the contours rather than the interior of regions, we can create the illusion of temporally coherent animation using only rough 2D image registration. This key observation allows us to design a simple yet effective algorithm that significantly reduces the amount of manual labor required to add visually complex detail to an animation, thus enabling efficient cartoon texturing for computer-assisted animation production pipelines. We demonstrate our technique on a variety of input animations as well as provide examples of post-processing operations that can be applied to simulate 3D-like effects entirely in the 2D domain.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, Shading, Shadowing, and Texture; I.4.3 [Image Processing and Computer Vision]: Enhancement—Registration; J.5 [Computer Applications]: Arts and Humanities—Fine arts

Keywords: cartoon animation, texture mapping, visual attention, deformable image registration

[*]e-mail: sykorad@fel.cvut.cz

1 Introduction

Texture mapping is a classic Computer Graphics technique for efficiently adding detail and richness to 3D models without increasing the geometric complexity. In contrast, traditional 2D animation requires any texture to be hand-drawn by artists in every single frame of the animation (see Fig. 2), resulting in a very labor-intensive and often tedious task. It thus seems natural to borrow the concept of texture mapping of 3D models and apply it to 2D cartoon animations.

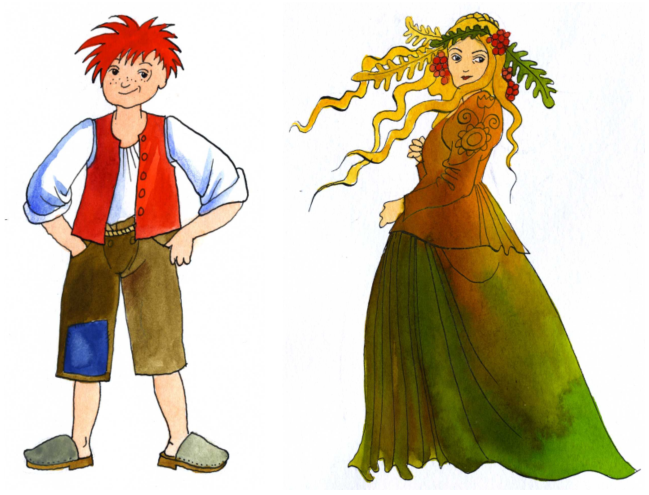
Figure 2: *Examples of cartoon images with hand-drawn textures.*

Unfortunately, there are two key obstacles inherent in cartoon animations that prevent the direct application of texture mapping to this domain. First, in order to texture map an animated object, correspondences must be established between object points across all frames of the animation. This correspondence ensures that each sample of a texture is consistently mapped to the same surface location through time. This requirement is trivial for a 3D animation due to the fact that the textured object in each frame is a single model that has been deformed and/or transformed and maintains

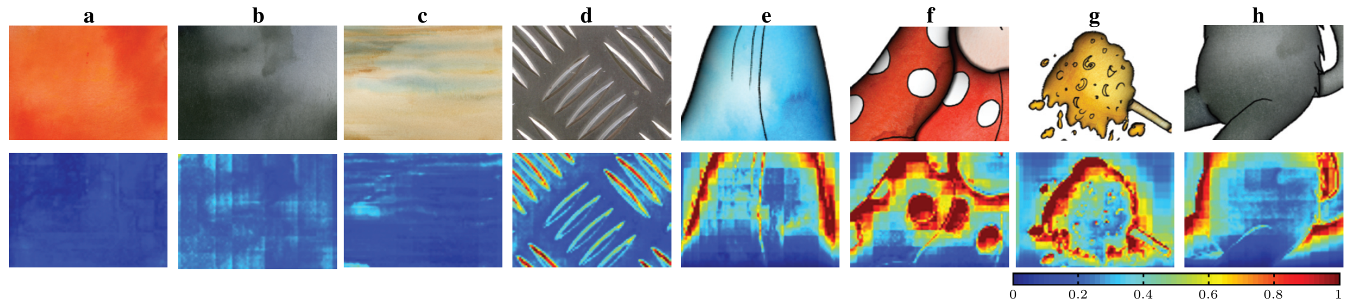

Figure 3: *Visual saliency of the textures. The top row contains example cartoon texture and contour samples. The bottom row visualizes saliency maps automatically predicted using a bottom-up model of human visual attention. (a, b, c) Examples of textures supported by the proposed technique. (d) A texture with strong features that exhibits relatively high visual saliency and therefore less suitable input for our technique. (e, f, g, h) Cartoon contours are visually much more salient than textures, and thus efficiently mask possible inconsistencies in texture registration.*

the same topology. In traditional cartoon animation, an object is drawn each frame by hand and therefore lacks any explicit geometry or structure that would allow texture mapping to be used in the same way. For this reason, correspondences must be defined explicitly. Unfortunately, it is often the case that a one-to-one correspondence between all points of all frames does not exist. This occurs, for example, when parts of the cartoon become occluded or revealed in different frames, resulting in topological changes in the drawing.

The second issue is that in cartoons, even if correspondences are known, the difficult inverse problem of recovering a 3D model from its artistic projection needs to be solved in order to use existing texture mapping algorithms. These two key obstacles prevent the widespread usage of texture mapping in traditional cartoon animation. This limits artists to use only simple techniques such as homogeneous colors, flat and static noise textures, or tedious manual drawing and painting of details for every frame.

To tackle these problems, previous methods [Corrêa et al. 1998; Ono et al. 2004; Chen et al. 2005] have focused on user-assisted texturing of cartoons based on an underlying 3D model which is either provided by the user or derived semi-automatically. These approaches require significant user-intervention and only support a narrow class of input shapes. We introduce a new technique for texturing cartoons that is easy to use, requires minimal user effort, and is applicable in a more general context than existing methods based on 3D proxies.

Our work is inspired by techniques for lossy video compression that leverage visual masking [Leung and Taubman 2009]. These approaches exploit the fact that the human visual system (HVS) tends to focus on visually *salient* regions, while devoting significantly less attention to other, less visually important, areas of the scene [Yarbus 1967; Itti 2000]. Our key observation is that for 2D animation, the salient features are the stroke contours, the structure and motion of which the HVS spends most of its time observing. Textures used in cartoons are typically less salient and thus attract considerably less attention [Walther and Koch 2006; Guo et al. 2008] (see Fig. 3). This fact motivated us to produce the illusion of temporal coherence using only rough image registration. Such an approximation can be easily computed by working directly with the 2D cartoons without the need for a corresponding 3D model. This greatly simplifies the task of texturing a hand-drawn animation while still producing visually compelling results.

2 Related work

Previous approaches to cartoon texturing rely on the existence of a 3D proxy. Corrêa et al. [1998] propose a solution that requires the user to provide a 3D model that approximates the geometry depicted in the 2D drawing. In addition, the user must manually specify the correspondence between the model and the drawing. The model is deformed in each frame, such that its projection matches the drawing. The texture coordinates are then transferred from the 3D model to the drawing. Although this method works nicely on simple shapes, it is not applicable to more complicated cartoons, such as those shown in Fig. 1, for which the creation of a 3D proxy model and specification of correspondences would be a time consuming task.

Ono et al. [2004] and later Chen et al. [2005] attempt to mitigate these problems by automating the creation of the 3D model using a sketch-based modeling tool similar to Teddy [Igarashi et al. 1999]. This tool allows the user to quickly create 3D proxy meshes with consistent triangulations by inflating a blobby surface whose boundary points lie on the cartoon's silhouette. However, this technique is applicable only to a limited set of cartoons whose shape can be approximated by inflation. Furthermore, extensive user intervention is still necessary in order to specify feature strokes and their correspondences.

Recently, Winnemöller et al. [2009] proposed a texture design and draping framework which utilizes diffusion curves [Orzan et al. 2008] and parallax mapping [Kaneko et al. 2001] to produce textured hand-drawn images with 3D-like shading and texture rounding effects. Although for static images this approach can provide visually comparable results to ours it is not suitable for hand-drawn cartoon animations as it requires extensive manual intervention when specifying diffusion curves and suffers from texture sliding artifacts caused by parallax mapping.

Our primary goal is to add visual richness to sequences of line drawings. This is similar in concept to non-photorealistic rendering techniques for stylization, such as: coherent dynamic canvas [Cunzi et al. 2003], solid textures [Bénard et al. 2009], dynamic 2D patterns [Breslav et al. 2007] and noise primitives [Bénard et al. 2010]. However, these applications assume that the correspondences are known and thus the registration computation is not required.

2.1 Image registration

A principal aspect of our algorithm is the extraction of dense correspondences between hand-made drawings. There has been a large body of research on image registration in recent years (see [Zitová and Flusser 2003] for a review). However, most of these techniques are not suitable for our task since they model the differences between images based on phenomena common in real world photographs, such as local consistency, projective distortion caused by camera motion, and subtle elastic deformations. Hand-drawn

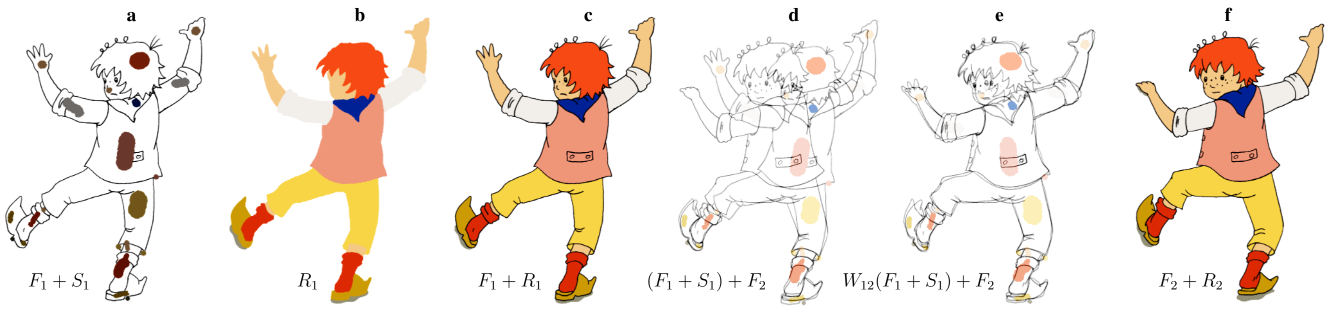

Figure 4: *Automatic painting using LazyBrush: (a) original drawing F_1 with color scribbles S_1, (b) segmentation R_1 of F_1 using scribbles S_1, (c) painted drawing $(F_1 + R_1)$, (d) initial overlap of the source F_1 with scribbles S_1 and the target frame F_2, (e) registration of the source frame F_1 with the target frame F_2, and transfer of scribbles S_1 using as-rigid-as-possible deformation field W_{12}, (f) painted drawing $(F_2 + R_2)$ using transferred scribbles $W_{12}(S_1)$.*

cartoons, on the other hand, exhibit very different properties. For example, they may contain large, non-realistic deformations and abrupt structural changes. Thus, we focus primarily on related work targeted for cartoon drawings.

Early approaches to cartoon registration were motivated by the application of “auto-painting”: transferring color information from painted exemplars to yet unpainted drawings. Many of these techniques segment the input image into a few separate regions and use shape similarity [Madeira et al. 1996], topology [Sýkora et al. 2005], stroke semantics [Kort 2002], hierarchies [Qiu et al. 2005], or skeletons [Qiu et al. 2008] to estimate correspondences between different regions. However, these techniques do not provide the dense correspondences needed for our approach since they are unnecessary for the auto-painting problem.

Another application for cartoon registration is cartoon motion capture, introduced by Bregler et al. [2002]. In this work, they extract a part of a cartoon, analyze its deformation as a non-linear function of a few chosen key frames, and then transfer this deformation to a different shape. Registration is required to track the extracted part across frames. However, because only the contour is tracked, no dense registration can be extracted from this process. This approach has been recently extended by Jain et al. [2009; 2010] who directly use hand-drawn skeletons to transfer artistic motion from a cartoon drawing to a three-dimensional shape. Although these techniques might be helpful for improving a cartoon texturing scheme based on 3D models such as [Corrêa et al. 1998], they are again not suitable for our task.

Shape contexts [Belongie et al. 2002], which have been primarily used for the registration of photographs, have recently been adopted for cartoon drawings. Zhang et al. [2009] apply shape contexts for coherent vectorization of hand-painted cartoon animation, whereas Xu et al. [2008], in an application which is closer in spirit to ours, use shape contexts for creating compelling animal motion from several poses extracted from a single photograph. Unfortunately shape contexts suffer from over-fitting, and thus can easily introduce unnatural distortion when occlusions or topological changes occur.

The most suitable techniques for our application are those that attempt to establish a dense deformation field between images. It has been shown that even a simple affine model can produce a reasonable approximation [Xie 1995]. De Juan and Bodenheimer [2006] use a more flexible free-form deformation model in their framework for segmentation and inbetweening. However, they use an elasticity-based model [Wirtz et al. 2004], which requires manual initialization and parameter tuning to avoid over-fitting.

Temporally coherent painterly renderings in the styles of oil [Hays and Essa 2004] and watercolor [Bousseau et al. 2007] paintings also require dense correspondences. However, such methods use a simple optical flow estimation suitable for temporally smooth image sequences such as live-action videos, but not for hand-drawn cartoon animations where the motion tends to be more rapid and concentrated in areas with outlines. Because the optical flow algorithm relies on tracking features, the absence of texture inside the homogeneous regions of a drawing would pose a considerable challenge.

In summary, most existing registration techniques either target cartoon images but compute only sparse correspondences, create dense correspondences but are only suited to real-world photographs, or are prone to over-fitting. Our framework instead builds upon the recent image registration algorithm proposed by Sýkora et al. [2009a], which is well suited for cartoons, provides dense correspondences, and relies on the robust as-rigid-as-possible deformation model proposed by Alexa et al. [2000].

3 Toon texturing

Our main goal is to generate an animation of “rendered” cartoons from a set of unpainted hand-drawn frames. The “geometry” of the cartoon is given by the hand-drawn strokes, whereas the “appearance” – colors, texture and lighting effects – are added automatically during the “rendering” process.

The input to our system is scanned hand-painted animation frames, hence the first step in our “rendering” pipeline is to assign colors to different parts of the cartoon. To do that we use the LazyBrush algorithm [Sýkora et al. 2009b] which allows an artist to quickly specify desired regions using a small number of color “scribbles”. See Fig. 4a-c for an example of scribble-based painting.

Next, we need to make sure the painting is temporally coherent across all animation frames. This is the well-known “auto-painting” scenario. To avoid specifying the scribbles repeatedly for all frames, we use as-rigid-as-possible (ARAP) image registration [Sýkora et al. 2009a]. This method allows us to register the first frame to the following frame, transfer the color “scribbles”, and finally use the LazyBrush algorithm to obtain the segmentation. See Fig. 4d-f for an example of color transfer. As the LazyBrush algorithm is robust to imprecise positioning of scribbles, small mismatches in the registration are allowed. However, for scenes where detailed painting is required (e.g., many small regions with different colors), the user may need to specify additional correction scribbles to keep the segmentation consistent.

A simple solution to the texturing problem would be to follow a similar route, where instead of specifying a single color value per region, the user would specify a texture. In this case, instead of all

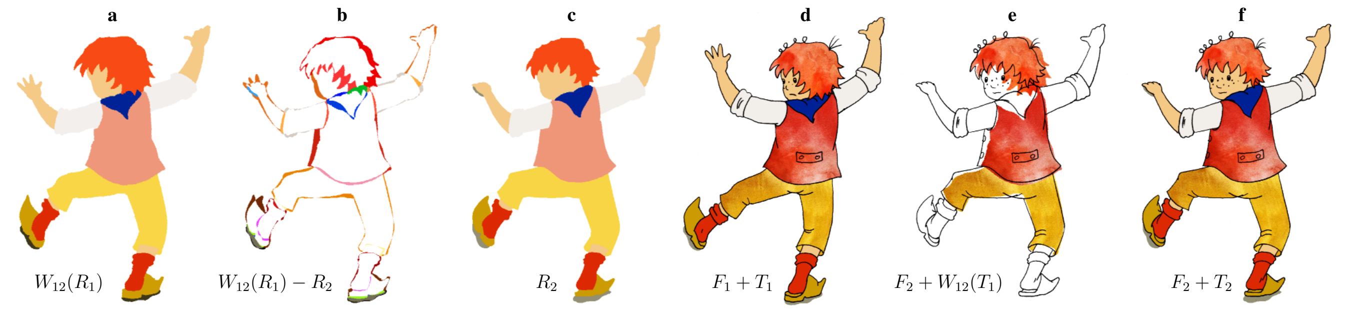

Figure 5: *Domain overlap and texture transfer: (a) segmentation R_1 deformed by W_{12}, (b) domain overlap visualised by symmetric difference of deformed segmentation R_1 and R_2, (c) segmentation R_2, (d) source frame F_1 with textured regions $r_t \in R_1$ mapped using initial texture coordinates T_1, (e) textured regions $r_t \in R_1$ mapped by deformed texture coordinates $W_{12}(T_1)$, (f) target frame F_2 with textured regions $r_t \in R_2$ mapped using deformed and extrapolated textures coordinates T_2,*

pixel values having the same color, the pixel value will be taken from a predefined texture. Since the drawing is two dimensional, we can use the identity as the texture coordinates for the first frame. However, for the texture to avoid the well-known "shower door" effect, the texture coordinates cannot be the identity for all frames: the texture coordinates of a point on any frame should match the coordinates of its corresponding point in the original frame. Our goal is to *deform* the textured region from the original frame to match its corresponding regions in subsequent frames.

To further complicate matters, not every point in a given frame has corresponding points in all other frames due to occlusions and changes of orientation. Fig. 5b shows the symmetric difference between the deformed regions of the source frame and the corresponding regions of the target frame. Notice in particular the differences between the deformed regions of the jacket and hair. Specifying the texture coordinates in the first frame is in fact not enough to generate texture coordinates for all other frames. Hence, we opt for the simple approach of extrapolating the unknown data from the known. Given the texture coordinates which we know (as they were transferred from the source frame), we extrapolate them to compute the texture coordinates in the regions which do not overlap with the transferred regions.

3.1 Algorithm

Our "rendering" pipeline is summarized in Figs. 4 and 5. For rendering a single frame F_1, we need its color scribbles S_1 (Fig. 4a), its segmentation into regions R_1 (Fig. 4b), and the texture coordinates for the textured regions T_1 for every textured region $r_t \in R_1$ (Fig. 5d).

The scribbles S_1 of the starting frame F_1 are supplied by the user, and the texture coordinates T_1 are the identity. Given this information, the frame will be colored and textured as follows: by applying LazyBrush [Sýkora et al. 2009b], every color scribble $s_i \in S_1$ produces a region $r_i \in R_1$, assigning it either a homogeneous color c, or a texture t. The textured region r_t is painted by picking the colors from a predefined texture, using the given texture coordinates T_1.

To render the target frame F_2, given the additional information from the source frame F_1 we proceed as follows. First, we register the frames using [Sýkora et al. 2009a], as shown in Fig. 4d-e. This produces the deformation map W_{12}, which assigns every pixel in R_1 to a pixel location in the frame F_2. Next, we use W_{12} to map the scribbles S_1 to S_2 and generate a segmentation of the target frame R_2 using LazyBrush as shown in Fig. 5c. Finally, Fig. 5e, we use W_{12} again, this time to deform the texture coordinates T_1 of all the textured regions. However, as discussed previously, there is a mismatch between the image of $W_{12}(R_1)$, and its matching segmented regions R_2 (see Fig. 5b). We therefore extrapolate the coordinates $W_{12}(T_1)$ for the region where they are not known, resulting in the new texture coordinates T_2. Now that all the information for the new frame is available to us, we can color and texture it as discussed before (Fig. 5f).

3.2 Implementation details

In this section we describe technical details of ARAP registration and texture coordinate transfer and extrapolation.

ARAP Registration. The registration step is an important part of our pipeline, as its output drives the rest of the components. In general, we follow the original ARAP image registration algorithm [Sýkora et al. 2009a], generating a dense uniform sampling of the source frame, and deforming it in an as-rigid-as-possible manner so that the feature lines align with the target frame. To improve its robustness we compute a distance field from feature lines [Borgefors 1986] and do the registration of the textured regions *separately* which helps to improve the accuracy when topology changes and occlusions occur.

Although in most cases the automatic registration yields good results, sometimes a quick user interaction can improve the accuracy considerably. Thus, we allow the user to pin and drag-and-drop several control points during the automatic registration process to guide the algorithm towards a better result. These control points are easily incorporated into the ARAP deformation scheme by setting high weights for them during the optimization process, effectively causing them to behave as soft or hard positional constraints.

UV Transfer and Extrapolation. The goal in this step is to generate texture coordinates for all of the points inside the textured regions. For that, we require the inverse mapping of all points in our current frame back to the first frame. However, as mentioned previously, the deformation map W_{12} is not onto, hence there are points in the current frame for which we do not have an inverse map. To solve this issue, we first transfer the texture coordinates to the current frame using W_{12}. Then, for each remaining un-mapped point in a textured region $r_t \in R_2$, we compute its texture coordinate value as a linear combination of existing values, using the thin-plate spline interpolation [Bookstein 1989]. To avoid distortions caused by using Euclidean distances, an approximation of geodesic distances can be used [Zhu and Gortler 2007] for which several fast algorithms exist [Yatziv and Sapiro 2006].

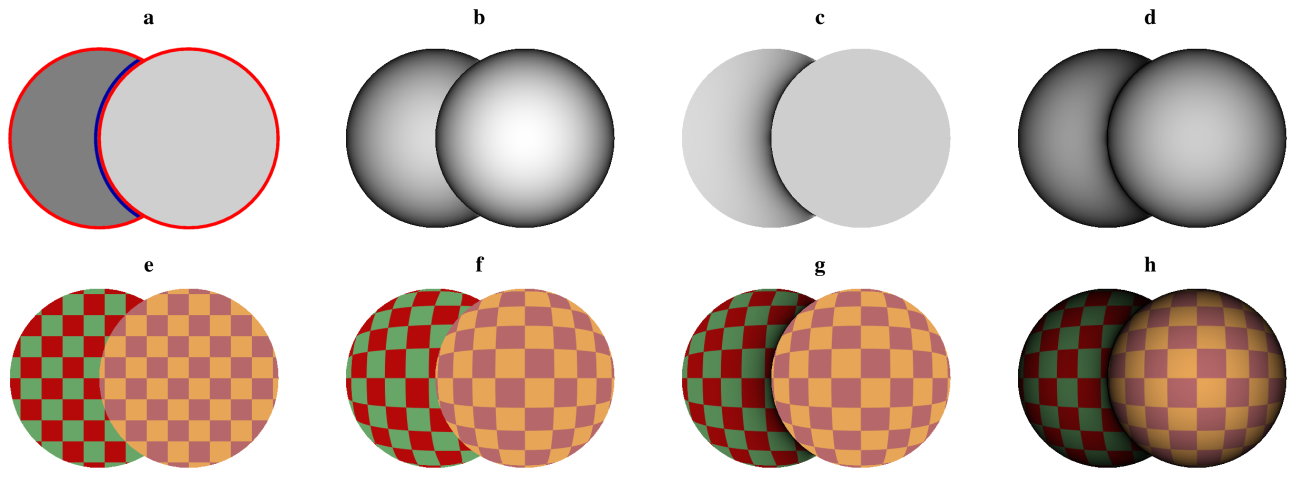

Figure 6: *3D-like effects: (a) depth map with Dirichlet (red) & Neumann (blue) boundary conditions, with initial texture mapping, (b) shading, (c) simulation of ambient occlusion, (d) shading with ambient occlusion, (e) texture mapping using flat UV coordinates, (f) texture rounding based on shading, (g) texture rounding with ambient occlusion, (h) texture rounding with shading & ambient occlusion.*

3.3 Extensions

The presented framework meshes nicely with existing schemes for adding richness to cartoon drawings, and allows for additional extensions and improvements. First, the initial texture need not be a flat one. In fact, as we are basing our framework on an image deformation machinery, it is easy to allow the user to deform the initial texture coordinates. Other variations in color, such as gradients, can be incorporated. A color gradient can be defined using control points which are then transferred using the deformation field W_{12}. In addition to specifying the color scribbles on the first frame, the user may also specify a set of depth (in)equalities (see green and blue arrows in Fig. 10a), which are transformed into a depth map of the image, using the algorithm from [Sýkora et al. 2010]. The resulting depth map can be used for improving the registration by avoiding layering problems and topology variations. The depth values can be easily transferred to the other frames along with the color scribbles, so that they can be further utilized for enhancing the texture with 3D-like effects, as discussed in the following section.

4 3D-like effects

In this section we describe post-processing operations that allow artists to simulate 3D-like effects entirely in the 2D domain, bypassing the need to reconstruct and render a 3D object (see Fig. 7). We first present a new formulation of the popular *Lumo* technique [Johnston 2002] using the depth map generated by [Sýkora et al. 2010] (Fig. 6b) and then show how to exploit such shading to simulate texture rounding (Fig. 6f). In addition, we simulate ambient occlusion effects, as described in [Sýkora et al. 2010] to enhance the perception of depth in the image (see Figs. 6c and 7d).

4.1 Shading

The original Lumo algorithm [Johnston 2002] approximates the normal field inside a region using the 2D normals computed on its boundaries. On the silhouette of an object the normal component in the viewing direction (the z-axis in our case) is 0, hence the normal is completely specified by its x and y components. Furthermore, the gradient of the image intensity is orthogonal to the silhouette, giving exactly the required normal components.

This simple workflow holds when the target shape contains only silhouette pixels. For interior strokes, depth discontinuities should be taken into account to produce convincing results. To address this issue Johnston utilized a manually painted over-under assignment map which is difficult to create and makes the overall process time-consuming. Recently, Sýkora et al. [2010] noted that such a map can be generated automatically from a depth map produced by their algorithm. In this paper we present a new formulation of Lumo which completely avoids this additional step and produces the final normal field in one step.

Our new solution is based on a *homogeneous Laplace* equation with specific boundary conditions defined at depth discontinuities provided by [Sýkora et al. 2010] (see Fig. 6a). For a given pixel, if it is on the boundary of a domain, its depth value is different than its neighbors. If the depth value is larger (Fig. 6a, red curves), it means the pixel lies on the silhouette of the object, and the gradient of the depth map should be used as the local components of the normal. If, on the other hand, the depth value is smaller (Fig. 6a, blue curve), then the pixel lies near the border of an occluding object, and nothing can be said about the values of the normals in that area, except that they should be continuous.

Hence, we can find the normal components by solving the homogeneous Laplace equation:

$$\nabla^2 f = 0 \tag{1}$$

where f is either the x or y component of the normal vector n and the Laplace operator ∇^2 is represented by a sparse matrix L: $L_{ij} = w_{ij}$ for all pixels j in the 4-connected neighborhood $\mathcal{N}_i$ of pixel i, and $L_{ii} = -\sum_{j\in\mathcal{N}_i} w_{ij}$, with $w_{ij} = 1$. As discussed in Section 4.1, the boundary conditions are given by the depth map d (see Fig. 6a):

$$\begin{array}{rlcl} \text{Dirichlet:} & f_p = d'_{pq} & \iff & d_p > d_q \\ \text{Neumann:} & f'_{pq} = 0 & \iff & d_p < d_q \end{array} \tag{2}$$

where q is a neighboring pixel to p, d'_{pq} is the derivative of the depth map at pixel p in the direction pq and f'_{pq} is the derivative of the normal component (n_x or n_y). This leads to a sparse system of linear equations with two different right hand sides (n_x and n_y) for which a fast GPU-based solver exists [Jeschke et al. 2009].

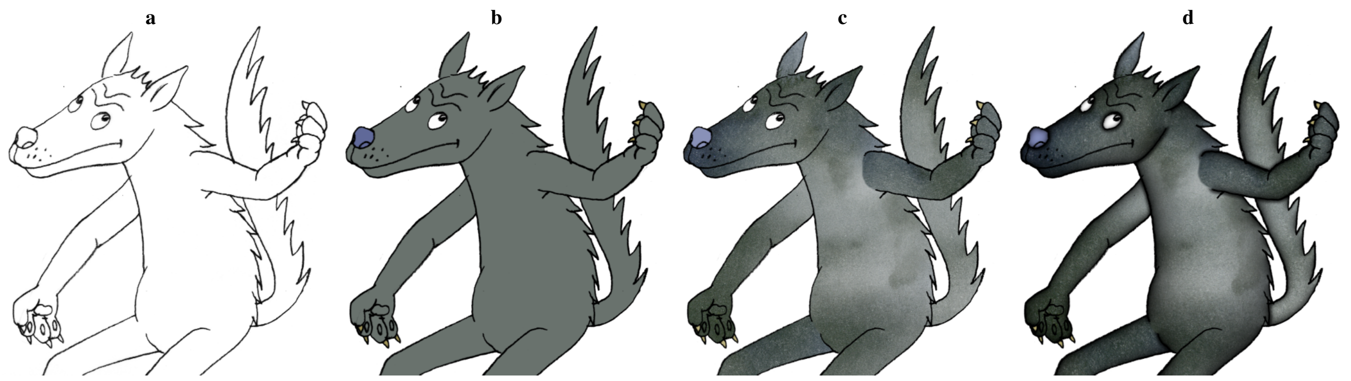

Figure 7: *Adding visual richness to a hand-drawn cartoon: (a) original line drawing, (b) painted with homogenous colors, (c) painted with textures, (d) added texture rounding, shading & simulation of ambient occlusion.*

Given the n_x and n_y components, we can estimate n_z using the sphere equation:

$$n_z = \sqrt{1 - n_x^2 - n_y^2} \tag{3}$$

The computed normal values can be directly used to approximate the shading of a *Lambertian* surface with normals $\vec{n}$, which is illuminated in the direction of the z-axis. More complicated lighting scenarios can be simulated using user-specified environment maps, as in [Johnston 2002].

4.2 Texture rounding

We can further utilize the values of n_z, by reinterpreting them as a height-function h to simulate another 3D effect – texture rounding. When texture mapping is applied to a 3D surface, the curvature of the surface generates an area distortion, effectively causing the texture in curved areas to scale (Fig. 6f). *Parallax mapping* [Kaneko et al. 2001] is one technique for simulating this effect [Winnemöller et al. 2009]. However, parallax mapping does not preserve the original UV coordinates at region boundaries and therefore produces noticeable texture sliding and can easily expose hidden "over-distorted" parts of the texture beyond the region boundaries (see Fig. 8).

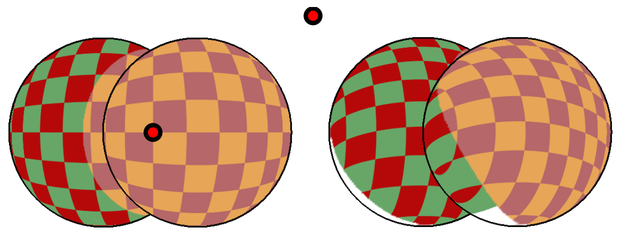

Figure 8: *Parallax mapping failure: resulting texture coordinates are not preserved at region boundaries and depend on the eye position (red dots). This can cause noticeable texture sliding.*

To avoid this artifact we propose a new approach that directly computes texture coordinates on a virtual 3D surface represented as a height function $S = (x, y, h(x, y))$ (see Fig. 9).

This can be done by mapping the boundary of the surface to the plane, and then solving a Laplace equation $\nabla^2_{LB} f = 0$ for the interior values, where ∇^2_{LB} is the Laplace-Beltrami operator. For a surface given as a height function, this operator is given by a matrix with a similar structure to L used in the previous section, but whose weights w_{ij} are different, since we are now measuring distances on the surface S, and not on the plane. However, there is no need to actually construct and flatten the 3D surface S. Instead, this procedure can be seen as solving an *inhomogeneous Laplace* equation on the plane:

$$\nabla^2_w f = 0 \tag{4}$$

where we are simulating the metric of a curved surface by manipulating the weights of the Laplacian matrix. The operator ∇^2_w is the same as the Laplace-Beltrami operator ∇^2_{LB} of S, and we take

$$w_{ij} = \frac{1}{\sqrt{1 + (h_i - h_j)^2}} \tag{5}$$

which is the inverse of the length of the edge connecting the two neighboring vertices i and j on S. This yields another large sparse system of linear equations, now with an irregular matrix and two different right hand sides. It can be efficiently solved using a direct solver such as PARDISO [Schenk and Gärtner 2004]. Interestingly, the inhomogeneous Laplace equation has been used in texture mapping applications such as [Yoshizawa et al. 2004] and [Zayer et al. 2005] to *remove* distortion.

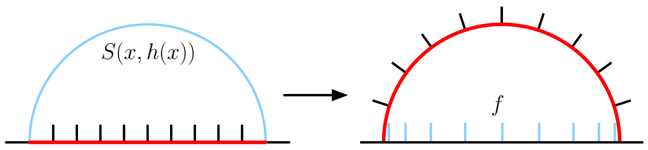

Figure 9: *Texture rounding (1D example): flat texture coordinates and target surface $S = (x, h(x))$ proportional to n_z (left), linearly interpolated texture coordinates on S and their projection back to flat domain f (right).*

Solving Equation (4) using Dirichlet boundary conditions given by the boundary of the domain yields texture coordinates for the (virtual) surface S. These are in fact a map from the plane to the plane, taking a point (x, y) to the texture coordinates of $(x, y, h(x, y))$. By composing this map with the texture coordinates generated in the previous section (Fig. 6e), we can add the required texture distortion (Fig. 6f). Finally, we combine the texture distortion with shading and simulation of ambient occlusion (Fig. 6d, g, h), to achieve a rich 3D-like effect (Fig. 7d).

5 Results

The proposed algorithm has been implemented as a part of a professional cartoon production pipeline and tested by artists on several animation sequences.

A typical workflow preferred by the artists is to paint the sequence first, then apply depth (in)equalities and finally do the texture transfer as a post-process. This helps the artists concentrate on similar tasks: as compared to painting where detailed corrections are sometimes necessary, texturing is much closer to an automatic process and does not require much interaction. Nevertheless, the user inspection and occasional interaction is still beneficial as it can considerably improve the quality of the result, especially when the structure of the target and source frames differs significantly.

Several examples of texture-mapped cartoons, including 3D-like effects are presented in Fig. 10. As can be seen in Fig. 10a, the majority of user interaction is devoted to scribbling and specification of depth (in)equalities. Once this task is completed, the texture transfer becomes a quick operation (Fig. 10b). Note that even if the registration is not accurate, the extrapolation of the UV coordinates keeps the result visually consistent (Fig. 10c, d).

To evaluate our method, we conducted an informal study and showed several textured sequences to numerous uninformed participants (20+) of varying age, sex and background. For textures with highly salient structure, some of the participants reported registration artifacts, however when less salient textures were used, subjects did not notice any inaccuracies in the registration. This implies that our technique will perform well in the common case since artists depict the visually important structure via contours and use less salient textures to add detail to interior regions.

6 Limitations and Future work

Although the proposed method produces convincing results on a variety of cartoon sequences, there are a few limitations we plan to address in future work: (1) Large movements out of the camera plane can be challenging to simulate using our approach. However, in practice this limitation is not as serious as it may seem. When there is some structure on the rotating "surface" (e.g., the belt on the walking boy's jacket in Fig. 10), the algorithm has a proper motion guide and the resulting ARAP deformation and texture rounding effect produce convincing results. This nicely corresponds to the observation made by Breslav et al. [2007] that only approximate 3D coherence is necessary for displaying 2D dynamic patterns on a rotating 3D object. In the case where there is no structure, the algorithm may fail, however the user can always manipulate the ARAP deformation during the registration to simulate this effect manually. (2) Simulation of ambient occlusion might require detailed depth maps, which are somewhat tedious to specify. We plan to incorporate the analysis of junctions to reduce the amount of required user editing. (3) 3D-like shading tends to produce temporal flickering when a part of the character is occluded. We plan to formulate an optimization framework to overcome this. (4) As shown in Fig. 3, not all textures are suitable input for our approach. We plan to use saliency maps generated from a model of the HVS (such as the technique [Walther and Koch 2006] used to generate the maps in Fig. 3 or [Guo et al. 2008]) as a perceptually motivated metric to estimate texture eligibility automatically.

7 Conclusions

We have presented a new approach to texture mapping of hand-drawn cartoon animations requiring considerably less manual work compared to previous techniques. By exploiting the lower sensitivity of the human visual system to motion inconsistency in less salient textured areas, we have shown that even a simple 2D image registration algorithm can produce convincing results, avoiding the creation of proxy 3D models and specification of 2D-to-3D correspondences. We believe our novel approach could motivate other researchers and artists to further develop this emerging visual style to become a standard in the world of computer assisted traditional animation.

Acknowledgements

We would like to thank to all anonymous reviewers for their fruitful comments. Images used in this paper are courtesy of Anifilm studio. Hand-drawn animations and textures were created by Zdena Krejčová and Andrey Iakimov. Scanning, painting, depth (in)equalities, and texture mapping by Ondřej Sýkora. This work has been supported by the Marie Curie action ERG, No. PERG07-GA-2010-268216 and partially by the MŠMT under the research program LC-06008 (Center for Computer Graphics).

References

ALEXA, M., COHEN-OR, D., AND LEVIN, D. 2000. As-rigid-as-possible shape interpolation. In *ACM SIGGRAPH Conference Proceedings*, 157–164.

BELONGIE, S., MALIK, J., AND PUZICHA, J. 2002. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 24, 509–522.

BÉNARD, P., BOUSSEAU, A., AND THOLLOT, J. 2009. Dynamic solid textures for real-time coherent stylization. In *Proceedings of Symposium on Interactive 3D Graphics and Games*, 121–127.

BÉNARD, P., LAGAE, A., VANGORP, P., LEFEBVRE, S., DRETTAKIS, G., AND THOLLOT, J. 2010. A dynamic noise primitive for coherent stylization. *Computer Graphics Forum 29*, 4, 1497–1506.

BOOKSTEIN, F. 1989. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Transactions of Pattern Analysis and Machine Intelligence 11*, 6, 567–585.

BORGEFORS, G. 1986. Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing 34*, 3, 344–371.

BOUSSEAU, A., NEYRET, F., THOLLOT, J., AND SALESIN, D. 2007. Video watercolorization using bidirectional texture advection. *ACM Transaction on Graphics 26*, 3, 104.

BREGLER, C., LOEB, L., CHUANG, E., AND DESHPANDE, H. 2002. Turning to the masters: Motion capturing cartoons. *ACM Transactions on Graphics 21*, 3, 399–407.

BRESLAV, S., SZERSZEN, K., MARKOSIAN, L., BARLA, P., AND THOLLOT, J. 2007. Dynamic 2D patterns for shading 3D scenes. *ACM Transactions on Graphics 26*, 3, 20.

CHEN, B.-Y., ONO, Y., AND NISHITA, T. 2005. Character animation creation using hand-drawn sketches. *The Visual Computer 21*, 8-10, 551–558.

CORRÊA, W. T., JENSEN, R. J., THAYER, C. E., AND FINKELSTEIN, A. 1998. Texture mapping for cel animation. In *ACM SIGGRAPH Conference Proceedings*, 435–446.

CUNZI, M., THOLLOT, J., PARIS, S., DEBUNNE, G., GASCUEL, J.-D., AND DURAND, F. 2003. Dynamic canvas for non-photorealistic walkthroughs. In *Proceedings of Graphics Interface*, 121–130.

GUO, C., MA, Q., AND ZHANG, L. 2008. Spatio-temporal saliency detection using phase spectrum of quaternion fourier transform. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*.

HAYS, J., AND ESSA, I. A. 2004. Image and video based painterly animation. In *Proceedings of International Symposium on Non-Photorealistic Animation and Rendering*, 113–120.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3D freeform design. In *ACM SIGGRAPH Conference Proceedings*, 409–416.

ITTI, L. 2000. *Models of Bottom-Up and Top-Down Visual Attention*. PhD thesis, California Institute of Technology.

JAIN, E., SHEIKH, Y., AND HODGINS, J. K. 2009. Leveraging the talent of hand animators to create three-dimensional animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 93–102.

JAIN, E., SHEIKH, Y. A., MAHLER, M., AND HODGINS, J. K. 2010. Augmenting hand animation with three-dimensional secondary motion. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 93–102.

JESCHKE, S., CLINE, D., AND WONKA, P. 2009. A GPU Laplacian solver for diffusion curves and Poisson image editing. *ACM Transaction on Graphics 28*, 5, 116.

JOHNSTON, S. F. 2002. Lumo: Illumination for cel animation. In *Proceedings of International Symposium on Non-photorealistic Animation and Rendering*, 45–52.

DE JUAN, C. N., AND BODENHEIMER, B. 2006. Re-using traditional animation: methods for semi-automatic segmentation and inbetweening. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 223–232.

KANEKO, T., TAKAHEI, T., INAMI, M., KAWAKAMI, N., YANAGIDA, Y., MAEDA, T., AND TACHI, S. 2001. Detailed shape representation with parallax mapping. In *Proceedings of International Conference on Artificial Reality and Telexistence*, 205–208.

KORT, A. 2002. Computer aided inbetweening. In *Proceedings of International Symposium on Non-photorealistic Animation and Rendering*, 125–132.

LEUNG, R., AND TAUBMAN, D. 2009. Perceptual optimization for scalable video compression based on visual masking principles. *IEEE Transactions on Circuits and Systems for Video Technology 19*, 309–322.

MADEIRA, J. S., STORK, A., AND GROB, M. H. 1996. An approach to computer-supported cartooning. *The Visual Computer 12*, 1, 1–17.

ONO, Y., CHEN, B.-Y., AND NISHITA, T. 2004. 3D character model creation from cel animation. In *Proceedings of International Conference on Cyberworlds*, 210–215.

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: A vector representation for smooth-shaded images. *ACM Transactions on Graphics 27*, 3, 92.

QIU, J., SEAH, H. S., TIAN, F., CHEN, Q., AND WU, Z. 2005. Enhanced auto coloring with hierarchical region matching. *Computer Animation and Virtual Worlds 16*, 3–4, 463–473.

QIU, J., SEAH, H. S., TIAN, F., CHEN, Q., WU, Z., AND MELIKHOV, K. 2008. Auto coloring with enhanced character registration. *International Journal of Computer Games Technology*, 1, 2.

SCHENK, O., AND GÄRTNER, K. 2004. Solving unsymmetric sparse systems of linear equations with PARDISO. *Journal of Future Generation Computer Systems 20*, 3, 475–487.

SÝKORA, D., BURIÁNEK, J., AND ŽÁRA, J. 2005. Colorization of black-and-white cartoons. *Image and Vision Computing 23*, 9, 767–782.

SÝKORA, D., DINGLIANA, J., AND COLLINS, S. 2009. As-rigid-as-possible image registration for hand-drawn cartoon animations. In *Proceedings of International Symposium on Non-photorealistic Animation and Rendering*, 25–33.

SÝKORA, D., DINGLIANA, J., AND COLLINS, S. 2009. LazyBrush: Flexible painting tool for hand-drawn cartoons. *Computer Graphics Forum 28*, 2, 599–608.

SÝKORA, D., SEDLACEK, D., JINCHAO, S., DINGLIANA, J., AND COLLINS, S. 2010. Adding depth to cartoons using sparse depth (in)equalities. *Computer Graphics Forum 29*, 2, 615–623.

WALTHER, D., AND KOCH, C. 2006. Modeling attention to salient proto-objects. *Neural Networks 19*, 9, 1395–1407.

WINNEMÖLLER, H., ORZAN, A., BOISSIEUX, L., AND THOLLOT, J. 2009. Texture design and draping in 2D images. *Computer Graphics Forum 28*, 4, 1091–1099.

WIRTZ, S., FISCHER, G., MODERSITZKI, J., AND SCHMITT, O. 2004. Superfast elastic registration of histologic images of a whole rat brain for 3d reconstruction. In *Proceedings of the SPIE*, vol. 5370, 328–334.

XIE, M. 1995. Feature matching and affine transformation for 2D cell animation. *The Visual Computer 11*, 8, 419–428.

XU, X., WAN, L., LIU, X., WONG, T.-T., WANG, L., AND LEUNG, C.-S. 2008. Animating animal motion from still. *ACM Transactions on Graphics 27*, 5, 117.

YARBUS, A. L. 1967. *Eye Movements and Vision*. Plenum Press.

YATZIV, L., AND SAPIRO, G. 2006. Fast image and video colorization using chrominance blending. *IEEE Transactions on Image Processing 15*, 5, 1120–1129.

YOSHIZAWA, S., BELYAEV, A., AND PETER SEIDEL, H. 2004. A fast and simple stretch-minimizing mesh parameterization. In *Proceedings of the Shape Modeling International*, 200–208.

ZAYER, R., ROSSL, C., AND PETER SEIDEL, H. 2005. Discrete tensorial quasiharmonic maps. In *Proceedings of the International Conference on Shape Modeling and Applications*, 276–285.

ZHANG, S.-H., CHEN, T., ZHANG, Y.-F., HU, S.-M., AND MARTIN, R. R. 2009. Vectorizing cartoon animations. *IEEE Transactions on Visualization and Computer Graphics 15*, 618–629.

ZHU, Y., AND GORTLER, S. J. 2007. 3d deformation using moving least squares. Tech. Rep. TR-10-07, Harvard Univeristy.

ZITOVÁ, B., AND FLUSSER, J. 2003. Image registration methods: A survey. *Image and Vision Computing 21*, 11, 977–1000.

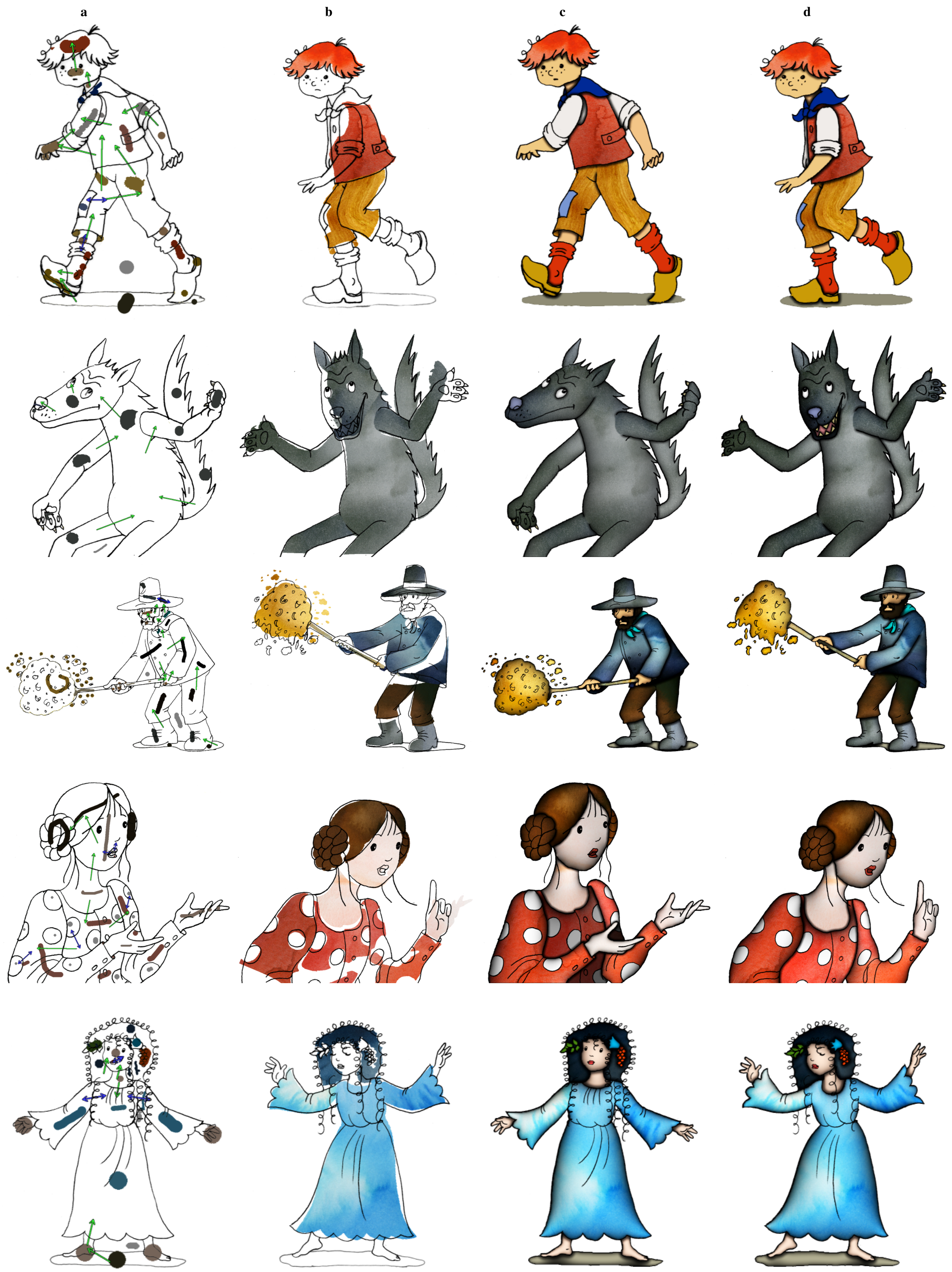

Figure 10: *Results: (a) source frame with user-defined color/texture scribbles and depth (in)equalities, (b) rough texture transfer to the target frame, (c, d) textured source & target frame enhanced using a combination of selected 3D-like effects.*