Habilitation Thesis

Computational Hand-Drawn Animation

Daniel Sýkora

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Graphics and Interaction
Karlovo nám. 13, 12135 Praha 2

Preface

This thesis presents nine research papers which I created with my collages during past five years. Seven papers directly address topics closely related to computational hand-drawn animation (segmentation, depth assignment, registration, texture mapping, 3D-like shading, and temporal noise control). There are also two miscellaneous papers which go slightly beyond the scope of this thesis and presents new results with a more general utility (efficient computation of minimal cuts and realistic example-based image synthesis). Their development was initiated by the research done on hand-drawn images and they also have direct applications in the field of cartoon animation. Four of the presented papers were published in impacted international journals, one paper have been recently accepted for a journal publication and will be in print in the next months. Four other papers were published at established conferences in the field.

The thesis contains an introductory part followed by a brief overview of basic algorithms, their applications, miscellaneous techniques, and concludes with possible new avenues for future work. Reprints of mentioned papers are presented in appendices.

Prague, February 19th, 2014 Daniel Sýkora

Acknowledgements

First of all I would like to thank to all my colleagues who contributed to the papers presented in this thesis (in alphabetic order): Jean-Charles Bazin, Mirela Ben-Chen, Steven Collins, Stelian Coros, Martin Čadík, John Dingliana, Jakub Fišer, Marcus Gross, Alec Jacobson, Ondřej Jamriška, Sun Jinchao, Ladislav Kavan, Michal Lukáč, Gioacchino Noris, David Sedláček, Maryann Simmons, Alexander & Olga Sorkine-Hornung, Robert Sumner, and Brian Whited. I also want to express my gratitude to Tomáš Rychecký from Anifilm studio, Vít Komrzý from Universal Production Partners as well as Maurizio Nitti from Diseny Reserach Zurich, Lukáš Vlček, Ondřej Sýkora, and Kristýna Mlynaříková for providing beautiful hand-drawn artwork which was always stimulating motivation for my research. Big thanks flies to Jiří Žára head of the Department of Computer Graphics and Interaction at CTU in Prague for creating a stimulating environment for research and education and for his encouragement to finalize this thesis. I also should not forget to mention Jiří Bittner and Vlastimil Havran who helped me a lot with perparation of this thesis and to all my colleagues at the Department of Computer Graphics and Interaction for fruitful discussions and support. Last but not least I want to thank my wife Pavla, daughters Štěpánka & Jolanka as well as step-sons Mikuláš & Matyáš for their constant support and endless patience.

Contents

Chapter 1

Introduction

Paper and pencil are the only tools a skilled artist needs to create a fascinating cartoon animations. With these the artist has a complete freedom which is tempered by the effort and time needed to complete the artwork especially in the case of colorful animation where hundreds of painted drawings are required.

Recently, CG animation systems have become very popular as they can save a great deal of manual work. Their key advantage is that they have internal representation of the structure and motion of animated objects, therefore the final artwork is created by an automated rendering algorithm without any additional effort. As a result, everything can be easily manipulated and modified. However, the compromise is that the artist loses a part of their freedom and expressivity.

In this thesis we present a set of new algorithms that enable modification, manipulation, and rendering similar to what can be achieved with CG animation systems, whilst preserving the expressivity and simplicity of the original hand-drawn animation. To achieve this, it is necessary to infer a part of the structural information hidden in the sequence of hand-drawn images, namely the partitioning into meaningful segments, their topology variations, depth ordering, and correspondences. Such inference can be very ambiguous and cannot be fully automated, therefore we let the artist provide a couple of rough hints that make this problem tractable.

The rest of the thesis is organized as follows. First we introduce a set of basic algorithms (Chapter 2) that help to introduce unknown structural information into a sequence of hand-drawn images and then we briefly demonstrate how this additional knowledge can help to solve practical tasks (Chapter 3) such as autopainting, temporally coherent texture mapping, example-based shape deformation, simulation of 3D-like effects, or temporal noise control. Finally, we describe two miscellaneous techniques (Chapter 4) that can improve performance of algo-

rithms presented in Chapter 2 and create visually rich example-based content for applications described in Chapter 3.

Chapter 2

Basic algorithms

In this chapter we introduce algorithms that enable to quickly partition the input drawing into a set of meaningful parts (Section 2.1), assign unknown depth information (Section 2.2), and retrieve correspondences between individual animation frames (Section 2.3).

2.1 Segmentation

In this section we briefly describe new algorithms we developed to allow for quick and accurate segmentation of hand-draw images: *LazyBrush* and *Smart Scribbles*.

2.1.1 LazyBrush

LazyBrush algorithm was developed to segment scanned hand-drawn images as well as rough digital sketches. It does not rely on a specific drawing style and can deliver clean segmentation with much less manual effort as compared to previous techniques based on flood-filling strategies. It overcomes typical issues such as leakage through gaps, anti-aliased outlines, or large number of small regions. It is also not sensitive to imprecise placement of segmentation strokes (scribbles) which makes the process less tedious and brings significant time savings when applied to animation. The segmentation of hand-drawn images is formulated as a discrete energy minimization problem which is in general NP-hard, however, we proposed an efficient approximative solution that facilitates a sequence of minimal cuts on a set of gradually reducing graphs. With this approach the computational overhead is much lower while the solution is visually comparable to a

more computationally demanding optimization techniques. Details can be found in Appendix A.

2.1.2 Smart Scribbles

Smart Scribbles algorithm was tailored to handle segmentation of hand-drawn digital sketches. Its primary goal is to cluster individual strokes and then use these clusters to encompass solid regions. It is based on a similar energy minimization framework as used in *LazyBrush* algorithm, however, instead of working on pixels it performs labelling directly on individual strokes which allows for more complex clustering. Another key advantage is that the algorithm takes into account also temporal and geometric information from the digital input. The observation here is that strokes drawn at a certain period of time typically correlates with a specific semantically important cluster and thus can help to produce meaningful segmentation even in cluttered scenes. Moreover, for the selection stroke also the orientation, curvature, and locality is considered so that artists can better express their intention to select strokes with specific directional and spatial properties. A user study was conducted to compare *Smart Scribbles* with common selection tools used in professional systems. The results demonstrate that our approach makes the selection process less tedious and notably faster. Details can be found in Appendix B.

2.2 Adding depth

In this section we briefly describe two algorithms we developed to add unknown depth information into hand-drawn images: *LazyDepth* and *Ink-and-Ray*.

2.2.1 LazyDepth

Perceptual studies show that for the human visual system the specification of absolute depth values in the scene is a difficult task while contrary binary decision whether some part of the scene is closer than another is much easier. Based on this observation we developed a novel depth assignment approach which does not require the user to specify absolute depth values while instead uses a set of sparse depth inequalities that express pairwise relationship between selected parts in the scene. To solve for absolute depth values we then formulated an optimization problem based on quadratic programming (QP) which enforces user-specified

depth inequalities while taking into account smoothness of the resulting depth field that is driven by the intensity in the input drawing. Since solving QP problems is a computationally demanding task we proposed an approximative scheme which decomposes the original QP problem into three simplified sub-problems that can be solved quickly: pre-segmentation, topological sorting, and depth interpolation. Such decomposition allows to deliver interactive responses and enables users to incrementally improve the solution. Details can be found in Appendix C.

2.2.2 Ink-and-Ray

One of the key limitations of the *LazyDepth* algorithm is that it produces only 2.5D flat piecewise continuous height field with arbitrary depth discontinuities. Although this simplified representation is already suitable for simulation of various 3D-like effects (see Chapter 3) it is not sufficient for more complex global illumination effects such as self-shadowing, color bleeding, or glossy reflections. To address this limitation we developed an extension of the *LazyDepth* algorithm which allows for a quick semi-automatic creation of smoothly interconnected stack of inflated layers that can mimic structure of bas-relief sculptures. A new type of optimization problem was formulated which combines inequality constraints with inflation. Moreover, it takes into account automatic estimation of relative depth order as well as reconstruction of occluded parts which considerably lower the number of required user interactions. Resulting proxy 3D mesh provides much richer geometric information sufficient to evoke impression of fully consistent 3D model rendered from orthographic view using global illumination algorithm. This was verified by a perceptual experiment which demonstrated that for observers without prior experience with computer graphics there is no statistically significant difference between a real 3D model and our approximation. Details can be found in Appendix D.

2.3 Registration

Estimation of correspondences between individual hand-drawn images is a challenging task. The key issue here is that individual animation frames are drawn from scratch and since typically a lower frame rate is used they undergo a large amount of free-form deformation as well as notable change in overall appearance. Popular computer vision techniques often fail in such scenario as they rely on unique local features or stable global configurations which are typical for real world photos but rare in hand-drawn images. Although state-of-the-art

deformable image registration approaches allow for retrieval of correspondences in presence of free-form deformations, they become computationally intractable for larger displacements due to exponentially increasing state space. We proposed a novel solution which uses popular as-rigid-as-possible deformation model (ARAP) that respects local rigidity as well as articulation of the deformed shape. The method iterates over two basic steps: (1) block matching algorithm that shifts selected points on the source shape so that their new position reduces local visual dissimilarity between the source and target images and (2) ARAP regularization that keeps the overall shape consistent. Thanks to robustness of ARAP model and capability of block matching algorithm to retrieve globally optimal shifts in a small neighbourhood the resulting algorithm yields state-of-the-art results when registering hand-drawn animation frames undergoing large free-form deformations as well as changes in appearance. Details can be found in Appendix E. For cases when parts for the source shape are occluded or glued together depth assignment algorithm presented in Appendix C can be utilized to improve accuracy of the registration.

Chapter 3

Applications

Techniques described in previous chapter can be utilized as basic building blocks for various practical applications that can bring ease of modification and manipulation from CG pipelines into the world of traditional hand-drawn animation.

3.1 Painting

A first straightforward application of segmentation and registration is *painting* or *colorization*. Here desired colors or color components are assigned to the resulting segments and, in each pixel, multiplied/combined with the original gray-scale intensity. To avoid repeated specification of selection strokes in all animation frames, proposed ARAP image registration scheme (Appendix E) can be utilized to register the first frame to the following frame, transfer the scribbles, and use the LazyBrush algorithm (Appendix A) to obtain the segmentation. 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.

3.2 Texture mapping

Instead of a single color, the user may also specify a texture and make the region filling more visually rich. However, in contrast to a single color there is an additional problem: the texture should follow the motion and/or deformation of its

corresponding regions in the subsequent frames to preserve temporal coherency. This can be problematic in hand-drawn animation as it is typically impossible to obtain one-to-one correspondence between individual frames. Fortunately, the human visual system tends to focus more on visually salient regions, while devoting significantly less attention to other, less visually important, areas. In hand-drawn animations contours are the salient features while textures are typically less salient and thus attract considerably less visual attention. By exploiting this property, an illusion of temporally coherent animation can be achieved using only rough correspondences obtained by ARAP image registration algorithm. This enables production of hand-drawn animations that convey visual richness of fully hand-colored artwork. Details can be found in Appendix F.

3.3 3D-like effects

The knowledge of segmentation, Section 2.1, and approximate depth information, Section 2.2, opens a potential to simulate 3D-like effects typical for CG pipelines:

Ambient occlusion. A popular technique that can approximate smooth light attenuation on diffuse surfaces caused by occlusion. Its key advantage is that it can enhance the perception of depth in the image. In our setting this effect can be simulated by superimposing a stack of regions with blurred boundaries in a back-to-front order (details can be found in Appendix F).

Shading. A simple 3D-like shading effect can be achieved by computing an approximation of a normal field inside each region. Environment mapping can then be used to map normal coordinates into an environment map where proper color information is retrieved. To obtain the normal field 2D normals computed on silhouettes can be interpolated inside the region while taking into account position of occlusion boundaries, i.e., boundaries of regions which have lower depth values. The problem can be formulated as a solution to Laplace equation with proper boundary conditions (Dirichlet on silhouettes and Neumann on occlusion boundaries). See Appendix F for further details.

Texture rounding. Estimated normal field can further be utilized to simulate 3D texture rounding effect, i.e., when the curvature of the surface generates an

area distortion and causes the texture to scale. This can be achieved by solving inhomogeneous Laplace equation with Laplace-Beltrami operator which measures real distances on the approximated surface (see Appendix F for details).

Stereo. The knowledge of depth information can also be useful to render pairs of images with different disparity for stereoscopic displays. In this case texture mapping can further improve the stereo effect as subtle structural details present in textures can help the human visual system to better estimate disparity and so improve the perception of depth in the scene (see an example in Appendix C).

Global illumination. With *Ink-and-Ray* algorithm (Appendix D) ambient occlusion as well as 3D-like shading effects can be computed more accurately using advanced light transport simulation algorithms (we use bidirectional path tracing). Thanks to this physically correct solution more complex global illumination effects such as self-shadowing, color bleeding, or glossy reflection can be achieved.

3.4 Shape manipulation

Depth maps generated by the algorithm described in Appendix C can be further utilized to resolve visibility of occluded parts during interactive shape manipulation. The user can freely interact with the shape and modify the visibility on the fly using additional depth inequalities. A similar problem can arise in systems where the user extracts and composes fragments of images. Here depth inequalities allow quick reordering of regions to obtain correct composition. Moreover, correspondences between consecutive animation frames allow for creation of smooth intermediate transitions that can be obtained by interpolating positions of individual points and performing several shape regularization iterations to enforce rigidity. The process of smooth inbetweening can further be controlled by the user. This yields an example-based shape manipulation technique which respects the original animation. The user can drag a specific vertex on the control lattice and move it to a different location. By projecting this new location on its inbetweening trajectory we can generate the closest transition frame and deform it to match the user-specified constraint, for details please refer to Appendix E.

3.5 Temporal noise control

A well-knonw issue in traditional hand-drawn animation is that when individual rough sketches of animation frames are played at a desired frame rate the resulting animation exhibits temporal noise that can significantly affect the viewing comfort and thus only production of short animation clips is tractable. For longer sequences clean-up frames need to be created manually to avoid this drawback. However, this cleaning step unfortunately suppresses visual richness and expressiveness of the original animation. To reduce the amount of temporal noise while preserving the expressiveness of the original artwork we applied our ARAP image registration algorithm (Appendix E) to estimate correspondences between individual frames and then proposed a novel interpolation scheme that enables control over the amount perceived temporal noise. Besides improving viewing comfort such manipulation can also provide additional artistic parameter to emphasize emotions as well as overall scene atmosphere. Details can be found in Appendix G.

Chapter 4

Miscellaneous algorithms

4.1 GridCut

GridCut algorithm was developed to further speed-up computation of minimal cuts on graphs with grid-like topology such as those used in *LazyBrush* algorithm (Appendix A) as well as in other computer graphics/vision problems including stereo, shape reconstruction/fitting/registration, video editing/synthesis, or pose estimation. It uses novel cache efficient scheme which substantially outperforms current state-of-the-art max-flow/min-cut solvers both in computational overhead and memory consumption. According to measurements performed on a comprehensive benchmark *GridCut* is currently the fastest max-flow/min-cut solver on the CPU for grid-like graphs emerging in mentioned computer graphics/vision problems. Details can be found in Appendix H.

4.2 Painting by Feature

TexToons described in Appendix F require specification of textures to fill regions delineated by hand-drawn contours. They can be created by hand and scanned, however, this can be a tedious process as it requires to work with real drawing medium. A more practical solution would be to have a database of reusable textural samples that can be directly applied. However, a problem can arose that in such database only a limited number of samples exists which may not cover all artistic needs. To increase variability and provide artistic control over the reusing process we proposed a novel example-based image synthesis algorithm that enables artists to paint in the visual style of the given example of drawing medium. They can use

entire textural examples of physical drawing medium as a palette, from which they select linear as well as areal structures and combine them seamlessly into a new textural image that on the local level preserves visual richness of the given example image while on the global level respects prescribed structural properties. A key improvement over previous example-based image synthesis techniques is that in our approach we propose a novel strategy where salient texture boundaries are synthesised independently by a randomized graph-traversal algorithm and then content-aware texture synthesis is applied to transfer textural information into the delimited regions. Since textural boundaries are prominent for the human visual system their proper synthesis notably improves visual fidelity of the resulting image. Details can be found in Appendix I.

Chapter 5

Conclusion and Future Work

In this thesis we presented techniques which enable usage of concepts from CG pipelines in the world of traditional hand-drawn animation. Using our tools artists can easily manipulate, modify, and enahce existing artwork while still retaining its hand-drawn nature. This opens a viable potential to deliver a fresh new look that may become an alternative to purely CG-based approaches.

Work on the presented papers reveals a vast pool of possibilities for further improvements. In the *LazyBrush* algorithm despite of the usage of fast *GridCut* solver still the performance is a limitation for larger resolutions (4K). Here some additional graph reduction techniques may improve the processing speed notably and allow for fully interactive response. The same limitation holds also for *Smart Scribbles* algorithm where the general graph structure is used for computation of minimal cuts and thus *GridCut* solver cannot be applied. Performance is issue also in *ARAP image registration* where, e.g., a multi-resolution scheme could help to lower computational overhead. This approach can also help to improve accuracy of the registration as finer grid is needed to reach pixel-level precision. Unfortunately, performance decreases significantly with increasing number of control points and thus some solution need to be found to keep the method tractable. In *LazyDepth* algorithm some additional image-based cues (such as T-junctions) can be utilized to predict depth inequalities. However, this automatic estimation introduces a problem of inaccuracies that may cause cycles in the depth order which should be resolved automatically. For this purpose a robust variant of topological sorting algorithm need to be developed. A key limitation of *TexToons* algorithm are motions out of camera plane including character rotation or scale changes. These cannot be simply handled by ARAP deformation model and thus may lead to disturbing shower door effect. Scaling can be partially resolved by replacing ARAP with as-similar-as-possible model, however, this model is not as robust as

ARAP and thus some additional constraints need to be specified and integrated into the algorithm. Off-plane rotations are challenging problem since they typically cannot be detected without integrating additional motion cues from different parts of the character. Therefore deeper understanding of global motion characteristics is necessary. In *Ink-and-Ray* framework processing speed is also one of the main issues. Here QP solver is applied only on mesh vertices to reduce the computational overhead, nevertheless, it is still far from interactive response. A better solution would be to use even more compressive 3D representation (e.g., distance fields or parametric surfaces) to further simplify calculations and deliver results at interactive rates. Another limitation of *Ink-and-Ray* is that currently each animation frame is processed independently. This may cause flickering in more complex animations. An extension of ARAP registration to 3D may help to establish rough correspondences and help to introduce additional constraints to enforce temporal coherency. Finally, *Painting by Feature* algorithm can be extended to produce animation sequences with controllable amount of perceived temporal noise. Also the whole interaction process can be simplified so that the user will draw only linear structures and then the algorithm picks corresponding textures for the content-aware fill automatically.

Appendices – Paper Reprints

Appendix A

LazyBrush: Flexible Painting Tool for Hand-drawn Cartoons

D. Sýkora, J. Dingliana, S. Collins: LazyBrush: Flexible Painting Tool for Hand-drawn Cartoons. Computer Graphics Forum, vol. 28, no. 2, pp. 599–608, March 2009. ISSN 0167-7055. **IF=1.638**

EUROGRAPHICS 2009 / P. Dutré and M. Stamminger
(Guest Editors)

Volume 28 (2009), Number 2

LazyBrush: Flexible Painting Tool for Hand-drawn Cartoons

Daniel Sýkora†, John Dingliana, and Steven Collins

Trinity College Dublin

Abstract

In this paper we present LazyBrush*, a novel interactive tool for painting hand-made cartoon drawings and animations. Its key advantage is simplicity and flexibility. As opposed to previous custom tailored approaches [SBv05, QWH06]* LazyBrush *does not rely on style specific features such as homogenous regions or pattern continuity yet still offers comparable or even less manual effort for a broad class of drawing styles. In addition to this, it is not sensitive to imprecise placement of color strokes which makes painting less tedious and brings significant time savings in the context cartoon animation.* LazyBrush *originally stems from requirements analysis carried out with professional ink-and-paint illustrators who established a list of useful features for an ideal painting tool. We incorporate this list into an optimization framework leading to a variant of Potts energy with several interesting theoretical properties. We show how to minimize it efficiently and demonstrate its usefulness in various practical scenarios including the ink-and-paint production pipeline.*

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.4]: Graphics Utilities—Graphics editors, Image Processing and Computer Vision [I.4.6]: Segmentation—Pixel classification, Computer Applications [J.5]: Arts and Humanities—Fine arts

1. Introduction

Painting, i.e. the process of adding colors to hand-made drawings, is a common operation in standard image manipulation programs starting from simple bitmap editors such as *Paintbrush* to professional digital ink-and-paint solutions like *Animo*, *Toonz*, or *Retas*. In these systems a variant of the flood-fill algorithm is typically used to speed up painting. This algorithm works well for images with homogenous regions and salient continuous outlines. However, many hand-made drawing styles contain more complicated structures (e.g. pencil drawing in Figure 1). For such images it is necessary to perform many detailed manual corrections to get clean results. This additional effort can be very time consuming and cost ineffective in the context of the ink-and-paint pipeline where thousands of frames must be painted.

Recently, significant effort has been devoted to a similar problem – the interactive colorization of gray-scale images [LLW04, YS06]. Although these approaches offer fascinating results on natural photographs and videos, they typically fail when applied to hand-made drawings which do not preserve a smooth image model (see Figure 2). Sýkora et al. [SBv05] addressed this issue by developing an unsupervised segmentation algorithm for black-and-white cartoon animations able to produce segmentation similar to that produced by *connected component analysis* [RK82] on a binary image. The main drawback of their approach is the assumption of large homogenous regions enclosed by distinct continuous outlines. When applied to more complicated styles, they tend to group salient regions due to gappy outlines or produce many small regions (see Figure 2).

Qu et al. [QWH06] proposed manga colorization framework that overcomes forementioned limitations by exploiting both pattern and intensity continuity in conjunction with a level-set optimization. According to user-specified examples of hatching patterns, they extract textural features and compute a similarity map having an intensity profile like a homogeneous region with distinct boundaries. Subsequently they propagate colors from user-specified scribbles until they reach salient barriers. During the propagation they also employ shape regularization to overcome possible leakage through gappy boundaries. Despite the success of this ap-

† e-mail: sykorad@cs.tcd.ie

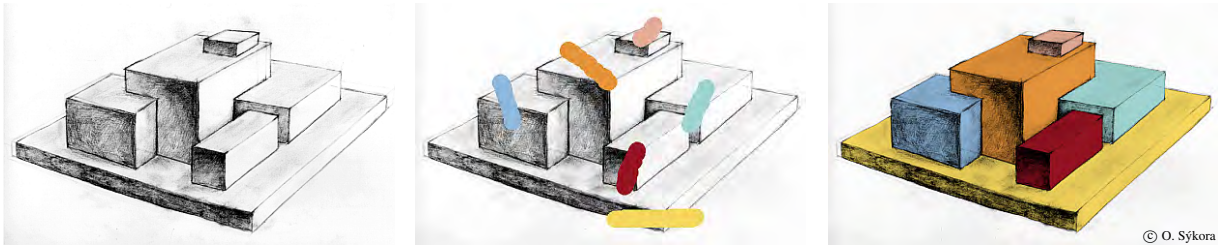

Figure 1: *LazyBrush in action – minimal effort is needed to paint this highly structured pencil drawing with fuzzy outlines and shaded regions (left). See how the algorithm handles imprecise placement of color strokes (middle) and is able to produce high quality anti-aliased output (right).*

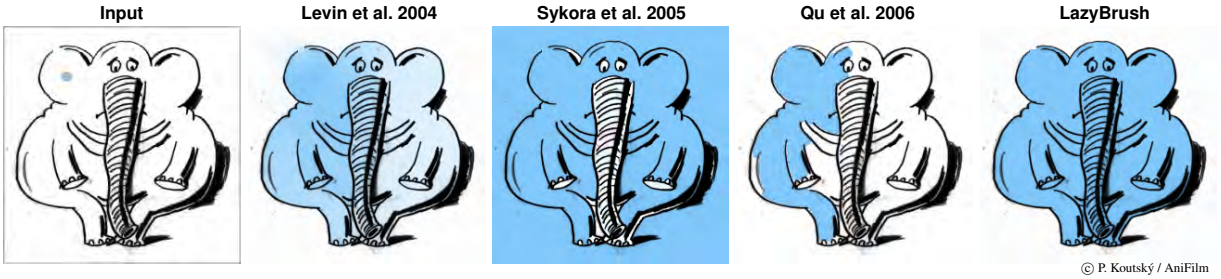

Figure 2: *LazyBrush vs. state-of-the-art – various algorithms applied on the same input data (background seeds around the image border and blue seed inside the elephant's ear): Levin et al. [LLW04] assume improper image model, Sýkora et al. [SBv05] do not handle gaps and produce many small regions, and Qu et al. [QWH06] get stuck in inappropriate local minima so that all remaining regions should be filled individually. In contrast to this, LazyBrush finds an optimal boundary and does not require further effort.*

proach, many important issues remain. Since the level-set optimization is based on gradient descent it can easily get stuck in some inappropriate local minima. This typically occurs when the algorithm is used for images which do not contain repetitive hatching patterns (see Figure 2). In this case the user has to specify many additional scribbles or tweak parameters of level-set optimization to allow crossing salient boundaries during front propagation. Another problem occurs when narrow or small regions are painted. Also in this case many thin scribbles must be drawn and parameters tweaked to achieve desired results. These limitations hinder the practical usability of manga colorization for images which do not contain repetitive patterns.

The aim of this paper is to present a novel flexible painting tool easily applicable to various drawing styles. We demonstrate an approach that is independent of style-specific features but, despite this, requires comparable or less manual effort than previous style-limited approaches. Our key contribution is hidden in a list of previously undiscussed properties presented in Section 3 which redefines behavior of an ideal painting tool. This list arose from a requirements analysis carried out with professional ink-and-paint illustrators. We reformulate it as an energy optimization problem and obtain an interesting and, to our knowledge, unexplored variant of energy function with Potts interaction [Pot52] and special sparse data term. We discuss its interesting theoretical properties and present an efficient approximation algorithm requiring only a few globally optimal decisions to obtain a nearly optimal solution.

The rest of the paper is organized as follows. First we briefly discuss related work, then we analyze some desired properties of a new painting tool, formulate the energy minimization problem and show how to solve it efficiently. Afterwards we use our new algorithm for painting real cartoon images in different drawing styles and analyze its practical strengths and limitations. Finally, we present a couple of promising applications in the cartoon production pipeline and conclude with several new avenues for future research.

2. Related work

Interactive filling of homogenous regions has been studied since several decades ago when large pixel frame-buffers became practical. Lieberman [Lie78] proposed an extension of the flood-fill algorithm for filling with arbitrary black-and-white patterns, Smith [Smi79] showed how to fill regions with shaded boundaries, and Fishkin and Barsky [FB84] presented recoloring of anti-aliased images. Although these approaches can simplify filling in some special cases, they still suffer from limitations of the original flood-fill algorithm, i.e. the inability to cope with gappy boundaries or to reach a salient boundary of a region with complicated hatching.

The same limitations also hold for auto-painting systems [SF00, QST*05] which build upon connected component analysis. This process is equivalent to sequential execution of the flood-fill algorithm with different labels on each unfilled pixel in a thresholded binary image. Sýkora et al. [SBv05] replaced the thresholding by a more sophisticated outline detection algorithm allowing auto-painting of black-and-white cartoon animations. Nevertheless, in the final stage, they still rely on connected component analysis and thus share the aforementioned limitations.

A related operation to filling is colorization based on color seeds. This method was pioneered by Horiuchi [Hor02] who used probabilistic relaxation to propagate colors. Levin et al. [LLW04] popularized this approach with their variant based on a weighted least squares optimization framework. Later Yatziv and Sapiro [YS06] proposed a different solution based on a blending of several nearest color seeds weighted by geodesic distance. Although these approaches require little effort for images satisfying a smooth image model, they become impractical for cartoon images due to color bleeding artifacts. Qu et al. [QWH06] and later Luan et al. [LWCO*07] addressed these issues by employing hard pre-segmentation based on texture classification schemes. However, this approach is applicable only for drawing styles containing repetitive textural patterns.

Painting has much in common with interactive image segmentation. This field was mainly motivated by the seminal work of Boykov and Jolly [BJ01] who demonstrated numerous benefits of a graph cut based solution. Grady [Gra06] later proposed a concurrent approach based on a weighted least squares framework (similar to [LLW04]) which is easily extendable to multi-label segmentation and obtains comparable results to a graph cut framework. Nevertheless, all these approaches do not take into account the specific requirements of painting which differ from those used in image segmentation.

3. Ideal painting tool

In this section, we formulate a set of desired properties for an ideal painting tool. This set arose from discussion with professional ink-and-paint illustrators who are familiar with standard image manipulation tools as well as professional ink-and-paint systems. They typically use a variant of the flood-fill algorithm, providing an effective solution for simple cartoon images with homogenous regions and distinct continuous outlines, but one rarely applicable to more complicated drawing styles.

One of the well-known problems of the flood-fill algorithm is color leakage through outline gaps. To overcome this issue, illustrators typically join problematic gaps manually. This is a tedious task requiring high concentration since the human visual system normally tends to connect weak edges [Kan79]. In professional ink-and-paint systems, automatic outline joining algorithms [SC94] are available. However, this process usually connects all gaps which is often counterproductive since in many drawings this operation removes the simplicity of one-click filling. A similar problem occurs also when the image contains hatching or many small regions. In these cases illustrators typically delineate the region of interest using some edge snapping selection tool (such as *intelligent scissors* [MB99]) and then fill the whole area. This however requires precise positioning of boundary seeds which is a tedious task. Manga colorization [QWH06] partially overcomes these limitations by virtually converting areas with repetitive patterns into homogenous regions with distinct boundaries. Nevertheless, such conversion works only for manga since repetitive patterns are rare in hand-made cartoon drawings.

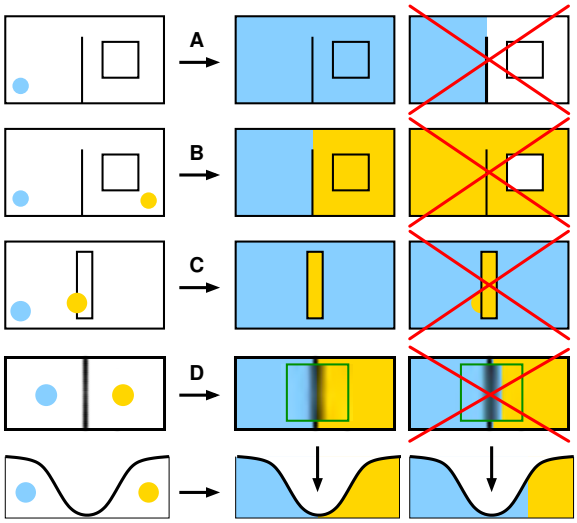

Figure 3: *An ideal painting tool tends to fill as much area as possible (A); when there are concurrent seeds, it finds an optimal boundary regardless of gappy outlines and produces compact regions without holes (B); it supports soft scribbles by preserving rule of majority so it is not necessary to paint precisely inside the region of interest (C); it handles anti-aliasing by pushing color boundaries to pixels with minimal intensity not with maximal gradient (D).*

Optimal boundary. The illustrators' wish is to have a tool that tends to fill as much area as possible by finding an optimal enclosing boundary (regardless of holes and gappy outlines) and then, when necessary, they can refine the interior using additional strokes (see cases A and B in Figure 3). Such workflow is not supported in manga colorization. Although it handles gappy outlines via region shape regularization, it is not able to find and optimal boundary due to getting stuck in inappropriate local minima (see Figure 2 or red crossed example in Figure 3, rule A).

Connected labelling. In manga colorization, user edits can produce color regions with arbitrary topology (i.e. they can consist of several disconnected parts). This functionality

brings considerable speed-up in a special case when there is a one-to-one correspondence between color and pattern. However, in a more general setting this behavior can be confusing since it breaks a locality assumption, which is essential for painting and is required by illustrators.

Soft scribble. Another feature which illustrators appreciate is a color brush resistant to imprecise placement. According to naming convention used in colorization and interactive image segmentation, we refer to strokes made with such a brush as *soft scribbles*. Soft scribbles should satisfy the so called *rule of majority*, meaning that a region is filled with a color whose strokes have most of their pixels lying in its interior (see case C in Figure 3). This simple rule can bring significant time savings when painting thin structures or small regions. Due to Fitts' law [Fit54] the time needed to reach thin objects can be greatly reduced by slightly increasing brush radius (see Figure 4). A great speed up can also be achieved in the context of the ink-and-paint pipeline when several aligned animation phases are painted simultaneously (*onion fill*) or when color patches are transferred from already painted frames to new ones (*patch pasting*, see Section 5 and Figure 9). In comparison to the manga colorization, soft scribbles are a completely new feature, however, a similar idea has been explored recently in the context of appearance editing [AP08]. The key difference is that the energy minimization framework used in [AP08] takes into account only coarse edits which are insufficient for painting.

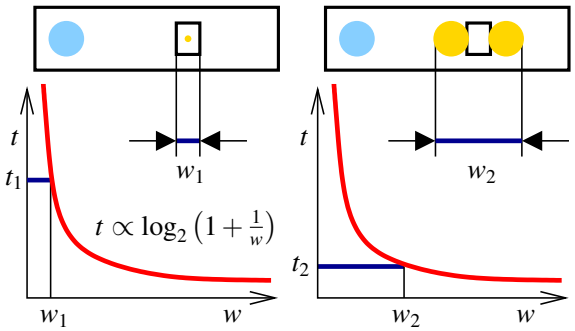

Figure 4: *Soft scribbles and Fitts' law [Fit54] – the task is to fill the small rectangle of width w_1. Using a pixel-wide brush the expected time needed to reach its interior is t_1. By increasing brush radius we can enlarge the target margin to w_2 and obtain considerably lower time t_2.*

Anti-aliasing. Since scanned hand-drawn images contain soft anti-aliased edges, it is necessary to have a mechanism that preserves such anti-aliasing during the painting phase (see case D in Figure 3). This feature can also be formulated as a goal to retrieve boundaries minimizing the visibility of color discontinuities. The reason is that in cartoon images dark outlines are used to emphasize region shape and since the color is typically multiplied by the original intensity, the optimal boundary should be in the place where this intensity is minimal. This finding is inconsistent with standard maximum gradient formulation used in interactive image segmentation [BJ01] (see intensity profiles in Figure 3 bottom). In manga colorization this feature was not discussed since authors considered only binary images.

4. Energy function

In this section, we formulate an energy minimization framework, the aim of which is to satisfy the requirements presented in the previous section.

As an input we have a gray-scale image I consisting of pixels P in a 4-connected neighborhood system $\mathcal{N}$ and a set of user-provided non-overlapping strokes S with colors C. The aim is to find a labelling, i.e. the color-to-pixel assignment c that minimizes the following energy function:

$$E(c) = \sum_{\{p,q\}\in\mathcal{N}} V_{p,q}(c_p, c_q) + \sum_{p\in P} D_p(c_p) \qquad (1)$$

where smoothness term $V_{p,q}$ represents the energy of color discontinuity between two neighbor pixels p and q, and data term D_p the energy of assigning color c_p to pixel p.

4.1. Smoothness term

As discussed in Section 3, the aim is to hide color discontinuities. Since typically multiplicative color modulation is used, the best locations for color discontinuities are at pixels where the original image intensity is low, e.g. inside dark outlines. According to this finding we let the energy $V_{p,q}$ be:

$$V_{p,q}(c_p, c_q) \propto \begin{cases} I_p & \text{for } c_p \neq c_q \\ 0 & \text{otherwise} \end{cases} \qquad (2)$$

However, the absolute values of $V_{p,q}$ should be set carefully since they have a fundamental impact on the resulting labelling. As we want to prefer compact and hole-free regions it is necessary to avoid zeros in $V_{p,q}$ for the case $c_p \neq c_q$, otherwise regions with outlines having zero intensity will not contribute to the minimum of (1). Such regions can be easily disconnected and produce holes in the final labelling. As opposed to that, non-zero smoothness term will lead to compact regions without holes. However, it can also produce unintended shortcuts through white areas. To suppress this shortcoming it is necessary to set high energies for the boundaries going through the white pixels. Theoretically, this energy should be higher than the longest outline in the image I. Nevertheless, a good estimate for this value is a perimeter of I. In most cases this setting effectively ensures that a region boundary will go through white pixels only when there is no other low energy path along dark outlines. Following these ideas, we map an interval of image intensities $\langle 0,1\rangle$ to $\langle 1,K\rangle$, where $K = 2\cdot(w+h)$, w is width and h height of I. For nearly binary images such mapping can be linear, i.e. $I'_p = K\cdot I_p + 1$, however, for black-and-white cartoons or soft pencil drawings (such as "blocks" image in Figure 1 or "robber" in Figure 10) the problem with

shortcuts persists due to lower contrast between homogenous areas and outlines.

To alleviate this issue it is possible to use some nonlinear mapping that enhances the contrast (e.g. $I'_p = K \cdot I_p^2 + 1$) or employ a more powerful technique previously used for outline detection in black-and-white cartoon images [SBv05]. Here, outlines are detected using the response of a Laplacian of Gaussian ($\mathbf{L}\circ\mathbf{G}$) filter. This filter corresponds to a light-over-dark mechanism used in the primary stages of the human visual system [MH80]. From a mathematical point of view, $\mathbf{L}\circ\mathbf{G}$ estimates the second order derivative of the image intensity, its zero-crossings correspond to edge locations, and local maxima to places with high curvature (e.g. centers of outlines). According to this we preprocess the image I by filtering with $\mathbf{L}\circ\mathbf{G}$ and produce a new image $I_f = 1 - \max(0, s \cdot \mathbf{L}\circ\mathbf{G}(I))$ where the negative response of $\mathbf{L}\circ\mathbf{G}$ is clamped to zero and positive values scaled by s to match the interval $\langle 0,1\rangle$. After this preprocessing, the contrast of outlines is emphasized and the interior of homogenous regions are turned to white regardless of their original intensity (see Figure 5). Finally, values in I_f are linearly mapped to the interval $\langle 1,K\rangle$ and used in smoothness term $V_{p,q}$.

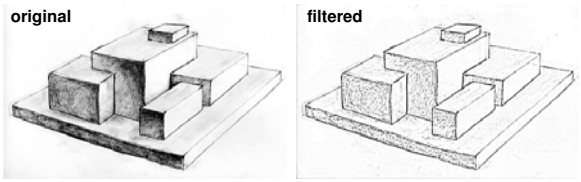

Figure 5: *An example of an image preprocessed by filtering with $\mathbf{L}\circ\mathbf{G}$ – the original image (left); normalized and clipped response of $\mathbf{L}\circ\mathbf{G}$ (right). See the improvement on the contrast of outlines.*

Note, how our smoothness term completely differs from terms used in interactive image segmentation [BJ01, Gra06]. The aim here is to push the segment boundary to pixels with maximal gradient. If the gradient magnitude is high (as in cartoon images), many pixels can have $V_{p,q}$ near or equal zero. As discussed in Section 3 this setting is unsuitable for painting since it reveals color discontinuities on soft edges and produces holes.

4.2. Data term

In manga colorization or interactive image segmentation the data term D_p is usually set to some image-based likelihood such as pattern or intensity similarity. The assumption behind this setting is that there is a one-to-one correspondence between color and pattern/intensity. However, repetitive patterns or intensity variations are not typical for hand-made drawings and even if they are present, one-to-one correspondences are rare. To address this fact *LazyBrush* does not rely on image-based likelihoods but uses completely user-driven data term allowing the implementation of a soft scribbles discussed in Section 3.

The key idea is to relax a common assumption, i.e. that all user-defined seeds are necessarily hard constraints. Instead we let the user to decide how to penalize labelling by setting:

$$D_p(c_p) = \lambda \cdot K, \tag{3}$$

where $\lambda \in \langle 0,1\rangle$ is a constant given by the user and K is the energy of discontinuity at white pixels that balances the influence of data and smoothness terms (therefore we use the same symbol as in Section 4.1). The value of λ indicates the presence of a brush stroke and its "strength": $\lambda = 1$ is for pixels that have not received a brush stroke, $\lambda = 0$ for hard scribbles, and for soft scribbles λ should satisfy the following inequality: $0 + K \cdot |S| < K \cdot \partial S + \lambda K \cdot |S|$, saying that the energy (1) is lower even if the pixels under a scribble S have not receive its color ($|S|$ is the area and ∂S the perimeter of S). From this constraint we obtain: $\lambda > 1 - \partial S/|S|$ which we can measure for each scribble but in practice most scribbles have $1 - \partial S/|S| < 0.95$ so we set $\lambda = 0.95$.

It is easy to verify that soft scribbles preserve the rule of majority. Imagine several seeded pixels S with $D_p = \lambda \cdot K$ inside a region R where the smoothness is assumed to by constant. Then the labelling with minimal energy should have lowest $\sum D_p = \lambda \cdot K \cdot |S| + K \cdot |R - S|$. After simplification: $\sum D_p = K \cdot (|R| - (1-\lambda) \cdot |S|)$ yields minimum for the largest $(1-\lambda) \cdot |S|$, i.e. when all scribbles have equal λ then the winner will have the largest number of seeded pixels $|S|$.

4.3. Minimization

Now we proceed to the minimization of (1). Since the smoothness term $V_{p,q}$ depends only on pixel intensity and not on the color labels, our energy function satisfies Potts model [Pot52]. As shown in [BVZ98] minimizing such a function is equivalent to solving a *multiway cut* problem on a certain undirected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ where $\mathcal{V} = \{P, C\}$ is a set of vertices and $\mathcal{E} = \{\mathcal{E}_p, \mathcal{E}_c\}$ a set of edges (see Figure 7).

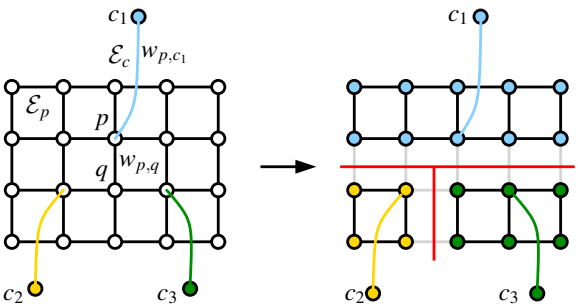

Figure 7: *Multiway cut – basic structure of graph $\mathcal{G}$ (left): pixels P (white dots), color terminals C (color dots), pixel edges $\mathcal{E}_p$ with weight $w_{p,q}$ (black lines), and links to color terminals $\mathcal{E}_c$ with weight $w_{p,c}$ (color lines). Resulting multiway cut and corresponding labelling of pixels (right).*

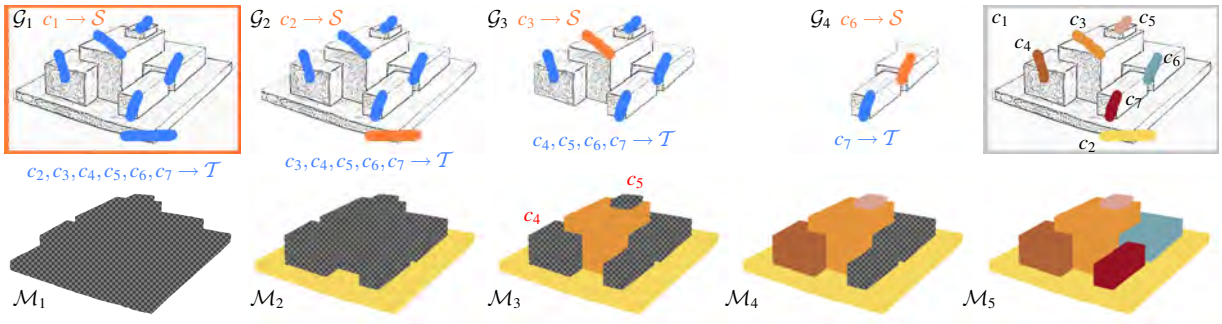

Figure 6: *Multiway cut algorithm in progress – gradually reducing max-flow/min-cut subproblems solved on graphs $\mathcal{G}$ with terminals $\mathcal{S}$ and $\mathcal{T}$ (top), corresponding masks of unlabelled pixels $\mathcal{M}$ (bottom, checkerboard pattern indicates unlabelled pixels). Note how two trivial subproblems c_4 and c_5 were pruned in the third iteration (middle).*

Vertices $\mathcal{V}$ consist of pixels P and color terminals C. Each pixel $p \in P$ is connected to its 4 neighbors via edges $\mathcal{E}_p$ having weight equal to smoothness term $w_{p,q} = V_{p,q}$ for case $c_p \neq c_q$. There are also auxiliary edges $\mathcal{E}_c$ that connect color terminals C to seeded pixels. Each $\mathcal{E}_c$ has weight $w_{p,c} = K - D_p(c)$ (hard scribbles have $w_{p,c} = K$ and soft $w_{p,c} = (1-\lambda) \cdot K$).

Note that in contrast to interactive image segmentation [BJ01] our graph is very sparse (has much less $\mathcal{E}_c$). This is due to the fact that most pixels have $D_p = K$ for all labels so the weight $w_{p,c} = 0$ and thus the corresponding $\mathcal{E}_c$ is redundant. Since there are no other links to terminals besides user-defined the resulting labelling will be always connected to seeds. This is in accordance with properties discussed in Section 3.

A multiway cut with 2 terminals is equivalent to a max-flow/min-cut problem for which efficient algorithms exist [BK04]. However, for 3 or more terminals the problem is NP-hard [DJP*92] even on our sparse graph. Nevertheless, it is interesting that we are very close to P, because if we assume only a set of hard scribbles each with unique terminal (e.g. as in Figure 7), we can always collapse seeded pixels to this terminal and obtain a planar graph for which an exact polynomial algorithm exists [Yeh01]. Nevertheless, we cannot collapse pixels seeded by soft scribbles and so we need to solve the full non-planar problem for which no polynomial approximation scheme exists. The best known approximation [KKS*04] based on geometric embedding and linear programming guarantees an optimal solution within a factor of $1.3438 - \varepsilon_k$, where ε_k goes to zero with increasing number of terminals k (for $k = 3$ the bound is $\frac{12}{11}$). This algorithm is not easy to implement and due to slow performance it is inappropriate for interactive applications. There are also other approximation algorithms based on the max-flow/min-cut subroutine [DJP*92, BVZ01]. Although they guarantee optimality only within a factor of $2 - \frac{2}{k}$ and 2 respectively, they are much easier to implement. The problem is that they are still relatively slow due to many max-flow/min-cut steps. For example it takes more than 11 seconds to compute labelling for 0.5 Mpix image in Figure 1 on a 2.4GHz CPU using α-expansion algorithm described in [BVZ01].

Inspired by the *isolation heuristic* used in [DJP*92] we propose a novel greedy multiway cut algorithm, which takes advantage of our special graph topology guaranteeing connected labelling. In practice, it provides similar results as the widely used α-expansion [BVZ01] but is significantly faster (18x for Figure 1, see also Table 1) and so more suitable for interactive applications. It works in a simple hierarchical fashion by solving less than N one-to-all max-flow/min-cut problems (where N is the number of colors). The significant speed up is obtained thanks to (1) gradually reducing size of max-flow/min-cut subproblems and (2) ability to prune trivial cases. It has the following steps (cf. work-in-progress example in Figure 6):

1. Initialize a set of active color labels C and a mask $\mathcal{M}$ of unlabelled pixels.
2. Find all unlabelled regions R in $\mathcal{M}$ that intersect strokes with only one color label c_r. For each such $r \in R$ set labels in $\mathcal{M}$ to c_r and if there is no other region in $\mathcal{M}$ containing strokes with label c_r, remove c_r from C.
3. If C is empty then stop.
4. Select an arbitrary color label $c \in C$.
5. Build a graph $\mathcal{G}$ from all unlabelled pixels in $\mathcal{M}$.
6. Connect pixels seeded with color label c to terminal $\mathcal{S}$, and pixels seeded with colors $C - \{c\}$ to terminal $\mathcal{T}$.
7. Solve max-flow/min-cut problem [BK04] on $\mathcal{G}$ with source $\mathcal{S}$ and sink $\mathcal{T}$.
8. At pixels where corresponding graph vertices were assigned to terminal $\mathcal{S}$, set label in mask $\mathcal{M}$ to c.
9. Remove color label c from C and go to (2).

Roughly speaking the algorithm selects an arbitrary color as a first terminal and all other colors as a second terminal. Then it solves the binary max-flow/min-cut problem and removes a part of the image assigned to the first terminal. It performs the same operation on the reduced image with reduced set of colors while avoiding max-flow/min-cut computation when there are regions containing only seeds with one color. If there is no other connected component with two different color labels the algorithm stops.

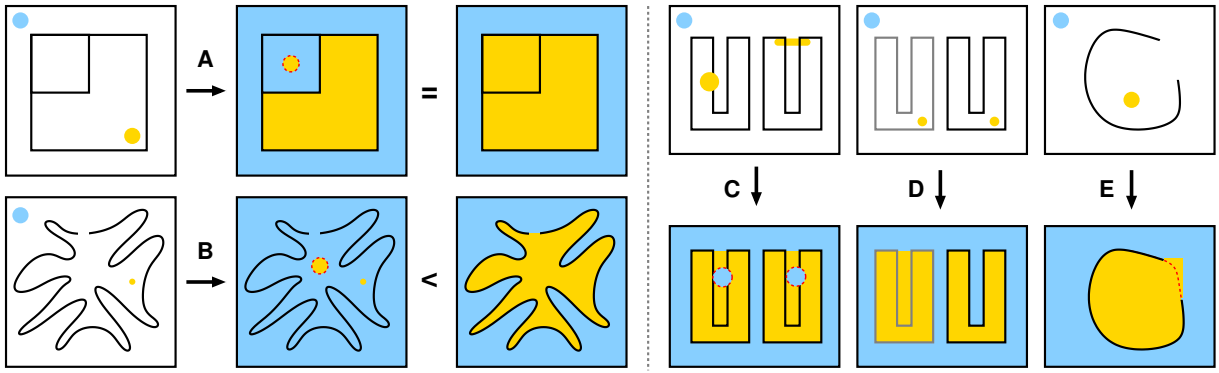

Figure 8: *Limitations – two different minimal solutions with equal energy (A); a shortcut encompassing a small scribble has lower energy than a boundary along the outline (B); the rule of majority is biased by thin creeks (C); low contrast between outlines and homogenous regions causes unintended labelling (D); long gaps or missing outlines can produce jaggy boundaries (E). Additional soft scribbles (marked with red dashed line) are necessary to resolve cases A-C. Case D can be suppressed by contrast enhancement and case E by post-processing using smooth active contour model [XAB07].*

Although such a greedy approach does not guarantee optimality within a factor of 2, in practice it produces labelling with energy close or even slightly better than α-expansion (see Table 1) so that the visual difference is imperceptible. Moreover, when the size of regions corresponding to individual colors is known beforehand (e.g. background seed or dominant color) it is possible to perform a selection of colors from the "largest" to the "smallest" and gain significant subproblem reduction after only a few initial steps. Another great optimization can be achieved if we can predict the topology of the resulting labelling. Then, thanks to the *four color theorem* [AH89], we can group color labels to 4 terminals and use only 4-way cut to obtain a constant time solution for an arbitrary number of colors.

name	resultion	colors	speed up	ΔE [%]
bottle	720x576	3	3x	-0.0452
robber	720x576	6	17x	0.0196
boy	720x576	7	17x	0.0274
picnic	1026x578	7	17x	0.0090
blocks	1026x578	7	18x	0.0038
footman	1026x578	9	9x	0.0025
manga	1026x578	11	16x	0.0395

Table 1: *Our algorithm vs. α-expansion – the speed up increases roughly with the number of colors while the change in labelling is imperceptible (negative ΔE means our algorithm found better local minimum and vice versa). Names correspond to drawings in Figure 10. The drawing "blocks" is shown in Figure 1.*

4.4. Limitations

There are several situations where the energy function (1) does not exactly preserve rules presented in Section 3. Although these cases are rare, the user should be aware of them, know the source of a problem and a way to resolve it.

Ambiguity. The first problematic situation is depicted in Figure 8 (case A). There are two different minimal solutions with equal energy. In this case the structure of the final labelling depends only on the order of labels. This ambiguity can be easily resolved by putting another decisive stroke inside the small square.

Shortcuts. When the user draws thin scribbles (e.g. one pixel wide) inside a region with a very long or gappy outline, the case can easily be that a shortcut encompassing the scribble will have lower energy than a long boundary along the outline (case B in Figure 8). To avoid such degenerate solutions, it is necessary to use wider brushes to ensure that the scribble's perimeter is much longer than the sum of lengths of all gaps.

Majority bias. Another problem is connected with the fact that the rule of majority can be biased by the image content. This bias becomes critical in the case of thin creeks (see case C in Figure 8). Here, the lower energy of soft scribbles can compensate for the high energy of shortcuts and produce unintended labelling. Another soft scribble is necessary to resolve this situation.

Low contrast. Our approach can fail on images where the contrast between outline and homogenous area is low (see case D in Figure 8). For such images it is recommended to use non-linear contrast enhancement or **L**∘**G**-based preprocessing as discussed in Section 4.1. Such modification is necessary only for setting up the smoothness term in (1), the resulting labelling can be then applied on the unmodified image (as done in Figure 1).

Metrication artifacts. When outlines have long gaps, the resulting boundary can have jaggy shape since its length is minimized in the sense of the L^1 norm (see case E in Figure 8). Although an extension exists [BK03], allowing the approximation of the L^2 norm to arbitrary extents, it requires

many additional $\mathcal{E}_p$ edges yielding significant slowdown of the optimization. Even if the L^2 norm is minimized, the boundary shape still does not need to be optimal in the sense of higher order continuity (C^1 or C^2). To solve this problem the shape can be post-processed using some active contour model with a more sophisticated smoothness term [XAB07].

5. Results

We implemented *LazyBrush* in a simple interactive application and validated its flexibility on various drawing styles including pen and pencil drawings, black-and-white cartoons [SBv05] and manga [QWH06]. Selected results are presented in Figure 10 (drawings "bottle" and "robber" were preprocessed using **L**○**G** to enhance contrast of outlines).

Note that the user effort required by *LazyBrush* is comparable to previous techniques, even though it does not assume any style-specific features. The roughly positioned soft scribbles also reduce the level of concentration required of the illustrator, which makes painting more enjoyable and thus less tedious. This feature can also be beneficial in applications where specific motor coordination abilities are taken into account (e.g. a painting tool for young children).

The advantage of soft scribbles becomes even more apparent in the context of the ink-and-paint pipeline where thousands of frames must be painted manually. A popular technique here is *onion fill* allowing to paint several superimposed animation in-betweens simultaneously (see Figure 9, top). In contrast to standard approaches *LazyBrush* does not require precise positioning of color seeds. By drawing longer soft scribbles it is possible to cover large movements.

LazyBrush can be also utilized in a more sophisticated auto-painting technique called *patch pasting* [SBv04] where color information is transferred automatically by registering interesting points in already painted and unpainted drawings. The original method requires pre-segmentation in order to compute the most frequently used color inside a region (see Figure 9, bottom). With *LazyBrush* the pre-segmentation is no longer needed since the color patches can be used as soft scribbles and the rule of majority will propagate to the most frequently used color. Thanks to this, patch pasting can be used also for more complicated drawing styles where meaningful segmentation is hard to obtain.

Since *LazyBrush* does not mix colors (as compared to [LLW04]) but produces hard segments, it is possible to use it for filling regions with different content beyond just color, e.g. gradients, patterns or depth information which can be utilized for composition, *unsharp masking* [LCD06] or in *Lumo* [Joh02] where it is necessary to produce correct normal orientation for 3D-like shading.

6. Conclusions and Future work

We have presented a new interactive tool for painting hand-made cartoon drawings called *LazyBrush*. It is based on an optimization framework that tries to mimic the behavior of an ideal painting tool as proposed by professional illustrators. The key advantage of *LazyBrush* is flexibility. It can be used for painting in various drawing styles with comparable user effort to that offered by previous style-limited approaches. We have also demonstrated its usability in the context of the ink-and-paint pipeline for which *LazyBrush* was mainly designated and can provide significant reduction of manual effort.

As future work we plan to integrate *LazyBrush* more into the spatio-temporal domain [WBC*05] and examine possible modifications of our energy function in order to utilize different optimization schemes such as *weighted least-squares* [Gra06, AP08]. Another interesting avenue for future research is *graph coloring algorithms* [RSST96], which allow the grouping of color terminals in such a way that only a fixed number of max-flow/min-cut steps is necessary to obtain a solution for arbitrary numbers of colors.

Acknowledgements

We are grateful to T. Jarkovský, V. Votýpka, and T. Rychecký from AniFilm studio for being initiators and first users of our system. Many thanks also go to L. Kavan for extensive discussions on the earlier version of the paper and to anonymous reviewers for their fruitful comments. Images in this paper are courtesy of P. Koutský / Anifilm, O. Sýkora, L. Vlček, Y. Qu et al., and studios UPP & DMP. This work has been supported by the Marie Curie action IEF, No. PIEF-GA-2008-221320.

References

[AH89] APPEL K., HAKEN W.: Every planar map is four-colorable. *Contemporary Mathematics 98* (1989), 1–741.

[AP08] AN X., PELLACINI F.: AppProp: All-pairs appearance-space edit propagation. *ACM Transactions on Graphics 27*, 3 (2008), 40.

[BJ01] BOYKOV Y., JOLLY M.-P.: Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. In *Proceedings of Internation Conference on Computer Vision* (2001), pp. I: 105–112.

[BK03] BOYKOV Y., KOLMOGOROV V.: Computing geodesics and minimal surfaces via graph cuts. In *Proceedings of IEEE International Conference on Computer Vision* (2003), pp. I: 26–33.

[BK04] BOYKOV Y., KOLMOGOROV V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 9 (2004), 1124–1137.

[BVZ98] BOYKOV Y., VEKSLER O., ZABIH R.: Markov random fields with efficient approximations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (1998), pp. 648–655.

[BVZ01] BOYKOV Y., VEKSLER O., ZABIH R.: Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence 23*, 11 (2001), 1222–1239.

[DJP*92] DAHLHAUS E., JOHNSON D. S., PAPADIMITRIOU C. H., SEYMOUR P. D., YANNAKAKIS M.: The complexity

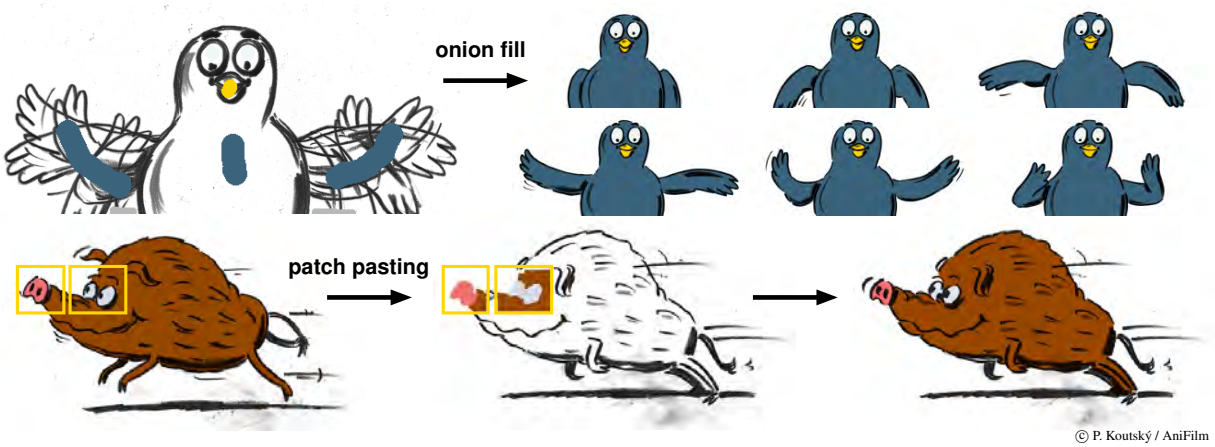

Figure 9: *LazyBrush for the ink-and-paint pipeline – a whole animation can be painted simultaneously in the onion fill setting using a few soft scribbles (top), auto-painting by patch pasting [SBv04] is possible even without pre-segmentation (bottom).*

of multiway cuts. In *Proceedings of ACM Symposium on Theory of Computing* (1992), pp. 241–251.

[FB84] FISHKIN K. P., BARSKY B. A.: A family of new algorithms for soft filling. *ACM SIGGRAPH Computer Graphics 18*, 3 (1984), 235–244.

[Fit54] FITTS P. M.: The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology 47*, 6 (1954), 381–391.

[Gra06] GRADY L.: Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 11 (2006), 1768–1783.

[Hor02] HORIUCHI T.: Estimation of color for gray-level image by probabilistic relaxation. In *Proceedings of IEEE International Conference on Pattern Recognition* (2002), pp. 867–870.

[Joh02] JOHNSTON S. F.: Lumo: Illumination for cel animation. In *Proceedings of International Symposium on Non-photorealistic Animation and Rendering* (2002), pp. 45–52.

[Kan79] KANIZSA G.: *Organization in Vision*. Praeger, New York, 1979.

[KKS*04] KARGER D. R., KLEIN P., STEIN C., THORUP M., YOUNG N. E.: Rounding algorithms for a geometric embedding of minimum multiway cut. *Mathematics of Operations Research 29*, 3 (2004), 436–461.

[LCD06] LUFT T., COLDITZ C., DEUSSEN O.: Image enhancement by unsharp masking the depth buffer. *ACM Transactions on Graphics 25*, 3 (2006), 1206–1213.

[Lie78] LIEBERMAN H.: How to color in a coloring book. *ACM SIGGRAPH Computer Graphics 12*, 3 (1978), 111–116.

[LLW04] LEVIN A., LISCHINSKI D., WEISS Y.: Colorization using optimization. *ACM Transactions on Graphics 23*, 3 (2004), 689–694.

[LWCO*07] LUAN Q., WEN F., COHEN-OR D., LIANG L., XU Y.-Q., SHUM H.-Y.: Natural image colorization. In *Proceedings Eurographics Symposium on Rendering* (2007), pp. 309–320.

[MB99] MORTENSEN E. N., BARRETT W. A.: Toboggan-based intelligent scissors with a four parameter edge model. In *Proceedings of IEEE Computer Vision and Pattern Recognition* (1999), pp. II: 452–458.

[MH80] MARR D., HILDRETH E. C.: Theory of edge detection. In *Proceedings of Royal Society* (1980), vol. B207, pp. 187–217.

[Pot52] POTTS R.: Some generalized order-disorder transformation. In *Proceedings of Cambridge Philosophical Society* (1952), vol. 48, pp. 106–109.

[QST*05] QIU J., SEAH H. S., TIAN F., WU Z., CHEN Q.: Feature- and region-based auto painting for 2D animation. *The Visual Computer 21*, 11 (2005), 928–944.

[QWH06] QU Y., WONG T. T., HENG P. A.: Manga colorization. *ACM Transactions on Graphics 25*, 3 (2006), 1214–1220.

[RK82] ROSENFELD A., KAK A. C.: *Digital Picture Processing*, vol. 1. Academic Press, Orlando, USA, 1982.

[RSST96] ROBERTSON N., SANDERS D. P., SEYMOUR P., THOMAS R.: Efficiently four-coloring planar graphs. In *Proceedings of ACM Symposium on Theory of Computing* (1996), pp. 571–575.

[SBv04] SÝKORA D., BURIÁNEK J., ŽÁRA J.: Unsupervised colorization of black-and-white cartoons. In *Proceedings of the 3rd International Symposium on Non-photorealistic Animation and Rendering* (2004), pp. 121–127.

[SBv05] SÝKORA D., BURIÁNEK J., ŽÁRA J.: Colorization of black-and-white cartoons. *Image and Vision Computing 23*, 9 (2005), 767–782.

[SC94] SEAH H. S., CHUA B. C.: A skeletal line joining algorithm. In *Proceedings of the Computer Graphics International* (1994), pp. 62–73.

[SF00] SEAH H. S., FENG T.: Computer-assisted coloring by matching line drawings. *The Visual Computer 16*, 5 (2000), 289–304.

[Smi79] SMITH A. R.: Tint fill. *ACM SIGGRAPH Computer Graphics 13*, 2 (1979), 276–283.

[WBC*05] WANG J., BHAT P., COLBURN R. A., AGRAWALA M., COHEN M. F.: Interactive video cutout. *ACM Transactions on Graphics 24*, 3 (2005), 585–594.

[XAB07] XU N., AHUJA N., BANSAL R.: Object segmentation using graph cuts based active contours. *Computer Vision and Image Understanding 107*, 3 (2007), 210–224.

[Yeh01] YEH W.-C.: A simple algorithm for the planar multiway cut problem. *Journal of Algorithms 39* (2001), 68–77.

[YS06] YATZIV L., SAPIRO G.: Fast image and video colorization using chrominance blending. *IEEE Transactions on Image Processing 15*, 5 (2006), 1120–1129.

Figure 10: *LazyBrush paintings – in each example see user-specified scribbles superimposed on the original drawing and the corresponding optimized painting (names correspond to Table 1). Background scribbles are sometimes invisible since they are implicitly drawn around the image border. Most color scribbles are soft therefore they can be positioned roughly.*

Appendix B

Smart Scribbles for Sketch Segmentation

G. Noris, D. Sýkora, A. Shamir, S. Coros, B. Whited, M. Simmons, A. Hornung, M. Gross, R. Sumner: Smart Scribbles for Sketch Segmentation. Computer Graphics Forum, vol. 31, no. 8, pp. 2516–2527, December 2012. ISSN 0167-7055. **IF=1.638**

Volume 31 (2012), Number 8 pp. 2516–2526 COMPUTER GRAPHICS *forum*

Smart Scribbles for Sketch Segmentation

G. Noris[†1,2], D. Sýkora[3], A. Shamir[4], S. Coros[1], B. Whited[5], M. Simmons[5], A. Hornung[1], M. Gross[1,2], and R. Sumner[1]

[1]Disney Research Zürich, [2]ETH Zürich, CGL, [3]CTU in Prague, FEE,
[4]The Interdisciplinary Center, [5]Walt Disney Animation Studios

Figure 1: *Sketch segmentation: For each example pair, Scribbles on the left produce the segmentation on the right.*

Abstract

We present Smart Scribbles*—a new scribble-based interface for user-guided segmentation of digital sketchy drawings. In contrast to previous approaches based on simple selection strategies, Smart Scribbles exploits richer geometric and temporal information, resulting in a more intuitive segmentation interface. We introduce a novel energy minimization formulation in which both geometric and temporal information from digital input devices is used to define stroke-to-stroke and scribble-to-stroke relationships. Although the minimization of this energy is, in general, a NP-hard problem, we use a simple heuristic that leads to a good approximation and permits an interactive system able to produce accurate labelings even for cluttered sketchy drawings. We demonstrate the power of our technique in several practical scenarios such as sketch editing, as-rigid-as-possible deformation and registration, and on-the-fly labeling based on pre-classified guidelines.*

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.4]: Graphics Utilities—Graphics editors, Computer Graphics [I.3.6]: Methodology and Techniques—Interaction techniques, Image Processing and Computer Vision [I.5.3]: Clustering—Similarity measures

Keywords: digital sketches, interactive segmentation, scribble-based user interface, energy minimization

1. Introduction

Sketchy drawings are prevalent across a wide range of applications and domains. In early development phases, rough drawings are used, for example, for concept art in product design, and for storyboards in animation environments, and are favored both for the speed of generation, and the expressiveness of the results. A sketchy style also has a place in finished art – providing a level of visual richness not found in "clean" line representations, i.e. drawings constructed from crisp, distinct outlines and minimal interior detail.

Modern digital devices and graphics software solutions offer powerful stylization, deformation, morphing, and animation capabilities for 2D drawings. However, in order to perform these high-level tasks, a certain degree of under-

† chino@disneyresearch.com

standing of the content of the drawing is required. This is a challenging problem due to the significant gap between the ability of a human to discern structure in a drawing and the capability of an algorithm to derive it from low level stroke information. This is true even for clean line drawings, and most existing approaches rely on the presence of a human user to provide sufficient information to guide the task.

The problem of extracting structure from drawings becomes substantially more difficult for sketchy input, and this is one reason it is far less common to find a consistently sketchy style in full-length animations or automatic support for sketchy input in high-level editing packages. One important category of drawing abstraction is segmenting the drawing into logical parts. To-date, there is no efficient method available for automatic segmentation in this domain. In contexts where a breakdown of the drawing is required, segmentation is typically achieved by design: the drawings are created in different layers, one for each logical component. This approach is too limiting in practice: it requires a priori knowledge of the use of the drawing, is cumbersome (especially when different tasks require different segmentations), and is an error-prone process, even for experienced artists.

We seek a semi-automated solution to segmenting sketchy drawings that is fast enough for interactive use, but also predictable and easy to use – making it accessible to even the most novice user.

To this end, we propose the concept of *Smart Scribbles* as an accurate and simple way for the user to specify semantically meaningful stroke clusters within a drawing. In contrast to previous methods that use scribbles as positional constraints for various image editing tasks [BJ01,LLW04,AP08, SDC09b], our formulation considers more detailed geometric (position, orientation, curvature) and temporal information (time of creation) when analyzing stroke-to-stroke and Scribble-to-stroke relationships. In addition, we introduce the concept of locality control as a way of conveniently trading off the Scribbles' areas of influence for accuracy. This allows our system to produce desired results with minimal user intervention even for cluttered sketches.

We evaluate our approach on a collection of digitally drawn sketches of varying complexity, and demonstrate its application to various tasks including sketch editing and as-rigid-as-possible (ARAP) deformation and registration. As our solution is fast to compute, our method enables tight integration of these tasks within an interactive digital drawing session.

2. Related Work

Relevant prior art can be divided into three main categories: sketch labeling interfaces, scribble-based image segmentation, and classification of vector fields in scientific visualization.

User-guided labeling of strokes in hand-drawn images plays a central role in many sketch-based editing systems. In Lank et al. [LS05], the authors present an approach for inferring user intent from the local velocities, accelerations and curvatures of the selection lasso. More recently, Wolin et al. [WSA07] presented a technique for labeling groups of strokes from a vectorized sketch where the system attempts to automatically fragment continuous strokes into logical pieces to assist the user. Both of these techniques ultimately utilize a region-based selection approach. ScanScribe, a system developed by Saund and colleagues [SFLM04], presents the user with an intuitive selection paradigm that allows for the creation of objects from collections of pixels and supports further grouping into composite objects. The system is able to automatically segment the image into basic primitives, such as linear curve fragments, and then group them into more complex objects, such as rectangles, using a fragment alignment metric (or by finding perceptually closed paths as proposed in [Sau03]). Two limitations of this automatic technique are 1) limited complexity of objects detected by the system and 2) the inability to handle sketchy overlapping curve fragments, thus requiring more traditional and tedious lasso/selection-box methods for more complex drawings.

The approach presented in this paper leverages previous works on interactive image segmentation in order to optimize the labeling process based on user scribbles. Boykov et al. [BJ01] developed such an approach based on graph cuts for segmenting images and finding optimal boundaries between objects. In [LLW04], Levin and colleagues present a similar framework based on a least-squares optimization for colorizing gray-scale images by roughly labeling regions with colored scribbles. More recently, An and Pellacini [AP08] developed an interactive energy minimization framework for propagating color edits to similar regions throughout the image. Our approach is most similar to LazyBrush [SDC09b], a graph cut based system for the selection of regions in sketchy drawings. The main difference is that this system cannot provide the labeling of the strokes that bound each painted region. From this point of view, our framework can be seen as a generalization of LazyBrush, since it extracts meaningful boundaries first, and then builds regions inferred from those boundaries. Because this process removes clutter from the input drawing, it greatly improves the accuracy of selection and reduces the amount of user interaction needed to obtain clean results.

Our approach also bears some resemblance to sketch-based clustering of vector fields in scientific visualization [WWYM10]. Here the aim is to allow the user to sketch 2D curves and use them as a query to retrieve 3D field lines whose view-dependent 2D projection is most similar to the input sketch. The curvature along the sketched input is used to measure the similarity between the input and projected curves using the edit distance [WF74]. In our approach, curvature is also used to distinguish between different shapes. However, the main advantage of our work is that we formu-

late an energy minimization problem where, in addition to shape similarity, we also take proximity, orientation, temporal information, and smoothness of the final labeling into account. As a result, our system can produce reasonable clustering even in cases when the shape of the input sketch is very rough or incomplete.

3. Method

The method we present allows users to intuitively segment digital sketches into semantically meaningful regions. The input to our framework consists of a digitally hand-drawn sketch and a small set of rough *Scribbles*. The input sketch is composed of a set of *strokes*, which are piecewise linear curves represented by sets of 2D vertices recorded from a digital input device such as a tablet. For each vertex of a stroke, we additionally store its time of creation. This helps us to differentiate strokes which are spatially close but are drawn at different moments in time.

The input Scribbles are special strokes that indicate the user's intent to segment a particular portion of the drawing. Two criteria related to the Scribble primitives are critical in order to ensure a useful and intuitive system. First, Scribbles should not have to closely follow the target region. However, if desired, the user should be able to precisely select localized regions. We call this property *locality control*. The second criterion specifies that the time of creation of the Scribble should not influence the segmentation results.

We observe that generally speaking, processing strokes as a whole is very difficult. A single stroke can be arbitrarily complex: it can cross or overlap with itself multiple times, and/or it can densely cover an area the artist wished to fill in. For this reason, we break strokes and Scribbles into linear *segments* by densely resampling the input. Any property defined locally over the stroke can easily be transferred to the segments.

The remainder of this section describes in detail each of the steps used by our method

We formulate the task of sketch clustering as an optimization problem, where the goal is to label each stroke in a way that minimizes an energy function. The concept of our design is depicted in Fig. 2 and the remainder of this section describes in detail each of the steps used by our method. The energy function is defined in Section 3.1. It relies on a smoothness and data term which are described in Sections 3.1.1 and 3.1.2, respectively. In Section 3.2 we discuss the minimization method used to compute the final solution to the stroke labeling.

3.1. Energy function

The input to our method consists of a set of stroke segments S and a set of Scribble segments R associated with a set of labels L. The goal is to find a labeling, i.e., an assignment ϕ of the labels in L to every segment in S, that minimizes the following energy function E:

$$E(\phi) = \sum_{i,j \in S} V_{i,j}(\phi_i, \phi_j) + \lambda \sum_{i \in S} D_i(\phi_i) \qquad (1)$$

where $V_{i,j}$ is a smoothness term that captures the cost of the labeling with respect to the similarity between two stroke segments i and j. The data term D_i measures the affinity between Scribbles and strokes. The parameter λ controls the relative influence of the smoothness and data terms.

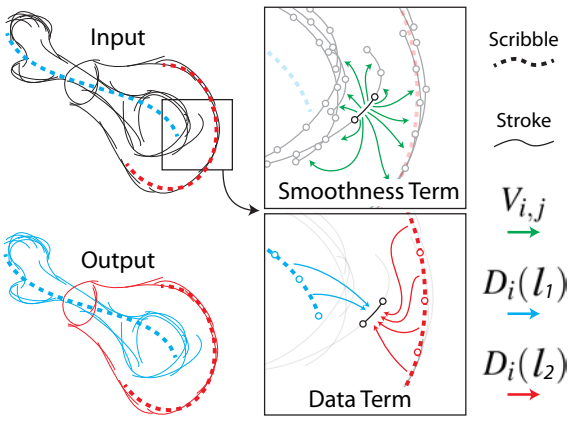

Figure 2: *Energy definition overview. The input consists of a set of strokes (black) and Scribbles (red and blue dotted lines). The output consists of a labeling of all strokes (the labeling is indicated here by the red/blue color assignment to the strokes in the output).* ***Smoothness Term:*** *For a segment i and a neighbor segment j, $V_{i,j}$ expresses the energy of assigning a different label to i and j, based on how similar they are.* ***Data Term:*** *Given a labeling $\phi_i = l_*$ (assigning label l_* to segment i), $D_i(\phi_i)$ expresses the energy of the labeling, which is a function of the similarity of segment i to all Scribbles associated with l_*.*

3.1.1. Smoothness Term

The smoothness term is defined as:

$$V_{i,j}(\phi_i, \phi_j) = \prod_{g \in G} \delta(g(i,j), \sigma_g) \qquad (2)$$

when $\phi_i \neq \phi_j$, otherwise it is zero. G is a set of similarity terms:

$$\begin{aligned} prox(i,j) &= ||\vec{p}_j - \vec{p}_i|| \\ dir(i,j) &= 1 - |\vec{d}_i \cdot \vec{d}_j| \\ curv(i,j) &= 1 - min(c_i, c_j)/max(c_i, c_j) \\ time(i,j) &= |t_j - t_i| \end{aligned}$$

where i and j are two segments, and p, d, c and t are the position, direction, radius of curvature, and time of creation associated with each segment. The fall-off function δ is defined as:

$$\delta(g(i,j),\sigma_g) = \exp\left(-\frac{g(i,j)^2}{\sigma_g^2}\right) \quad (3)$$

3.1.2. Data Term

The data term is defined as:

$$D_i(\phi_i) = 1 - \max_{r \in R(\phi_i)} A(i,r) \quad (4)$$

where $R(\phi_i)$ denotes a set of Scribble segments r with label ϕ_i. The affinity $A(i,r)$ is defined as:

$$A(i,r) = \prod_{g \in G_{data}} \delta(g(i,r),\sigma_g) \quad (5)$$

Here, as with the smoothness term, we measure the similarity between segments rather than strokes. However, as Scribbles have no associated time information, we reduce the set of similarity terms to $G_{data} = \{prox, dir, curv\} \subset G$. Additionally, we alter the definition of curvature to become oriented: $curv(i,j) = ||\vec{c}_i - \vec{c}_j||$. This allows extra control in separating curves with the same curvature but different orientation (e.g. the tangled lines in Fig. 5).

One of our main goals is to allow users, if desired, to have precise local control over the strokes that get affected by each Scribble. To illustrate this, we consider a scenario where the user draws a single Scribble, as shown in Fig. 3a. In this case, because no concurrent label exists, all strokes are selected. This behavior, though reasonable, is not in line with a user's expectations of having local control.

To address this, we introduce an artificial background label $b \in L$, in addition to the labels prescribed by the user. This new label has a constant influence on each stroke segment i regardless of the existence of any particular user-defined Scribble, i.e., $A(i,b) = B$, where B is a threshold to override the influence of distant Scribbles. The background label therefore serves as a lower bound for computing the *max* component in the data term (4).

Furthermore, we control the locality of each Scribble r by modifying its proximity fall-off δ (3) as follows:

$$\delta(prox(i,r),\sigma_{prox}) = \frac{1}{\sigma_{prox}} \exp\left(-\frac{prox(i,r)^2}{\sigma_{prox}^2}\right). \quad (6)$$

Here σ_{prox} follows the desired locality (i.e., is large for global influence and small for local influence) and the normalization term $1/\sigma_{prox}$ ensures the integral over the fall-off function stays equal for different values of σ_{prox} (i.e., amplitude is high for small values and low for large ones). In other words, the overall energy remains constant, while its spatial spread is controlled. When σ_{prox} becomes very low, the response of the fall-off function (6) for distant stroke segments also becomes very low and can therefore be easily overridden when computing the *max* value in (4) as illustrated in Fig. 3b-f.

There are several possible ways to control the parameter σ_{prox}. One natural way is to use the speed of the Scribble based on the experimentally demonstrated linear relationship between speed and perceived locality [AZ97]:

$$W = \frac{\beta \cdot L}{T - \alpha}. \quad (7)$$

Here W is the selection radius, L is length of the Scribble, T is time spent on drawing it, and α and β are empirically measured constants. This rule was used to control the selection locality in systems having limited modality [LS05]. Since the spatial spread of the fall-off function (6) grows linearly with the increasing σ_{prox} we can set $\sigma_{prox} = W/2$. Alternatively, one could consider the use of pen-pressure, or—in the case of binary modality—a simple key toggle to switch between two locality values.

3.2. Optimization method

As shown in [BVZ98], minimizing the energy function defined in Equation 1 is equivalent to solving a multi-way cut on a specific weighted graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V} = \{S, L\}$ is a set of vertices and $\mathcal{E} = \{\mathcal{E}_s, \mathcal{E}_l\}$ is a set of edges (See Fig. 4). The graph vertices $\mathcal{V}$ consist of stroke segments S and label terminals L. Each stroke segment $i \in S$ is connected to all other stroke segments $j \in S - \{i\}$ via edges $\mathcal{E}_{i,j}$ having weight $w_{i,j}$ equal to the smoothness term $V_{i,j}$ when $\phi_i \neq \phi_j$. In addition, auxiliary edges $\mathcal{E}_{i,l}$ connect stroke segments $i \in S$ to label terminals $l \in L$. Each $\mathcal{E}_{i,l}$ has weight $w_{i,l} = \lambda(1 - D_i(l))$, where λ is the parameter defined in Equation 1.

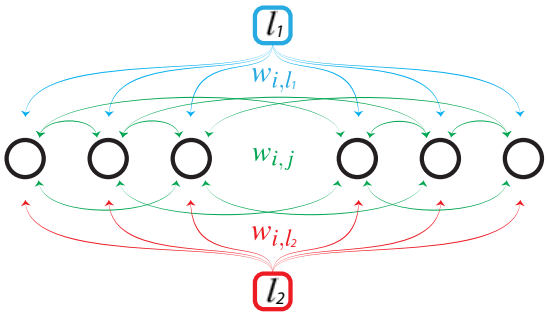

Figure 4: *Graph Construction. Stroke segments are shown as black circles. Terminal labels (in this example l_1 and l_2) are shown as colored squares. The graph edges $w_{i,j}$ reflect the smoothness terms $V_{i,j}$ between the stroke segments $i, j \in S$, while the data terms $D_i(l)$ for stroke segment $i \in S$ and label $l \in L$ are captured by the weights $w_{i,l}$.*

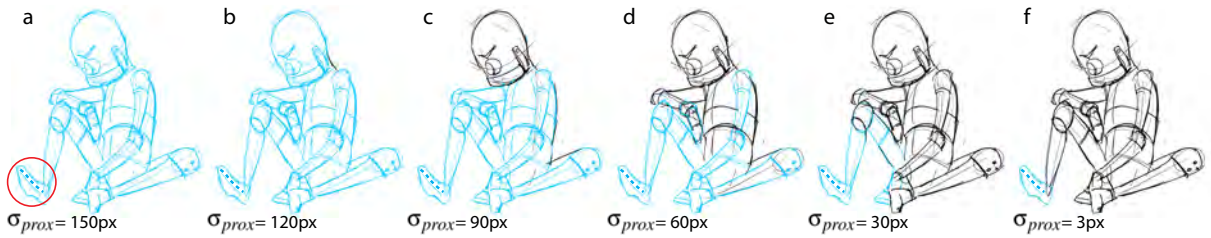

Figure 3: *The effect of the locality control by varying σ_{prox}: A blue Scribble is drawn on the foot (circled in red). On the right, the value of σ_{prox} is progressively decreased. Notice how the selection becomes progressively more local as the influence of the blue label gets overruled by the background label (shown in black).*

The multi-way cut problem with two terminals is equivalent to a max-flow/min-cut problem for which efficient polynomial algorithms exist [BK04]. However, for three or more terminals the problem is NP-hard [DJP*92]. To obtain a good approximate solution we use a simple divide-and-conquer heuristic previously proposed in [SDC09b] to gradually simplify the N-terminal problem into a sequence of $N-1$ binary max-flow/min-cut sub-problems. This approach provides results similar to more advanced techniques (such as α-expansion or α/β-swap [BVZ01]), but is significantly faster and therefore better suited for interactive applications.

4. Results

We demonstrate the effectiveness of our algorithm on a variety of input sketches. All results were generated using the parameters in Table 1.

Fig. 5 shows a collection of simple input sketches and Scribbles, together with the color-coded stroke labeling output by our system. These results show that desirable sketch segmentations can be obtained using very different scribbling strategies. We note that the input Scribbles do not have to closely match the sketch in order for our algorithm to work well—approximate similarity in terms of position, orientation and curvature is sufficient.

Figs. 1 and 6 show results from more complex input sketches. To correctly segment these images, users typically start with rough, fast strokes, and then refine the output locally using slower, more accurate strokes. Our method robustly handles scenarios where strokes that are close together and almost parallel belong semantically to different regions (as shown on the waiter's legs and snake and pole example in Fig. 6). In these cases, the time metric plays an important role in the labeling process.

Our framework does not require artists to draw the input sketches in any particular manner. It is possible that strokes representing the same region can be drawn at very different moments in time. This happens, for instance, when artists first draw silhouettes for the whole scene, and then proceed to refine the drawing. This can diminish the advantage of taking timing into account in the similarity metric. Correct segmentations can still be obtained, but more Scribbles may be required. Alternatively the similarity metric can be adjusted to apply a smaller weight to the time parameter, or it can be removed as is done for the Scribble metric.

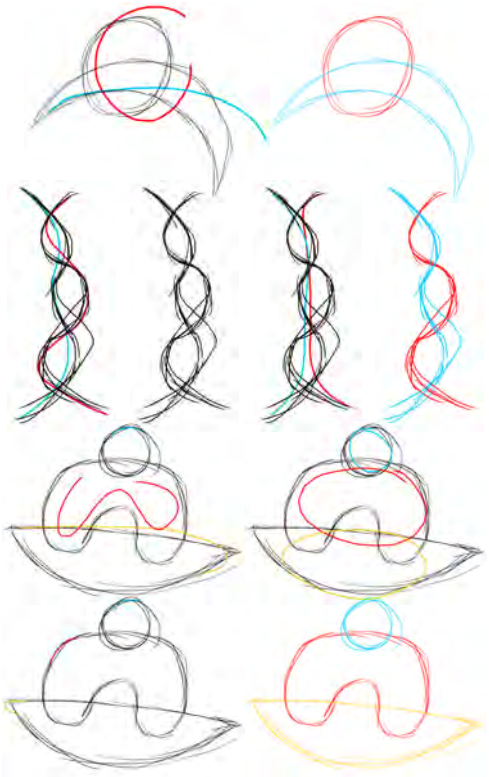

Figure 5: *Results for simple sketches: several different inputs produce the same segmentation.*

4.1. User Study

In order to test the efficiency and ease of use of our method, we conducted a user study comparing Smart Scribbles to our implementation of several commonly-used selection tools,

Figure 6: *Example Results: For each example, the colored Scribbles are shown on the input drawing and the adjacent image shows the resulting color-coded labelings.*

Parameter	Value	Unit
λ	4	
$\sigma_{prox\ smooth}$	100	px
$\sigma_{dir\ smooth}$	0.5	
$\sigma_{time\ smooth}$	1000	ms
$\sigma_{curv\ smooth}$	0.1	
$\sigma_{prox\ data}$	[10 ; 90]	px
$\sigma_{dir\ data}$	0.1	
$\sigma_{curv\ data}$	0.25	
B	0.0001	
artboard width	1200	px
artboard heigth	1200	px

Table 1: *Parameter settings for the user study and all examples in this paper.*

drawing	speed-up	$t(df)$	p-value
combo	1.23x	−1.8798	0.07847
snake	1.53x	−2.4807	0.02461
skull	1.83x	−8.3931	0.00000
house	1.85x	−5.2488	0.00001
abstract	2.36x	−5.5759	0.00003
characters	1.65x	−3.8896	0.00118

Table 2: *Median speed-ups and results of paired t-tests comparing times spent on labeling different drawings using Smart Scribbles and common selection tools.*

namely point, box, and lasso (these tools are typically included in professional vector graphics software such as Adobe Illustrator or Inkscape). This section includes an overview of the study results.

Our user study had 35 participants (8 female and 27 male with ages ranging from 18 to 62). Subjects had no prior experience using Smart Scribbles, and varying levels of proficiency (from none to expert level) with the professional software packages.

Participants were asked to use the different tools to match given labelings on four different drawings of varying complexity (see Fig. 7). The system recorded the time taken to complete the tasks and the mouse mileage, as well as the accuracy of the final labeling. Overall distributions of times and mouse mileage measured during the experiment are depicted in Fig. 8. There is a notable performance gain (1 : 23x to 2 : 36x median speed-up) when comparing Smart Scribbles to the common selection tools. Paired t-tests (see Table 2) indicate that this gain is statistically significant ($p < 0.005$) for 4 out of 6 drawings. The lower confidence level for the snake and combo drawings is reflected by notable intersection between interquartile ranges of box-and-whisker plots in Fig. 8. Although the median speed-up is apparent, the advantage of Smart Scribbles is not as convincing in this case. The main reason here could be the relatively low complexity of these examples.

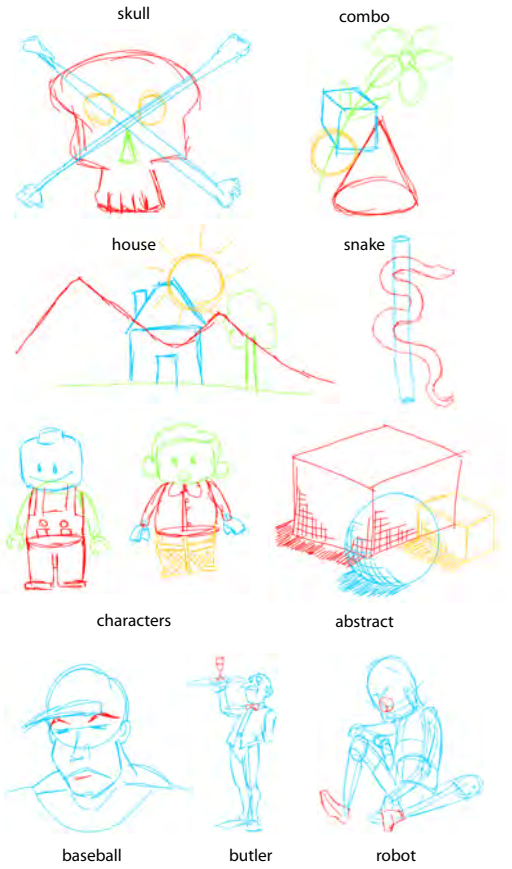

Figure 7: *Given drawings and labelings from the user study.*

In addition, participants were presented with different ways of controlling the locality. We tested the linear relationship proposed in [AZ97], as well as a simple binary modality associated with the extreme values of the parameter $\sigma_{prox\ data}$ as shown in Table 1. When asked about their experience, a majority (31) preferred the binary switching approach. We believe this is due to two reasons. First, in previous work, stroke speed had a direct associated visual feedback. This cannot easily be done with our Scribbles, as the result of the selection is not strictly bounded by a spatial radius. A simpler, more explicit interface may therefore be more appropriate. Secondly, a majority of the participants (27) indicated that they do not want to be forced to draw Scribbles slowly.

5. Applications

The labeling produced by our approach can be utilized to generate input to perform region as well as stroke segmentation (see Fig. 10a). Once the labels for the segments of each stroke are computed, we can automatically obtain an area mask of the enclosed region using the LazyBrush [SDC09b]

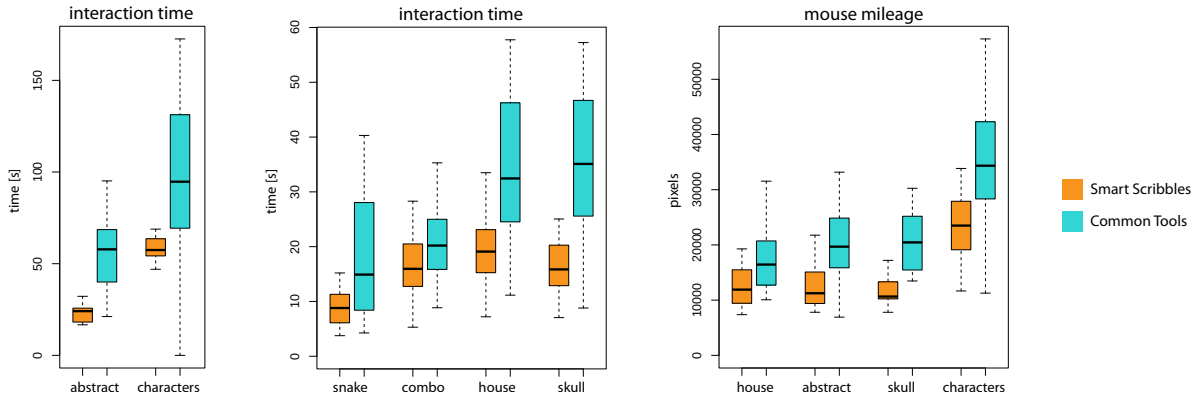

Figure 8: *Interaction times and mouse mileage of participants for different drawings using Smart Scribbles (orange) and common selection tools (blue).*

algorithm (see Fig. 9). To this end, we first render all segments assigned to a specific label to a raster image. This image is used both as an input gray-scale image (Fig. 9a) and as foreground soft scribbles (blue in Fig. 9b) for input to LazyBrush. In addition, we use a default background hard scribble around the image boundary (red in Fig. 9b). Given this input, LazyBrush produces the desired area mask (Fig. 9c). As compared to other naive methods (like convex-hull or flood-fill), this approach works with concave regions and is robust to small gaps.

For more complex sketches, the user may need to specify additional Scribbles (Fig. 9d) to classify interior strokes and use them as additional background soft scribbles for LazyBrush (Fig. 9e). These new scribbles enable the area computation method to produce masks that contain holes (Fig. 9f).

We note that similar masks (Fig. 9i) can be produced with the original LazyBrush algorithm directly. However, the area segmentation alone is not sufficient to provide a labeling of the individual strokes, because strokes at the boundaries between different area masks cannot be consistently assigned to one mask or another (Fig. 9g). Moreover, with the original LazyBrush algorithm, the user must be more careful, since the optimization only takes into account the position of the scribbles. In contrast, our framework also considers orientation, curvature, and time (compare Fig. 9d and h).

The ability to easily label both strokes and areas empowers a large variety of applications. One can easily alter the individual drawing style for all strokes that have the same label. It is also possible to accurately separate the different parts of a sketch, specify their depth ordering [SSJ*10], and then deform them independently using ARAP shape deformation techniques [IMH05] (see Fig. 10b). These operations can help, for instance, in the context of image registration [SDC09a] to produce better alignment.

When artists start a drawing, they typically begin with a simple, high-level sketch that depicts a set of of primitive shapes (see examples and references in [GIZ09]) that are called volume or scaffold lines. If available, we can use these aiding structures as Scribbles to segment the final detailed sketch (see Fig. 10c). This would let the artist focus on drawing without having to switch between different brushes. ARAP deformation, for instance, could then be used to correct the shape of semantically meaningful sketch regions.

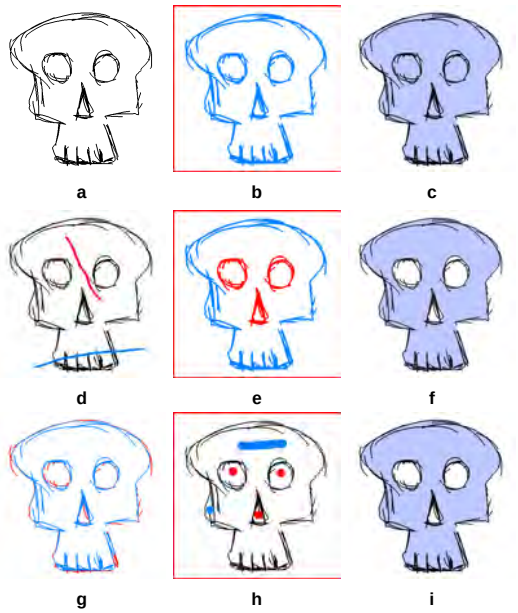

Figure 9: *Area mask computation: strokes of an input sketch (a) can be used as LazyBrush soft scribbles (b) to automatically fill the drawing (c). Additional Scribbles (d) can be used to segment the strokes (e) for better control of the paint fill (f). Using the original LazyBrush algorithm (h) to paint the figure also produces a good result (i), however, the strokes cannot be classified based on the painting alone (g).*

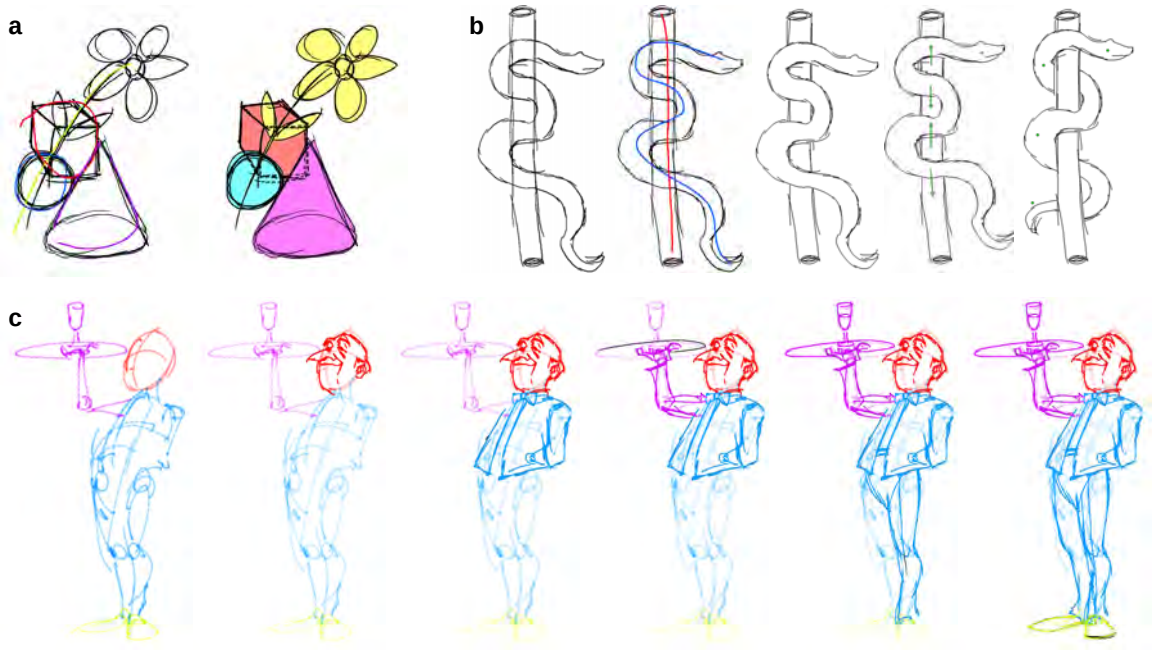

Figure 10: *Applications. Opaquing of the segmented clusters (a), ARAP deformation and opaquing with depth inequalities (b), on-the-fly labeling (c).*

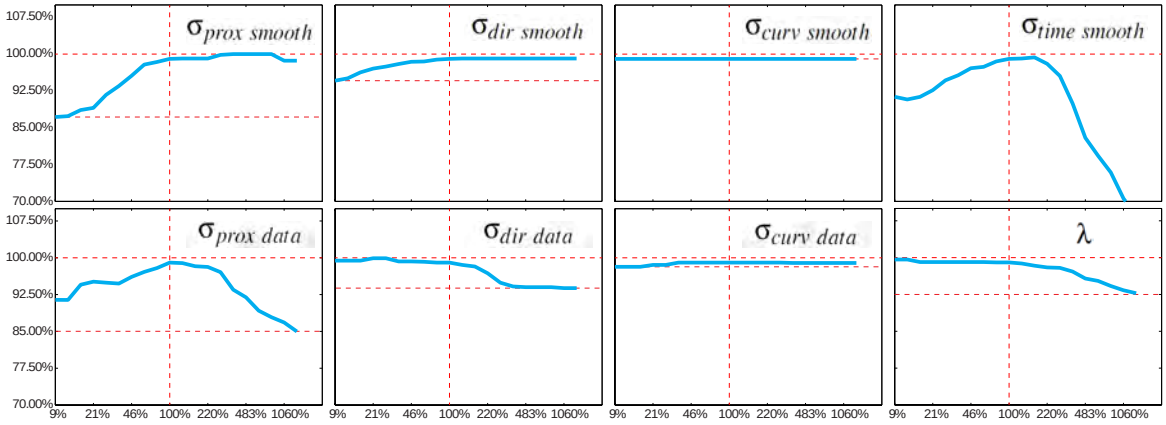

Figure 11: *Single parameter perturbation. Given a database of 8 drawings, each with 5 different sets of Scribbles drawn to match a desired segmentation, we measure the segmentation accuracy obtained with perturbations of the empirically chosen settings of Table 1. In each graph, the horizontal axis shows the multiplication factor for one of the parameters in exponential scale. The vertical axis shows how the correctness of the segmentation evolves. The graphs show that the system is mostly sensitive to time and proximity information, while direction and curvature have less influence.*

6. Limitations and Future Work

The selection of good parameters for the similarity terms and the energy function requires some effort. As can be observed in the parameter sensitivity graphs in Fig. 11, the system is robust when parameters are perturbed one at a time. This is due to the correlation that exists between the similarity terms. However, it is possible that the perturbation of multiple parameters can lead to significant changes in the result. We also tested the benefit of including the stroke creation time in the stroke analysis. After removing this information from the similarity terms, and re-tuning the remaining parameters, we achieved the results shown in Fig. 12. In our experience, omitting this temporal information reduces the effectiveness of our method, as overlapping strokes require

more effort to be separated. In the future we plan to develop a system that allows automatic parameter tuning based on a database of ground truth data.

Although we aim to produce accurate labeling with minimal user effort, detailed selection is necessary when ambiguities exist. One such ambiguous case occurs when an object is occluded by another object and parallel strokes from each are very close together or even overlap. In this case, only the time constraint can provide a distinctive metric to obtain correct labeling. However, when the time is not available or when the user does not preserve temporal coherency of strokes, our approach requires additional user guidance.

The proposed graph-cut energy minimization strategy is generally very fast and produces the labeling at interactive rates. However, in the worst case, when a large number of stokes are close to each other as defined by our similarity measure, the number of edges in the graph can grow quadratically with the number of strokes and the computation can become prohibitively slow (see Fig. 13). The problem can be alleviated by subsampling the strokes and processing disconnected components individually. Another problem is related to the non-polynomial complexity of the core max-flow algorithm [BK04]. In certain situations where the cost of the minimal cut is very high and the graph topology is complex, the number of augmentation paths can grow very quickly along with the computation time. This issue can be solved by a recently proposed incremental breadth-first search solution [GHK*11] that works in polynomial time and is typically notably faster than [BK04].

The use of previously labeled drawings as Scribbles offers another avenue for future work. These annotations could be used on-the-fly to label new sketches as they are created, thus simplifying further interactions. This approach could be used, for instance, as an extension to the recently presented ShadowDraw system [LZC11], by augmenting each sketch in the database with Scribbles. In this way the segmentation could be provided automatically as new drawings are created. A similar use case arises in the context of sketchy animations where image registration [SDC09a] can be used to transfer already labeled strokes and treat them as Scribbles for the next frame. This can help, for instance, to better control temporal noise [NSC*11]. Scribbles could also potentially be used to improve the accuracy of drawing simplification methods [GDS04,BTS05,SC08], as a typical problem with the current, fully automatic, approaches is that they do not take into account any semantic information such as provided by our approach.

7. Conclusions

We have presented Smart Scribbles, a scribble-based interface for sketch segmentation. Our method is fast, supports multi-label segmentation, and acts as an enabling technology for a variety of applications in the context of drawing, editing, and animation.

In the long term, we envision a next-generation drawing application, where drawing, editing, and animation are tightly integrated, and where the simplicity of the interaction is the key. This work represents a step in this direction; a bridge between classic drawing and digital editing.

Acknowledgements

We would like to thank Adam Sporka for help with the user study, Maurizio Nitti for creating some of the drawings, and all anonymous reviewers for their insightful comments and suggestions.

References

[AP08] AN X., PELLACINI F.: AppProp: All-pairs appearance-space edit propagation. *ACM Transactions on Graphics 27*, 3 (2008), 40. 2517

[AZ97] ACCOT J., ZHAI S.: Beyond Fitts' law: Models for trajectory-based HCI tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (1997), pp. 295–302. 2519, 2522

[BJ01] BOYKOV Y., JOLLY M.-P.: Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. In *Proceedings of Internation Conference on Computer Vision* (2001), pp. 105–112. 2517

[BK04] BOYKOV Y., KOLMOGOROV V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 9 (2004), 1124–1137. 2520, 2525

[BTS05] BARLA P., THOLLOT J., SILLION F. X.: Geometric clustering for line drawing simplification. In *In Proceedings of the Eurographics Symposium on Rendering* (2005), pp. 183–192. 2525

[BVZ98] BOYKOV Y., VEKSLER O., ZABIH R.: Markov random fields with efficient approximations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (1998), pp. 648–655. 2519

[BVZ01] BOYKOV Y., VEKSLER O., ZABIH R.: Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence 23*, 11 (2001), 1222–1239. 2520

[DJP*92] DAHLHAUS E., JOHNSON D. S., PAPADIMITRIOU C. H., SEYMOUR P. D., YANNAKAKIS M.: The complexity of multiway cuts. In *Proceedings of ACM Symposium on Theory of Computing* (1992), pp. 241–251. 2520

[GDS04] GRABLI S., DURAND F., SILLION F. X.: Density measure for line-drawing simplification. In *Proceedigns of Pacific Conference on Computer Graphics and Applications* (2004), pp. 309–318. 2525

[GHK*11] GOLDBERG A. V., HED S., KAPLAN H., TARJAN R. E., WERNECK R. F. F.: Maximum flows by incremental breadth-first search. In *ESA* (2011), pp. 457–468. 2525

[GIZ09] GINGOLD Y., IGARASHI T., ZORIN D.: Structured annotations for 2D-to-3D modeling. *ACM Transactions on Graphics 28*, 5 (2009), 148. 2523

[IMH05] IGARASHI T., MOSCOVICH T., HUGHES J. F.: As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics 24*, 3 (2005), 1134–1141. 2523

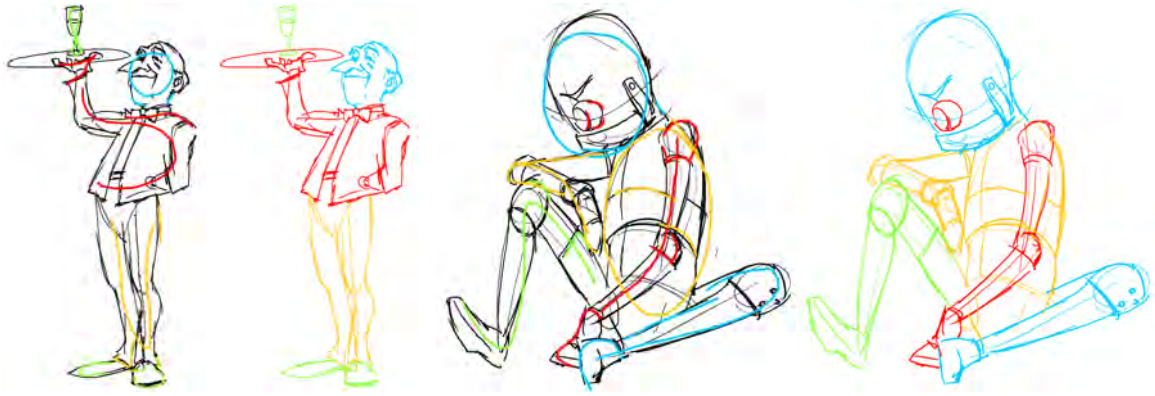

Figure 12: *No-Time-Test: this image shows how the system works when no temporal information is used. Notice how the cluttered regions require more Scribbles to produce a proper segmentation.*

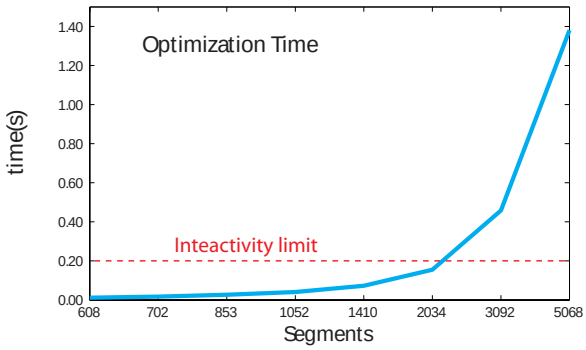

Figure 13: *Performance Limit. The graph shows the computation time for the optimization in our implementation. The data was generated by progressively subsampling a complex drawing. Assuming a interactivity limit of* 0.2 *seconds, our implementation can optimize the labeling for up to 2000 segments. The corresponding subsampled drawing is shown on the right.*

[LLW04] LEVIN A., LISCHINSKI D., WEISS Y.: Colorization using optimization. *ACM Transactions on Graphics 23*, 3 (2004), 689–694. 2517

[LS05] LANK E., SAUND E.: Sloppy selection: Providing an accurate interpretation of imprecise selection gestures. *Computers & Graphics 29*, 4 (2005), 490–500. 2517, 2519

[LZC11] LEE Y. J., ZITNICK C. L., COHEN M. F.: ShadowDraw: real-time user guidance for freehand drawing. *ACM Transactions on Graphics 30* (2011), 27. 2525

[NSC*11] NORIS G., SÝKORA D., COROS S., WHITED B., SIMMONS M., HORNUNG A., GROSS M., SUMNER R.: Temporal noise control for sketchy animation. In *Proceedings of International Symposium on Non-photorealistic Animation and Rendering* (2011), pp. 93–98. 2525

[Sau03] SAUND E.: Finding perceptually closed paths in sketches and drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence 25*, 4 (2003), 475–491. 2517

[SC08] SHESH A., CHEN B.: Efficient and dynamic simplification of line drawings. *Computer Graphics Forum 27*, 2 (2008), 537–545. 2525

[SDC09a] SÝKORA D., DINGLIANA J., COLLINS S.: As-rigid-as-possible image registration for hand-drawn cartoon animations. In *Proceedings of International Symposium on Non-photorealistic Animation and Rendering* (2009), pp. 25–33. 2523, 2525

[SDC09b] SÝKORA D., DINGLIANA J., COLLINS S.: LazyBrush: Flexible painting tool for hand-drawn cartoons. *Computer Graphics Forum 28*, 2 (2009), 599–608. 2517, 2520, 2522

[SFLM04] SAUND E., FLEET D., LARNER D., MAHONEY J.: Perceptually-supported image editing of text and graphics. *ACM Transactions on Graphics 23*, 3 (2004), 728–728. 2517

[SSJ*10] SÝKORA D., SEDLACEK D., JINCHAO S., DINGLIANA J., COLLINS S.: Adding depth to cartoons using sparse depth (in)equalities. *Computer Graphics Forum 29*, 2 (2010), 615–623. 2523

[WF74] WAGNER R., FISCHER M.: The string-to-string correction problem. *Journal of the ACM 21*, 1 (1974), 168–173. 2517

[WSA07] WOLIN A., SMITH D., ALVARADO C.: A pen-based tool for efficient labeling of 2D sketches. In *Proceedings of Eurographics Workshop on Sketch-Based Interfaces and Modeling* (2007), pp. 67–74. 2517

[WWYM10] WEI J., WANG C., YU H., MA K.-L.: A sketch-based interface for classifying and visualizing vector fields. In *Proceedings of IEEE Pacific Visualization Symposium* (2010), pp. 129–136. 2517

Appendix C

Adding Depth to Cartoons Using Sparse Depth (In)equalities

D. Sýkora, D. Sedláček, S. Jinchao, J. Dingliana, S. Collins: Adding Depth to Cartoons Using Sparse Depth (In)equalities. Computer Graphics Forum, vol. 29, no. 2, pp. 615–623, May 2010. ISSN 0167-7055. **IF=1.638**

EUROGRAPHICS 2010 / T. Akenine-Möller and M. Zwicker
(Guest Editors)

Volume 29 (2010), Number 2

Adding Depth to Cartoons Using Sparse Depth (In)equalities

D. Sýkora,[†1] D. Sedlacek,[2] S. Jinchao,[1] J. Dingliana,[1] and S. Collins[1]

[1]Trinity College Dublin
[2]Czech Technical University in Prague, Faculty of Electrical Engineering

Abstract

This paper presents a novel interactive approach for adding depth information into hand-drawn cartoon images and animations. In comparison to previous depth assignment techniques our solution requires minimal user effort and enables creation of consistent pop-ups in a matter of seconds. Inspired by perceptual studies we formulate a custom tailored optimization framework that tries to mimic the way that a human reconstructs depth information from a single image. Its key advantage is that it completely avoids inputs requiring knowledge of absolute depth and instead uses a set of sparse depth (in)equalities that are much easier to specify. Since these constraints lead to a solution based on quadratic programming that is time consuming to evaluate we propose a simple approximative algorithm yielding similar results with much lower computational overhead. We demonstrate its usefulness in the context of a cartoon animation production pipeline including applications such as enhancement, registration, composition, 3D modelling and stereoscopic display.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.4]: Graphics Utilities—Graphics editors, Image Processing and Computer Vision [I.4.8]: Scene Analysis—Depth cues, Image Processing and Computer Vision [I.4.6]: Segmentation—Pixel classification, Computer Applications [J.5]: Arts and Humanities—Fine arts

1. Introduction

Recovering depth from a single image is a challenge, which has remained an open problem after decades of active research. In this paper we focus on a specific variant of the problem where the input image is a hand-made line drawing. As opposed to previous attempts to provide complete 3D reconstruction either by imposing various geometric assumptions [LS96, VM02, LFG08] or using sketch-based interfaces to create the 3D model incrementally [IMT99, KH06, NISA07, JC08, GIZ09], we seek a simple 2.5D pop-up consistent with the observer's perception of depth in the scene. Such representation is crucial for maintaining correct visibility and connectivity of individual parts during interactive shape manipulation [IMH05, WXXC08], deformable image registration [SDC09a] or fragment composition [SBv05] (see Figure 1 top). It can also be used in the context of image enhancement to generate 3D-like shading effects [Joh02], improve perception of depth [LCD06], or produce stereoscopic images which have become increasingly popular due to emerging 3D display technologies (see Figure 1 bottom).

Although in general it is almost impossible to obtain consistent 2.5D pop-up automatically we show that by using a few user-provided hints the task becomes very simple. Our novel approach is motivated by perceptual studies that have tried to understand how a human reconstructs depth information from a single image [KvDK96, Koe98]. These studies show that in contrast to machines, humans have no internal representation of the scene and rather rely on a set of local judgements from which they reconstruct global observation. More specifically they have found that humans typically fail to specify absolute depth values but are much more accurate in telling whether some part of the object is in front of another and vice versa. This observation leads us to the formulation of an optimization problem that tries to emulate the human depth reconstruction process. Its key advantage is that it completely avoids specification of absolute depth values and instead uses depth (in)equalities which better cor-

† e-mail: sykorad@cs.tcd.ie

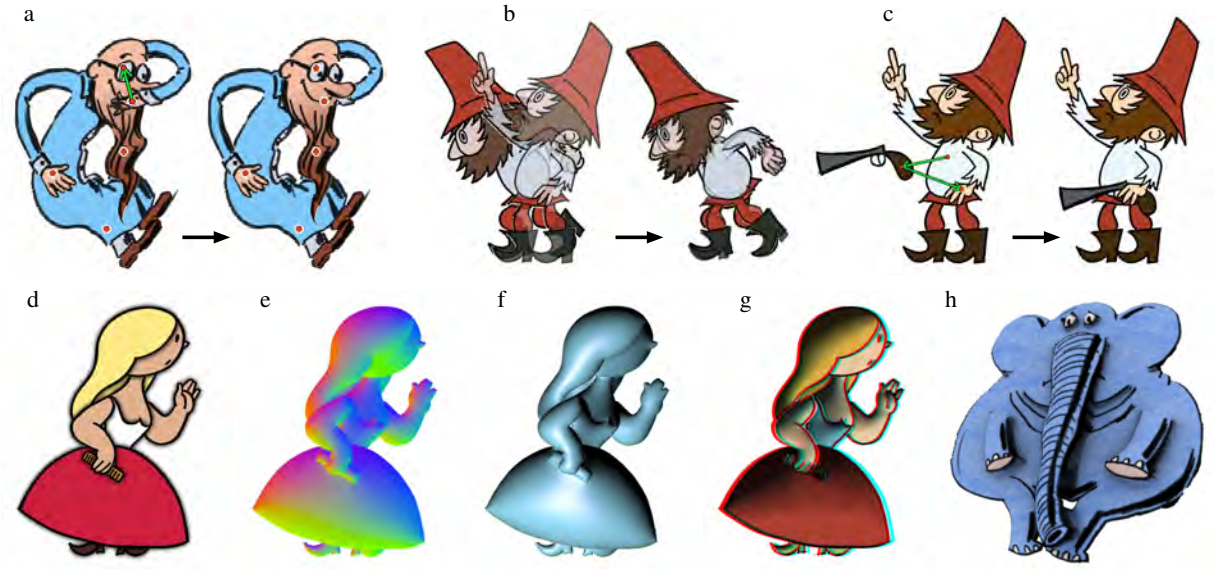

Figure 1: *Applications – depth maps generated using our system can be utilized to maintain correct visibility in (a) as-rigid-as-possible shape manipulation, (b) deformable image registration, and (c) fragment composition. They can also help to (d) enhance perception of depth, (e) interpolate normals with respect to depth discontinuities, and produce (f) 3D-like shading, (g) stereoscopic images, or (h) simple 3D models.*

respond to the user's intuition. Using this framework we implement an interactive user interface that significantly lowers the manual effort needed to create simple cartoon pop-ups.

The rest of the paper is organized as follows. We first summarize previous work in recovering depth from single images and analyze the main drawbacks with respect to our cartoon-oriented scenario. Then we proceed to a description of our novel approach, show some results and discuss implementation issues and limitations. Finally we demonstrate various applications in the context the cartoon animation production pipeline and conclude with possible avenues for future work.

2. Related work

Classical approaches to depth from a single image exploit various photographic depth cues such as shading [Hor90, WSTS08], texture [SB95, For01], depth-of-field [NN94, AW07], or haze [Fat08, HST09]. There are also techniques that treat the problem as a statistical inference and use supervised learning [HEH05, SSN09] or a large database of manually annotated images [RT09] to provide a sample of prior knowledge. However, due to their statistical nature they require the input data to be very similar to that provided in examples or databases and if not they typically produce only very coarse approximations.

When depth cues or statistical inference do not provide sufficient results one can let the user paint with depth directly as with color [Wil90, Kan98]. Although several simple geometric constraints [HiAA97] or custom tailored editing tools [OCDD01] can speed-up the process, it is still very tedious and time consuming.

The problem becomes slightly simpler when the input image is a line drawing with visible contours that delineate object boundaries. When the depicted object consists of several planar surfaces and when some basic geometric constraints are satisfied, a complete 3D model can be generated automatically [LS96, VM02, LFG08]. However, such constraints are typically not satisfied in the case of cartoon images where the aim is to provide highly stylized depiction. A common assumption here is that the object has a blobby surface therefore it is possible to use its contours as an input to some sketch-based 3D modelling tool [IMT99, KH06, NISA07, JC08, GIZ09] to create a simple blobby object that can be further used in various applications [PFWF00, OCN04]. However, such a workflow is not suitable for our 2.5D scenario where the aim is to provide depth values consistent with the internal structure of the cartoon image.

Our technique is most closely related to the work of Zhang et al. [ZDPSS01], who propose a system that allows reconstruction of free-form surfaces from several user-specified hints which form a linearly constrained quadratic optimization problem. Although the method is very successful, there are several drawbacks that make the depth assignment difficult in our cartoon pop-up scenario: (1) they require the user to delineate depth discontinuities manually, (2) it is necessary to specify absolute depth values for regions separated by discontinuities, and (3) the system does not pro-

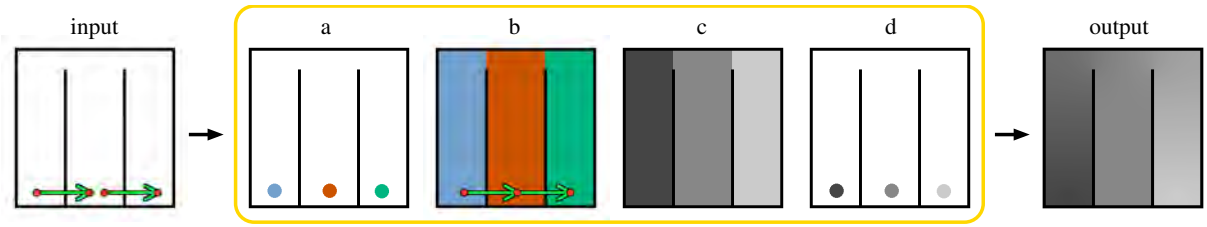

Figure 2: *Depth from sparse inequalities – input image (left) with two user-specified depth inequalities (green arrows), output depth map obtained by solving quadratic program (right), and approximative solution allowing interactive feedback (middle yellow box): (a) multi-label segmentation, (b) topological sorting, (c) intermediate depth map, (d) boundary conditions for Laplace equation.*

vide interactive feedback since the optimization is extremely time consuming even when sophisticated solvers with hierarchical preconditioning are used. Assa and Wolf [AW07] later extended Zhang's approach by adding automatic pre-segmentation and depth inequalities similar to that used in our system. However, their system is still not interactive. The whole pipeline takes several hours per single image. They first generate constraints automatically using various depth cues extracted from an input photograph and then solve a complex optimization problem to obtain the final depth map.

Recently, Ventura et al. [VDH09] presented an interactive sketch-based interface for photo pop-up that also bears some resemblance to our system. They first pre-segment the image and then use distance on a ground plane to assign coarse depths to each layer. After that they manipulate details by painting depth gradients and integrate them to produce the final depth map.

Our approach can also be viewed as an extension of colorization [LLW04] or multi-label segmentation [Gra06] where, instead of colors or labels, the aim is to propagate depth values. The added value of our system is that we do not require the user to specify exact depth values as boundary conditions but instead use depth (in)equalities which make the process more intuitive.

3. Our approach

In this section we present our novel approach to cartoon pop-up based on a set of sparse depth (in)equalities. We first formulate an optimization problem and show how to solve it using quadratic programming, then we proceed to an approximative solution that allows interactive feedback and, finally, we demonstrate how to extend it to handle drawing styles with thick contours and cartoon animations.

3.1. Problem formulation

Let us assume that we have an image $\mathcal{I}$ for which the user has already specified desired depth equalities $\{p,q\} \in \mathcal{U}_=$ and inequalities $\{p,q\} \in \mathcal{U}_>$ (see Figure 2 left), where $p,q \in \mathcal{I}$ are pixels or a set of pixels (scribbles). In the following section we assume that p and q are pixels but it is trivial to extend the approach to work also with scribbles.

Now the task is similar to colorization or segmentation. We want to assign depths to all pixels so that the user-specified constraints are satisfied and discontinuities are preferred at locations where the intensity gradient of $\mathcal{I}$ is high (see Figure 2 right). This can be formulated as an optimization problem where the aim is to find a minimum of the following energy function:

$$\text{minimize:} \quad \sum_{p\in\mathcal{I}} \sum_{q\in\mathcal{N}_p} w_{pq}(d_p - d_q)^2 \qquad (1)$$

$$\text{subject to:} \quad \begin{array}{ll} d_p - d_q = 0 & \forall\{p,q\} \in \mathcal{U}_= \\ d_p - d_q \geq \varepsilon & \forall\{p,q\} \in \mathcal{U}_> \end{array}$$

where d_p denotes the depth value assigned to a pixel p, $\mathcal{N}_p$ is a 4-connected neighborhood of p, and ε is some positive number greater than zero (e.g. $\varepsilon = 1$). The weight w_{pq} is set as in [Gra06]:

$$w_{pq} \propto \exp\left(-\frac{1}{\sigma}(\mathcal{I}_p - \mathcal{I}_q)^2\right),$$

where $\mathcal{I}_p$ is the image intensity at pixel p, and σ is a mean contrast of edges. A closer inspection of (1) reveals that our problem can be rewritten as a quadratic program:

$$\text{minimize:} \quad \frac{1}{2} d^T L d \qquad (2)$$

$$\text{subject to:} \quad \begin{array}{ll} d_p - d_q = 0 & \forall\{p,q\} \in \mathcal{U}_= \\ d_p - d_q \geq \varepsilon & \forall\{p,q\} \in \mathcal{U}_> \end{array}$$

where L is a large sparse matrix of size $|\mathcal{I}|^2$ representing the so called Laplace-Beltrami operator [Gra06]. This program can be solved directly using, for instance, an active set method [GMSW84], however, in practice the computation can take tens of seconds even for very small images. To allow instant feedback we propose an approximative solution that yields similar results with significantly lower computational overhead.

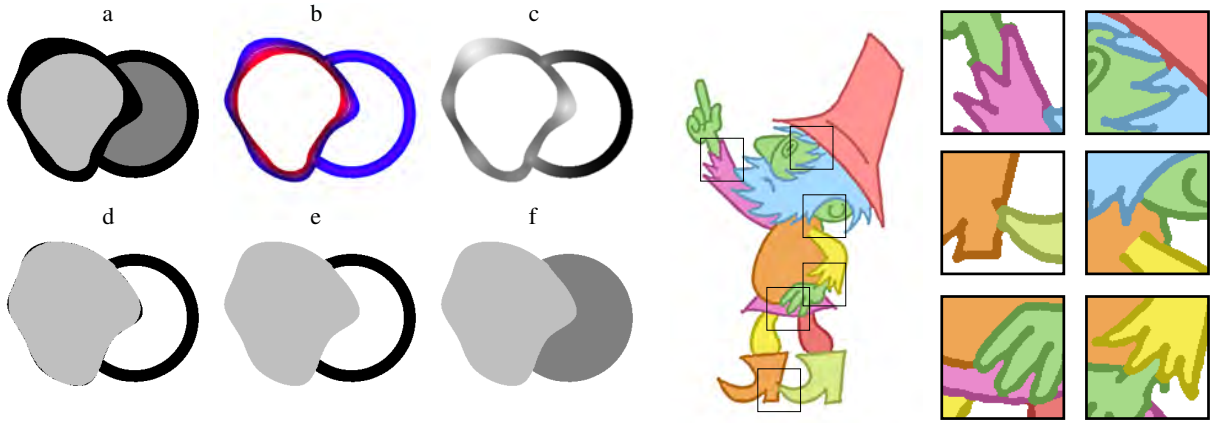

Figure 3: *Depth expansion – synthetic example (left): (a) input depth map with contours, (b) medial axis obtained using two distance transforms computed from active region (red) and all other regions (blue), (c) propagation of contour thickness from medial axis to all contour pixels, (d) depth expansion based on local thickness estimate, (e) filling small gaps, (f) expanded depth map. Practical example (right): several minor artifacts are depicted in selected zoom-ins.*

3.2. Interactive approximation

The key idea of our approximative solution is to decompose the problem into two separate steps: (1) multi-label segmentation (Figure 2a) and (2) depth assignment (Figure 2b). To do so we first treat the user-specified constraints as an unordered set of labels $\mathcal{L}$. Then we exploit an interactive multi-label segmentation algorithm tailored to cartoon images – LazyBrush [SDC09b]. Another alternative is random walker segmentation [Gra06], however, since this algorithm requires us to solve a number of poorly conditioned linear systems it does not provide instant feedback. In contrast LazyBrush exploits fast discrete optimization and furthermore is resistant to imprecise scribble placement.

Once regions are separated we can apply depth inequalities $\mathcal{U}_<$ but now all constraints that fall inside a region are associated with one graph node that corresponds to the underlying region. Such nodes are interconnected by a set of oriented edges representing desired inequalities (see Figure 2b). In general this process can result in an arbitrary oriented graph for which consistent depth assignment (see Figure 2c) exists only if it does not contain oriented loops. The algorithm that solves this problem is known as topological sorting [Kah62] (see Appendix). A part of this algorithm is a detection of oriented loops so that one can easily recognize whether the newly added constraint is consistent. If not, the segmentation phase can be invoked to create a new region and then update the depths with another topological sorting step. Note that since segmentation and depth assignment steps are independent they can be executed incrementally until the desired depth assignment is reached.

When the absolute depths are known, we can easily reconstruct smooth depth transitions (see Figure 2 right) to mimic the result provided by the quadratic program (2). To do so we use a simplified version of (1):

$$\text{minimize:} \sum_{p\in\mathcal{I}} \sum_{q\in\mathcal{N}_p} v_{pq}(d_p - d_q)^2 \qquad (3)$$

$$\text{subject to:} \quad d_p = \hat{d}_p \quad \forall p \in \mathcal{U}_\circ$$

where

$$v_{pq} \propto \begin{cases} \mathcal{I}_p & \text{for } \hat{d}_p \neq \hat{d}_q \\ 1 & \text{otherwise,} \end{cases}$$

$\hat{d}$ denotes depth values from a depth map produced by the topological sorting, and $\mathcal{U}_\circ$ is a subset of pixels used in $\mathcal{U}_<$ or $\mathcal{U}_=$ (see Figure 2d). We let the user decide whether these constraints will be included in $\mathcal{U}_\circ$. The reason we use $\mathcal{I}_p$ instead of 0 is that we have to decide whether the depth discontinuity is real or virtual. Real discontinuities have $\mathcal{I}_p = 0$ but those which are not covered by contours (like upper gaps in Figure 2) have $\mathcal{I}_p = 1$, i.e. they preserve continuity and therefore produce smooth depth transitions. The energy (3) can be minimized by solving the Laplace equation

$$\nabla^2 d = 0$$

with the following boundary conditions ($q \in \mathcal{N}_p$):

$$\begin{array}{lrcl} \text{Dirichlet:} & d_p = \hat{d}_p & \Longleftrightarrow & p \in \mathcal{U}_\circ \\ \text{Neumann:} & d'_{pq} = 0 & \Longleftrightarrow & \hat{d}_p \neq \hat{d}_q \wedge \mathcal{I}_p = 0 \end{array}$$

for which a fast GPU-based solver exists [JCW09].

3.3. Depth expansion

Up to this point we have not yet taken care of pixels covered by contours. As the LazyBrush algorithm places region boundaries inside the contour it is not clear to which region the contour actually belongs. Since this information is inevitable for most applications discussed in this paper

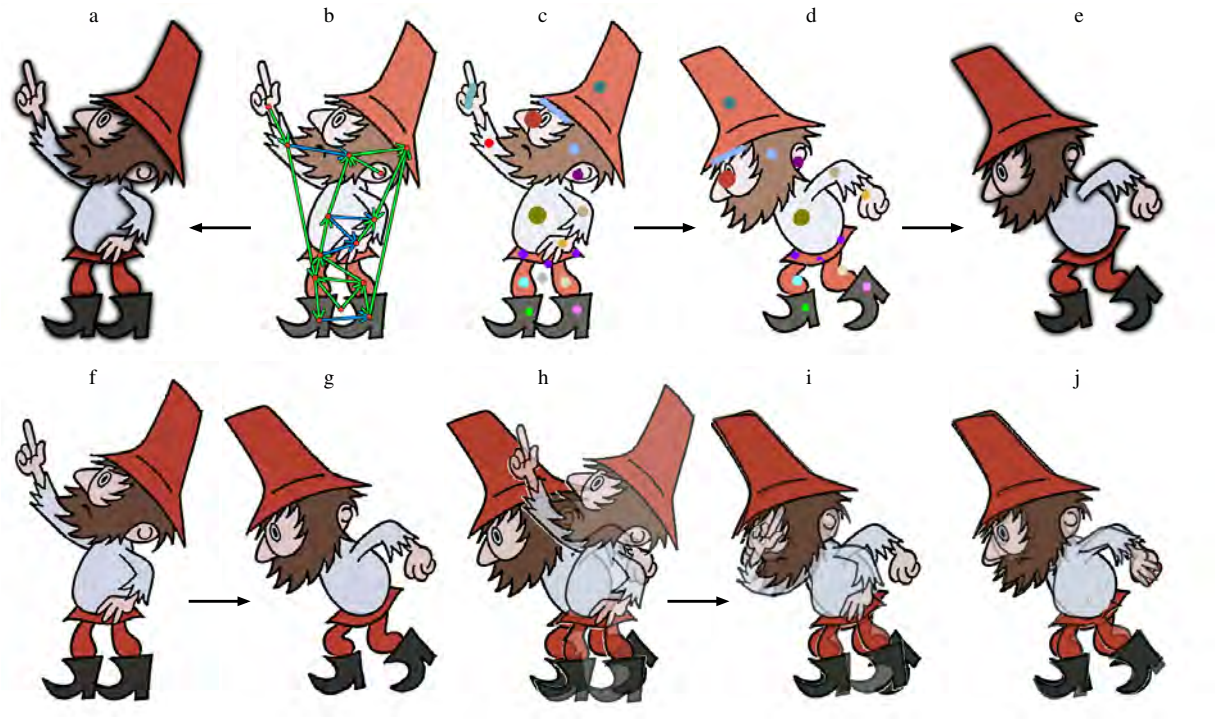

Figure 4: *Depth propagation – transfer of user-provided constraints (top): (a) source image enhanced by a depth map, produced using constraints (b,c), (d) constraints transferred to a target image using as-rigid-as-possible image registration, (e) target image enhanced by a depth map produced using (d). As-rigid-as-possible image registration (bottom): (f) source image, (g) target image, (h) initial overlap, (i) erronous registration without depth map, (j) correct registration using depth map (a) and additional cracks at depth discontinuities specified by blue arrows in (b).*

we have to provide a solution to this depth expansion problem. For drawings containing only thin contours, regions can be consecutively eroded in a front-to-back order. However, such a simple approach can easily lead to noticeable artifacts when used for thick contours. In this case the depth expansion process is equivalent to the figure-ground separation problem described in [PGR99] which is not trivial since it typically requires semantic knowledge that cannot be recovered without additional user intervention.

To solve this we improve upon a previous approach proposed in [SBv05] that in practice produces good results with minor artifacts. We first estimate local contour thickness using two distance maps [FH04]: $\mathcal{D}^1$ computed from the boundaries of regions being expanded (red color in Figure 3b) and $\mathcal{D}^2$ from all other regions (blue color in Figure 3b). Pixels where $\mathcal{D}^1_p = \mathcal{D}^2_p$ form a medial axis from which we can propagate estimates of contour thickness t to all other pixels (Figure 3c) by solving Laplace equation

$$\nabla^2 t = 0$$

using the following boundary conditions ($q \in \mathcal{N}_p$):

$$\begin{array}{lll} \text{Dirichlet:} & t_p = 2\mathcal{D}^1_p & \Longleftrightarrow \quad \mathcal{D}^1_p = \mathcal{D}^2_p \\ \text{Neumann:} & t'_{pq} = 0 & \Longleftrightarrow \quad \mathcal{D}^1_p = 0 \vee \mathcal{D}^2_p = 0. \end{array}$$

Then we expand the region to pixels where $\mathcal{D}^1_p < t_p$ (Figure 3d) and fill in small gaps by removing connected components of which the size is below a predefined threshold (Figure 3e). The expanded region is then removed from the depth map and the same process is applied to all other remaining regions in a front-to-back order to obtain the resulting expansion (Figure 3f).

3.4. Depth propagation

Our depth assignment framework can be easily extended to handle cartoon animations. We can utilize the as-rigid-as-possible image registration algorithm proposed in [SDC09a]. The depth propagation phase is the same as auto-painting: a pre-annotated animation frame (Figure 4c) is registered with a target image (Figure 4g) and user-defined labels $\mathcal{L}$ are automatically transferred (Figure 4d) between corresponding pixels (Figure 4j). After that the LazyBrush algorithm [SDC09b] is used to obtain the final depth map (Figure 4e). Note that since the absolute depth values are already known in the source frame we do not need to execute a topological sorting step.

Known depth values can be also utilized during as-rigid-as-possible image registration to avoid inconsistent con-

figurations which can arise when several parts of the deformed shape overlap each other (Figure 4i). Moreover, the user can additionally specify a subset of depth inequalities (blue arrows in Figure 4b) to mark out discontinuities that will be understood as cracks during the square embedding phase used in the as-rigid-as-possible registration algorithm [SDC09a]. This modification allows more accurate image registration and consequently more accurate depth propagation in cases when several parts of the deformed shape are glued together in the 2D projection although they can move independently (e.g. hands or boots in Figure 4f).

4. Results and Limitations

We have implemented our algorithm in a simple interactive application where the user can freely combine multi-label segmentation with the specification of depth inequalities. A typical workflow suitable for novice users is to first define the segmentation and then add depth inequalities, however, experienced users will tend to combine these two modes to achieve faster progress. They can, for instance, directly generate scribbles with a newly added depth inequality to produce a region and constraint in one step.

Several examples of cartoon pop-ups generated using our algorithm are presented in Figures 5 and 6. User interactions are recorded in two separate layers – one contains equality scribbles and the second contains inequalities. These two layers are hidden during the interaction to avoid clutter. It is typically sufficient to see only the currently added constraint together with an intermediate depth map (Figure 5c) enhanced using a technique that will be discussed later.

Note that the user does not tend to specify a minimal set of inequalities but instead uses a more complicated graph (see Figure 5b). She typically starts with local inequalities and then refines the result by adding more distant constraints. This incremental process nicely corresponds to how the human visual system works. It starts from local cues and then proceeds to more complicated global ones to reach the overall perception of depth in the scene.

As the LazyBrush algorithm [SDC09b] suffers from metrication artifacts when a large part of the contour is missing, we additionally allow the user to specify a set of virtual contours using scribbles or spline strokes (see pink curves with black dots in Figure 6a). However, these curves are required only when the output is expected to be the intermediate discrete depth map (Figure 6c). In the smooth case (Figure 6d), the shape of the virtual contours is not important since it does not correspond to a depth discontinuity and therefore will be smoothed out. Nevertheless, the virtual contours can still be used in order to remove real contours that do not represent depth discontinuities (upper part of trunk in Figure 6) or to introduce them (lower part of dog's head in Figure 6). From this perspective our approximative algorithm brings another important benefit as the pre-segmentation phase automatically removes unimportant details (e.g. trunk wrinkles in Figure 6) that will normally be understood as candidates for depth discontinuities. To obtain the same result using the solution based on quadratic programming (2) it is necessary to mark out all such details as virtual contours.

Limitations. An important limitation of the proposed algorithm is the depth expansion phase described in Section 3.3. Although in general it gives better results as compared to the simple median filter used in [SBv05] it still produces several artifacts (see separate zoomed-in insets in Figure 3). These arise mainly due to the incorrect estimation of contour thickness and/or nontrivial overlapping of contours. Although these small errors are typically imperceptible in most applications it is necessary to correct them manually when a clean depth map is expected. Another drawback lies in the expert annotation workflow when the user starts to combine segmentation with the specification of depth inequalities. Although the proposed approximative algorithm detects cycles automatically it is not able to decide where to put a new scribble in order to resolve the conflict. This limitation makes the interface more complicated as compared to the solution based on quadratic programming, where the the problem is resolved automatically.

5. Applications

In this section we discuss several applications that use depth maps generated by the proposed algorithm. As our research is mainly motivated by the needs of the cartoon animation production pipeline we focus on this field, however, we believe that similar principles are applicable also in a more general image editing context.

Composition and manipulation. Depth maps generated using our system provide invaluable information for interactive shape manipulation [IMH05, WXXC08] where several parts of deformed shapes can overlap each other. Using our approach the user can quickly modify the depth order to enforce the desired visibility (see Figure 1a). The same problem arises in systems where the users extract and compose image fragments [SBv05]. Here depth inequalities allow quick reordering of regions to obtain correct composition (see Figure 1c). Although a similar workflow is used also in [MP09], our approach is more general as it naturally handles self occlusion and does not require any special data structure.

Enhancement and shading. Per-pixel depth values are important also for various image enhancement techniques. In our framework we enhance depth perception by superimposing stacks of regions with blurred boundaries in a back-to-front order (see Figures 1d and 5). This effect is similar to that produced by depth-buffer unsharp masking [LCD06]. Another popular technique that can profit from our algorithm is Lumo [Joh02]. Here per-pixel depth maps help to obtain correct normal interpolation (Figure 1e) which is later used to emulate 3D-like shading (Figure 1f). In the original system the user had to trace region boundaries manually and

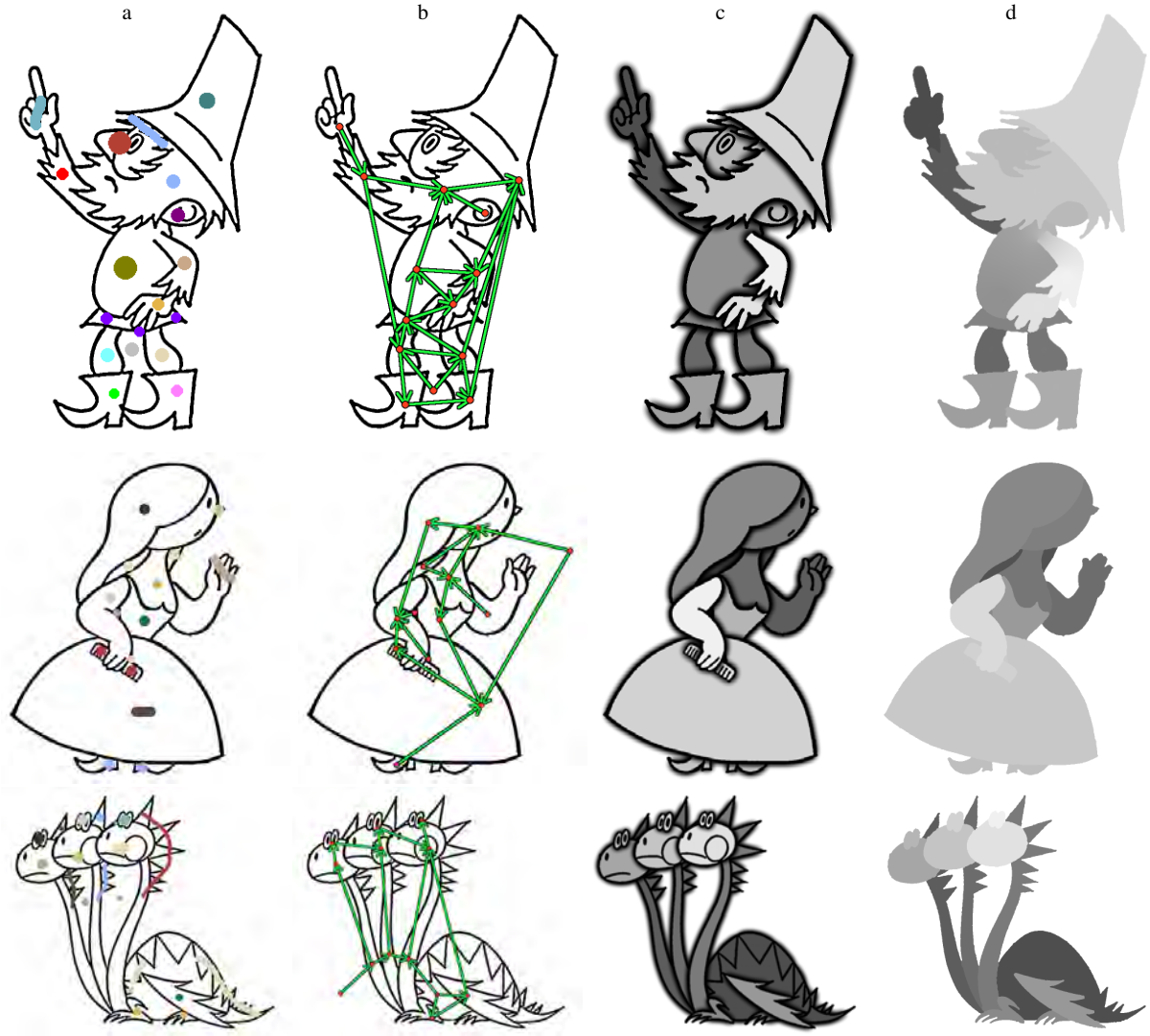

Figure 5: *Examples of cartoon pop-ups generated using our system: (a) user-specified depth equalities, (b) inequalities, (c) depth visualization used during the interactive session to provide visual feedback, (d) final depth map.*

then specify absolute depth values, which is time consuming and counterintuitive. Using our technique correct normal maps can be obtained very quickly.

Modelling and stereo. Using an analogy to shape from shading we can reconstruct an initial blobby surface using the Z-component or normals generated by Lumo [Joh02]. Such surfaces can be further refined using smooth depth maps produced by our framework. We set up a consistent gradient field that respects smooth depth transitions as well as user-provided discontinuities and then integrate it to reconstruct the final height field (see Figure 1h). Such a simple 3D model can be further refined in some modelling tool or rendered from two different viewpoints to obtain stereoscopic images (see Figure 1g).

6. Conclusions and Future work

We have presented a novel interactive approach to cartoon pop-up that enables artists to quickly annotate their handmade drawings and animations with additional depth information. A key benefit of the proposed method as compared to previous depth assignment techniques is that it uses depth (in)equalities instead of absolute depth values. Since this type of constraint better corresponds to the depth recovery mechanism used in the human visual system, it makes the depth assignment more intuitive and less tedious.

As a future work we plan to improve the depth expansion phase and develop an automatic approach for resolving the cycling problem in the expert annotation workflow. We also

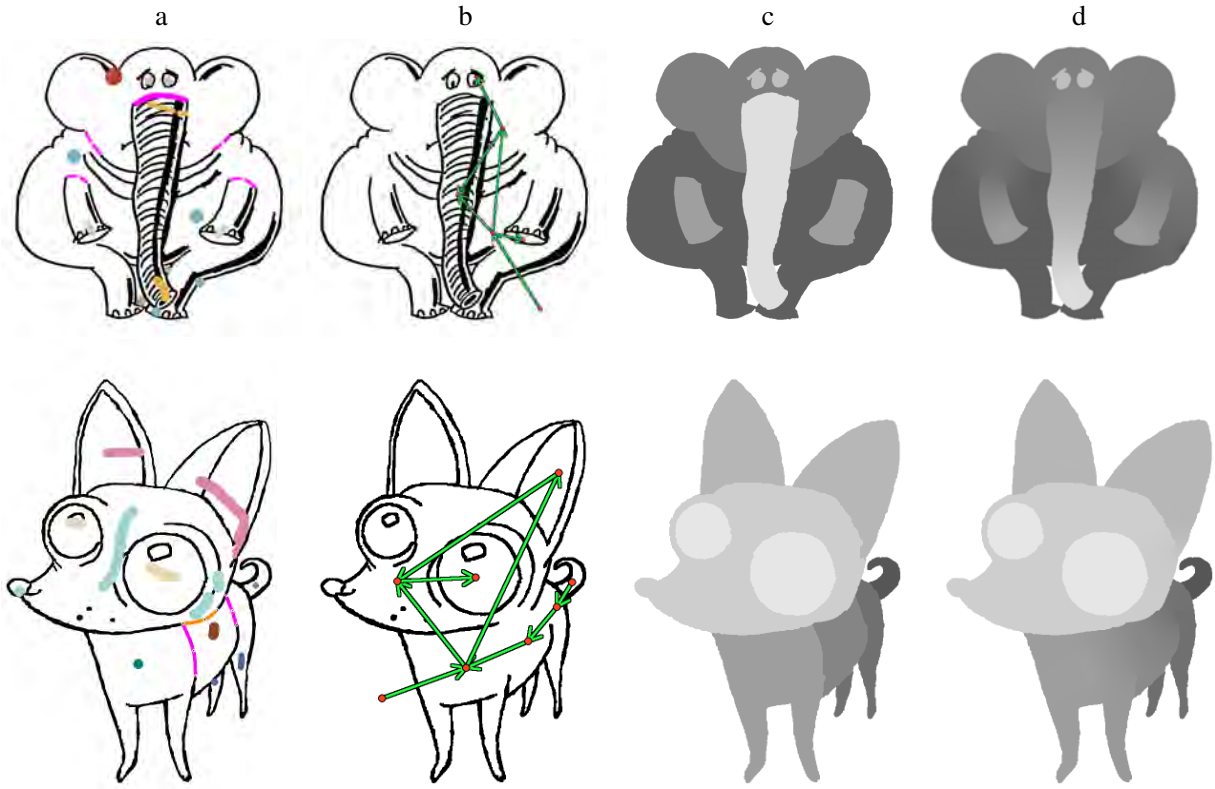

Figure 6: *Examples of cartoon pop-ups generated using our system: (a) user-specified virtual contours (pink) and depth equalities, (b) inequalities, (c) intermediate discrete depth map, (d) final smooth depth map.*

want to extend our approach to work with natural images and find some other applications for using inequalities instead of absolute values in the more general context of interactive image editing.

Acknowledgements

We thank L. Kavan and anonymous reviewers for their fruitful comments. Images used in this paper are courtesy of studios UPP, DMP & Anifilm, P. Koutský, and K. Mlynaříková. This work has been supported by the Marie Curie action IEF, No. PIEF-GA-2008-221320 and partially by the MŠMT under the research program LC-06008 (Center for Computer Graphics).

References

[AW07] ASSA J., WOLF L.: Diorama construction from a single image. *Computer Graphics Forum 26*, 3 (2007), 599–608.

[Fat08] FATTAL R.: Single image dehazing. *ACM Transactions on Graphics 27*, 3 (2008), 72.

[FH04] FELZENSZWALB P. F., HUTTENLOCHER D. P.: *Distance Transforms of Sampled Functions*. Tech. Rep. TR2004-1963, Cornell University, 2004.

[For01] FORSYTH D. A.: Shape from texture and integrability. In *Proceedings of IEEE International Conference on Computer Vision* (2001), pp. 447–453.

[GIZ09] GINGOLD Y., IGARASHI T., ZORIN D.: Structured annotations for 2D-to-3D modeling. *ACM Transactions on Graphics 28*, 5 (2009), 148.

[GMSW84] GILL P. E., MURRAY W., SAUNDERS M. A., WRIGHT M. H.: Procedures for optimization problems with a mixture of bounds and general linear constraints. *ACM Transactions on Mathematical Software 10*, 3 (1984), 282–298.

[Gra06] GRADY L.: Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 11 (2006), 1768–1783.

[HEH05] HOIEM D., EFROS A. A., HEBERT M.: Automatic photo pop-up. *ACM Transactions on Graphics 24*, 3 (2005), 577–584.

[HiAA97] HORRY Y., ICHI ANJYO K., ARAI K.: Tour into the picture: Using a spidery mesh interface to make animation from a single image. In *ACM SIGGRAPH Conference Proceedings* (1997), pp. 225–232.

[Hor90] HORN B. K. P.: Height and gradient from shading. *Internation Journal of Computer Vision 5*, 1 (1990), 37–75.

[HST09] HE K., SUN J., TANG X.: Single image haze removal using dark channel prior. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (2009), pp. 1956–1963.

[IMH05] IGARASHI T., MOSCOVICH T., HUGHES J. F.: As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics 24*, 3 (2005), 1134–1141.

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy:

A sketching interface for 3D freeform design. In *ACM SIGGRAPH Conference Proceedings* (1999), pp. 409–416.

[JC08] JOSHI P., CARR N. A.: Repoussé: Automatic inflation of 2D artwork. In *Proceedings of Eurographics Workshop on Sketch-Based Interfaces and Modeling* (2008), pp. 49–55.

[JCW09] JESCHKE S., CLINE D., WONKA P.: A GPU Laplacian solver for diffusion curves and Poisson image editing. *ACM Transaction on Graphics 28*, 5 (2009), 116.

[Joh02] JOHNSTON S. F.: Lumo: Illumination for cel animation. In *Proceedings of International Symposium on Non-photorealistic Animation and Rendering* (2002), pp. 45–52.

[Kah62] KAHN A. B.: Topological sorting of large networks. *Communications of the ACM 5*, 11 (1962), 558–562.

[Kan98] KANG S. B.: *Depth Painting for Image-Based Rendering Applications*. Tech. rep., Cambridge Research Laboratory, 1998.

[KH06] KARPENKO O. A., HUGHES J. F.: SmoothSketch: 3D free-form shapes from complex sketches. *ACM Transactions on Graphics 25*, 3 (2006), 589–598.

[Koe98] KOENDERINK J. J.: Pictorial relief. *Philosophical Transactions of the Royal Society of London 356*, 1740 (1998), 1071–1086.

[KvDK96] KOENDERINK J. J., VAN DOORN A. J., KAPPERS A. M. L.: Pictorial surface attitude and local depth comparisons. *Perception & Psychophysics 58*, 2 (1996), 163–173.

[LCD06] LUFT T., COLDITZ C., DEUSSEN O.: Image enhancement by unsharp masking the depth buffer. *ACM Transactions on Graphics 25*, 3 (2006), 1206–1213.

[LFG08] LEE S., FENG D., GOOCH B.: Automatic construction of 3D models from architectural line drawings. In *Proceedings of Symposium on Interactive 3D Graphics* (2008), pp. 123–130.

[LLW04] LEVIN A., LISCHINSKI D., WEISS Y.: Colorization using optimization. *ACM Transactions on Graphics 23*, 3 (2004), 689–694.

[LS96] LIPSON H., SHPITALNI M.: Optimization-based reconstruction of a 3D object from a single freehand line drawing. *Computer-Aided Design 28*, 8 (1996), 651–663.

[MP09] MCCANN J., POLLARD N. S.: Local layering. *ACM Transactions on Graphics 28*, 3 (2009), 84.

[NISA07] NEALEN A., IGARASHI T., SORKINE O., ALEXA M.: FiberMesh: Designing freeform surfaces with 3D curves. *ACM Transactions on Graphics 26*, 3 (2007), 41.

[NN94] NAYAR S. K., NAKAGAWA Y.: Shape from focus. *IEEE Transactions on Pattern Analysis and Machine Intelligence 16*, 8 (1994), 824–831.

[OCDD01] OH B. M., CHEN M., DORSEY J., DURAND F.: Image-based modeling and photo editing. In *ACM SIGGRAPH Conference Proceedings* (2001), pp. 433–442.

[OCN04] ONO Y., CHEN B.-Y., NISHITA T.: 3D character model creation from cel animation. In *Proceedings of International Conference on Cyberworlds* (2004), pp. 210–215.

[PFWF00] PETROVIĆ L., FUJITO B., WILLIAMS L., FINKELSTEIN A.: Shadows for cel animation. In *ACM SIGGRAPH Conference Proceedings* (2000), pp. 511–516.

[PGR99] PAO H.-K., GEIGER D., RUBIN N.: Measuring convexity for figure/ground separation. In *Proceedings of IEEE International Conference on Computer Vision* (1999), pp. 948–955.

[RT09] RUSSELL B., TORRALLBA A.: Building a database of 3D scenes from user annotations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (2009), pp. 2711–2718.

[SB95] SUPER B. J., BOVIK A. C.: Shape from texture using local spectral moments. *IEEE Transactions on Pattern Analysis and Machine Intelligence 17*, 4 (1995), 333–343.

[SBv05] SÝKORA D., BURIÁNEK J., ŽÁRA J.: Sketching cartoons by example. In *Proceedings of Eurographics Workshop on Sketch-Based Interfaces and Modeling* (2005), pp. 27–34.

[SDC09a] SÝKORA D., DINGLIANA J., COLLINS S.: As-rigid-as-possible image registration for hand-drawn cartoon animations. In *Proceedings of International Symposium on Non-photorealistic Animation and Rendering* (2009), pp. 25–33.

[SDC09b] SÝKORA D., DINGLIANA J., COLLINS S.: LazyBrush: Flexible painting tool for hand-drawn cartoons. *Computer Graphics Forum 28*, 2 (2009), 599–608.

[SSN09] SAXENA A., SUN M., NG A. Y.: Make3D: Learning 3D scene structure from a single still image. *IEEE Transactions on Pattern Analysis and Machine Intelligence 31*, 5 (2009), 824–840.

[VDH09] VENTURA J., DIVERDI S., HÖLLERER T.: A sketch-based interface for photo pop-up. In *Proceedings of Eurographics Symposium on Sketch-Based Interfaces and Modeling* (2009), pp. 21–28.

[VM02] VARLEY P. A. C., MARTIN R. R.: Estimating depth from line drawings. In *Proceedings of ACM Symposium on Modeling and Application* (2002), pp. 180–191.

[Wil90] WILLIAMS L.: 3D paint. *ACM SIGGRAPH Computer Graphics 24*, 2 (1990), 225–233.

[WSTS08] WU T.-P., SUN J., TANG C.-K., SHUM H.-Y.: Interactive normal reconstruction from a single image. *ACM Transactions on Graphics 27*, 5 (2008), 119.

[WXXC08] WANG Y., XU K., XIONG Y., CHENG Z.-Q.: 2D shape deformation based on rigid square matching. *Computer Animation and Virtual Worlds 19*, 3–4 (2008), 411–420.

[ZDPSS01] ZHANG L., DUGAS-PHOCION G., SAMSON J.-S., SEITZ S. M.: Single view modeling of free-form scenes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (2001), pp. 990–997.

Appendix

Let us assume we have a graph $G(V,E)$ consisting of nodes V interconnected by a set of oriented edges E. In addition to that we have an empty set L and a set S that contains all nodes having no incoming edges. Topological sorting of V can be obtained using the following algorithm [Kah62]:

while $S \neq \emptyset$ **do**

$S := S - \{n\}$ and $L := L \cup \{n\}$

for $\forall m \in V$ having edge $e : n \rightarrow m$ **do**

$E := E - \{e\}$

if m has no other incoming edges **then**

$S := S \cup \{m\}$

endfor

endwhile

if $E \neq \emptyset$ **then**

G has at least one oriented cycle **else**
L contains topologically sorted nodes.

Appendix D

Ink-and-Ray: Bas-Relief Meshes for Adding Global Illumination Effects to Hand-Drawn Characters

D. Sýkora, L. Kavan, M. Čadík, O. Jamriška, A. Jacobson, B. Whited, M. Simmons, O. Sorkine-Hornung: Ink-and-Ray: Bas-Relief Meshes for Adding Global Illumination Effects to Hand-Drawn Characters. ACM Transactions on Graphics, in press, accepted for publication in December 2013. ISSN 0730-0301. **IF=3.361**

Ink-and-Ray: Bas-Relief Meshes for Adding Global Illumination Effects to Hand-Drawn Characters

DANIEL SÝKORA
CTU in Prague, FEE
and
LADISLAV KAVAN
University of Pennsylvania / ETH Zurich
and
MARTIN ČADÍK
MPI Informatik / CTU in Prague, FEE / Brno University of Technology
and
ONDŘEJ JAMRIŠKA
CTU in Prague, FEE
and
ALEC JACOBSON
ETH Zurich
and
BRIAN WHITED and MARYANN SIMMONS
Walt Disney Animation Studios
and
OLGA SORKINE-HORNUNG
ETH Zurich

We present a new approach for generating global illumination renderings of hand-drawn characters using only a small set of simple annotations. Our system exploits the concept of bas-relief sculptures, making it possible to generate 3D proxies suitable for rendering without requiring side-views or extensive user input. We formulate an optimization process that automatically constructs approximate geometry sufficient to evoke the impression of a consistent 3D shape. The resulting renders provide the richer stylization capabilities of 3D global illumination while still retaining the 2D hand-drawn look-and-feel. We demonstrate our approach on a varied set of hand-drawn images and animations, showing that even in comparison to ground-truth renderings of full 3D objects, our bas-relief approximation is able to produce convincing global illumination effects, including self-shadowing, glossy reflections, and diffuse color bleeding.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Color, shading, shadowing, and texture*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Curve, surface, solid, and object representations*

General Terms: Algorithms, Design, Human Factors

Additional Key Words and Phrases: Cartoons, global illumination, non-photorelasitic rendering, 2D-to-3D conversion, bas-relief

ACM Reference Format:

Daniel Sýkora, Ladislav Kavan, Martin Čadík, Ondřej Jamriška, Alec Jacobson, Maryann Simmons, Brian Whited, Olga Sorkine-Hornung. 2013. Ink-and-Ray: Bas-Relief Meshes for Adding Global Illumination Effects to Hand-Drawn Characters. *ACM Trans. Graph.* 28, 4, Article 106 (September 2013), 15 pages.
DOI: http://dx.doi.org/?

Authors' addresses: Daniel Sýkora, (Current address) DCGI FEE CTU in Prague, Karlovo náměstí 13, 121 35 Prague 2, Czech Republic.

DOI: http://dx.doi.org/?

1. INTRODUCTION

Despite the recent success and proliferation of 3D computer-generated imagery, traditional 2D hand-drawn animation is still a popular medium for animated films. Even in modern 3D production pipelines, 2D animation plays an important role during preproduction (such as in the story, layout, and animatic phases). The key

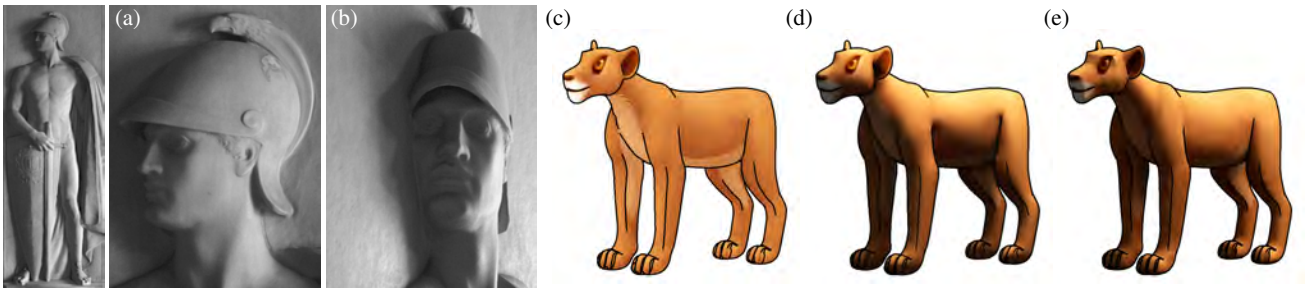

Fig. 1. Bas-relief sculpture—(a) from a frontal view the sculpture appears like a full 3D object, (b) while a side view reveals unreal depth proportions [Belhumeur et al. 1999]. (c) Input 2D character for which global illumination is computed using (d) our bas-relief like approximation and (e) the full 3D model. (Photographs © Kluwer Academic Publishers. All rights reserved.)

advantage of the 2D drawing metaphor is that it provides complete artistic freedom without the restrictions imposed by a CG environment (e.g., limited flexibility due to the 3D model and its animation rig). However, those very limitations are what make complex effects such as global illumination feasible in the computer-generated medium. Such effects are extremely labor-intensive to create for every frame of a traditionally animated film.

In this paper we introduce a new approach to the production of CG images which preserves the look-and-feel and artistic advantages of the 2D domain but at the same time delivers a look similar to that produced by a complex full 3D pipeline. Our technique is inspired by bas-relief sculptures [Read 1961] which demonstrate that fully consistent 3D models are not necessary for an appealing and convincing depiction of 3D shapes (Fig. 1a–b). The phenomenon behind this observation is known as the *bas-relief ambiguity* [Belhumeur et al. 1999] which states that an orthographic projection of a 3D object under global illumination is ambiguous under stretching or shearing transformations in depth. This is true for the lit areas of the surface, as well as cast and attached shadows. In other words the bas-relief ambiguity says that the knowledge of accurate depth values is not necessary to produce believable advanced illumination effects.

Instead, as one can observe from bas-relief sculptures, the models need to contain explicit, though possibly simplified, surface geometry (as opposed to a normal field only). In addition, the representation needs to capture consistent depth order, and appropriate separation, contact and continuity between individual surface components. Explicit geometry, ordering and separation are necessary features to capture effects produced by light bouncing between surface components, passing through gaps between surfaces, and being occluded by surfaces. Without appropriate contact, light will leak through components that are supposed to be connected/coincident and continuity is needed to control whether the surface interacts with light as a smoothly connected or discontinuous component. See Fig. 3 for an illustration of the importance of individual requirements. Our novel *ink-and-ray* framework facilitates fast creation of proxies containing the above features which we demonstrate are accurate enough to produce 3D global illumination effects including complex self shadowing, glossy reflections, and color bleeding.

To enable the proposed concept several new technical contributions have been made. A key part is the stitching phase where individual inflated layers are joined together in a prescribed relative depth order. Here a new Dirichlet type of energy with inequality constraints followed by $\mathcal{C}^1$–continuous biharmonic smoothing is formulated and solved. In addition, we present an approach for relative depth order estimation based on the concept of illusory surfaces [Geiger et al. 1998] solved via diffusion curves [Orzan et al. 2008] and a new Poisson-based formulation of the layer inflation process.

2. RELATED WORK

The bas-relief ambiguity was studied only with Lambertian reflectance, however, similar observations about the lower sensitivity of the human visual system to inconsistencies in illumination of 3D objects have been made in more complex scenarios [Ostrovsky et al. 2005]. Khan et al. [2006] demonstrated the practical utility of this phenomenon in their framework for editing materials in natural images. They reconstruct the approximate geometry of 3D objects using simple filtering of image luminance and a rough estimation of environment lighting in order to produce renderings of the objects with different materials. A similar workflow is used by Lopez-Moreno et al. [2010] to create stylized depictions of images. This type of approach is not applicable to hand-drawn characters lacking reliable shading information from which to extract depth information.

Another popular technique for approximating 3D-like shading without leaving the 2D domain is the use of estimated normal fields. *Lumo* [Johnston 2002] generates such an approximation from a line drawing by interpolating the normals on region boundaries. Extensions to the base Lumo approach [Okabe et al. 2006; Winnemöller et al. 2009; Shao et al. 2012] obtain better control over the values in the generated normal field. However, due to the lack of depth information, more complex global illumination effects such as reflections, color bleeding, and self shadowing are not supported.

Toler-Franklin et al. [2007] use curvature and directional discontinuities of estimated normal fields to emulate effects such as ambient occlusion and local self-shadowing. Similarly in *TexToons* [Sýkora et al. 2011], depth layering is used to enhance textured images with ambient occlusion, shading, and texture rounding effects. Recently Vergne et al. [2012] presented a new approach to simulate complex lighting effects where a user-defined flow field deforms the input image so that it appears to be reflected from a specific surface. These methods produce a 3D-like look, however the overall appearance can still feel synthetic due to the lack of subtle global illumination effects which require some approximation of a 3D model.

In general, approaches for constructing 3D models from 2D input can be classified into two main groups: methods focused on creating fully consistent 3D models and methods producing 2.5D approximations such as height fields.

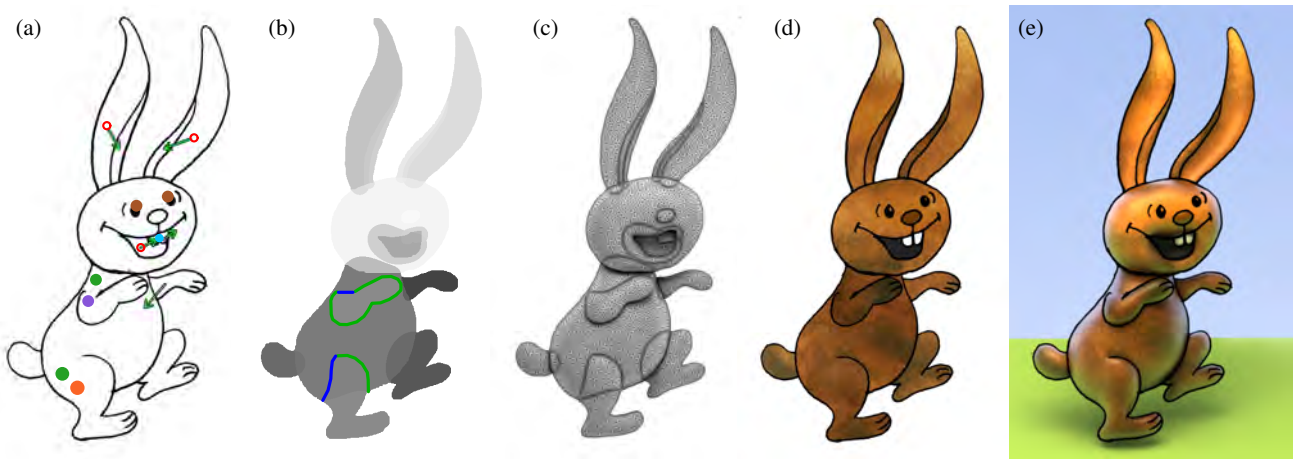

Fig. 2. Ink-and-ray pipeline—(a) Hand-drawn image with user-specified annotations to guide segmentation and depth ordering. (b) Regions shaded according to depth order including estimated occluded silhouettes and grafting boundary conditions shown in blue and green. (c) Proxy 3D mesh. (d) Original drawing used as a texture in render. (e) Global illumination render. (Source drawing © Anifilm. All rights reserved.)

Reconstruction of 3D models from line drawings is a central challenge in the field of computer vision [Malik 1986]. In general, the problem is highly under-constrained, and additional knowledge is typically required to obtain reasonable results automatically [Wang et al. 2009; Cole et al. 2012]. The situation becomes even more complicated when the input is man-made drawings or sketches. In this scenario, important geometric rules are broken due to inaccuracies or deliberate stylization, and thus manual intervention becomes inevitable, making the process more akin to modeling from scratch than reconstruction.

With sketch-based approaches for creating full 3D models, the user often traces curves over the original image or sketches regions directly with a mouse or a tablet. A concept of surface inflation [Igarashi et al. 1999] enhanced by intelligent topological embedding [Karpenko and Hughes 2006], additional control curves [Nealen et al. 2007], or replaced by geometric primitives [Gingold et al. 2009], is then used to produce an initial approximation of the 3D object. The user can add more shapes, make connections, and arrange them in the 3D space by working with side views. Some additional annotations [Gingold et al. 2009; Olsen et al. 2011] can be used to improve the shape's structure and topology. Similarly, other sketch-based methods have been proposed for producing 2.5D approximations instead of full 3D surfaces [Ono et al. 2004; Chen et al. 2005; Joshi and Carr 2008; Andrews et al. 2011]. A common drawback of these approaches is that they require tedious specification of control curves with positional and directional constraints to produce the desired results. Moreover, they typically assume the resulting surface is a height field which inherently limits the range of illumination effects (e.g., light cannot pass through holes in the object).

Sýkora et al. [2010] presented a depth assignment framework for hand-drawn images utilizing simple user-specified annotations to produce flat height fields with smooth depth transitions that can be further inflated using a reconstructed normal field. Although this approach shares interaction simplicity and produces approximations close to our bas-relief structure, the resulting surface is only a 2.5D height field with arbitrary discontinuities. As such, subsequent inflation may not respect the prescribed depth order and can easily lead to large inconsistencies such as self-intersections.

The approach of Rivers et al. [2010] allows quick creation of 2.5D models. While the result supports 3D rotation, it does not produce 3D surfaces which are necessary for simulating global illumination effects. Petrović et al. [2000] show that a simple approximation of a 3D model is sufficient for generating believable cast shadows for cel animation. Their technique bears some resemblance to our approach, but requires extensive manual intervention and working with side views to obtain the desired results.

Although some of the previous approaches could be used to create proxy 3D models for rendering, our new ink-and-ray pipeline greatly simplifies the process. The bas-relief mesh contains enough 3D information to produce global illumination effects without requiring absolute depth proportions. This enables artists to work entirely in the 2D domain. Based on a few simple annotations, the system automatically creates a stack of inflated layers which preserve relative depth order, contact, and continuity, without requiring image tracing or tedious specification of control curves.

3. INK-AND-RAY PIPELINE

In our proposed ink-and-ray pipeline, the process of creating a bas-relief-type mesh from a hand-drawn image consists of six main steps: segmentation, completion, layering, inflation, stitching, and grafting. In this section we give an overview and provide motivation for each step.

Segmentation In the segmentation phase, the image is partitioned into a set of regions which preserve the individual components delineated by outlines in the original image (see Fig. 6b). Without segmentation, the resulting global illumination render would omit important details, producing only a simplistic balloon-like appearance (see Fig. 3a).

Completion When an extracted region includes occluded silhouettes, the hidden parts are estimated using shape completion (see Fig. 6d). This completion aids in the layering and inflation phases (see Fig. 3b and Fig. 12). It also helps to produce correct cast shadows for occluded parts (see Fig. 4).

Layering Based on the shape completion, a relative depth order can be assigned to the regions, producing a stack of layers (see Fig. 6c). Layering has a direct impact on the quality of the resulting illumination – it helps to keep the overall structure con-

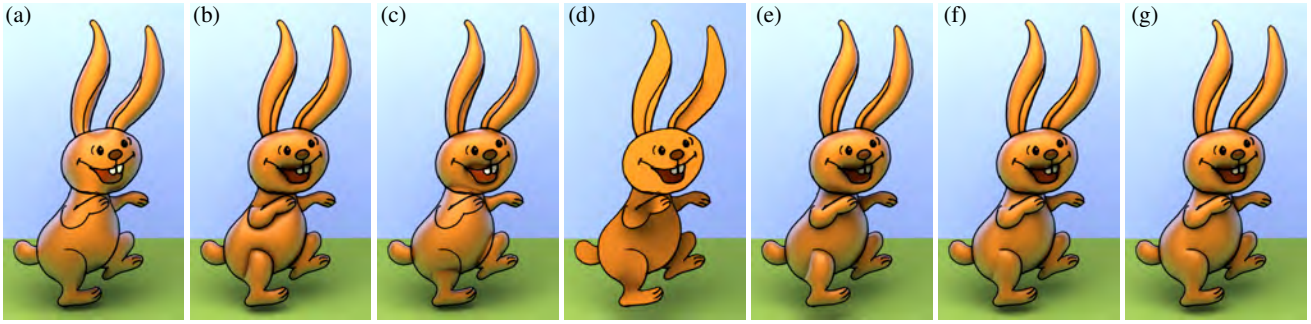

Fig. 3. Influence of the individual steps of the ink-and-ray pipeline on the resulting global illumination render (g). In each example above, one of the steps is omitted: (a) segmentation – geometric details are missing, (b) completion – occlusion destroys shape consistency, (c) layering – inflated segments penetrate, (d) inflation – lack of volume, (e) stitching – segments hover in space, (f) grafting – $\mathcal{C}^1$ discontinuities. (Source drawing © Anifilm. All rights reserved.)

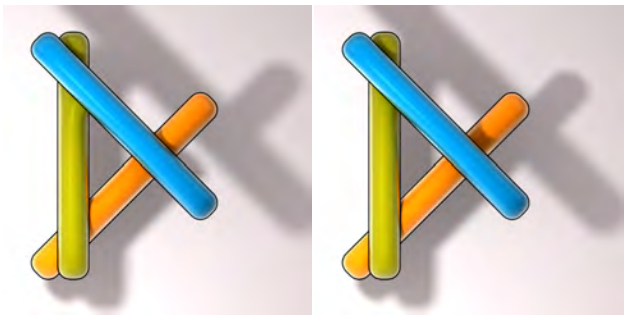

Fig. 4. Artifacts can appear when the shape of an occluded part is not completed (left). With completion, occluded parts of the reconstructed mesh are correctly reflected in the cast shadows (right).

sistent by avoiding surface intersections during the stitching phase (see Fig. 3c) and allows holes which are important for plausible light transport (see Fig. 5).

Inflation Each region is inflated to produce a 3D surface with volume (see Fig. 2c). This enables direct illumination effects such as diffuse shading and glossy highlights as well as global illumination effects arising from the light interaction between objects. Without inflation the resulting render would look like a scene from a pop-up book (see Fig. 3d).

Stitching Adjacent inflated regions are stitched together at contours according to their relative depth order (see Fig. 13a–b). This influences global illumination effects such as self-shadowing, color bleeding, and glossy reflections. Without stitching, the resulting render would reveal missing contacts between individual parts (see Fig. 3e).

Grafting Finally, where the outline is missing in the original image, grafting replaces unwanted $\mathcal{C}^0$ contours with smooth $\mathcal{C}^1$ transitions (see Fig. 13c–d). This step is important in preserving the smoothness of direct illumination, preventing visible discontinuities (see Fig. 3f).

The resulting proxy can then be textured and rendered.

4. ALGORITHM

In this section, each step of the proposed ink-and-ray pipeline outlined above is described in more detail.

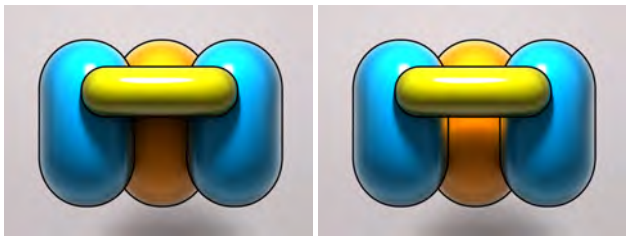

Fig. 5. Approaches that use representations based on height fields (left) can not represent holes. Our layering approach preserves plausible depth proportions and supports holes (right).

4.1 Segmentation

The input to our system is a scanned hand-drawn image or a digital sketch (Fig. 6a). The first step partitions the drawing into a set of regions, each of which should correspond to a logical structural component of the hypothetical 3D object depicted in the drawing (Fig. 6b).

The original image will be used as a texture during the rendering phase, and therefore it is not necessary for the segmentation to be carried out to the finest detail level (e.g., eyes can be part of the head region and the teeth together can form a single region as in Fig. 6b). On the other hand, it is important to separate out regions that represent articulated parts (for example, the limbs in Fig. 6), and in some cases to join parts broken into multiple components due to occlusion (e.g., the ear in Fig. 6 or the wolf's tail in Fig. 26).

For "clean" drawings, where regions are distinctly separated by closed outlines, the base partitioning can be done without user intervention [Sýkora et al. 2005]. In cases where this is not sufficient, the user can then perform detail reduction and region joining (see selection strokes in Fig. 6a). For articulations and more complex drawings containing rough strokes with gaps, scribble-based segmentation tools tailored to scanned images (e.g., LazyBrush [Sýkora et al. 2009]) or digital sketches (e.g., SmartScribbles [Noris et al. 2012]) can be used.

When separating articulations, the *LazyBrush* algorithm may produce artifacts (e.g., the squaring off of the top of the arm in Fig. 6b). This problem typically does not affect the final geometry as the region will be smoothly grafted to the base mesh. However, for cases where the shape is important, the silhouette completion mechanism described in the following section can be utilized (see leg in Fig. 6d).

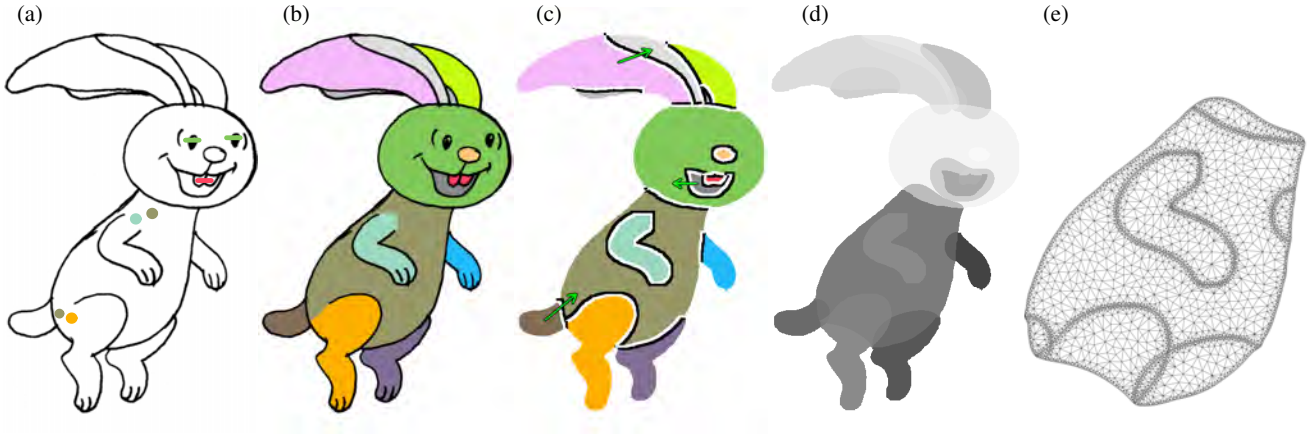

Fig. 6. Converting an input image into a set of regions with depth order—(a) the input image with user-specified annotations for region identification, (b) resulting segmentation, (c) estimation of relative depth order (black pixels mark the side of the region boundary with greater depth value, white pixels lie on the side which is relatively closer), together with relative depth corrections specified by the user (green arrows), (d) regions with depth order and illusory silhouettes in occluded parts (lighter portions are closer), (e) conforming constrained Delaunay triangulation with Steiner points at boundaries of intersecting regions. (Source drawing © Anifilm. All rights reserved.)

4.2 Completion

Once the image is partitioned into a set of regions, the next step is to complete silhouettes where hand-drawn outlines are missing in the original image either due to occlusion or articulation. The completion approach described below is inspired by techniques for reconstruction of illusory surfaces [Geiger et al. 1998].

Consider the problem of estimating the illusory surface in the classical Kanizsa square example (Fig. 7a). The boundary of the illusory square at the corner pixels is known and therefore a hypothesis can be initialized where these pixels are set appropriately to be inside ($p = 1$, white in Fig. 7b) or outside ($p = 0$, black in Fig. 7b) the square. As in Geiger et al. [1998], a diffusion process similar to that used in diffusion curves [Orzan et al. 2008] (Fig. 7c) can be applied to propagate these probabilities into the unknown pixels (gray in Fig. 7b). Finally pixels with probabilities $p > 0.5$ are classified to lie inside the illusory surface (Fig. 7d).

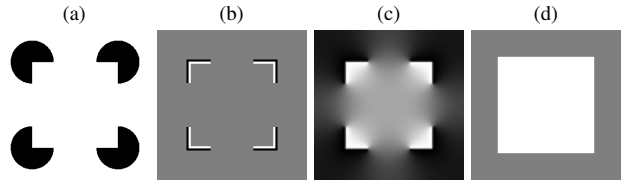

Fig. 7. Estimation of illusory surface—(a) Kanizsa square, (b) initial hypothesis: white pixels $p = 1$ are inside, black $p = 0$ are outside, gray are unknown, (c) diffusion of p into unknown pixels, (d) pixels with $p > 0.5$ lie inside the illusory surface.

4.3 Layering

The silhouette completion mechanism is used in the layering step to predict the relative depth order between regions.

Given two regions A and B (Fig. 8a), we would like to test if region B is occluded by region A. This can be determined by computing B's illusory surface B' (Fig. 8c) and checking whether the area of B' is greater than the area of B (Fig. 8c).

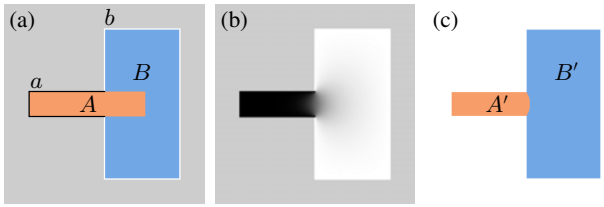

Fig. 8. Inferring relative depth order—(a) an illusory surface B' of the region B is estimated using two diffusion curves a and b with values $p_a = 0$ black and $p_b = 1$ white, (b) p after the diffusion, (c) pixels which belong to B' have $p > 0.5$. As the area of B' is bigger than area of B, we can conclude region B was occluded by region A.

The illusory surface B' is created by first constructing two diffusion curves [Orzan et al. 2008]: a with value $p_a = 0$ and b with $p_b = 1$ (Fig. 8a), which represent silhouette pixels of the corresponding regions, i.e., all boundary pixels of the region that do not touch the other region. These values are then diffused into the compound area of both regions (Fig. 8b). Interior pixels with $p > 0.5$ then correspond to the illusory surface B'.

This process can be applied to each pair of neighboring regions (see resulting predicted depth order in Fig. 6c). The user can then correct possible prediction errors using additional depth inequalities (arrows in Fig. 6c). A graph of relative inequalities is constructed [Sýkora et al. 2010] and topologically sorted to obtain absolute depth values for each region (Fig. 6d).

Once the relative depth order of regions is known, the silhouettes of occluded regions can be further refined. Given a region A that is known to be occluded by a region B (Fig. 9), the diffusion curve a can be modified to include the boundary of region B (Fig. 9a). The diffusion process is then applied (Fig. 9b) and pixels with probability p lower than 0.5 correspond to the refined illusory surface A' (Fig. 9c). B is known to be in front of A, and therefore its shape remains unchanged.

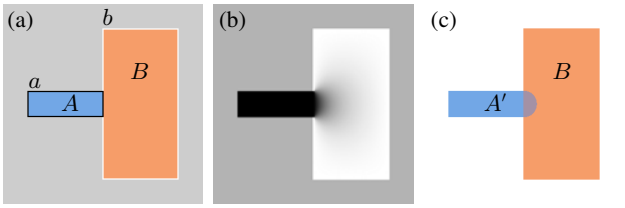

Fig. 9. Completion of occluded silhouettes—(a) an illusory surface A' of the region A occluded by the region B is estimated using two diffusion curves a and b with values $p_a = 0$ (black) and $p_b = 1$ (white), (b) p after the diffusion, (c) pixels which belong to A' have $p < 0.5$, the shape of the region B remains unchanged.

4.4 Inflation

Given a labeling of segmented regions $\Omega_i \subseteq \mathbb{R}^2$, where i is the region index, the goal of the inflation phase is to find functions $\tilde{f}_i : \Omega_i \rightarrow \mathbb{R}$ corresponding to a buckled shape. We find $\tilde{f}_i$ by solving a Poisson equation

$$-\nabla^2 \tilde{f}_i(\mathbf{x}) = c_i \quad \forall \mathbf{x} \in int(\Omega_i) \tag{1}$$

where $c_i > 0$ is a scalar specifying how much the inflated surface should buckle. Both Dirichlet and Neumann boundary constraints are supported on $\partial\Omega_i$:

$$\tilde{f}_i(\mathbf{x}) = 0 \quad \forall \mathbf{x} \in B_D \tag{2}$$

$$\frac{\partial \tilde{f}_i}{\partial \mathbf{n}}(\mathbf{x}) = 0 \quad \forall \mathbf{x} \in B_N \tag{3}$$

where the disjoint sets $B_D \cup B_N = \partial\Omega_i$ describe where each type of boundary constraint applies. We solve this Poisson equation using the standard finite element discretization with piecewise linear shape functions, reducing the problem to a linear system solve. To produce this discretization, we employ boundary tracing [Ren et al. 2002] of Ω_i and apply conforming constrained Delaunay triangulation [Shewchuk 2002]. Additional Steiner points are added where silhouettes of other regions intersect the interior of Ω_i (Fig. 6e) and used during the stitching phase to impose constraints on coincident vertices.

The resulting $\tilde{f}_i$ produces a parabolic profile (Fig. 10a). If desired, the user can specify a custom cross-section function to convert the profile into an alternative shape. A typical example of such a function is $f_i(x) = d_i\sqrt{\tilde{f}_i(x)}$, where $d_i \in \mathbb{R}$ is a scaling factor which makes it possible to obtain spherical profiles (Fig. 10b). This can be applied, for example, when the user wants to produce a concave or flatter profile (see the ears and mouth in Fig. 2, where a lower negative d_i was used to simulate flatter concavity).

Fig. 10. Shape profiles—(a) initial parabolic profile produced by the Poisson equation, (b) spherical profile obtained by applying a cross-section function with square root.

The shape of the inflated region can further be modified by varying the boundary conditions B. Dirichlet boundary conditions are used by default. To prevent rounding at places where the surface should remain straight, Neumann boundary conditions can be utilized (see the sleeve and lower part of the jacket in Fig. 11). To selectively change the type of boundary condition, the user clicks on two endpoints and the system automatically finds the path that connects the points along a boundary of the nearest region (see dots and curves in Fig. 11).

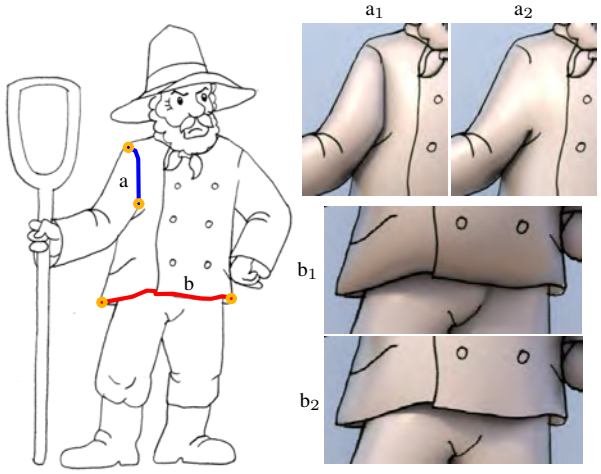

Fig. 11. Modifying boundary conditions—(a,b) the user clicks on endpoints (orange dots) and the system automatically finds the path that connects them along a boundary of the nearest region (red and blue curves). (a_1,b_1) The resulting shapes when the default settings (Dirichlet boundary conditions B_D, stitching equalities $C^{=}_{i,j}$, no grafting) are used. (a_2) Result when Neumann boundary conditions B_N are used together with grafting to produce $\mathcal{C}^1$-continuous stitching. (b_2) Result when Neumann boundary conditions B_N are used with stitching inequalities $C^{\geq}_{i,j}$ to avoid rounded boundaries. (Source drawing © Anifilm. All rights reserved.)

A similar approach to inflation was previously used in TexToons [Sýkora et al. 2011]. Here normal interpolation (originally proposed in Lumo [Johnston 2002]) is reformulated based on solving a Laplace equation. A key drawback to this method is that normal estimation is required at region boundaries, which is not necessary in our Poisson-based solution. In TexToons, normal interpolation is further guided by Neumann boundary conditions at depth discontinuities. Our approach could utilize this technique, removing the need for our shape completion phase. However, if a large portion of the boundary has Neumann conditions, it can produce an undesirable inflation which does not respect the underlying structure (see Fig. 12).

4.5 Stitching

The shapes f_i obtained by inflation are all based at the same height $z = 0$ (see Fig. 13a). Note that we assume that Dirichlet boundary conditions are used at least at one point. To obtain a bas-relief-type structure, the inflated shapes need to be stitched together in a way that satisfies the relative depth order (see Fig. 13b). This requires translation in z and deformation of the initial shapes. We accomplish this by solving for functions $g_i : \Omega_i \rightarrow \mathbb{R}$ such that the sum $f_i + g_i$ satisfies *stitching constraints*. The stitching constraints can either be equality (specifying that two regions should exactly meet at given boundary points) or inequality (specifying that one region should be above/below another one).

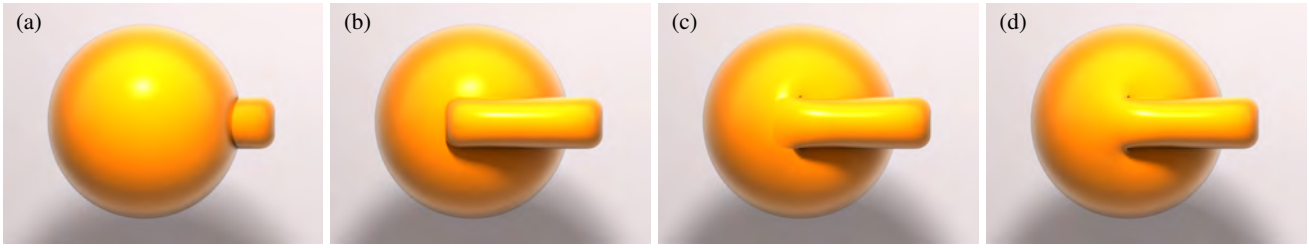

Fig. 13. Stitching & grafting—(a) Initial inflated shapes are all at the same depth. (b) Prescribed $C^{=}$ and $C^{\geq}$ constraints enforce correct absolute depth values. (c) Grafting merges meshes together, and (d) removes visible $\mathcal{C}^1$ discontinuities.

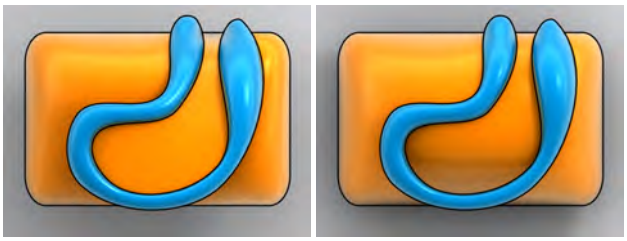

Fig. 12. If Neumann boundary conditions are used to guide the inflation at depth discontinuities, the resulting shape does not respect the underlying structure (left). A better result is obtained if the shape completion mechanism is applied prior to inflation of the occluded rectangle (right).

For two overlapping regions Ω_i, Ω_j (Fig. 14b), we define the sets $C^{=}_{i,j}, C^{\leq}_{i,j}, C^{\geq}_{i,j} \subseteq \partial\Omega_i \cup \partial\Omega_j$ which describe the type of applied constraints (Fig. 14c–d). Then the functions g_i are found by minimizing the Dirichlet energy:

$$\int_{\Omega_i} \|\nabla g_i\|^2 d\mathbf{x} \tag{4}$$

subject to the stitching constraints:

$$f_i(\mathbf{x}) + g_i(\mathbf{x}) = f_j(\mathbf{x}) + g_j(\mathbf{x}) \quad \forall \mathbf{x} \in C^{=}_{i,j} \tag{5}$$

$$f_i(\mathbf{x}) + g_i(\mathbf{x}) \leq f_j(\mathbf{x}) + g_j(\mathbf{x}) \quad \forall \mathbf{x} \in C^{\leq}_{i,j} \tag{6}$$

$$f_i(\mathbf{x}) + g_i(\mathbf{x}) \geq f_j(\mathbf{x}) + g_j(\mathbf{x}) \quad \forall \mathbf{x} \in C^{\geq}_{i,j} \tag{7}$$

Intuitively, this corresponds to seeking g_i which is as constant as possible while satisfying the constraints (Fig. 14e–f).

Finite element discretization of this problem is straightforward because, by construction, each constrained vertex is present in meshes corresponding to both Ω_i and Ω_j (Fig. 6e). The constraints can then be directly expressed as a system of linear equalities and inequalities, resulting in one quadratic program for all regions which is solved using MOSEK [Andersen and Andersen 2000].

4.6 Grafting

The $\mathcal{C}^0$ connections created during the stitching step are desirable for most parts of the proxy 3D mesh. However, for regions representing shape articulations, the user may prefer smooth $\mathcal{C}^1$ transitions (see Fig. 13c–d). In this case, a *grafting* step can be used to smoothly merge the meshes of individual regions together.

Grafting works with a subset of vertices $G_i \subseteq \partial\Omega_i$, which specify where the regions Ω_i will be smoothly connected. These *grafting vertices* can be detected automatically as boundary points that are not covered by a hand-drawn contour, or the user can use

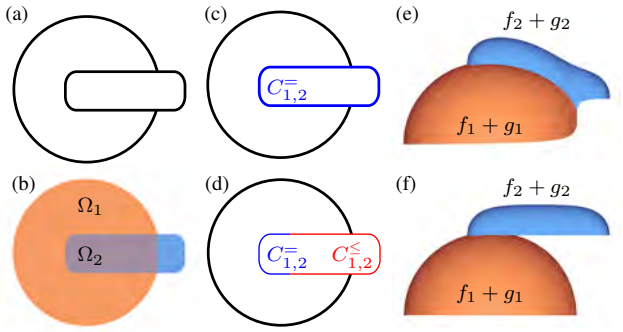

Fig. 14. Stitching—(a) an input drawing containing two regions Ω_1 and Ω_2 (b). When equality constraint $C^{=}_{1,2}$ is active on the whole $\partial\Omega_2$ (c), then $f_2 + g_2$ is glued together with $f_1 + g_1$ (e). When the equality constraint is replaced by inequality constraint $C^{\leq}_{1,2}$ on a subset of $\partial\Omega_2$ (d), the corresponding part of $f_2 + g_2$ will hover over $f_1 + g_1$. Side views are shown in images (e) and (f).

the aforementioned two-click interface to specify them manually (Fig. 11a). The regions are then processed and merged one by one in a bottom-top order. Given regions $\Omega_1, \ldots, \Omega_{i-1}$ that have already been joined into region Υ, the task is to graft Ω_i (Fig. 15a). We search for vertices in Υ coincident (in x, y coordinates) with G_i, and reconnect the triangles in such a way that the visible top surface forms a connected mesh (Fig. 15b). Note that this process only changes the connectivity, no re-indexing of vertices is necessary.

After the grafting, smoothing is performed in two steps. First, a smoothing region S is determined (Fig. 15c) which ensures that only a local neighborhood of G_i is affected. This can be done using an approach similar to the illusory surface reconstruction performed during the layering phase (see Sec. 4.3). We apply harmonic regularization to a delta-type function:

$$\delta(\mathbf{x}) = \begin{cases} 0 & \mathbf{x} \notin G_i \\ 1 & \mathbf{x} \in G_i \end{cases} \tag{8}$$

Specifically, we find $s : \Upsilon \to \mathbb{R}$ which minimizes

$$\int_{\Upsilon} \left((s - \delta)^2 + \alpha \|\nabla s\|^2 \right) d\mathbf{x} \tag{9}$$

where $\alpha > 0$ is a scalar controlling the amount of diffusion. The smoothing region S is then defined as

$$S = \{\mathbf{x} \in \Upsilon : s(\mathbf{x}) > 0.5\} \tag{10}$$

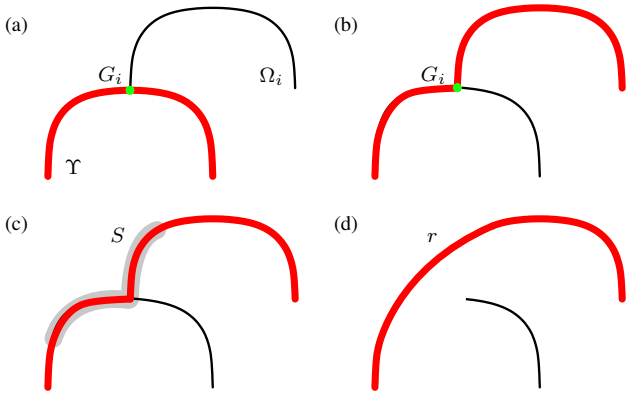

Fig. 15. Grafting—(a) region Ω_i is grafted on Υ at grafting vertices G_i, (b) triangles of Υ are reconnected to Ω_i at vertices coincident with G_i, (c) smoothing region S is computed, and (d) $\mathcal{C}^1$-continuous surface r is produced. For clarity sectional views are used to show the surface profile.

Once the smoothing region S is defined, the second step is to obtain a new smoothed surface r (Fig. 15d) by minimizing:

$$\int_{\Upsilon} (\Delta r)^2 d\mathbf{x} \tag{11}$$

subject to:

$$r(\mathbf{x}) = f(\mathbf{x}) + g(\mathbf{x}) \quad \forall \mathbf{x} \notin S \tag{12}$$

i.e., replacing the values of the original function $f + g$ by a $\mathcal{C}^1$ biharmonic interpolant in the smoothing region S.

4.7 Texturing

Since only the z coordinates of the points on the mesh are modified, the $[x, y]$ coordinates can be directly used as texture coordinates $[u, v]$. Any property attached to the original image can therefore be directly transferred to the 3D proxy surface, e.g., flat colors assigned to regions [Sýkora et al. 2009] (Fig. 16a–b), or textures [Sýkora et al. 2011] with bump maps (Fig. 16d–e). The only limitation of this transfer is that occluded surfaces are assigned the same data as the occluding top layer. In most cases, this does not produce noticeable artifacts. However, in cases when correct coloring is important (e.g., due to color bleeding effects), one can employ the illusory surfaces computed during the layering phase. Data (e.g., flat colors or $[u, v]$ coordinates) can then be extended from the visible portion of the region to the occluded part using the extrapolation approach in TexToons [Sýkora et al. 2011] for filling small gaps caused by misalignment.

4.8 Rendering

Once the proxy mesh is generated, it can be rendered using an off-the-shelf global illumination engine or, for a more stylized look, simple flat shading alone can be computed with, for example, a cartoon shader (Fig. 16a). The final look can be customized by adding environment geometry with flat colors or textures (see ground and background plane in Fig. 16b–e), setting multiple light sources (Fig. 16b), specifying materials (e.g., glossy or diffuse in Fig. 16d–e), adding fur (Fig. 16c), etc. The only limitation is the use of an orthographic camera with optical axis perpendicular to the plane of the source image. Near perspective or different viewpoints may reveal the approximate bas-relief structure (see Fig. 28) and will break the linear mapping between vertex coordinates and the original image space. This mapping is crucial for the post-processing phase where the reduced details and hand-drawn look are restored by superimposing the input image as a texture in the final render (Fig. 16).

5. RESULTS

We have implemented the ink-and-ray prototype using a combination of C++ and Matlab. Our prototype utilizes the following packages: the fast max-flow/min-cut solver GridCut [Jamriška et al. 2012] for the LazyBrush segmentation, a fast Laplacian solver [Jeschke et al. 2009] for computing diffusion curves, and the Triangle package [Shewchuk 1996] for triangulation. In Matlab, the backslash operator was used to solve linear systems and MOSEK [Andersen and Andersen 2000] for quadratic programs. The whole process from input image to output mesh takes less than 7 seconds for the *bunny* scene in Fig. 2 (0.5Mpix, 32k triangles) on a Xeon 3.5GHz with 16GB RAM and GeForce GTX 660. See the following table for timing breakdown of the individual steps:

segmentation	0.2 sec
completion & layering	0.4 sec
triangulation	0.9 sec
inflation & stitching	2.6 sec
grafting & smoothing	2.7 sec
total	6.8 sec

To render the image, we implemented a progressive path tracer with Monte-Carlo integration [Kajiya 1986] based on the OptiX ray tracing engine [Parker et al. 2010]. Although the current performance of the whole prototype pipeline is not fully interactive, it still provides reasonable enough response, making incremental edits feasible. In future work we plan to employ parallel processing and greater GPU utilization to deliver truly interactive performance.

To demonstrate the versatility of the proposed method, we selected a set of scanned hand-drawn images in various drawing styles, with different poses and structural complexity (see Fig. 2, Fig. 26, and Fig. 27). In each example, we show the original hand-drawn image, the result of the completion and layering phases, the generated proxy mesh, texture used for rendering, and the final global illumination render. The original and layered images are superimposed with the annotations made by the user (segmentation scribbles, depth ordering correction arrows, and dots specifying boundary conditions).

An example of the annotation workflow is shown in Fig. 27. The initial segmentation is either established automatically (see Fig. 2b and *robber* in Fig. 26b) or a couple of rough scribbles are drawn to help the system extract important regions (see Fig. 27c). The system next estimates the relative depth order of extracted regions and produces the initial bas-relief-type approximation. If depth ordering artifacts are present in the rendering of this initial approximation (Fig. 27d), additional depth inequalities can be specified (Fig. 27e) to correct them. Grafting and $C^{\geq}$ boundary conditions are detected automatically (see Fig. 2b and *wolf* in Fig. 26b) or specified by the user (Fig. 27g) to improve on the results of the default inflation and stitching operations (Fig. 27f). Using this type of annotation workflow, a trained user was able to specify the annotations in less than a minute for each frame. No tedious manual

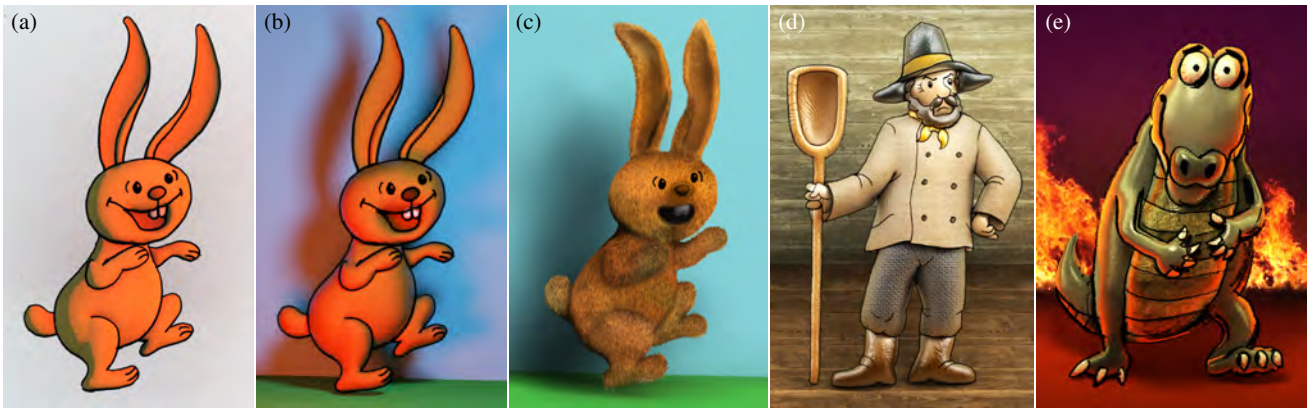

Fig. 16. The generated proxy mesh can be rendered using an off-the-self physically-based renderer (b–e). (a) One can also employ a stylized shader to simulate cartoon-like shading. Global illumination examples include: (b) flat colors, (c) fur, and (d–e) textures with bump maps applied on the mesh. Multiple light sources (b) as well as diffuse (d) or glossy (e) materials can be used as in a standard rendering pipeline. (Source drawings © Anifilm, textures © CGTextures. All rights reserved.)

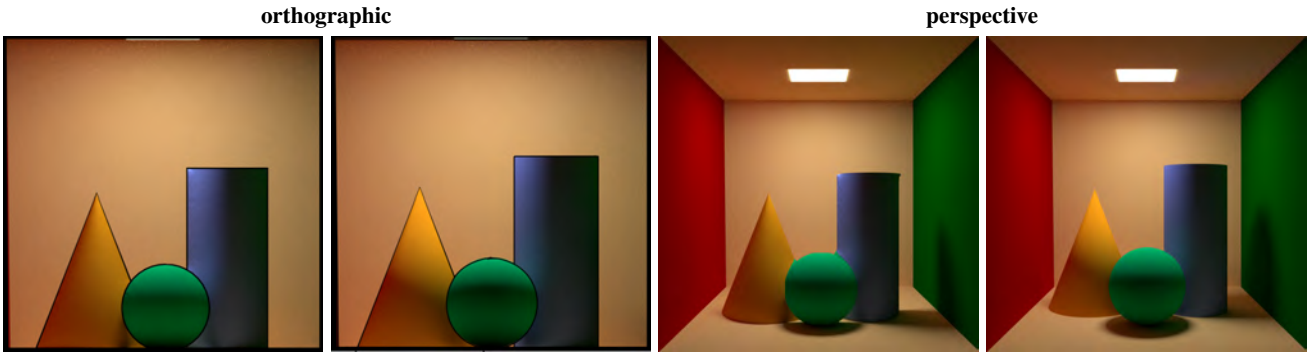

Fig. 17. A simple Cornell box—global illumination computed using our approximation (left) and full 3D model (right). Differences are less noticeable when orthographic projection is used.

labor such as rotoscoping was required. In contrast, the creation of similar 3D models using standard modeling tools can take hours.

The final render produces believable illumination despite the simplifications in the generated proxy mesh (see tilted views in Fig. 28). To show that our bas-relief approximation is able to deliver comparable render quality to that of a complete 3D model, we set up a simple Cornell box scene where a set of 3D objects (sphere, cone, cylinder) occluding each other is illuminated by an area light source (Fig. 17). We rendered visible outlines in orthographic projection using a simple cartoon shader and applied our method using the resulting 2D image as input. A global illumination render from the same viewpoint was run for the full 3D model and for our proxy mesh. The resulting images look similar under orthographic projection. Similar illumination effects are produced even when rendered with a perspective camera, but in this case the approximation does cause some artifacts, most notably in the attached shadows. To further test the quality of our results against a rendering of a full 3D model, we performed a similar ground-truth test on a cartoon model of a lion (Fig. 1c–e) and conducted a perceptual experiment (see Sec. 6).

Our method can be applied to individual drawings as well as sequences of animated frames. We use the concept of *scribble transfer* described in [Sýkora et al. 2009] to reuse annotations in subse-

Fig. 19. Comparison with previous approaches—(a) normal map based shading used in Lumo [Johnston 2002], (b) simulation of 3D-like shading used in TexToons [Sýkora et al. 2011], (c) our approach. (Source drawing © Anifilm. All rights reserved.)

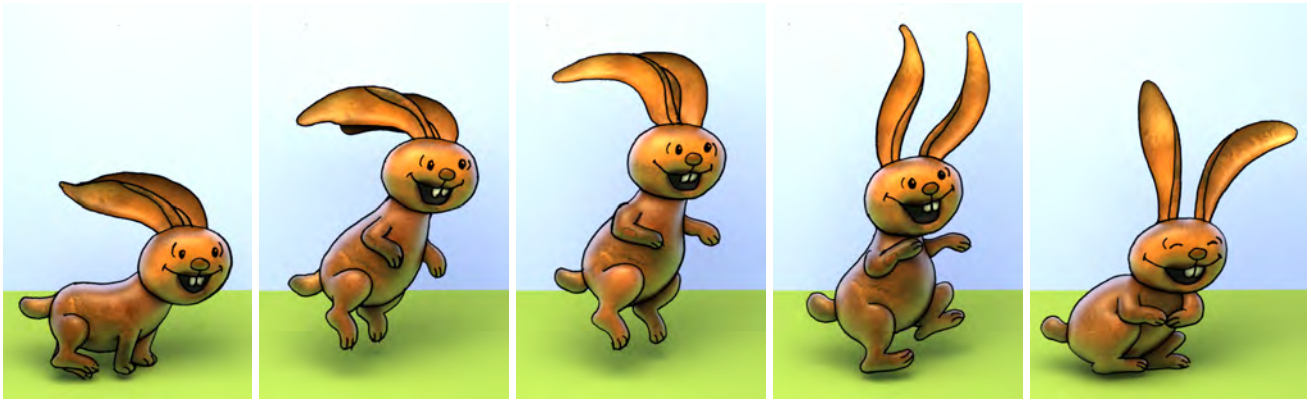

Fig. 18. Example animation sequence created using the ink-and-ray framework. Annotations from Fig. 2a were transferred to the remaining frames using as-rigid-as-possible image registration [Sýkora et al. 2009] (each region was registered separately to handle topology changes) and the TexToons [Sýkora et al. 2011] framework was used to transfer texture coordinates from Fig. 2b. (Source drawings © Anifilm. All rights reserved.)

quent frames. Our method was then used to produce a proxy mesh for each frame. Although our approach does not address temporal coherency explicitly, the shading produced by the bas-relief-type approximation generally appears consistent over animated sequences (see Fig. 18). One reason for the lack of noticeable temporal artifacts is the typically lower frame rate (12 frames per second) found in many hand-drawn animations. The lower frame rate (1) reduces manual effort and (2) allows greater stylization, i.e., larger structural changes between individual frames are less noticeable to the viewer.

We compared our ink-and-ray method to other related approaches for enhancing hand-drawn images: Lumo [Johnston 2002] and TexToons [Sýkora et al. 2011]. To simulate Lumo we extracted a normal field from our bas-relief approximation and rendered it using the same lighting conditions. As visible in Fig. 19a, due to lack of 3D structure, Lumo does not offer important depth-related global illumination effects such as complex self shadowing, glossy reflections, and diffuse color bleeding. In TexToons, Lumo-like shading is combined together with the knowledge of relative depth order to simulate the effect of ambient occlusion. However, the result is still purely a 2D approximation and does not respect 3D structure and produces a limited stylized look (Fig. 19b).

6. PERCEPTUAL EXPERIMENT

We conducted a subjective perceptual experiment to verify the overall visual quality of renderings generated using the output of our ink-and-ray pipeline.

6.1 Stimuli

The observers compared images rendered using our bas-relief-type approximation with those using the ground-truth (full 3D model), normal map based shading (Lumo) [Johnston 2002], and a simple inflation from silhouette as used in [Ono et al. 2004] (see Fig. 20). We projected the ground-truth 3D model using an orthographic camera and used NPR techniques to produce a synthetic 2D line drawing which was then used to generate the approximate models. The three test models (*lioness, rings, snake*) were rendered using two different global illumination conditions (diffuse and diffuse plus specular).

6.2 Experimental Design and Procedure

We conducted a two-alternatives-forced-choice (2AFC) designed experiment [David 1988] with 89 observers (20 females and 69 males; age 18 to 71 years), all reporting to have normal or corrected-to-normal vision.

Each subject was introduced to the problem before the experiment. They were told the goal was to compare different methods for adding 3D effects to 2D drawings. For a series of inputs, the observer was presented with a reference line drawing and a pair of side-by-side rendered images and asked to choose the rendered image with the most correct 3D appearance. They were told to focus on shading, shadows, lighting and reflectance. The evaluated images were presented on uncalibrated displays. The observing distance was not controlled and no time constraints were imposed. The sequences of images were randomized. The experiment took on average 10 minutes per observer.

6.3 Agreement and Consistency

The agreement between observers was quantified by a coefficient of agreement u, ranging from $u = -1/(s-1) = -0.011$, where $s = 89$ is the number of observers, (which indicates no agreement) to $u = 1$ (all observers responded the same). The obtained value for our case $u = 0.47$ is rather high, meaning the observers had similar preferences during the judgment. Accordingly, the χ^2 test of significance clearly shows the statistical significance of u ($\chi^2 = 1515.1$, $p < 0.001$, 6 degrees of freedom), meaning the measured subjective responses are not random.

The coefficient of consistency of responses within subject ζ ranges from $\zeta = 0$ (no consistency) to $\zeta = 1$ (ideally consistent responses). The average consistency over all the observers $\zeta_{avg} = 0.93$ indicates that the observers were highly consistent in their responses and most of them did not change their preference during the experiment.

6.4 Study Results

Each observer performed $m \times n(n-1)/2 = 36$ comparisons, where $m = 3(\text{models}) \times 2(\text{illuminations})$ and $n = 4$ is the number of tested methods. The image chosen by an observer was given a score of 1, the other a score of 0. The data was stored in a 4×4 frequency

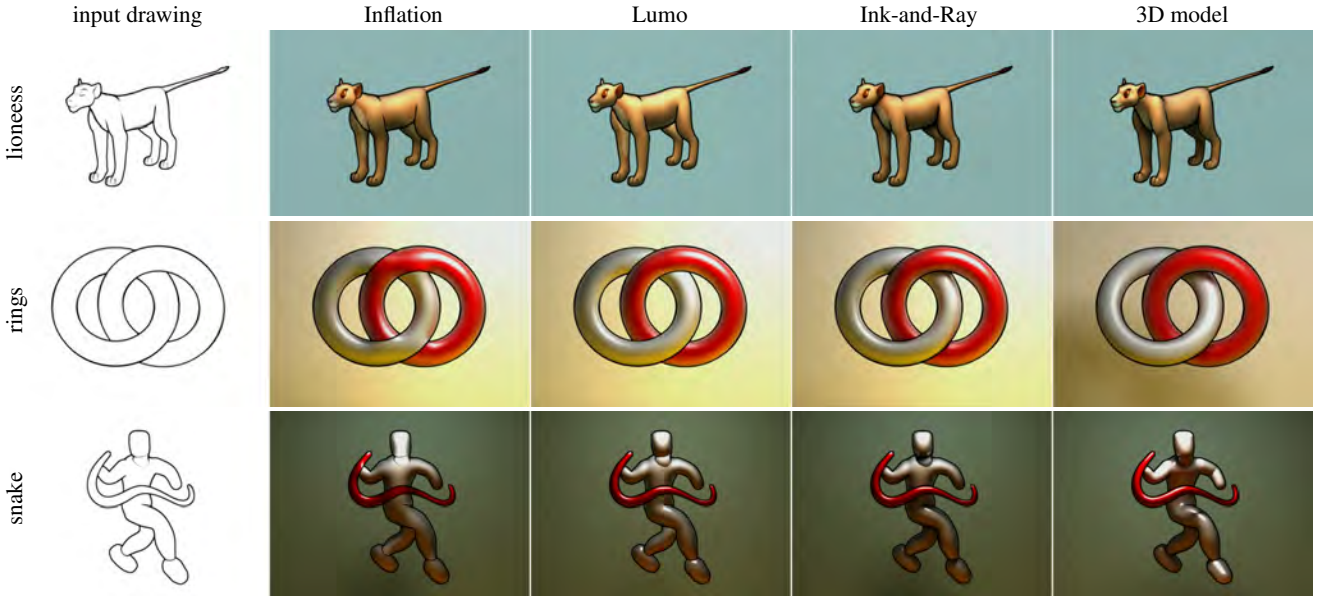

Fig. 20. Perceptual experiment stimuli (each model is shown for one illumination condition only). (Snake 3D model © Yotam Gingold. All rights reserved.)

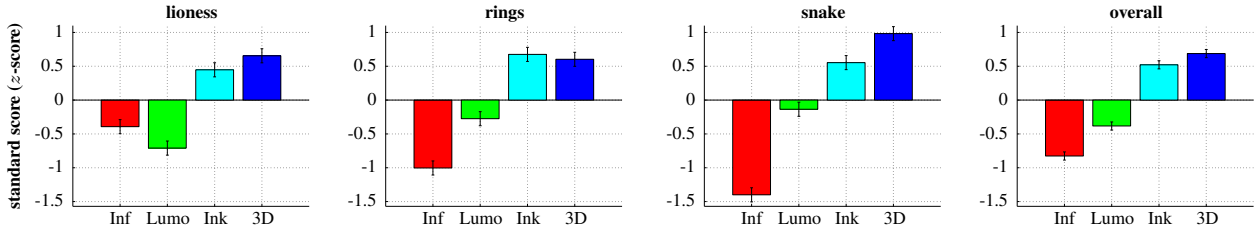

Fig. 21. Results of the perceptual experiment (standard z-scores) per stimulus and overall results.

matrix for each observer, where the value in column i and row j represents the score of tested method i compared with method j.

The overall results as well as results for each stimulus separately are depicted in Fig. 21. This plot shows standard z-scores for all observers along with the confidence intervals. The z-scores were acquired from frequency matrices using the incomplete matrix Bradley-Terry-Luce model. The overall results using the full 3D model exhibit the best perceptual quality (the highest z-score), followed by our proposed ink-and-ray method, Lumo technique, and finally the simple inflation from silhouette.

To evaluate the significance of the results we apply the Kruskal-Wallis (non-parametric version of one-way ANOVA) test [Mongomery and Runger 1999]. The results of the test are summarized in Fig. 22. The null hypothesis "there is no difference in perceived quality of the presented stimuli" can clearly be rejected ($p \ll 0.001$), meaning the methods produce perceptually different results. The multiple comparison test (Tukey's honestly significant differences [Hochberg and Tamhane 1987]) returns an overall ranking of the conversions with an indication of the significance of the differences (see Fig. 23, top row). The results show that there is a statistically significant difference between all the tested methods. This means that inflation from the silhouette is statistically worse than the Lumo technique, the Lumo technique produces overall perceptually worse results than our proposed method, and that the rendering using the full 3D model is better than our approximation. An interesting result can be obtained when we deliberately do not take into account subjects who considered themselves as graphics experts. For this case (49 subjects; 19 females and 30 males), the difference between our method and the full 3D model is not statistically significant (see Fig. 23, bottom row). This corresponds to our empirical observation that non-expert observers typically do not recognize differences between our method and the full 3D model.

6.5 Study Summary

The results of this study show that observers prefer the renderings generated using a full 3D model (when available). However, our model was shown to be better with statistical significance than models produced by simple inflation from the silhouette and by the Lumo technique. Moreover, for non-expert observers, results generated using our ink-and-ray method can be considered perceptually comparable to those produced from a full 3D model.

7. LIMITATIONS AND FUTURE WORK

In this section we discuss some limitations of the current prototype and propose avenues for improvements in future work.

The current implementation is unable to provide updates at interactive rates. Incremental updates are feasible, however further

	SS	$d.f.$	MS	χ^2	p
method	$439.1 \cdot 10^6$	3	$146.4 \cdot 10^6$	1233.1	$\ll 0.001$
error	$321.2 \cdot 10^6$	2132	150677		
total	$760.4 \cdot 10^6$	2135			

Fig. 22. Results of Kruskal-Wallis test (where SS denotes Sum of Squares, $d.f.$ means Degrees of Freedom, MS denotes Mean Square, χ^2 is χ^2 value, and p is p-value for the null hypothesis [Tabachnick and Fidell 2007]).

	Inflation	Lumo	Ink-and-Ray	3D model
with experts	-0.8263	-0.3829	0.5216	0.6875
without experts	-0.7093	-0.3438	0.4833	0.5698

Fig. 23. Overall performances of the tested methods (z-scores). The best result is *3D model*. In the first case when all participants are considered (top row), no methods were considered perceptually similar, however, in the second case where graphics experts were not taken into account (bottom row) our method and 3D model are considered perceptually similar (highlighted).

optimizations are necessary to deliver immediate visual feedback. One possible approach is to compute a quick on-the-fly approximation of the proxy while editing.

A key advantage of our bas-relief-type approximation is that the user is not required to draw side-views or to specify absolute depth values for any part of the drawings. However, this might be limiting in cases when the range of depths in the generated proxy mesh is either too small or large such that the resulting error starts to be visible in cast shadows and reflections (see Fig. 25). This drawback can be fixed by adding lower and upper bounds to the inequalities $C^{\geq}$ used during the stitching phase (see Sec. 4.5).

Our method assumes hand-drawn objects consist mainly of rounded shapes and does not provide larger control over sharp features. The user can globally reduce smoothness by applying an arbitrary cross section function to the inflation, however, it is currently not possible to control surface sharpness locally. To address this we plan to employ more advanced shape control approaches such as Cauchy constraints [Andrews et al. 2011].

Another practical limitation is that, due to the unreal depth proportions produced by our bas-relief-type approximation and due to the use of an orthographic camera, to be consistent, surrounding geometry must be constructed to be compatible within this context. This means the artist needs to take these factors into account and adjust surrounding geometry to follow the used bas-relief principle (e.g., tilt the ground plane to simulate perspective projection as shown in Fig. 24) or use bas-relief conversion [Weyrich et al. 2007] to deliver a similar solution automatically.

Fig. 24.

Although processing the animation frames separately produces reasonable results in general (see Fig. 18), the smoothness of transitions between individual frames still strongly depends on the coherency of the original source animation. By considering rough correspondences [Sýkora et al. 2009], temporal flickering could be reduced overall in the final rendered animation. In addition, visible portions of occluded objects from previous frames could be used to improve the shape completion phase.

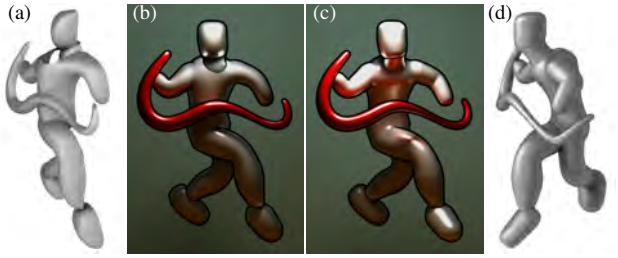

Fig. 25. Limitation—our bas-relief-type approximation (a) could contain too small (snake vs. body) or too large (neck vs. body) depth discontinuities which may influence the quality of the resulting global illumination (b). A full 3D model (d) rendered using the same lighting conditions is shown for comparison (c). (Source 3D model © Yotam Gingold. All rights reserved.)

8. CONCLUSIONS

We present a new ink-and-ray framework to enhance hand-drawn 2D artwork with global illumination effects. Our approach uses a bas-relief-type approximation to produce a visual appearance typically only found in complex 3D CG pipelines. As it requires very little manual input, we believe our approach could open a new potential for rapid prototyping in early phases of current CG pipelines, as well as support the production of a richer look for traditional hand-drawn animation.

ACKNOWLEDGMENTS

We would like to thank Yotam Gingold for the help with perceptual experiment, Anifilm and UPP for hand-drawn images, Petr Šťastný for adding materials and compositing, Jan Tománek for fur, and all anonymous reviewers for their constructive comments. This research has been supported by the Technology Agency of the Czech Republic under the research program TE01020415 (V3C – Visual Computing Competence Center), by the Czech Science Foundation under research program P202/12/2413 (OPALIS), COST Action IC1005 on "HDRi: The digital capture, storage, transmission and display of real-world lighting", Swiss National Science Foundation award 200021_137879, and ERC grant iModel (StG-2012-306877).

REFERENCES

E. D. Andersen and K. D. Andersen. 2000. The MOSEK Interior Point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm. In *High Performance Optimization*. Kluwer Academic Publishers, 197–232.

J. Andrews, P. Joshi, and N. A. Carr. 2011. A Linear Variational System for Modelling From Curves. *Computer Graphics Forum* 30, 6 (2011), 1850–1861.

P. N. Belhumeur, D. J. Kriegman, and A. L. Yuille. 1999. The Bas-Relief Ambiguity. *International Journal of Computer Vision* 35, 1 (1999), 33–44.

B.-Y. Chen, Y. Ono, and T. Nishita. 2005. Character Animation Creation using Hand-drawn Sketches. *The Visual Computer* 21, 8-10 (2005), 551–558.

F. Cole, P. Isola, W. T. Freeman, F. Durand, and E. H. Adelson. 2012. Shapecollage: Occlusion-Aware, Example-Based Shape Interpretation. In *Proceedings of European Conference on Computer Vision*. 665–678.

Fig. 26. Results—(a) the original drawing including user-defined annotations for segmentation, (b) result of the layering and completion phase with boundary condition annotations (red: Neumann and $C^{\geq}$, green: Dirichlet and $C^{\geq}$, blue: Neumann and C^{1} grafting), (c) generated bas-relief-type mesh, (d) original texture, (e) resulting global illumination render. (Source drawings: lady, wolf, dino, and farmer © Anifilm, robber © UPP. All rights reserved.)

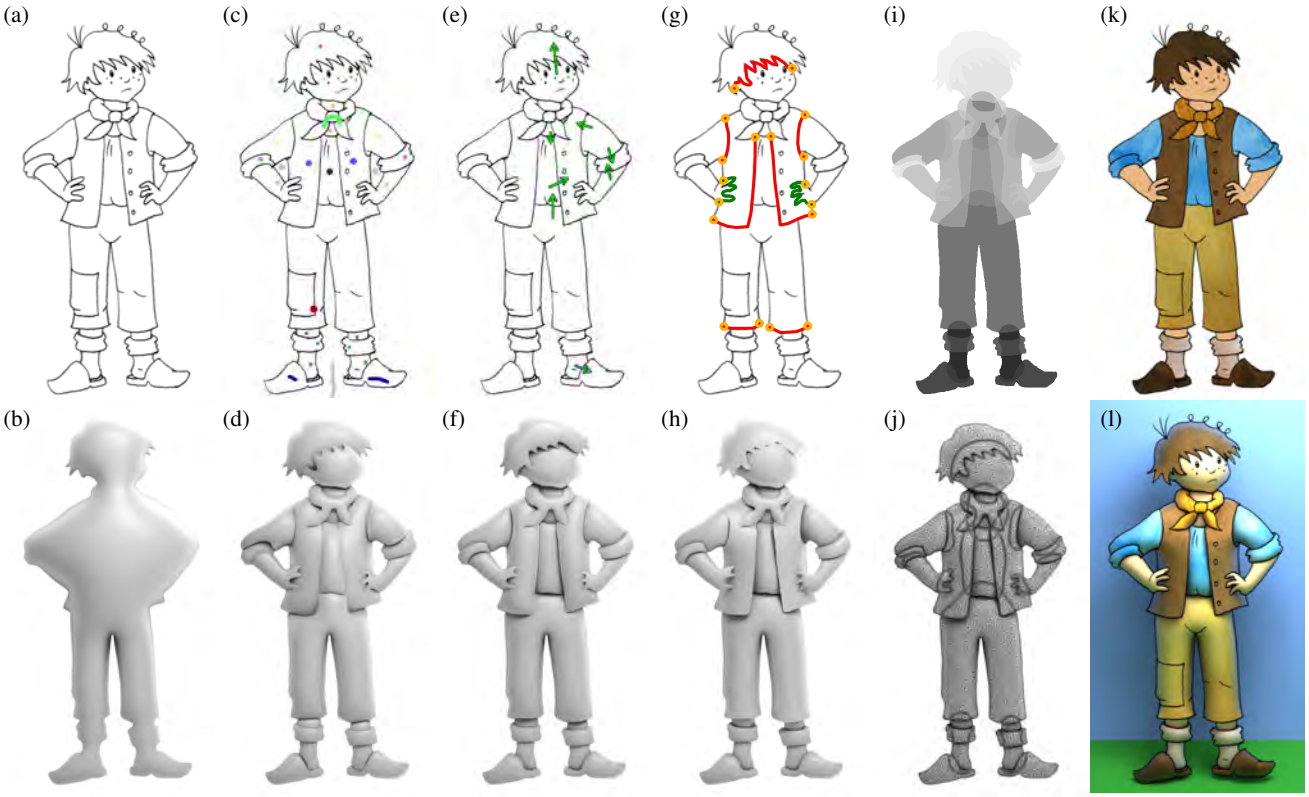

Fig. 27. Annotation workflow—(a) the original hand-drawn image, (b) simple inflation from silhouette, (c) segmentation scribbles specified by the user, (d) resulting bas-relief-type approximation with visible artifacts in depth order, (e) depth inequalities added by the user to fix incorrect depth ordering, (f) resulting bas-relief model has unwanted discontinuities using the default inflation and stitching parameters, (g) user-defined boundary conditions to improve shape of selected segments (green: $C^{\geq}$, red: Neumann with $C^{\geq}$), (h) final proxy mesh, (i) result of depth ordering and completion phases, (j) triangulation and topology of the resulting proxy mesh, (k) original texture, (l) resulting global illumination render. (Source drawing © Anifilm. All rights reserved.)

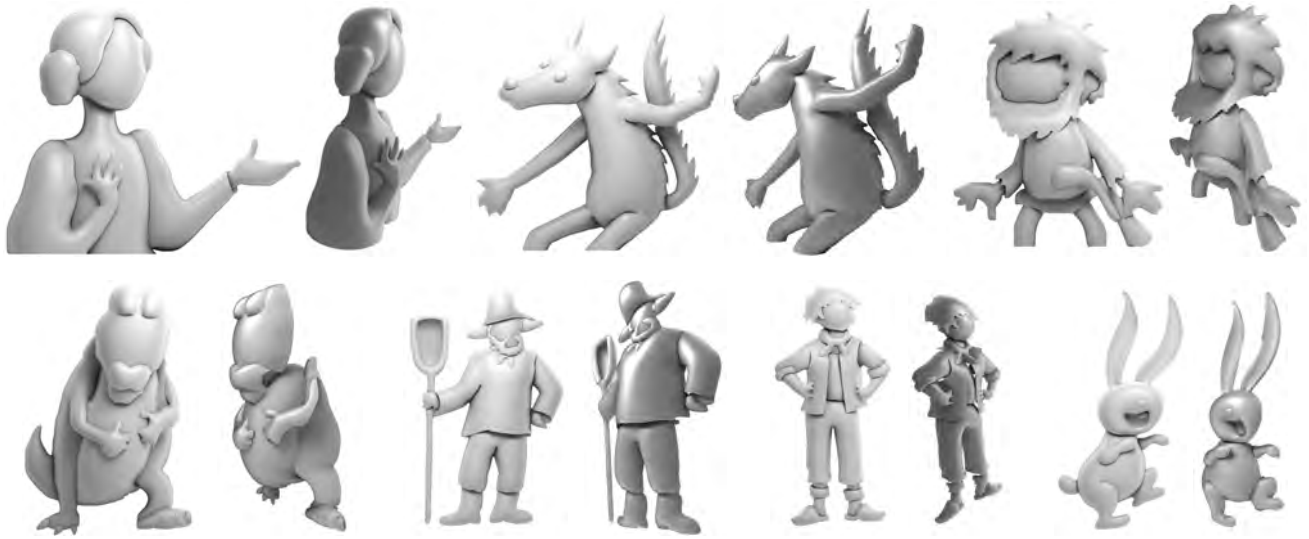

Fig. 28. Proxy meshes generated using our approach expose their approximate nature when rendered from sideviews using a perspective camera. Note the bas-relief-type false-depth structure which is not visible in the orthographic projection.

H. A. David. 1988. *The Method of Paired Comparisons* (2nd ed.). Oxford University Press.

D. Geiger, H.-K. Pao, and N. Rubin. 1998. Salient and Multiple Illusory Surfaces. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 118–124.

Y. Gingold, T. Igarashi, and D. Zorin. 2009. Structured Annotations for 2D-to-3D Modeling. *ACM Transactions on Graphics* 28, 5, Article 148 (2009).

Y. Hochberg and A. C. Tamhane. 1987. *Multiple Comparison Procedures* (1st ed.). Wiley.

T. Igarashi, S. Matsuoka, and H. Tanaka. 1999. Teddy: A Sketching Interface for 3D Freeform Design. In *ACM SIGGRAPH Conference Proceedings*. 409–416.

O. Jamriška, D. Sýkora, and A. Hornung. 2012. Cache-efficient Graph Cuts on Structured Grids. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 3673–3680.

S. Jeschke, D. Cline, and P. Wonka. 2009. A GPU Laplacian Solver for Diffusion Curves and Poisson Image Editing. *ACM Transaction on Graphics* 28, 5, Article 116 (2009).

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

P. Joshi and N. A. Carr. 2008. Repoussé: Automatic Inflation of 2D Artwork. In *Proceedings of Eurographics Workshop on Sketch-Based Interfaces and Modeling*. 49–55.

J. T. Kajiya. 1986. The Rendering Equation. *SIGGRAPH Computer Graphics* 20, 4 (1986), 143–150.

O. A. Karpenko and J. F. Hughes. 2006. SmoothSketch: 3D Free-form Shapes from Complex Sketches. *ACM Transactions on Graphics* 25, 3 (2006), 589–598.

E. A. Khan, E. Reinhard, R. Fleming, and H. Buelthoff. 2006. Image-Based Material Editing. *ACM Transactions on Graphics* 25, 3 (2006), 654–663.

J. Lopez-Moreno, J. Jimenez, S. Hadap, E. Reinhard, K. Anjyo, and D. Gutierrez. 2010. Stylized Depiction of Images Based on Depth Perception. In *Proceedings of International Symposium on Non-Photorealistic Animation and Rendering*. 109–118.

J. Malik. 1986. *Interpreting Line Drawings of Curved Objects*. Ph.D. Dissertation. Stanford University.

D. C. Mongomery and G. C. Runger. 1999. *Applied Statistics and Probability for Engineers* (2nd ed.). John Wiley & Sons.

A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa. 2007. FiberMesh: Designing Freeform Surfaces with 3D Curves. *ACM Transactions on Graphics* 26, 3, Article 41 (2007).

G. Noris, D. Sýkora, A. Shamir, S. Coros, B. Whited, M. Simmons, A. Hornung, M. Gross, and R. Sumner. 2012. Smart Scribbles for Sketch Segmentation. *Computer Graphics Forum* 31, 8 (2012), 2516–2527.

M. Okabe, G. Zeng, Y. Matsushita, T. Igarashi, L. Quan, and H.-Y. Shum. 2006. Single-View Relighting with Normal Map Painting. In *Proceedings of Pacific Conference on Computer Graphics and Applications*. 27–34.

L. Olsen, F. Samavati, and J. A. Jorge. 2011. NaturaSketch: Modeling from Images and Natural Sketches. *IEEE Computer Graphics and Applications* 31, 6 (2011), 24–34.

Y. Ono, B.-Y. Chen, and T. Nishita. 2004. 3D Character Model Creation from Cel Animation. In *Proceedings of International Conference on Cyberworlds*. 210–215.

A. Orzan, A. Bousseau, H. Winnemöller, P. Barla, J. Thollot, and D. Salesin. 2008. Diffusion Curves: A Vector Representation for Smooth-Shaded Images. *ACM Transactions on Graphics* 27, 3, Article 92 (2008).

Y. Ostrovsky, P. Cavanagh, and P. Sinha. 2005. Perceiving Illumination Inconsistencies. *Perception* 34, 11 (2005), 1301–1314.

S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich. 2010. OptiX: A General Purpose Ray Tracing Engine. *ACM Transactions on Graphics* 29, 4, Article 66 (2010).

L. Petrović, B. Fujito, L. Williams, and A. Finkelstein. 2000. Shadows for Cel Animation. In *ACM SIGGRAPH Conference Proceedings*. 511–516.

H. Read. 1961. *The Art of Sculpture* (2nd ed.). Bollingen Foundation, New York.

M. W. Ren, J. Y. Yang, and H. Sun. 2002. Tracing Boundary Contours in a Binary Image. *Image and Vision Computing* 20, 2 (2002), 125–131.

A. Rivers, T. Igarashi, and F. Durand. 2010. 2.5D Cartoon Models. *ACM Transactions on Graphics* 29, 4, Article 59 (2010).

C. Shao, A. Bousseau, A. Sheffer, and K. Singh. 2012. CrossShade: Shading Concept Sketches Using Cross-Section Curves. *ACM Transactions on Graphics* 31, 4, Article 45 (2012).

J. R. Shewchuk. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Proceedings of ACM Workshop on Applied Computational Geometry*. 203–222.

J. R. Shewchuk. 2002. Delaunay Refinement Algorithms for Triangular Mesh Generation. *Computational Geometry: Theory and Applications* 22, 1–3 (2002), 21–74.

D. Sýkora, M. Ben-Chen, M. Čadík, B. Whited, and M. Simmons. 2011. TexToons: Practical Texture Mapping for Hand-drawn Cartoon Animations. In *Proceedings of International Symposium on Non-photorealistic Animation and Rendering*. 75–83.

D. Sýkora, J. Buriánek, and J. Žára. 2005. Colorization of Black-and-White Cartoons. *Image and Vision Computing* 23, 9 (2005), 767–782.

D. Sýkora, J. Dingliana, and S. Collins. 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.

D. Sýkora, J. Dingliana, and S. Collins. 2009. LazyBrush: Flexible Painting Tool for Hand-drawn Cartoons. *Computer Graphics Forum* 28, 2 (2009), 599–608.

D. Sýkora, D. Sedlacek, S. Jinchao, J. Dingliana, and S. Collins. 2010. Adding Depth to Cartoons Using Sparse Depth (In)equalities. *Computer Graphics Forum* 29, 2 (2010), 615–623.

B. G. Tabachnick and L. S. Fidell. 2007. *Using Multivariate Statistics* (5th ed.). Pearson Education.

C. Toler-Franklin, A. Finkelstein, and S. Rusinkiewicz. 2007. Illustration of Complex Real-World Objects Using Images with Normals. In *Proceedings of International Symposium on Non-Photorealistic Animation and Rendering*. 111–119.

R. Vergne, P. Barla, R. W. Fleming, and X. Granier. 2012. Surface Flows for Image-based Shading Design. *ACM Transactions on Graphics* 31, 4, Article 94 (2012).

Y. Wang, Y. Chen, J. Z. Liu, and X. Tang. 2009. 3D Reconstruction of Curved Objects from Single 2D Line Drawings. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1834–1841.

T. Weyrich, J. Deng, C. Barnes, S. Rusinkiewicz, and A. Finkelstein. 2007. Digital Bas-Relief from 3D Scenes. *ACM Transactions on Graphics* 26, 3, Article 32 (2007).

H. Winnemöller, A. Orzan, L. Boissieux, and J. Thollot. 2009. Texture Design and Draping in 2D Images. *Computer Graphics Forum* 28, 4 (2009), 1091–1099.

Received January 2013; accepted December 2013

Appendix E

As-Rigid-As-Possible Image Registration for Hand-drawn Cartoon Animations

D. Sýkora, J. Dingliana, S. Collins: As-Rigid-As-Possible Image Registration for Hand-drawn Cartoon Animations. Proceedings of International Symposium on Non-photorealistic Animation and Rendering (NPAR'09), pages 25–33, August 2009. ISBN: 978-1-60558-604-5.

As-Rigid-As-Possible Image Registration for Hand-drawn Cartoon Animations

Daniel Sýkora* Trinity College Dublin
John Dingliana Trinity College Dublin
Steven Collins Trinity College Dublin

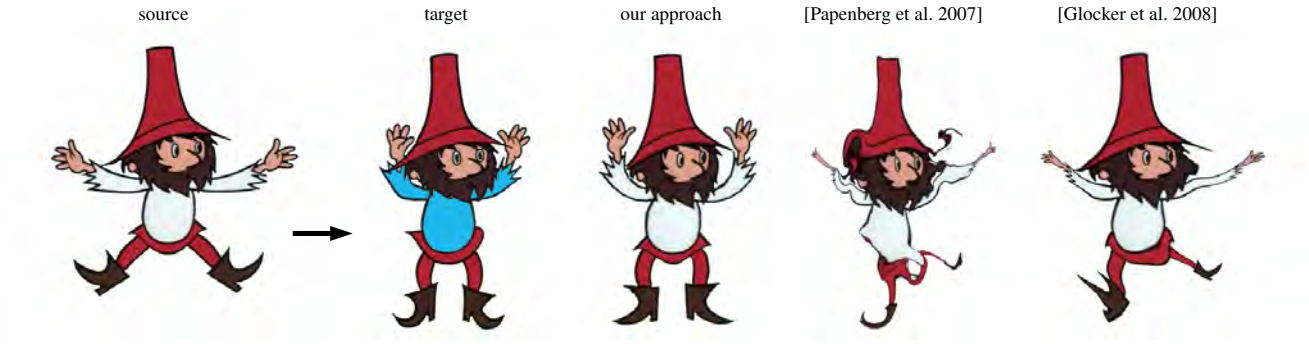

Figure 1: *Compared with the state-of-the-art in deformable image registration, our novel approach reaches plausible results even for challenging configurations undergoing large amounts of free-form deformation and notable changes in appearance.*

Abstract

We present a new approach to deformable image registration suitable for articulated images such as hand-drawn cartoon characters and human postures. For such type of data state-of-the-art techniques typically yield undesirable results. We propose a novel geometrically motivated iterative scheme where point movements are decoupled from shape consistency. By combining locally optimal block matching with as-rigid-as-possible shape regularization, our algorithm allows us to register images undergoing large free-form deformations and appearance variations. We demonstrate its practical usability in various challenging tasks performed in the cartoon animation production pipeline including unsupervised inbetweening, example-based shape deformation, auto-painting, editing, and motion retargeting.

CR Categories: I.4.3 [Image Processing and Computer Vision]: Enhancement—Registration; I.3.4 [Computer Graphics]: Graphics Utilities—Graphics editors; J.5 [Computer Applications]: Arts and Humanities—Fine arts

Keywords: deformable image registration, as-rigid-as-possible deformation, interactive shape manipulation

1 Introduction

In a traditional cartoon animation each animation frame is drawn by hand so that the correspondences between them are unknown. Such a drawback considerably limits the usage of traditional approaches in recent computer-based animation systems, where the knowledge of correspondences between individual key-frames plays an important role.

*e-mail: sykorad@cs.tcd.ie

Obtaining correspondences automatically is a challenging task since each hand-drawn image is unique and typically undergos a large amount of free-form deformation and notable change in appearance. In this context popular computer vision techniques based on local similarity [Lowe 2004] or global contexts [Belongie et al. 2002] fail since they rely on unique local features or stable global configurations. Although such features are typical for real world photographs they are rare in hand-made drawings. Moreover, the aforementioned techniques provide only isolated point correspondences and do not consider global consistency. Thus, they can easily lead to spatially inconsistent mapping.

A more powerful approach to this problem is deformable image registration [Maintz and Viergever 1998; Modersitzki 2004; Gholipour et al. 2007] which allows the retrieval of dense correspondences between images and simultaneously maintains spatial consistency of the resulting mapping. It is typically formulated as a non-linear optimization problem where a predefined energy function is minimized through some established numerical optimization technique [Klein et al. 2007]. However, there are two key difficulties which make the solution challenging: (1) non-convexity of the energy function and (2) sensitivity to outliers (i.e. appearance variations that do not fit the selected deformation model). To overcome these, an initial guess close to the global minima is required. It can be obtained through various heuristics such as the popular multi-scale approach [Lucas and Kanade 1981] or by using a hierarchy of deformation models [Bergen et al. 1992]. Unfortunately, for large displacements or appearance variations, even these heuristics yield erroneous results. This fundamental problem has been recently addressed by techniques that attempt to minimize the energy not through the iterative numerical optimization, but directly via discrete labelling [Glocker et al. 2008; Shekhovtsov et al. 2008]. These are built upon recent advances in algorithms for inference from random fields [Szeliski et al. 2008] which allow fast approximate solutions to non-linear problems with effective avoidance of local minima. Nevertheless, they still do not a guarantee global optimum (since the problem is NP-hard) and become computationally intractable for large displacements due to significantly increasing number of labels.

In this paper we develop a new approach to deformable image registration which addresses the issue of local minima and is able to reach a desired pose even from large initial displacements or no-

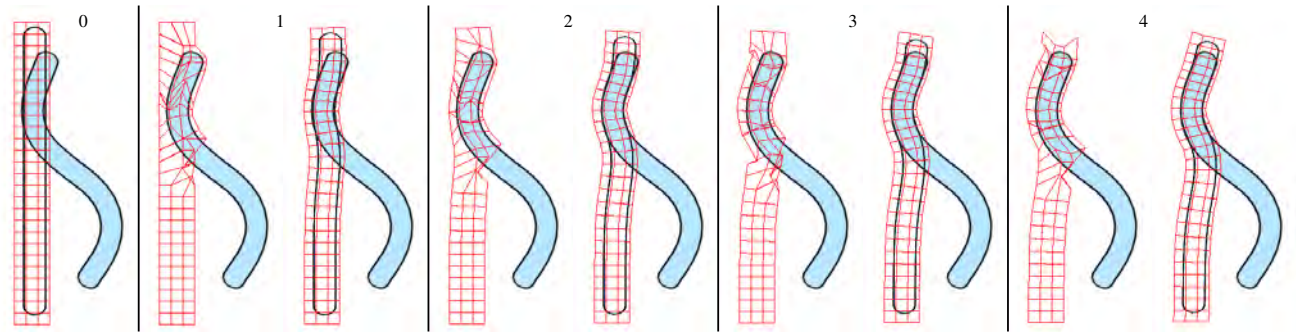

Figure 2: *Deformable image registration in progress – we want to register a straight stripe (filled with transparent color and embedded in a square lattice) with its S-shaped counterpart filled with light blue color (column 0). In each iteration (columns 1–4) two steps are repeated: points are first pushed towards locations with minimal visual difference without considering shape consistency (left) and then the shape is regularized using a variant of as-rigid-as-possible shape matching algorithm (right). Note, how the shape gradually approaches the desired configuration.*

table changes in appearance. It is inspired by the success of recent work in real-time simulation of deformable objects [Müller et al. 2005; Rivers and James 2007], where points are pushed directly towards desired positions and then as-rigid-as-possible shape regularization is used to ensure consistency of the original shape. In our case, points are not influenced by gravity or inertial forces, but instead attracted to locations with minimal visual difference. A key benefit of this new approach lies in the fact that the aforementioned shifts can be arbitrary and only the shape regularization ensures consistency. This is the fundamental difference to numerical optimization where incremental shifts are predicted by minimizing locally linearized version of the energy function, which typically leads to an inappropriate local minima. Our new technique is closer to the approaches based on discrete optimization in the sense it can recover from inappropriate local minima and lead to more plausible results. However, the key difference is that we do not minimize an energy function but instead use a geometrically motivated shape regularization scheme which preserves local rigidity and does not require computationally demanding discrete optimization inapplicable to large initial displacements.

Since our work is mainly motivated by the needs of the cartoon animation production pipeline, we demonstrate the practical usability of our new algorithm in this context. We show how it can reduce the amount of manual work in tasks such as inbetweening, painting, retargeting, shape deformation, and reusing traditional animation. We believe that these examples demonstrate the practical potential of our new technique and will motivate developers of recent professional cartoon animation systems to incorporate our technique as a versatile building block applicable to various practical scenarios.

The rest of the paper is organized as follows. First we briefly overview related work and analyze key drawbacks which motivated us to develop a new approach. Then we describe the proposed algorithm in more detail and discuss its strengths and limitations. Finally we demonstrate its practical usability in the context of the cartoon animation production pipeline and conclude with several ideas for future work.

2 Related work

Obtaining correspondences between hand-drawn images is a challenging task that has captured the attention of many researches within the last two decades. Xie [1995] used simple affine transformations to match two line drawings and perform automatic inbetweening. Madeira et al. [1996] pioneered a region-centered approach where drawings are first sub-divided into regions and then matched using shape similarity and topological relations. This approach has been later improved by several authors who assume additional semantic information about the image [Kort 2002], cater specifically for black-and-white cartoons [Sýkora et al. 2005], rely on a hierarchy of regions [Qiu et al. 2005] or employ skeleton matching [Qiu et al. 2008]. Their common limitation is that they are applicable only to specific easy-to-analyze drawing styles and do not provide dense correspondences.

Bregler et al. [2002] presented a more general approach in their famous framework for cartoon motion capture. They overcome the problem of deformable registration by sampling the space of possible deformations using as-rigid-as-possible interpolation [Alexa et al. 2000] and then infer optimal linear combinations of these samples to fit the target pose. Although this approach works in the context of motion retargeting, it is not applicable to our problem since it does not directly provide dense correspondences between two animation frames. De Juan and Bodenheimer [2006] utilize dense correspondences in their framework for re-using traditional animation. However, they completely rely on manual initialization and refine dense mapping using an existing algorithm [Wirtz et al. 2004] based on numerical optimization. Recently, Xu et al. [2008] proposed a system for animating motion from several snapshots captured in a single image. However, since they rely on shape contexts [Belongie et al. 2002], unstable under large free-form deformations, extensive manual intervention is needed to identify stable features.

3 Our approach

Our novel approach to deformable image registration stems from the successful workflow recently used for dynamic simulation of deformable objects by Müller et al. [2005] and later extended by Rivers and James [2007]. A core idea of this technique is to decouple point movements from shape consistency so that the physical simulation can be applied directly on points, treating them as particles without considering their mutual connectivity. To keep the shape consistent, a geometrically motivated shape matching phase is then used to regularize point locations.

A key observation is that such a workflow can bring significant benefit also to deformable image registration since it allows retrieval of locally optimal shifts and still keeps the shape consistent. This is in contrast to energy-based approaches where shifts are limited by the deformation model which does not allow temporary increase of the overall energy and so typically leads to undesirable local minima.

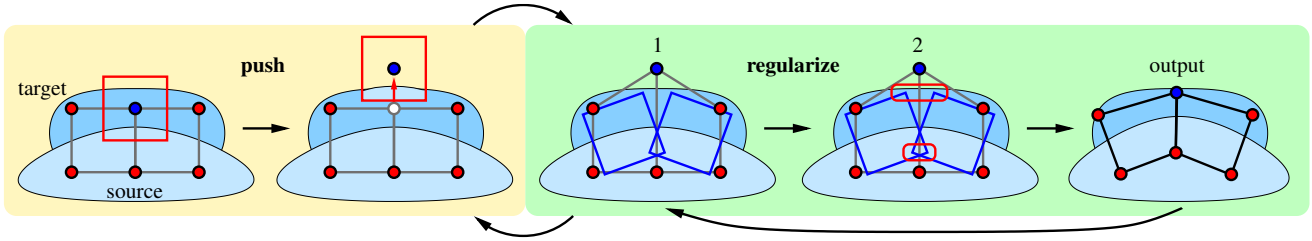

Figure 3: *A schematic overview of the proposed algorithm – the aim is to register a light blob (source) with its darker counterpart (target). The light blob is embedded in a square lattice (partly visible). The algorithm iterates two main phases:* ***push*** *(yellow part) &* ***regularize*** *(green part). In the pushing phase block matching is used to move lattice points towards locations where a sum of absolute differences over a local neighborhood (red square) is minimal. Then in the shape regularization phase two steps are iterated: (1) optimal rigid transformation is computed for each lattice square and then (2) lattice points are moved to the centroid of their shared instances in all connected squares.*

The only necessary modification as compared to the original concept is to replace physically motivated forces by an attraction to locations with visually similar neighborhood. Based on this setup, the resulting algorithm is as follows. Similarly to [Rivers and James 2007] we embed the input image into a regular square lattice respecting its articulated shape and then iterate two following steps until a stable configuration is reached:

1. **Push** all points to locations with minimal visual difference (Section 3.1).
2. **Regularize** the point locations to keep the shape consistent (Section 3.2).

These steps are illustrated in Figure 3 and a practical example is presented in Figure 2. Note how in each iteration, points are first pushed arbitrarily towards desired locations and how the overall shape becomes messy. Nevertheless, after regularization the shape is consistent and better fits the target pose.

In the following sections we describe these two steps in more detail, discuss implementation issues, stopping criteria, and possible limitations.

3.1 Push

The aim of the pushing phase is to find a new location for each point on the embedding lattice that minimizes visual difference in its local neighborhood. Since we are not limited by shape consistency we can utilize simple block matching which guarantees globally optimal shift within a predefined search area (see Figure 3, yellow part). Formally, the aim is to find a shift vector $\mathbf{t}^*$ from search area M that minimizes the sum of absolute differences over a neighborhood N, i.e.:

$$\mathbf{t}^* = \arg\min_{\mathbf{t}\in M} \sum_{\mathbf{p}\in N} |\mathrm{S}(\mathbf{p}+\mathbf{t}) - \mathrm{T}(\mathbf{p})| \tag{1}$$

where S denotes the source and T the target image. Note that in spite of shift optimization, the overall algorithm is not limited to pure translation, since S is slightly deformed in each iteration local neighborhoods of points gradually adapt to more complicated deformations.

The important parameters of the block matching phase are the size of the neighborhood $|N|$ and the size of search area $|M|$. In general $|N|$ should be large enough to contain substantial information but also small enough to preserve locality, whereas $|M|$ should allow attraction to further locations but also avoid ambiguity. The other limiting factor is the computational overhead which can increase dramatically due to the worst case complexity of the block matching algorithm $\mathcal{O}(|N||M|)$. If we consider that in each iteration typically hundreds of block matching operations are executed, the complexity can be very high even for small neighborhoods and search areas. However, due to the fact that optimal block positions typically remain constant during most iterations and all other shifts have much higher sums of absolute differences, the early termination heuristic [Li and Salari 1995] can be used to gain considerable speed up. Based on this observation, we set the width of N to 16 pixels and the width of M to 48 pixels (providing that images are in PAL resolution). This setting yields good results both in robustness and computational overhead in all examples shown in this paper.

3.2 Regularize

The second step of our algorithm is a geometrically motivated routine that iteratively regularizes the point locations so that local rigidity of the shape is preserved. This is another important difference from state-of-the-art techniques, which typically use elastic models that do not preserve rigidity and produce undesirable deformations when the initial displacements are large or when there is a notable variation in appearance (see Figure 1).

In Müller's original algorithm, the aim was to find an optimal rigid transformation (rotation $\mathbf{R}^*$ and translation $\mathbf{t}^*$) that moves points of the original shape $p_i \in P$ so that the sum of squared distances to the desired pose q_i is minimized:

$$(\mathbf{R}^*, \mathbf{t}^*) = \arg\min_{\mathbf{R},\mathbf{t}} \sum_i |\mathbf{R}\cdot\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i|^2 \tag{2}$$

In 3D the computation of $\mathbf{R}^*$ is non-linear, thus polar decomposition is required to solve this problem. However, as shown by Schaefer et al. [2006], a simple closed form solution exists in 2D. It can be obtained when we compute centroids $\mathbf{p}_c$ and $\mathbf{q}_c$ of the source and target pose and then substitute $\hat{\mathbf{p}}_i = \mathbf{p}_i - \mathbf{p}_c$ and $\hat{\mathbf{q}}_i = \mathbf{q}_i - \mathbf{q}_c$:

$$\mathbf{R}^* = \frac{1}{\mu}\sum_i \begin{pmatrix} \hat{\mathbf{p}}_i \\ \hat{\mathbf{p}}_i^{\perp} \end{pmatrix} \begin{pmatrix} \hat{\mathbf{q}}_i^T & \hat{\mathbf{q}}_i^{\perp T} \end{pmatrix}, \tag{3}$$

where

$$\mu = \sqrt{\left(\sum_i \hat{\mathbf{q}}_i \hat{\mathbf{p}}_i^T\right)^2 + \left(\sum_i \hat{\mathbf{q}}_i \hat{\mathbf{p}}_i^{\perp T}\right)^2}, \tag{4}$$

T denotes transposition, and the operator $\perp$ denotes the perpendicular vector, i.e.: $(x,y)^{\perp} = (y,-x)$. Once the rotation matrix $\mathbf{R}^*$ is known, the translation vector $\mathbf{t}^*$ can be computed directly:

$$\mathbf{t}^* = \mathbf{p}_c - \mathbf{R}^* \cdot \mathbf{q}_c \tag{5}$$

Our embedding lattice consists of several connected squares. In this case local rigid transformations are computed individually for each square and then the global smoothing step is used to ensure consistency. This simple extension enables more flexible deformations and still preserves local rigidity of the original shape (see Figure 4).

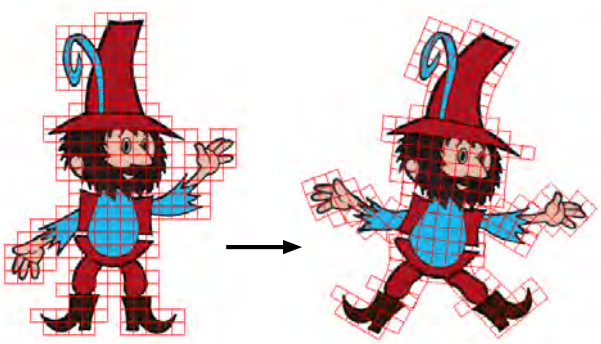

Figure 4: *An example of as-rigid-as-possible image deformation – the original image embedded in a square lattice (left) and its deformed counterpart (right).*

A similar mechanism is also used in the context of interactive shape deformation [Igarashi et al. 2005; Sumner et al. 2007; Botsch et al. 2007; Sorkine and Alexa 2007], however, the key difference is that in these techniques user-specified point locations are treated as hard constraints and the aim is to find an optimal deformation to satisfy them. In our case we do not have hard constraints. What we want is to smooth out point locations so that the shape becomes consistent. To perform this smoothing we exploit a very simple iterative approach inspired by recent work in interactive shape deformation [Wang et al. 2008]. It produces similar results to [Rivers and James 2007] but allows smooth control over shape rigidity (see Figure 3, green part):

1. For each square on the embedding lattice, use equations (3), (4) and (5) to obtain $(\mathbf{R}^*, \mathbf{t}^*)$ and use this to transform its points.
2. Move each point on the embedding lattice to the centroid of its transformed instances in all connected squares.

The only difference to the original shape deformation technique is that Wang et al. additionally simulate hard constraints by setting very large weights to points that represent manipulation handles. In fact their algorithm is nearly identical to [Sorkine and Alexa 2007], where instead of computing centroids a sparse linear system is solved. This modification clarifies the aforementioned difference between shape regularization and interactive shape deformation. In our case no points are fixed therefore after sufficient number of iterations the shape will return to the original configuration up to some global rigid body transformation. Such behavior is depicted in Figure 5 where one point is fixed at a different location and then the evolution of the deformation is captured during several shape regularization iterations. Initially the shape is flexible but with an increasing number of iterations, global rigidity is enforced so that the deformation gradually reduces to pure translation. This is caused by the diffusive nature of the averaging phase which gradually propagates rigidity through the whole shape.

This gradual diffusion of rigidity has several practical applications. By changing the number of shape regularization iterations we can smoothly vary between rigid and flexible deformation. It allows us to implement a smooth analogy to a hierarchy of deformation models [Bergen et al. 1992] (see Section 3.4) and also one-point interactive shape deformation (see Section 4).

3.3 Stopping criteria

Although our method does not explicitly minimize predefined energy function we can still estimate plausibility of the resulting registration by computing the average sum of absolute differences over all blocks during the block matching phase. In Figure 9 there are several graph plots of this average (blue curve) measured during a hundred iterations for different registration tasks. In most cases this value decreases monotonically and after several iterations the change is negligible. However, when a part of the shape undergoes a large non-overlapping deformation the change can be negligible for several iterations (see Figure 9b) and after that period the algorithm suddenly approaches new configurations with much lower value. This is caused by the fact that although a part of the image moves towards the desired pose it still remains in an area with no overlap where the sum of absolute differences is nearly constant. To overcome such ambiguity we instead monitor the average distance to the initial rest pose (red curve in Figure 9):

$$d_{\text{avg}} = \frac{1}{|P|} \sum_i ||\mathbf{p}_i - \mathbf{q}_i|| \tag{6}$$

This value informs us whether the control points on the embedding lattice are moving or not. We stop push-regularize iterations when d_{avg} has not changed considerably in the last 20 iterations.

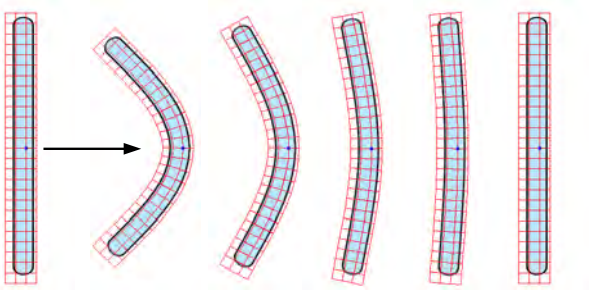

Figure 5: *The evolution of the shape deformation through several shape regularization iterations – one point is fixed at different location (left). During the first iterations the shape is flexible but when the number of iterations increases the deformation gradually approaches pure translation (from left to right).*

3.4 Limitations

Although our approach produces good results in most practical scenarios, there are some limiting factors which should be taken into account since they can lead to unexpected behavior. In this section we discuss these in more detail and address how they can be dealt with.

Limited resolution. Since we embed the image into a coarse lattice we cannot directly obtain pixel or even sub-pixel precision. Although a multi-scale extension is possible, increasing the number of squares makes the overall iterative process ineffective. This is caused mainly by an increasing number of block matching instances and shape regularization iterations. However, the coarse approximation produced by our algorithm is typically close enough to the desired pose so that classical energy-based approaches can be utilized to refine the registration to sub-pixel precision (we use a publicly available implementation of [Glocker et al. 2008]).

Occlusion and topology. The presence of occluders and topology variations in 2D projections of 3D articulated objects is a long-standing and challenging computer vision problem. It also limits

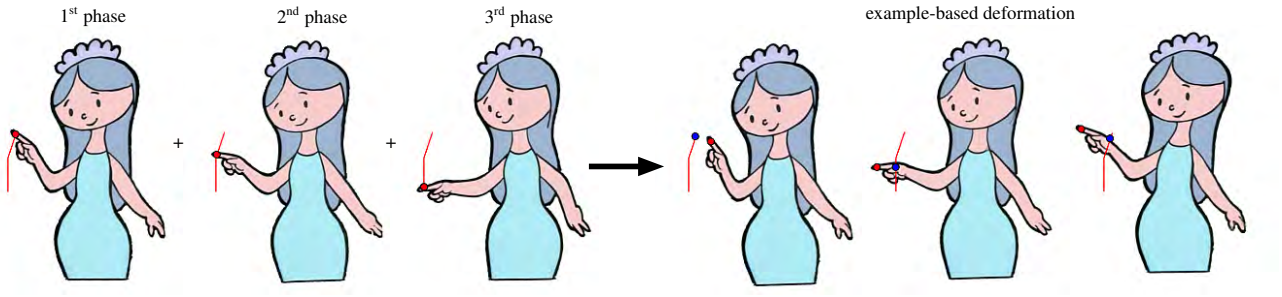

Figure 6: *Example-based shape deformation – by registering several consecutive animation phases (left) a smooth sequence of intermediate frames can be generated. This can be utilized for a synthesis of new poses satisfying a user-given positional constraint (right): the current position of the dragged point (red dot) is projected (blue dot) on its key-frame trajectory (red curve) to retrieve the corresponding intermediate frame which is subsequently deformed to match the actual position of the dragged point.*

the usage of our algorithm since occluded parts move together with their occluders and topology variations impose false connectivity. A possible solution to this problem is to reconstruct a layered representation of the image where each layer has its own depth information and shares common control points with other layers. Using this structure one can perform the pushing step only between layers having equal depth and then use the shape regularization phase to propagate these movements to other connected layers.

Scaling and shearing. As our method exploits the as-rigid-as-possible deformation model it is not able to handle deformations which do not preserve local rigidity (such as scaling or 3D rotation). This limitation can be partially reduced by exploiting an approach analogous to the use of a hierarchy of deformation models. Initially we can treat the image as more rigid and perform a higher number of rigid shape matching iterations. After that the number is gradually decreased to allow more flexible deformations. However, even using this extension, significant changes in scale and/or shearing are still intractable. In cases when such deformations are required we can switch to a different local deformation model allowing similarity or even affine transformations. According to [Schaefer et al. 2006] for similarity this can be done by replacing (4) with:

$$\mu = \sum_i \hat{\mathbf{p}}_i \hat{\mathbf{p}}_i^T \tag{7}$$

and for affine transform by replacing (3) and (5) with a full affine matrix:

$$\mathbf{A} = \left(\sum_i \hat{\mathbf{p}}_i^T \hat{\mathbf{p}}_i \right)^{-1} \sum_j \hat{\mathbf{p}}_j^T \hat{\mathbf{q}}_j \tag{8}$$

However, since similarity and affine models do not tend to preserve area they are not as stable as the original as-rigid-as-possible model therefore are suitable only for final refinement when the source and target pose are close enough, otherwise they can distort the image considerably and lead to unacceptable results.

Insufficient overlap. A key feature of our technique is the ability of the block matching phase to overcome configurations which correspond to local minima in energy-based techniques. However, to avoid matching ambiguity, the size of searching windows has to be limited. Because of this reason, our method requires partial overlap and consistent scale & orientation between source and target images. For images that do not satisfy these requirements we recommend that the initial rigid-body transformation be estimated by hand or that some automatic rigid-body registration technique should be used. When the insufficient overlap is caused by large free-form deformation (as in Figure 9h), the algorithm may get stuck in some inappropriate pose. In this case, the user can provide additional hints by dragging a problematic part towards a desired position or use bidirectional registration, i.e. to alternate pushing and regularization steps on both source and target image. As compared to single image deformation where the target image is static, this approach provides better flexibility and so can overcome challenging configurations.

4 Results and Applications

We implemented our algorithm and tested it on various hand-drawn cartoon characters and human postures undergoing both small and large free-form deformations and changes in appearance. Selected results are presented in Figures 1 and 9. All examples are in PAL resolution. The width of squares on the embedding lattice is the same as the width of neighborhood N in equation (1), i.e. 16 pixels (blue squares in Figure 9) and the width of the search area M is 48 pixels (red squares). The number of inner iterations in the shape regularization phase is linearly decreased from 256 to 32 during the first 50 push-regularize steps.

To have an unified overview of the algorithm convergence we measured the average sum of absolute differences (blue curve) and the average distance to the starting pose (red curve) during the first 100 iterations for all examples in Figure 9. The actual number of iterations needed to reach stable configuration varies with the complexity of deformation. In simple cases it does not exceed 30, however, for large deformations such as human postures in Figures 9f and 9g it increases to 80. A typical processing speed is 20 iterations per second on a 3 GHz machine while the most demanding part is the block matching phase. However, it can be easily parallelized and thus much better processing speeds could be reached on some parallel architectures.

As stated in Section 3.4 the accuracy of our algorithm depends mainly on the resolution of the embedding lattice. Such precision is typically sufficient for applications where exact dense correspondences are not required such as auto-painting or motion capture. However, for inbetweening and example-based shape deformation, where smooth transitions are required, subsequent refinement is necessary to obtain sub-pixel accurate dense mapping. When a local appearance does not change considerably it is possible to take the result of our method as an initial guess for an energy-based approach (we use [Glocker et al. 2008]) and obtain refined sub-pixel accurate mapping. In Figure 9 we show both the registrations produced by our algorithm and also the corresponding refined results.

In the rest of this section we discuss several applications. Since our work is mainly motivated by the needs of the professional cartoon animation production pipeline we focus on this field. However, we believe that our algorithm is versatile enough to be applied

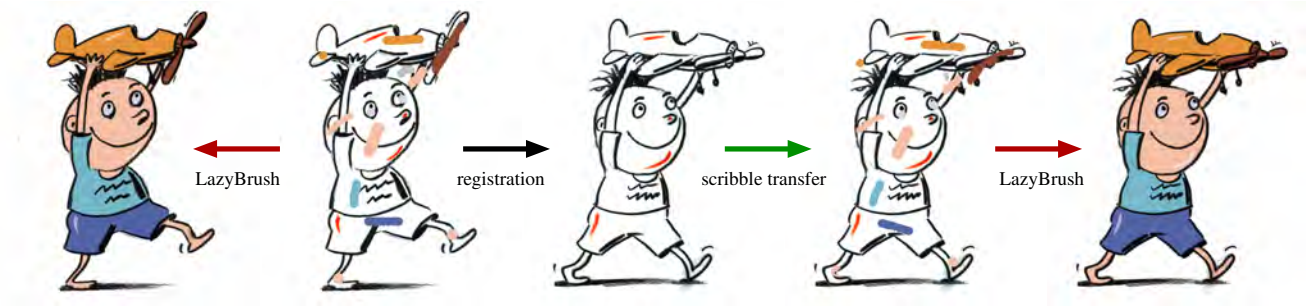

Figure 7: *Auto-painting by unsupervised scribble transfer – color scribbles can be transferred from already painted to yet unpainted animation frames using our deformable image registration algorithm. The LazyBrush [Sýkora et al. 2009] algorithm is then utilized to compute the final painting.*

in other contexts such as pedestrian registration, deformable object snapping or improving interactive shape deformation by providing dynamic feedback that looks like physical simulation.

Unsupervised inbetweening. The knowledge of dense correspondences between several consecutive frames allows us to create smooth intermediate transitions. One possibility for achieving this is to linearly interpolate positions of corresponding pixels. However, this is applicable only for small motions since the local rigidity is not preserved. A better solution is to divide the transition to coarse and fine level. The coarse level consists of the same square lattice as used for registration and the fine level is represented by two dense displacement maps (source-target and target-source) computed by the energy-based method [Glocker et al. 2008]. To generate the intermediate frame we first linearly interpolate the coarse lattices of the source and target frame and perform several shape regularization iterations to enforce rigidity. On the pixel level we scale transformed source-target and target-source displacements and resample source and target images accordingly. Finally we blend co-located pixels to obtain C^0 continuity.

Example-based shape deformation. User-driven shape deformation based on intuitive positional constraints has recently become popular particularly due to the work of Igarashi et al. [Igarashi et al. 2005]. Although many researchers attempt to improve this technique [Schaefer et al. 2006; Weng et al. 2006; Wang et al. 2008] they still offer only single image deformation. Thanks to our deformable image registration algorithm, we can easily extend this technique to multiple images (see Figure 6). By registering several animation phases we obtain smooth transitions as we do for inbetweening, however, a key difference here is that we allow the user to drag a specific point and move it to a different location. We project this new location on its inbetweening trajectory and generate a closest transition frame that is subsequently deformed to match the user-specified position. This enables interactive shape deformation which respects the original animation but is more flexible than simple inbetweening. Moreover, the ease of manipulation is improved considerably since in contrast to classical approaches we do not need to place other positional constraints to fix the global pose. Instead we apply a lower number of shape regularization iterations as described in Section 3.2 to suppress the diffusion of rigidity so that parts of the shape having long geodesic distances from the selected point remain untouched.

Auto-painting and editing. The process of adding colors to hand-drawn images is one of the most challenging tasks in the classical cartoon animation pipeline. In the last decade researchers have developed various auto-painting approaches allowing significant reduction of manual effort [Madeira et al. 1996; Chang and Lee 1997; Seah and Feng 2000; Sýkora et al. 2005; Qiu et al. 2008]. As these techniques exploit similarity of regions they require drawing styles that can be easily converted to a set of homogenous regions. Recently, Sykora et al. [2009] introduced a more general approach based on color scribbles that is applicable to a broad class of different drawing styles. By registering painted and yet unpainted frames, we can transfer scribbles between animation frames and considerably speed up the process (see Figure 7). Besides painting, similar workflow can be utilized to perform various editing operations, e.g. retouching, insertion, and deletion.

Motion capture and retargeting. In this application pioneered by [Bregler et al. 2002] the aim is to transfer specific motion captured in a sequence of images to a novel animation having a different visual appearance. In our case this can be done by superimposing a skeleton on a reference pose and then using deformable image registration to find corresponding positions of joints and bones in subsequent animation frames (see Figure 8). Moreover, the superimposed skeleton can be utilized to form a set of rigid clusters and perform skeletal-like deformation via rigid square matching as in [Wang et al. 2008].

5 Conclusions and Future work

We presented a new approach to deformable image registration based on an approach analogous to the dynamic simulation of deformable objects. In contrast to previous techniques it handles large free-form deformations and notable changes in appearance. Although the algorithm prevails in challenging situations it is surprisingly easy to implement. We believe that, due to its simplicity and robustness, it will find numerous applications in tasks where the knowledge of correspondences between images plays an important role. As an example of such usage we presented several use cases in the context of the cartoon animation production pipeline.

As future work we plan to extend our approach to handle occlusions and also to develop efficient multi-resolution schemes to avoid dependance on energy-based techniques for applications where a pixel or sub-pixel precision is required.

Acknowledgements

We are grateful to Ladislav Kavan for numerous fruitful discussions. Thanks must also go to anonymous reviewers for their insightful comments and to Tomáš Jarkovský, Vojtěch Votýpka, and Tomáš Rychecký from AniFilm studio for being initiators of this work. Cartoon images used in this paper are courtesy of Pavel Koutský / AniFilm, and studios Universal Production Partners & Digital Media Production. This work has been supported by the Marie Curie action IEF, No. PIEF-GA-2008-221320.

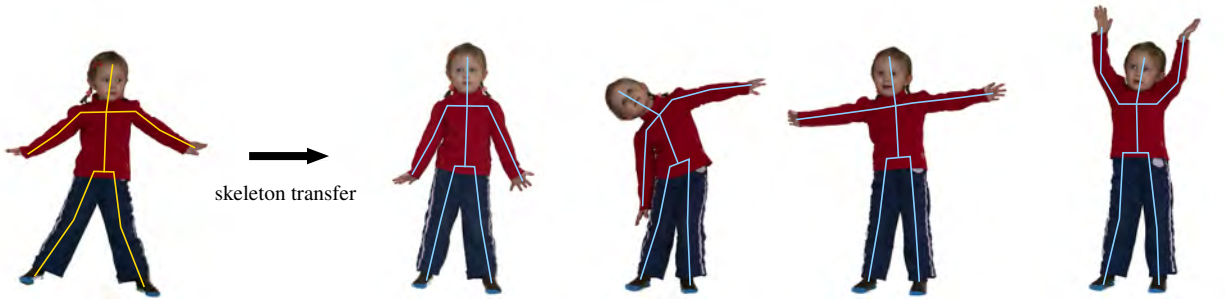

Figure 8: *Motion capture by skeleton transfer – the image of the rest pose was manually annotated by a skeleton (left). Its corresponding positions on several new postures were obtained without user intervention using our deformable image registration algorithm (right).*

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.

BERGEN, J. R., ANANDAN, P., HANNA, K. J., AND HINGORANI, R. 1992. Hierarchical model-based motion estimation. In *Proceedings of European Conference on Computer Vision*, 237–252.

BOTSCH, M., PAULY, M., WICKE, M., AND GROSS, M. H. 2007. Adaptive space deformations based on rigid cells. *Computer Graphics Forum 26*, 3, 339–347.

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.

CHANG, C. W., AND LEE, S. Y. 1997. Automatic cel painting in computer-assisted cartoon production using similarity recognition. *The Journal of Visualization and Computer Animation 8*, 3, 165–185.

GHOLIPOUR, A., KEHTARNAVAZ, N., BRIGGS, R. W., DEVOUS, M., AND GOPINATH, K. S. 2007. Brain functional localization: A survey of image registration techniques. *IEEE Transactions on Medical Imaging 26*, 4, 427–451.

GLOCKER, B., KOMODAKIS, N., TZIRITAS, G., NAVAB, N., AND PARAGIOS, N. 2008. Dense image registration through MRFs and efficient linear programming. *Medical Image Analysis 12*, 6, 731–741.

IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. F. 2005. As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics 24*, 3, 1134–1141.

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.

KLEIN, S., STARING, M., AND PLUIM, J. P. W. 2007. Evaluation of optimization methods for nonrigid medical image registration using mutual information and B-splines. *IEEE Transactions on Image Processing 16*, 12, 2879–2890.

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

LI, W., AND SALARI, E. 1995. Successive elimination algorithm for motion estimation. *IEEE Transactions on Image Processing 4*, 1, 105–107.

LOWE, D. G. 2004. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision 60*, 2, 91–110.

LUCAS, B. D., AND KANADE, T. 1981. An iterative image registration technique with an application to stereo vision. In *Proceedings of International Joint Conference on Artificial Intelligence*, 674–679.

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

MAINTZ, J. B. A., AND VIERGEVER, M. A. 1998. A survey of medical image registration. *Medical Image Analysis 2*, 1, 1–16.

MODERSITZKI, J. 2004. *Numerical Methods for Image Registration*. Oxford University Press, UK.

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformations based on shape matching. *ACM Transactions on Graphics 24*, 3, 471–478.

PAPENBERG, N., SCHUMACHER, H., HELDMANN, S., WIRTZ, S., BOMMERSHEIM, S., ENS, K., MODERSITZKI, J., AND FISCHER, B. 2007. A fast and flexible image registration toolbox. In *Bildverarbeitung für die Medizin*, 106–110.

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.

RIVERS, A. R., AND JAMES, D. L. 2007. FastLSM: Fast lattice shape matching for robust real-time deformation. *ACM Transactions on Graphics 26*, 3, 82.

SCHAEFER, S., MCPHAIL, T., AND WARREN, J. 2006. Image deformation using moving least squares. *ACM Transactions on Graphics 25*, 3, 533–540.

SEAH, H. S., AND FENG, T. 2000. Computer-assisted coloring by matching line drawings. *The Visual Computer 16*, 5, 289–304.

SHEKHOVTSOV, A., KOVTUN, I., AND HLAVÁČ, V. 2008. Efficient MRF deformation model for non-rigid image matching. *Computer Vision and Image Understanding 112*, 1, 91–99.

SORKINE, O., AND ALEXA, M. 2007. As-rigid-as-possible surface modeling. In *Proceedings of Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, 109–116.

SUMNER, R. W., SCHMID, J., AND PAULY, M. 2007. Embedded deformation for shape manipulation. *ACM Transactions on Graphics 26*, 3, 80.

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. LazyBrush: Flexible painting tool for hand-drawn cartoons. *Computer Graphics Forum 28*, 2, 599–608.

SZELISKI, R. S., ZABIH, R., SCHARSTEIN, D., VEKSLER, O., KOLMOGOROV, V., AGARWALA, A., TAPPEN, M., AND ROTHER, C. 2008. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence 30*, 6, 1068–1080.

WANG, Y., XU, K., XIONG, Y., AND CHENG, Z.-Q. 2008. 2D shape deformation based on rigid square matching. *Computer Animation and Virtual Worlds 19*, 3–4, 411–420.

WENG, Y., XU, W., WU, Y., ZHOU, K., AND GUO, B. 2006. 2D shape deformation using nonlinear least squares optimization. *The Visual Computer 22*, 9, 653–660.

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.

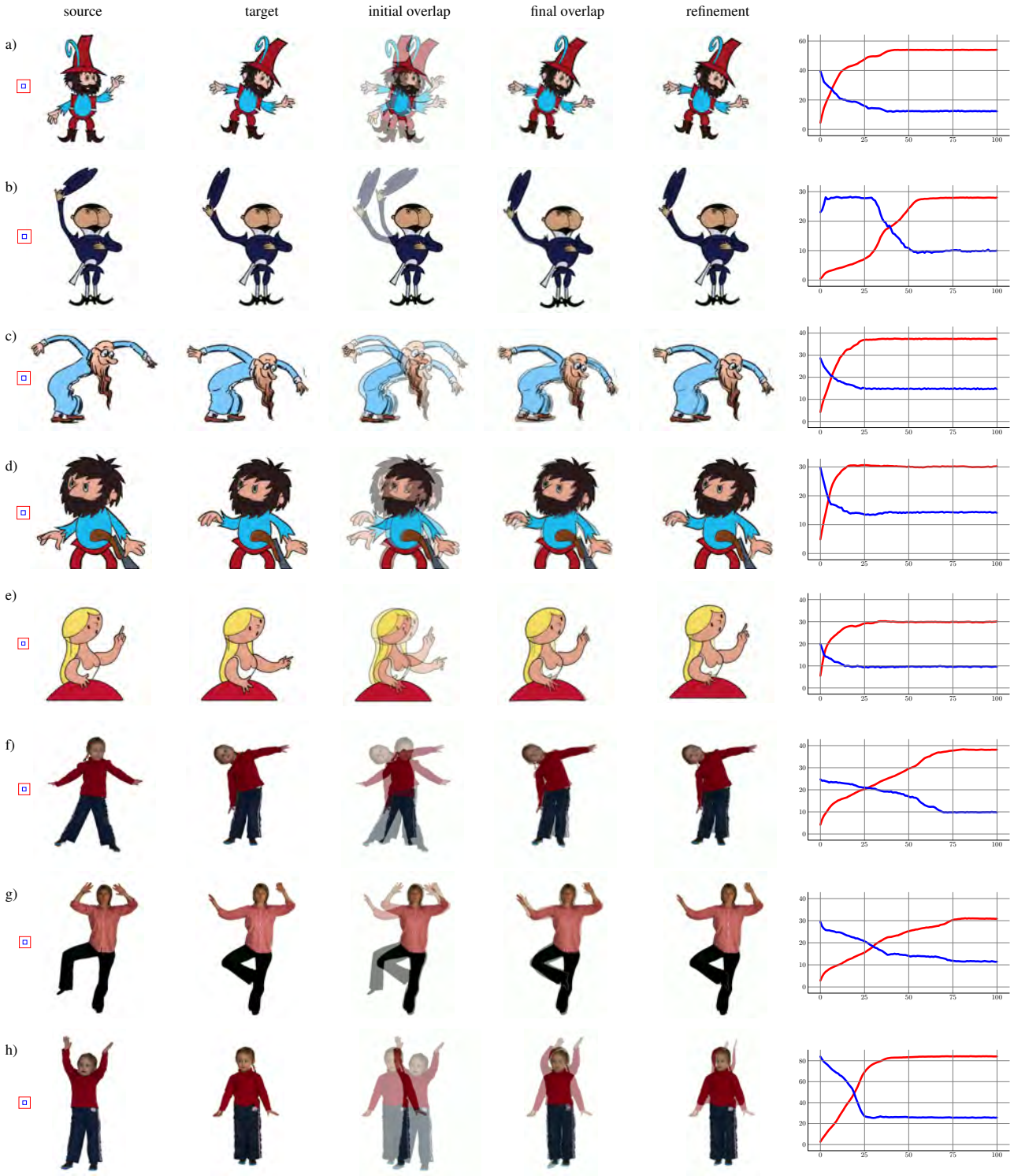

Figure 9: *Selected examples of deformable image registration produced by our algorithm – each example contains (from left to right): size of the square on the embedding lattice equal to the local neighborhood N (blue square), size of the local searching area M (red square), source & target image, their initial overlap, resulting overlap after registration using our approach, overlap refined by energy-based approach [Glocker et al. 2008], and the graph of the average sum of absolute differences (blue curve) and the average distance to starting pose (red curve) for first 100 iterations. The last registration result (h) presents a failure example when our algorithm gets stuck in an undesirable pose due to very large free-form deformation.*

Appendix F

TexToons: Practical Texture Mapping for Hand-drawn Cartoon Animations

D. Sýkora, M. Ben-Chen, M. Čadík, B. Whited, M. Simmons: TexToons: Practical Texture Mapping for Hand-drawn Cartoon Animations. Proceedings of International Symposium on Non-photorealistic Animation and Rendering (NPAR'11), pages 75–83, August 2011. ISBN: 978-1-4503-0907-3.

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

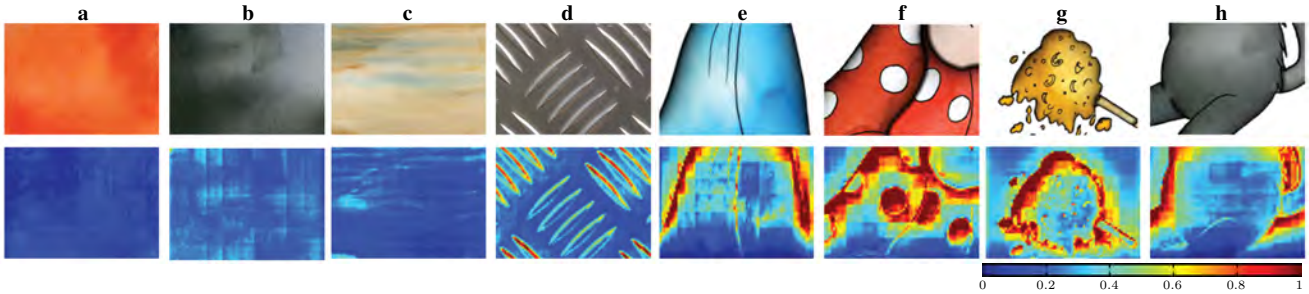

Figure 3: *Visual saliency of the textures. The top row contains example cartoon texture and contour samples. The bottom row visualizes saliency maps automatically predicted using a bottom-up model of human visual attention. (a, b, c) Examples of textures supported by the proposed technique. (d) A texture 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

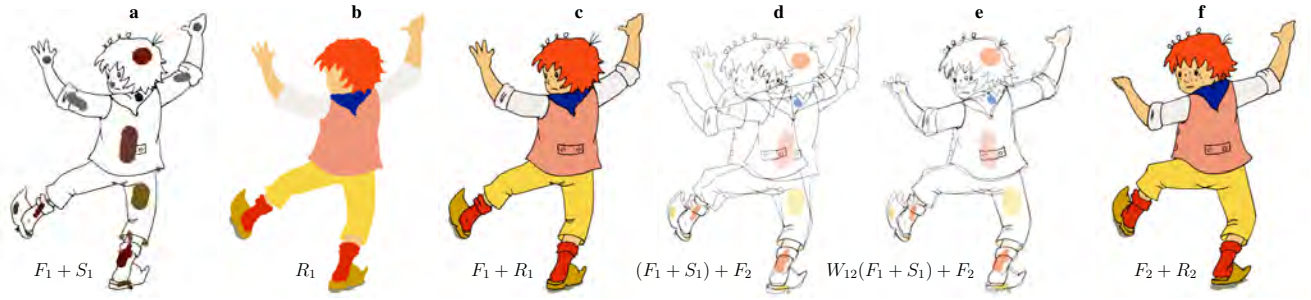

Figure 4: *Automatic painting using LazyBrush: (a) original drawing F_1 with color scribbles S_1, (b) segmentation R_1 of F_1 using scribbles S_1, (c) painted drawing $(F_1 + R_1)$, (d) initial overlap of the source F_1 with scribbles S_1 and the target frame F_2, (e) registration of the source frame 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

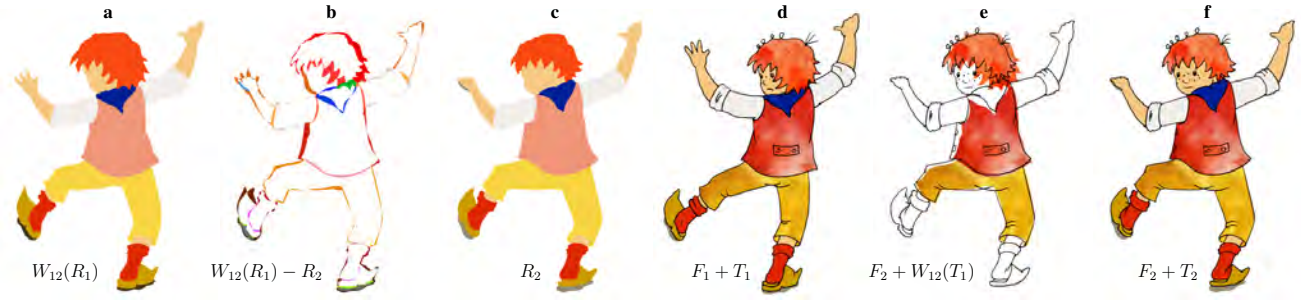

Figure 5: *Domain overlap and texture transfer: (a) segmentation R_1 deformed by W_{12}, (b) domain overlap visualised by symmetric difference of deformed segmentation R_1 and R_2, (c) segmentation R_2, (d) source frame F_1 with textured regions $r_t \in R_1$ mapped using initial texture coordinates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

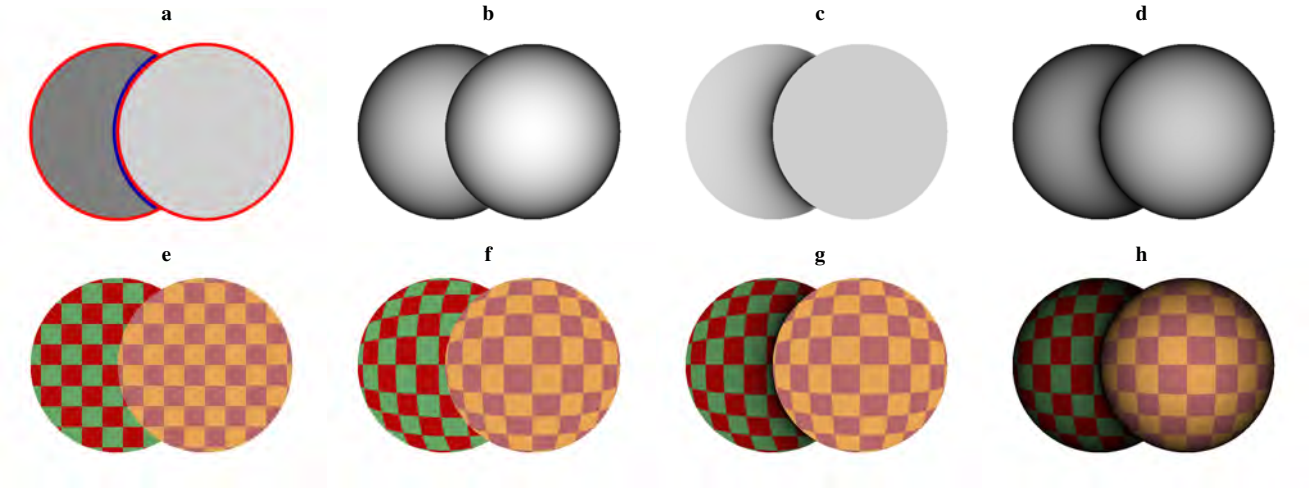

Figure 6: *3D-like effects: (a) depth map with Dirichlet (red) & Neumann (blue) boundary conditions, with initial texture mapping, (b) shading, (c) simulation of ambient occlusion, (d) shading with ambient occlusion, (e) texture mapping using flat UV coordinates, (f) texture rounding based on shading,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

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

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].

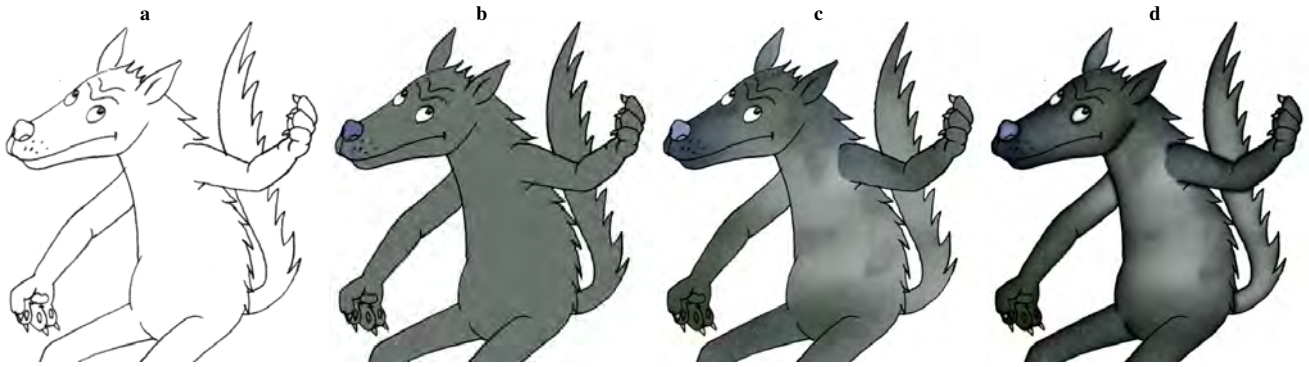

Figure 7: *Adding visual richness to a hand-drawn cartoon: (a) original line drawing, (b) painted with homogenous colors, (c) painted with textures, (d) added texture rounding, shading & simulation of ambient occlusion.*

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).

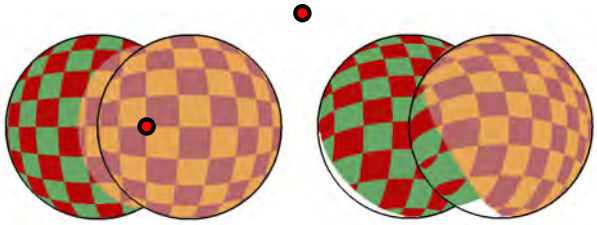

Figure 8: *Parallax mapping failure: resulting texture coordinates are not preserved at region boundaries and depend on the eye position (red dots). This can cause noticeable texture sliding.*

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.

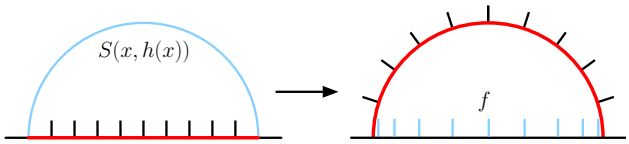

Figure 9: *Texture rounding (1D example): flat texture coordinates and target surface $S = (x, h(x))$ proportional to n_z (left), linearly interpolated texture coordinates on S and their projection back to flat domain f (right).*

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.*

Appendix G

Temporal Noise Control for Sketchy Animation

G. Noris, D. Sýkora, S. Coros, B. Whited, M. Simmons, A. Hornung, M. Gross, and R. Sumner: Temporal Noise Control for Sketchy Animation. Proceedings of International Symposium on Non-photorealistic Animation and Rendering (NPAR'11), pp. 93–98, Vancouver, Canada, August 2011. ISBN: 978-1-4503-0907-3.

Temporal Noise Control for Sketchy Animation

G. Noris[1,2] D. Sýkora[3] S. Coros[2] B. Whited[4] M. Simmons[4] A. Hornung[2] M. Gross[1,2] R. W. Sumner[2]
[1]ETH Zurich, CGL [2]Disney Research Zurich [3]CTU in Prague, FEE [4]Walt Disney Animation Studios

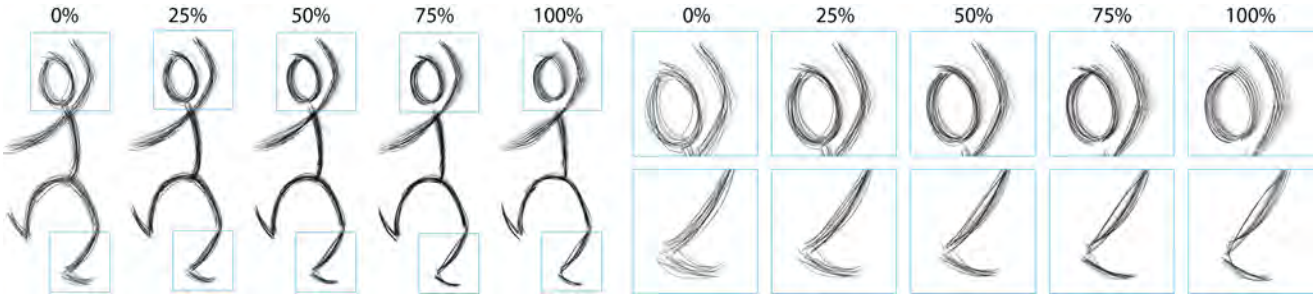

Figure 1: *Balancing scene. We compare different noise reduction values from* 0% *(input animation) to* 100% *(noise-free). Close-ups are marked by blue squares.*

Abstract

We propose a technique to control the temporal noise present in sketchy animations. Given an input animation drawn digitally, our approach works by combining motion extraction and inbetweening techniques to generate a reduced-noise sketchy animation registered to the input animation. The amount of noise is then controlled by a continuous parameter value. Our method can be applied to effectively reduce the temporal noise present in sequences of sketches to a desired rate, while preserving the geometric richness of the sketchy style in each frame. This provides the manipulation of temporal noise as an additional artistic parameter, e.g. to emphasize character emotions and scene atmosphere, and enables the display of sketchy content to broader audiences by producing animations with comfortable noise levels. We demonstrate the effectiveness of our approach on a series of rough hand-drawn animations.

1 Introduction

Compared to traditional cleaned-up drawings, sketches present a kind of visual richness, where both silhouette and interior lines are composed of many rough strokes. This style allows another dimension of expressiveness - emotion, action, and other features can be conveyed through the "sketchy" drawings.

The richness provided by the sketchy style can be considered to be a form of geometric noise. Despite its positive benefits in still images, geometric noise becomes temporal noise in sequences of sketches and is generally unpleasant to view. The industry solution to this problem is to remove the geometric noise. In production environments, early versions of animation (both 2D and 3D) are often composed of sequences of rough sketches. Later in the pipeline, these are systematically replaced either with clean-line drawings or with renderings of 3D scenes, which typically present cleaner visuals. Animations completely made of sketches are less common and generally confined to short sequences or small productions[1].

Our goal is to preserve the expressiveness inherent in sketchy drawings while removing unpleasant temporal issues. We propose to reduce *temporal* noise while keeping *geometric* noise by supporting interpolation at a finer level, down to the individual sketchy strokes. Instead of appearing and disappearing, the strokes transition smoothly from frame to frame. Enforcing these constraints manually in typical production environments would be impractical: establishing a fine-scale correspondence of strokes within the sequence of sketches and generating the proper animation path is too labor-intensive.

A key insight of our approach is that we can first construct a noise-free animation using only a representative subset of the input frames such that effectively all temporal noise is removed. Then, the desired amount of noise can be continuously varied on top of this noise-free animation. Another key idea is the use of motion extraction to allow local stroke searches within the global motion - enabling an automated solution to the fine-scale stroke correspondence problem.

2 Related Work

Research efforts related to sketchy or hand-drawn styles of illustration can be divided into two main topics: simplification and generation (static images as well as temporally coherent animations).

In the area of automated simpification/beautification, several techniques have been developed that reduce the number of lines in a drawing using a density measure to prune possibly redundant lines while still conveying the notion of the original shape [Wilson and Ma 2004; Grabli et al. 2004]. Instead of solely performing stroke reduction, Barla et al. [2005] propose a perceptually motivated technique for synthesizing representative lines. Input strokes are first grouped using a greedy clustering algorithm that pairs strokes according to screen-space proximity. A geometric reconstruction step then follows to produce a single line for each group. The described approach is for static drawings, though they propose incorporating a temporal aspect of perceptual grouping- "common fate", expressed by grouping line segments with similar velocity to produce coherent animation. Shesh et al. [2008] later extended this approach to handle time coherence by building a simplification hierarchy of strokes and using opacity blending to interpolate between a pair of strokes and its simplified version to create smooth transitions.

Generation of static sketchy or pen-and-ink style illustrations from input 2D images/photographs and from 3D models is a popular topic in the field of non-photorealistic animation and rendering (NPAR) (e.g. [Winkenbach and Salesin 1994; Salisbury et al. 1997; Coconu et al. 2006]). Far less attention has been focused

[1]Notable examples include Frédéric Back's short "L'homme qui plantait des arbres" and a brief series of sketches in the "Colors of the Wind" sequence in Disney's animated feature *Pocahontas*.

on developing temporally coherent results suitable for animation. Existing techniques [Curtis 1998; Bourdev 1998; Kalnins et al. 2002; Kalnins et al. 2003] focus on rendering stylized silhouettes of animated 3D models. After silhouettes are extracted from the 3D model the main challenge is to generate coherent "sketchiness" along the silhouette strokes over time, e.g. through assigning temporally coherent parameterizations to strokes in the image plane [Kalnins et al. 2003; Bourdev 1998] or alternatively achieving coherence via a particle system seeded along the silhouette[Curtis 1998] or through stroke texture representations [Bénard et al. 2010] designed for temporal coherence. The key issue with all of these approaches is that they require an underlying 3D model or a clean 2D image with known stroke correspondences. In our case the challenge is that we have a set of unordered strokes in each frame which are not necessarily moving coherently and we need to determine how to best match and interpolate them over time.

3 Method

The method we introduce offers artistic control over the level of temporal noise in hand-drawn animations. Mismatches in the individual strokes used to define the same silhouette in consecutive frames can be a major source of temporal noise, particularly in rough sketches. Generally speaking, we seek to maintain the global motion of an input animation, as well as the overall drawing style, while manipulating the temporal noise level. Smooth output animations are typically preferred, but we note that high-frequency noise can be an effective artistic tool. The animation of a character who is scared, for instance, could benefit from a certain amount of temporal noise to emphasize emotion.

Before describing our method, we introduce the notation used throughout the paper. Our algorithm operates on sequences of frames, where each frame $\mathcal{F}$ contains a set of strokes that appear in the animation at the same moment in time. Each stroke s is a piece-wise linear curve defined by a sequence of vertices. The i-th stroke in a frame $\mathcal{F}$ is given by $\mathcal{F}(i)$.

A motion field is a function $\mathcal{M} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ that tracks the movement of every point on the 2D canvas. In particular, $\mathcal{M}_{\mathcal{F}_1}^{\mathcal{F}_2}$ is the motion field that describes the relative motion between frames $\mathcal{F}_1$ and $\mathcal{F}_2$. We define $\mathcal{D}(\mathcal{M}, \mathcal{F})$ to be the deformed frame that is obtained by taking the vertices of every stroke in $\mathcal{F}$ and displacing them according to $\mathcal{M}$.

3.1 Overview

Our approach takes as input a sequence of frames from a hand-drawn animation. The core algorithm works in two passes (see Figure 2). First, the input sequence is processed to create a *noise-free* animation (see Figure 2a). This is done by sampling the original input animation to choose representative frames. These frames are smoothly interpolated to create a new animation, and, for the time-being, all other frames not in the representative subset are ignored. The output of this stage is a set of smooth, automatically-created inbetween frames.

In the limit if we choose only the first and very last frame of the animation as the representative set, replacing frames from the original sequence with these newly synthesized inbetweens would effectively produce a noise-free animation. However, much of the finer scale motion and sketchy details would also be removed. On the other hand, if the representative set includes all of the original frames, then no interpolation is performed and we have the original, noisy content.

Our goal is to allow an artist to precisely control the level of temporal noise. This is enabled by the second pass of our algorithm, which smoothly interpolates the original noisy animation with the smooth inbetweens created during the first pass (see Figure 2b).

Both passes must solve the same problem of creating smooth inbetween frames. In other words, given two frames of animation, we must establish a fine-scale stroke correspondence and interpolate smoothly between each stroke pair.

Algorithm 1 CreateInbetweens

input $\mathcal{F}_1, \mathcal{F}_2$: animation frames
input t: interpolation parameter, $0 \leq t \leq 1$
output $\mathcal{F}_t$: an inbetween
$\hat{\mathcal{F}}_1 \leftarrow \mathcal{D}(\mathcal{M}_{\mathcal{F}_1}^{\mathcal{F}_2}, \mathcal{F}_1)$
$\mathcal{S} \leftarrow computeStrokeCorrespondencePairs(\hat{\mathcal{F}}_1, \mathcal{F}_2)$
$\mathcal{F}_t \leftarrow \{\}$
for all $(i, j) \in S$ **do**
  $s \leftarrow interpolateStrokes(\mathcal{F}_1(i), \mathcal{F}_2(j), t)$
  $\mathcal{F}_t \leftarrow \mathcal{F}_t \cup \{s\}$
end for

3.2 Representative Frame Sampling

The first phase of our algorithm generates a noise-free sequence which ideally should resemble the original animation as much as possible. The sampling scheme (i.e. choice of representative frames) is crucial to the quality of the resulting animation. Our method is comprised of two main components: the sampling strategy and timing control.

We propose two selection strategies for choosing representative frames. *Uniform sampling* selects representative frames distributed at equal intervals specified by a window size w. *Keyframes* uses important or extreme frames that are manually selected from the original animation input.

Once the representative frames are selected, our system assumes a uniform division of time units inside each interval, resulting in an approximation of the input timing. Our experience is that, by respecting the input animation keyframes, this approximation is acceptable. However, for particular scenes it might be desirable to have a finer control. This can be achieved by altering the timing values used to generate the inbetweens in [Whited et al. 2010].

It is assumed that the keyframe specification and timing information when needed is explicitly provided as input, along with the original animation. This is suitable for a classic animation environment, where both keyframes and timing information are captured by timing charts.

3.3 Creating Smooth Inbetween Frames

Algorithm 1 describes the steps to create smooth inbetween frames. Both passes of our method use this algorithm. The input consists of a pair of representative animation frames, $\mathcal{F}_1$ and $\mathcal{F}_2$, and an interpolation parameter $t, 0 \leq t \leq 1$. Our goal is to create a new frame $\mathcal{F}_t$ using solely the strokes from frames $\mathcal{F}_1$ and $\mathcal{F}_2$. This process ensures continuity in the strokes that are output as t varies between 0 and 1, and thus results in smooth animations.

The first step towards creating smooth inbetween frames consists of identifying the pairs of strokes from $\mathcal{F}_1$ and $\mathcal{F}_2$ that represent the same features in the drawing at different moments in time. Every stroke from one input frame is matched to the most likely candidate stroke from the other frame, as expressed by a stroke correspondence measure. We refer to this process as finding the stroke-to-stroke correspondences (see Section 3.3.2).

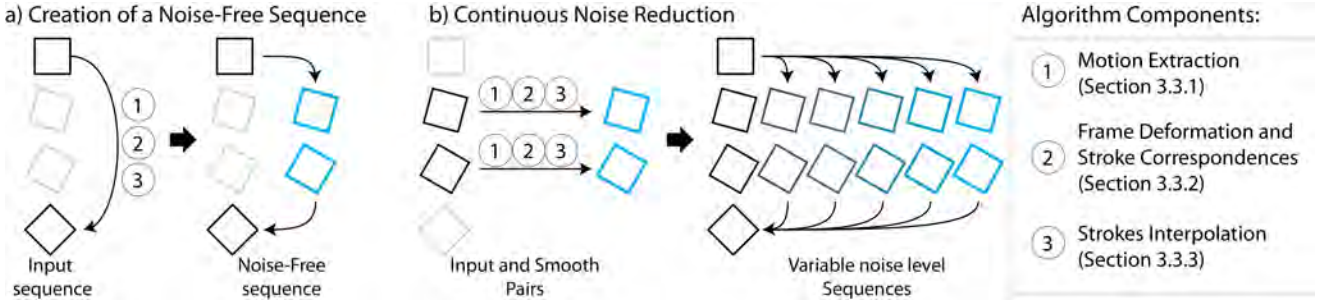

Figure 2: *Method Overview. Our method works in two phases. (a) The input sequence is processed to create a Noise-Free sequence. To do so, three steps are performed: (1) Motion extraction (2) Frames Deformation and Stroke-to-Stroke correspondences, and (3) Interpolation. (b) For each pair of input vs. noise-free frames, an arbitrary number of sequences with increasing noise reduction can be generated, again using the same three steps.*

Strokes that are used to define the same element in a drawing (e.g. a portion of the silhouette) can be far apart spatially from frame to frame. Similarly, strokes that represent different elements of the drawing can become spatially close. Computing appropriate stroke-to-stroke correspondences in this setting is therefore a very difficult task, since proximity is not a reliable measure of similarity. To mitigate this problem we compute the stroke correspondence measure after deforming the strokes of $\mathcal{F}_1$ according to the motion field $\mathcal{M}_{\mathcal{F}_1}^{\mathcal{F}_2}$ (see Section 3.3.1). More precisely, we compute the stroke-to-stroke correspondences between the frames $\hat{\mathcal{F}}_1 = \mathcal{D}(\mathcal{M}_{\mathcal{F}_1}^{\mathcal{F}_2}, \mathcal{F}_1)$ and $\mathcal{F}_2$. In general, the spatial distances between the strokes in the deformed version of $\mathcal{F}_1$ and $\mathcal{F}_2$ are significantly smaller, so computing stroke-to-stroke correspondences in this setting is less prone to mis-matches.

The stroke-to-stroke correspondence step results in a set of pairs (i, j), where each pair indicates a correspondence between stroke i of $\mathcal{F}_1$ and stroke j of $\mathcal{F}_2$. All pairs of corresponding strokes are then interpolated to create the inbetween frames $\mathcal{F}_t$ (see Section 3.3.3).

The remainder of this section outlines the method used to create the motion fields, the process of deforming the input frame, the criteria used to compute the stroke-to-stroke correspondences, and the stroke interpolation method.

3.3.1 Motion Extraction

An important part of our processing pipeline consists of computing the relative motion, or motion field, between two frames $\mathcal{F}_1$ and $\mathcal{F}_2$. We use a slightly modified version of the As-Rigid-As-Possible (ARAP) method described in [Sýkora et al. 2009]. The original approach assumes the mask of the registered image is known beforehand. This allows the control lattice, which represents the motion field $\mathcal{M}_{\mathcal{F}_1}^{\mathcal{F}_2}$, to be adapted to the topology variations of the underlying shape.

In our problem domain, we are dealing with a more complicated scenario, as the input consists of an unordered set of strokes without any connectivity information. However, it is still important to take into account the topology of the input sketch, as opposed to using a uniform grid. To overcome this difficulty we compute a rasterized distance field, as illustrated in Fig. 3, which we use as a mask. The computed distance field closes small gaps between the input strokes. In addition, the distance field provides a better cue for image registration (similar to *chamfer matching* [Borgefors 1988]).

In addition to the distance field, we use a hierarchical coarse-to-fine refinement. We build a multi-resolution pyramid by recursively reducing the image scale by a factor of two. Then we proceed from the coarse to fine level and run the registration algorithm with a control lattice of constant quad size. When moving up each level, we render a pixel accurate motion field to initialize the position of the control points on the finer lattice. This helps us to speed up the convergence of the motion field extraction algorithm and increases robustness under large motions. The hierarchical refinement also allows us to adapt to fine details and provide tight image registration.

As noted in [Sýkora et al. 2009] the image registration algorithm can potentially get trapped in a local optimum. These cases are typically rare but when they occur we let the user drag-and-drop selected control points in order to guide the algorithm towards a better solution. This operation can be implemented simply by fixing the position of the selected control point and changing its weight to some very large value as in [Wang et al. 2008].

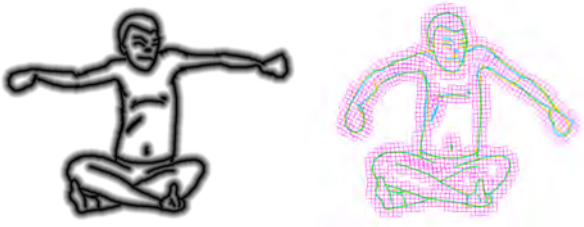

Figure 3: *Building the control lattice for the ARAP image deformation: distance field computed from rasterized strokes (left), ARAP registration using the control lattice based on the distance field (right).*

3.3.2 Frame Deformation and Stroke Matching

The motion field $\mathcal{M}_{\mathcal{F}_1}^{\mathcal{F}_2}$ represents the relative motion between frames $\mathcal{F}_1$ and $\mathcal{F}_2$. This provides a way of estimating the location of each stroke from $\mathcal{F}_1$, if it had been drawn at the time represented by frame $\mathcal{F}_2$. The computation of the deformed frame is straightforward. The vertices v (which are simply 2-dimensional points on the digital canvas) defining the strokes in $\mathcal{F}_i$ are displaced according to the motion field: $v \leftarrow v + \mathcal{M}_{\mathcal{F}_1}^{\mathcal{F}_2}(v)$.

The stroke correspondence step is used to compute a measure of how well two strokes from different frames will interpolate. Intuitively, the better aligned and spatially close the two strokes are, the better their correspondence measure should be. For our work, we define the correspondence measure between two strokes s_1 and s_2 as $h(s_1, s_2) * h(s_2, s_1)$, where $h(A, B)$ is a component of the Hausdorff distance. More precisely, $h(A, B) = max_{a \in A}(min_{b \in B}(d(a, b)))$ and $d(a, b)$ is the Euclidean distance

Figure 5: *Generation of the motion ground truth. A planar grid is generated and deformed with Autodesk Maya (left). A reference image is applied as texture on the mesh (center), generating a reference animation over which the final tree animation is sketched (right). The animation motion is therefore captured by the mesh deformation.*

between points a and b, i.e. the vertices of strokes A and B. We note that this is one of many choices of similarity measures [Seah and Feng 2000; Veltkamp 2001; Lie et al. 2010]. However, in our experiments, we found the Hausdorff distance to work well and be more robust than other measures due to the lack of structure in sketchy drawings.

3.3.3 Stroke Interpolation

The stroke correspondence algorithm is used to find all pairs of strokes that need to be interpolated to create the inbetween frames. We use the three-step deformation method introduced in Whited et al. [2010] to create smooth blends for each pair of strokes. The strength of this approach over other techniques [Fu et al. 2005; Baxter and Anjyo 2006; Baxter et al. 2009] is that in addition to interpolating the stroke curvatures, it also computes the global motion between pairs of strokes along logarithmic spiral trajectories, thus capturing the affinity of the global motion (rotation, translation and uniform scaling).

4 Results

We tested the effectiveness of our algorithm on hand-drawn animation sequences containing between 30 and 90 frames. In order to evaluate our motion extraction algorithm, the animation of the *Tree* shown in Figure 4 was generated procedurally (see Figure 5). All other animations were created manually with a digital drawing tool. The *Square* animation, shown in Figure 6, presents a simple case with an almost rigid motion. The use of temporal noise as an artistic tool is investigated using the *Face* animation, which is shown in Figure 11. Lastly, the *Balancing* animation, Figure 1, illustrates another challenging example handled by our framework.

For the examples in this Section, to emphasize the temporal progression, consecutive animation frames are displayed with decreasing opacity. For the *Square*, *Balancing*, and *Face* scenes, we used a uniform representative frame sampling, with a window size of 7 frames. The *Tree* scene uses a set of 8 selected keyframes and the resulting window sizes are between 4 and 7 frames. Temporal reduction values are displayed as percentages, where 0% is the input animation and 100% is the Noise-Free animation.

Our algorithm was developed in C++ and runs as a single thread. On a standard workstation, the execution of Algorithm 1 takes up to 10 seconds per pair of frames, with the motion extraction and the search for correspondences being the most time-consuming tasks. Overall, computing a full scene takes a few minutes with the unoptimized prototype.

Figure 6: *Square Scene. This image shows the effect of increasing window sizes. Each close-up shows 5 animation frames overlayed with decreasing opacity. When the noise is high, the lines look evenly distributed and unstructured (Input Sequence). With low temporal noise, lines appear to follow a structure and tend to be clustered (5 and 7).*

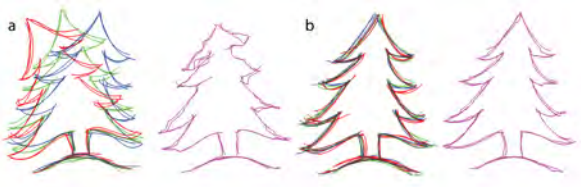

Figure 8: *Neighborhood Averaging. A window of three frames (red, green, blue) is used to compute a per point neighborhood average (purple). When the motion is large (a), the resulting frame is strongly degenerated. Even for slow animations (b), this approach leads to undesirable kinks and deformations.*

4.1 Ground Truth Comparison

To evaluate our motion extraction method we created a "ground truth" animation of the Tree in Maya. Starting with a planar textured mesh of the undeformed tree (see Figure 5), the mesh was deformed by using the bend deformer tool and keyframing the curvature. The deformed tree texture was then used as a reference over which an artist sketched each frame of the animation.

Figure 7 compares the extracted motion field with the ground truth generated in Maya. In general, the extracted motion captures both the global animation motion and local deformations due to the geometric noise. As a result, it provides a very precise stroke alignment, which greatly simplifies the task of finding stroke to stroke correspondences.

4.2 Neighborhood Averaging Comparison

We compare our results with those obtained through neighborhood interpolation, which proceeds as follows. For each sample point p_1 of a stroke j in a frame i, we collect the two nearest neighbor points p_2 and p_3 in frames $i-1$ and $i+1$. p_1 is then updated to $p_1 \leftarrow (p_1 + p_2 + p_3)/3$. An example result is shown in Figure 8. Since each sample point p_k is free to move independently, divergent attractions result in breaks of the line continuity. This usually happens when sample points of one stroke are influenced by different sets of strokes.

This result shows that a simple averaging approach is not desirable. In general, we observe that when the motion is large, this approach produces obvious artifacts, losing important features of the frame. This motivates the use of motion extraction to alleviate the effects of large animation motions. Even when the motion is small, kinks and undesirable deformations are present. This motivates the consideration of strokes as atomic entities, therefore shifting the correspondence from the individual points samples to the stroke level. Our approach benefits from both considerations.

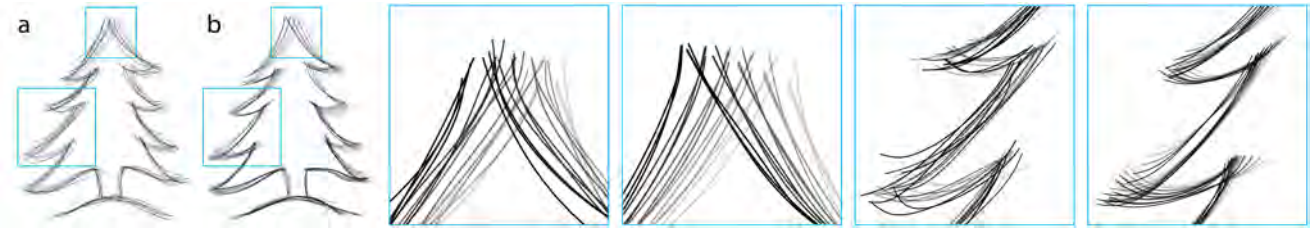

Figure 4: *Tree scene. We compare the input animation (a) with the noise free result (b). Close-ups are marked by blue squares.*

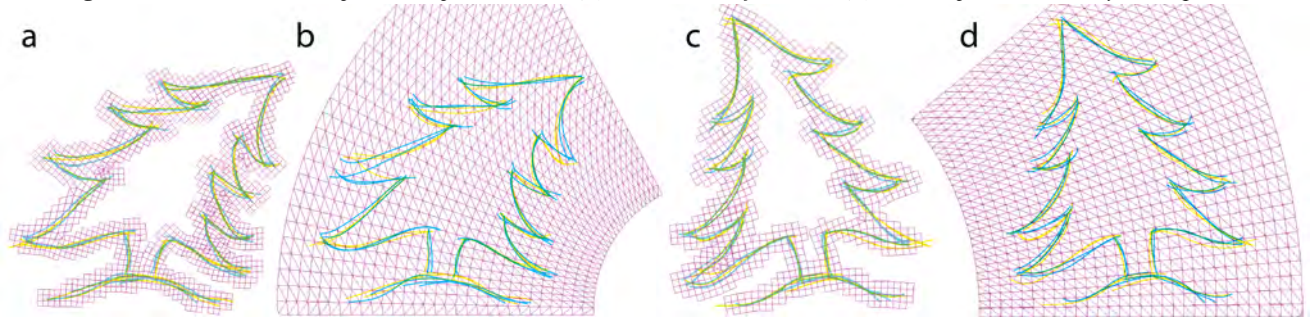

Figure 7: *Motion Extraction Comparison. The extracted motion (a and c) is compared with the ground truth motion (b and d). The blue and yellow frames represent $\mathcal{D}_i^j$ and $\mathcal{F}_j$, with $i = 22, j = 26$ (a,b) and $i = 29, j = 33$ (c,d). The extracted motion is precise and simplifies the search of stroke to stroke correspondences.*

Figure 9: *Sampling Strategy. This image shows the maximum motion error (visualized as misalignment) obtained in the Tree animation using two different sampling strategies: Uniform Sampling, with a window size of 7 frames, and Keyframes, with 6 keyframes marked at the important animation times.*

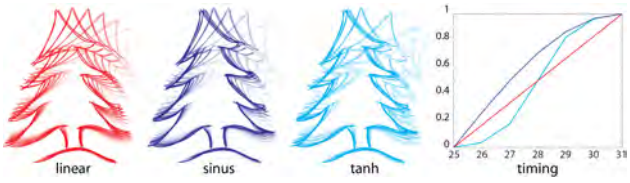

Figure 10: *Timing Control. This image shows the effect of different timing functions applied to the same set of frames.*

4.3 Sampling and Timing Control

As discussed in Section 3.2, we implemented two selection strategies for choosing representative frames, *uniform sampling*, and manually designated *keyframes*. Figure 9 compares results of using the different selection strategies. Notice that using the input animation keyframes, as opposed to uniformly sampling the frames, greatly reduces the motion error.

Figure 10 shows the effect of different timing charts applied to the same set of frames where the timing values used to generate the in-betweens in [Whited et al. 2010] are altered. Notice how the spaces are affected by the choice of the timing functions - most notably at the tip of the tree.

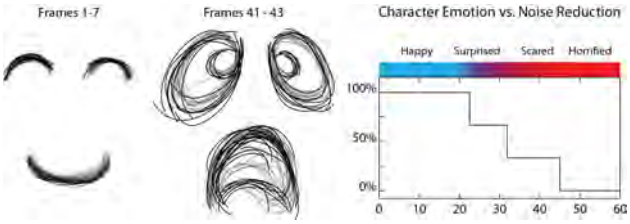

Figure 11: *Face. This scene presents a character with an emotional evolution from happy to horrified. We adapt the noise reduction to these emotions.*

4.4 Temporal Noise as an Artistic Tool

The second phase of our method allows the generation of sequences with varying temporal noise. By manipulating the noise reduction level in different parts of the animation, an artist has the ability to use the noise as an additional storytelling element. Noisy animations can be used to portray certain feelings, such as anger or fear. A proof of concept is shown in Figure 11.

5 Conclusion

We have presented a novel technique for noise reduction/manipulation in sketchy animations and demonstrated its use on a series of hand-drawn inputs. Our approach not only makes the production of larger scale sketchy animations feasible through automated noise reduction, but also widens the scope of artistic control to include noise as a first-class creative device.

5.1 Limitations and Extensions

One limitation of this work is due to the intrinsic difficulty of handling occlusions, topology changes, and disconnected components moving independently in a 2D environment. For the motion extraction step, distinct objects with different motions can create divergent motion fields and occlusions can force unnatural compres-

sions; both cases are difficult to capture with a ARAP model. In order to handle these scenarios in complex scenes, the input would need to be first segmented into coherent layers.

Additionally, complex curved strokes that self intersect may cause problems in the correspondence and interpolation steps. A simple solution, similar to what is proposed in Barla et al. [2005], is to cut these strokes into separate pieces. However, every piece would then behave independently, which may not be the desired animation effect.

Another limitation of our work lies in the selection of keyframes. As shown in Figure 9 the set of selected representative frames has a significant impact on the output animation. In particular, as the complexity of the animation increases (i.e. higher frequency motions), a larger number of keyframes is needed to preserve the motion. This effectively limits the number of smooth inbetween frames that we can create to reduce temporal noise.

Our frame inbetweening technique needs to be extended to provide smooth interpolation across multiple keyframes. Although the interpolating motions computed by the logarithmic spiral technique [Whited et al. 2010] are smooth between consecutive keyframes, the approach has the limitation that continuity is not necessarily preserved across longer sequences. However, multiple techniques have been proposed to preserve this continuity at the keyframes, while still interpolating them. In one approach [Powell and Rossignac 2008], subdivision is used to smoothly interpolate 3D poses by blending consecutive screw motions. A more recent approach [Rossignac and Vinacua 2011] also produces continuous motions as a blending of consecutive "steady affine motions". In 2D, both of these algorithms may be applied to logarithmic spirals with little modification.

Another possible area of exploration is the use of our noise reduction technique as a post-processing operation on sketchy animations synthesized using NPAR techniques.

References

BARLA, P., THOLLOT, J., AND SILLION, F. X. 2005. Geometric clustering for line drawing simplification. In *In Proceedings of the Eurographics Symposium on Rendering*, 183–192.

BAXTER, W., AND ANJYO, K.-I. 2006. Latent doodle space. *Computer Graphics Forum 25*, 3, 477–486.

BAXTER, W., BARLA, P., AND ANJYO, K. 2009. N-way morphing for 2D animation. *Computer Animation and Virtual Worlds (proc. CASA 2009) 20*, 79–87.

BÉNARD, P., COLE, F., GOLOVINSKIY, A., AND FINKELSTEIN, A. 2010. Self-similar texture for coherent line stylization. In *NPAR 2010: Proceedings of the 8th International Symposium on Non-photorealistic Animation and Rendering*, ACM, 91–97.

BORGEFORS, G. 1988. Hierarchical chamfer matching: A parametric edge matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence 10*, 849–865.

BOURDEV, L. 1998. *Rendering Nonphotorealiztic Strokes with Temporal and Arc-length Coherence*. Master's thesis, Brown University.

COCONU, L., DEUSSEN, O., AND HEGE, H.-C. 2006. Real-time pen-and-ink illustration of landscapes. In *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, ACM, NPAR '06, 27–35.

CURTIS, C. 1998. Loose and sketchy animation. In *Technical Sketch SIGGRAPH 1998*, ACM.

FU, H., TAI, C.-L., AND AU, O. K.-C. 2005. Morphing with laplacian coordinates and spatial-temporal texture. In *Proceedings of Pacific Graphics 2005*, 100–102.

GRABLI, S., DURAND, F., AND SILLION, F. X. 2004. Density measure for line-drawing simplification. In *Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, 309–318.

KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. Wysiwyg npr: drawing strokes directly on 3d models. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '02, 755–762.

KALNINS, R. D., DAVIDSON, P. L., MARKOSIAN, L., AND FINKELSTEIN, A. 2003. Coherent stylized silhouettes. In *Proceedings of SIGGRAPH 2003*, ACM, SIGGRAPH '03, 856–861.

LIE, D., CHEN, Q., YU, J., GU, H., TAO, D., AND SEAH, H. S. 2010. Stroke correspondence construction for vector-based 2d animation inbetweening. In *Proceedings of Computer Graphics International 2010*.

POWELL, A., AND ROSSIGNAC, J. 2008. Screwbender: Smoothing piecewise helical motions. *IEEE Comput. Graph. Appl. 28* (January), 56–63.

ROSSIGNAC, J., AND VINACUA, A. 2011. Steady affine motions and morphs. *ACM Transactions on Graphics (to appear)*.

SALISBURY, M. P., WONG, M. T., HUGHES, J. F., AND SALESIN, D. H. 1997. Orientable textures for image-based pen-and-ink illustration. In *Proceedings of the ACM SIGGRAPH Conference (SIGGRAPH-97)*, ACM Press, New York, 401–406.

SEAH, H., AND FENG, T. 2000. Computer-assisted coloring by matching line drawings. *The Visual Computer 16*, 269–304.

SHESH, A., AND CHEN, B. 2008. Efficient and dynamic simplification of line drawings. *Proceedings of Eurographics, Computer Graphics Forum 27*, 2, 537–545.

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.

VELTKAMP, R. 2001. Shape matching: similarity measures and algorithms. In *Shape Modeling and Applications, SMI 2001 Intl. Conference on.*, 188–197.

WANG, Y., XU, K., XIONG, Y., AND CHENG, Z.-Q. 2008. 2D shape deformation based on rigid square matching. *Computer Animation and Virtual Worlds 19*, 3–4, 411–420.

WHITED, B., NORIS, G., SIMMONS, M., SUMNER, R. W., GROSS, M., AND ROSSIGNAC, J. 2010. Betweenit: An interactive tool for tight inbetweening. In *Proceedings of Eurographics, Computer Graphics Forum*.

WILSON, B., AND MA, K.-L. 2004. Rendering complexity in computer-generated pen-and-ink illustrations. In *Proceedings of the 3rd International Symposium on Non-Photorealistic Animation and Rendering 2004, Annecy, France, June 7-9, 2004*, ACM, 129–137.

WINKENBACH, G., AND SALESIN, D. H. 1994. Computer-generated pen-and-ink illustration. In *Proceedings of SIGGRAPH 1994*, ACM, 91–100.

Appendix H

Cache-efficient Graph Cuts on Structured Grids

O. Jamriška, D. Sýkora, A. Hornung: Cache-efficient Graph Cuts on Structured Grids. Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'12), pp. 3673–3680, June 2012. ISBN 978-1-46731-226-4.

Cache-efficient Graph Cuts on Structured Grids

Ondřej Jamriška Daniel Sýkora
Czech Technical University in Prague, FEE
{jamriond,sykorad}@fel.cvut.cz

Alexander Hornung
Disney Research Zurich
hornung@disneyresearch.com

Abstract

Finding minimal cuts on graphs with a grid-like structure has become a core task for solving many computer vision and graphics related problems. However, computation speed and memory consumption oftentimes limit the effective use in applications requiring high resolution grids or interactive response. In particular, memory bandwidth represents one of the major bottlenecks even in today's most efficient implementations.

We propose a compact data structure with cache-efficient memory layout for the representation of graph instances that are based on regular N-D grids with topologically identical neighborhood systems. For this common class of graphs our data structure allows for 3 to 12 times higher grid resolutions and a 3- to 9-fold speedup compared to existing approaches. Our design is agnostic to the underlying algorithm, and hence orthogonal to other optimizations such as parallel and hierarchical processing. We evaluate the performance gain on a variety of typical problems including 2D/3D segmentation, colorization, and stereo. All experiments show an unconditional improvement in terms of speed and memory consumption, with graceful performance degradation for graphs with increasing topological irregularities.

1. Introduction

Minimal cuts on graphs have been studied over decades [12] and have developed into a fundamental solution to problems in various disciplines. In computer vision and graphics, applications range from segmentation [4, 24, 28], stereo and shape reconstruction [7, 17, 35], editing and synthesis [1, 23, 29, 32], fitting and registration [5, 13, 22] to pose estimation and more [20].

Typically, in those application domains the underlying graph structures can be characterized as regular N-D grids, where all nodes have topologically identical neighborhood systems, *i.e.*, each node is connected in a uniform fashion to all other nodes lying within a given radius (Fig. 1). One of the algorithms which are efficient on such graph constructions is the popular method by Boykov and Kolmogorov (BK) [6].

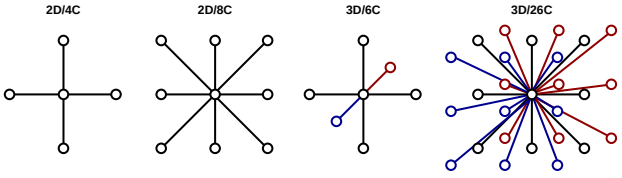

Figure 1: Typical examples of neighborhood connectivity.

Central to this method is the concept of the residual graph; based on the seminal Ford-Fulkerson's algorithm [12], the BK algorithm performs a bi-directional search for augmenting paths on the residual graph, together with a tree-reuse strategy which avoids rebuilding the search trees from scratch after each augmentation. Although its worst case complexity may be worse than alternative algorithms it was shown [6,8,11] that, for the specific graph structures encountered in vision and graphics, BK typically outperforms other approaches such as the push-relabel algorithm [15] and its more recent variants [14].

However, despite many advances, using min-cuts for processing high resolution data at sufficient speed is still a significant challenge. For example, in the previously mentioned application domains, segmenting and editing HD video in real-time or high resolution 3D model reconstruction from megapixel images remain very active fields of research. Recent research on parallelization [25] and strictly polynomial extensions [16] shows that there is still considerable need and room for improvement of the BK algorithm.

In this paper we show that significant further improvements can be achieved by optimizations of the memory bandwidth. Our main contribution is a compact data structure specifically designed for an efficient encoding of commonly used structured N-D graphs, and, in particular, their residual graphs. We exploit the typical connectivity patterns observed in the previously mentioned application domains for a highly efficient memory layout. Our method allows for 3 to 12 times higher grid resolutions and achieves a 3- to 9-fold performance gain. Our optimizations are agnostic

of the actual implementation and hence complementary to other types of min-cut algorithms. Moreover, our method does not strictly rely on the graph regularity assumptions, and shows superior performance on graphs with moderate amount of topological irregularities as well.

2. Previous work

The body of work on optimization strategies for graph cut computation comprises a variety of fundamentally different approaches, including hierarchical approximations [18, 26, 27], capacity scaling [19], strategies for reusing computations [20], or parallel processing [10, 25, 30, 31, 34]. We therefore concentrate our review on those works related to cache-efficient data structures that are closest related to our paper.

One of the first approaches specifically focusing on memory layout and cache optimizations is the method by Bader and Sachdeva [3]. Their focus is on a parallel implementation of the push-relabel method [15] for general, irregular graphs on multi-processor machines with shared memory. For each edge they store its capacity, current flow and also link to its reverse edge to reduce the number of memory accesses when the reverse edge is just read and not updated. They also use contiguous allocation of memory for parallel pairs of edges to ensure spatial locality of the data structures during the updating. Delong and Boykov [10] employed similar strategies and presented a different parallel variant of the push-relabel approach that targets the specific grid-like graph structures encountered in computer vision. A particular strength of their approach is the control over locality of memory accesses that allows for more efficient processing of very large graph structures. Liu and Sun [25] describe a parallel implementation of the BK algorithm, using a grid-like partitioning to maintain locality of computations within subgraphs. Their approach is implicitly cache-friendly and partially achieved super-linear speedups on shape fitting problems. Recently Goldberg et al. [16] proposed several low-level optimizations to avoid cache violating access. They also replace adjacency lists with arrays and reported that this optimization resulted on average in a 20% performance gain over the original BK implementation.

Complementary to these works we show that the optimization of *memory bandwidth* is a further important resource for optimization, gaining significant improvements in terms of speed *and* memory utilization.

3. Our approach

Computation of maximum flow on graphs typically requires frequent data transfers between memory and CPU causing significant latencies in the run time of max-flow algorithms. Our aim is to ease this memory bandwidth bottleneck by employing a compact graph representation with cache-friendly memory layout that exploits the regular structure of grid-like graphs.

3.1. Compact representation of the residual graph

Central to most max-flow algorithms [6, 10, 16] is the so called *residual graph* which maintains the distribution of flow during the algorithm run time. This data structure requires constant access and updates, as such its compact representation is crucial in order to achieve a significant reduction in memory bandwidth.

For the encoding of general graphs one usually requires a representation that stores the connectivity information explicitly. These are realized by pointer-heavy data structures (like the adjacency list) which reference elements from separate collections of nodes and edges. The pointers often comprise the majority of the graph's memory footprint, in particular on 64-bit CPUs where single pointer occupies eight bytes.

However, by exploiting prior knowledge of the graph structure we can eliminate the need for pointers altogether by determining connectivity information on the fly. To that end we employ a single 32-bit integer index for addressing individual nodes. Since nodes are arranged in a N-D grid with repeated neighborhood connectivity we can enumerate them in such a way that their neighbors are always at constant offsets. Thanks to this property accessing to neighboring nodes requires only picking the right offset and adding it to the current node's index.

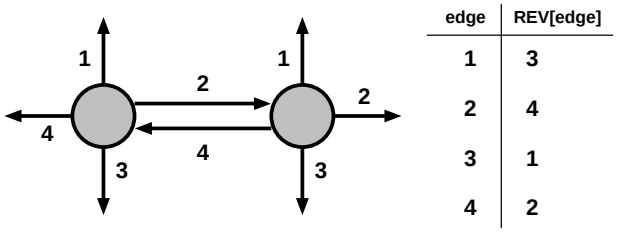

Figure 2: Enumeration of outgoing edges and reverse edge lookup using the REV table.

As the connectivity information is implicit in our representation, there is no need to maintain a separate collection of edges. Instead, we store the information associated with edges in their tail nodes, *i.e.*, each node is equipped with the data of its outgoing edges. Each edge in the residual graph is associated with residual capacity rc, which is the amount of flow that can be further pushed along the edge without exceeding its capacity. In our representation we impose fixed ordering on the node's k outgoing edges and enumerate them with index $1 \dots k$ (see Fig. 2). Each node is then associated with k-tuple $(rc_1, \dots, rc_k)$ of its outgoing edges' residual capacities. This scheme assumes that each

node has the same degree, however, it can be used even in cases when the graph is not perfectly k-connected by assigning zero residual capacity to missing edges or to edges adjacent to missing nodes.

When traversing the residual graph, max-flow algorithms often have to access residual capacities of reverse edges. We facilitate this access by referencing the opposite direction edge in the rc tuple of neighbor's outgoing edges. To determine the index of the opposite direction edge, we use a small lookup table REV (see Fig. 2). When a node lies at the grid boundary, the requested neighbor might not exist. We avoid handling this situation as a special case by extending the grid with a border layer of sentinel nodes whose outgoing edges have residual capacities all set to zero. In this way there always exists a valid neighbor, and referencing of reverse edges can never cause an out-of-bounds access.

In addition to the regular edges between neighboring nodes, each node can be potentially connected to both terminals by s/t links. Similarly to the implementation[1] of [6], we perform a trivial augmentation of the source-node-sink path during the graph initialization and only store the residual capacity $rc_{s/t}$ of the link that remained unsaturated (or set it to zero when both became saturated). In the context of push-relabel algorithms, this value corresponds to the node's initial excess/deficit.

In summary, to represent the residual graph, we maintain a collection of nodes, and for each node we only store the residual capacities of its outgoing edges and the residual capacity of the remaining unsaturated s/t link. In this collection, the nodes are addressed by a 32-bit index and indices of neighbor nodes are computed on the fly. Each node has the same number of outgoing edges, and the edges have a fixed ordering which allows quick determination of the opposite direction edge using a lookup table. This approach is similar to representations employed by [25], the implementation[2] of [10], and the GPU oriented graph encoding of [34], but crucially, we further combine it with structure splitting and blocked array reordering to achieve even better utilization of caches, as described in the following sections.

3.2. Structure splitting

Besides residual capacities max-flow algorithms usually introduce additional algorithm-specific fields. For example, BK maintains two search trees with marking heuristic [21] requiring for each node to store the tree membership tag, a reference to a parent node, an estimation of the distance to its terminal, and a time-stamp indicating when the distance was computed. However, from the optimization standpoint the key observation is that these fields are typically not accessed at the same time during the max-flow computation. For instance, algorithms based on augmenting paths alternate between search and augmentation while accessing different subsets of fields in each stage. In literature on cache optimization [9] these frequently accessed fields are typically referred to as *hot* while unused fields are *cold*. To improve cache utilization we want to preserve space for *hot* fields and avoid transfers of *cold* fields. This can be achieved by rearranging the node information using the structure-of-arrays layout (SoA) [9]. Instead of storing all nodes in a single array of structures with all fields packed together, we split the individual fields into separate arrays. With this layout the data can be naturally split into *hot* and *cold* portion.

3.3. Cache-friendly layout of arrays

We observe that even though the data access pattern of max-flow algorithms is irregular, it usually exhibits a certain amount of spatial locality. For example, when some node is visited during the computation, it is likely that other nearby nodes will be visited afterwards. The aim is to exploit this behavior to further improve utilization of caches. As transfers between cache and memory are executed in blocks (*cache lines*) with fixed size (typically 64 bytes) we can rearrange arrays into a *blocked layout*. Here all blocks have a size equal to the size of cache lines and their fields are arranged in such a way that spatially close nodes on the grid become spatially close in memory (see Fig. 3). When a field from the block is accessed for the first time, a cache miss occurs and the field is loaded into the cache along with fields of other nodes lying in the same block. Thanks to this behavior fields of nearby nodes can be quickly accessed in further steps of the algorithm.

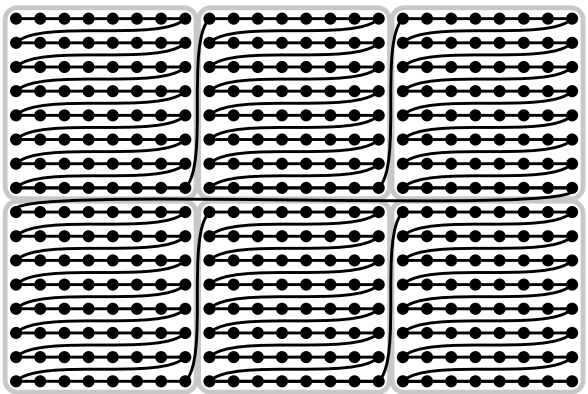

Figure 3: Blocked array layout.

The only issue associated with this rearrangement is an efficient computation of a node's neighbor indices. A naive approach would be to reconstruct the grid coordinates back from the nodes's index, translate them by the desired offset and then recompute a neighbor's index from the new coordinates. However, this computation can be very costly

[1] vision.csd.uwo.ca/code/maxflow-v3.01.zip
[2] vision.csd.uwo.ca/code/regionpushrelabel-v1.03.zip

requiring several division and modulo operations. To keep the index computation cheap, we consider only blocks with power of two extents (not necessarily equal) and with scan-line ordering of nodes inside the block as well as the blocks themselves (see Fig. 3). Grids with dimensions that are not evenly divisible by block size are padded by dummy nodes. In this setting coordinates inside the block can be easily separated from the node's index by bit-mask operations. This helps to quickly distinguish between different neighbor offsets for nodes inside a block. We demonstrate this computation for 2D 4-connected grids in Appendix, other common neighborhood systems (2D8C, 3D6C, and 3D26C) are listed in the supplementary PDF.

3.4. Implementation details

To demonstrate the effectiveness of our method, we incorporated the proposed optimizations in our own re-implementation of BK which we call OBK. The source code of OBK is available at *http://gridcut.com*. Our implementation always uses the full combination of the compact residual graph representation (CR), the structure of arrays (SoA), and the blocked array layout (BLK).

Because the growth and adoption stages of BK access residual capacities only to determine whether the edges are saturated or not, we found it beneficial to store the saturation status of a node's k outgoing edges in an additional k-bit field. Each bit in this field indicates whether the corresponding edge has non-zero residual capacity. Even though the bit field must be updated whenever the edge saturates or desaturates (in the augmentation stage), the overall benefit of fetching less data from memory more than compensates for the overhead of updates. The use of a saturation bit field is advantageous mainly on denser graphs with large capacities. We used it in all our experiments except for the 4-connected grids which are not dense enough to amortize its maintenance.

Our implementation also employs the "marking" heuristic described in [21], except for 4-connected grids where we observed an average 7% increase in algorithm run time when the heuristic was enabled. However, we still make use of the timestamping mechanism to mark the nodes whose origin was already traced to one of the terminals. In its original form the heuristic is applied in both the growth and the adoption stages of BK. We use a slightly different approach where we apply the heuristic only during adoption stage, as in our experiments the path length reduction achieved by parent reassignment in the growth stage did not account for its overhead, and on average increased the run time by 9%.

4. Results and Discussion

To verify the efficiency of our method we performed an exhaustive benchmark (see Tab. 1) which combines graphs from well-known data sets provided by The University of Western Ontario[3] and Middlebury College[4] (α-expansion [7, 33]). These graphs originate from various applications such as restoration (1-2), stereo (3-8), 2D segmentation (9-11), photomontage (12-13), 3D shape fitting (21-22), and 3D segmentation (23-48). In addition to that we also include graphs (14-20) used to solve the colorization problem [32] which is similar to 2D image segmentation [4], but with only very sparse links to the s/t terminals. Besides various applications and density of s/t links we also provide variability in grid dimension (2D/3D), number of nodes (20k-12M), neighborhood topology (4/6/26-connected), and number of bits needed to store the maximal capacity (8/16/32-bit).

To perform the comparison we downloaded the latest implementation[1] of BK and used our optimized implementation OBK. We compiled both codes in 32-bit and 64-bit modes with identical compiler settings and ran them on various CPUs (Xeon & Core i3/i7) with different cache sizes (3/6/8M). Exact configurations as well as used compilers and switches are listed in Tab. 1. For each graph instance we selected an appropriate code template having an optimal data type able to store the maximal capacity in the graph (same for BK and OBK). In contrast to previous benchmarks we decided to measure processing time not only for maximum flow computation, but also for graph initialization and output phase. This is crucial for a fair comparison with BK as in OBK we have to perform initial data rearrangement which takes some additional CPU time.

The resulting memory footprint reductions as well as speedups on selected CPUs are listed in Tab. 1. As shown in the table, the memory footprint reduction increases with the number of nodes and graph connectivity. It ranges from 2x to 6x for 32-bit and 3x to 12x for 64-bit modes. The average speedup over all instances and CPUs is 2.7x for 32-bit and 4.4x for 64-bit modes. However, as is visible from the table and graphs in Fig. 4, there is a notable variability. Despite of this fact, we found a few interesting trends which are worth mentioning.

First we observed that the main performance gain is always caused by the compact data representation (CR on the bottom graph in Fig. 4). The importance of the lower memory consumption is also apparent when comparing 32-bit with 64-bit modes (top and middle graph in Fig. 4). Today 64-bit mode is preferred in practice as it allows for a significantly larger addressing space. On the other hand it requires 64-bits to store all the pointers used in the BK method, therefore a larger amount of data needs to be stored/transferred compared to OBK where only a few array pointers are needed and all nodes are indexed by 32-bit integers. Another important factor which influences speedup is the size of frequently accessed area in the memory. The

[3] http://vision.csd.uwo.ca/maxflow-data/
[4] http://vision.middlebury.edu/MRF/

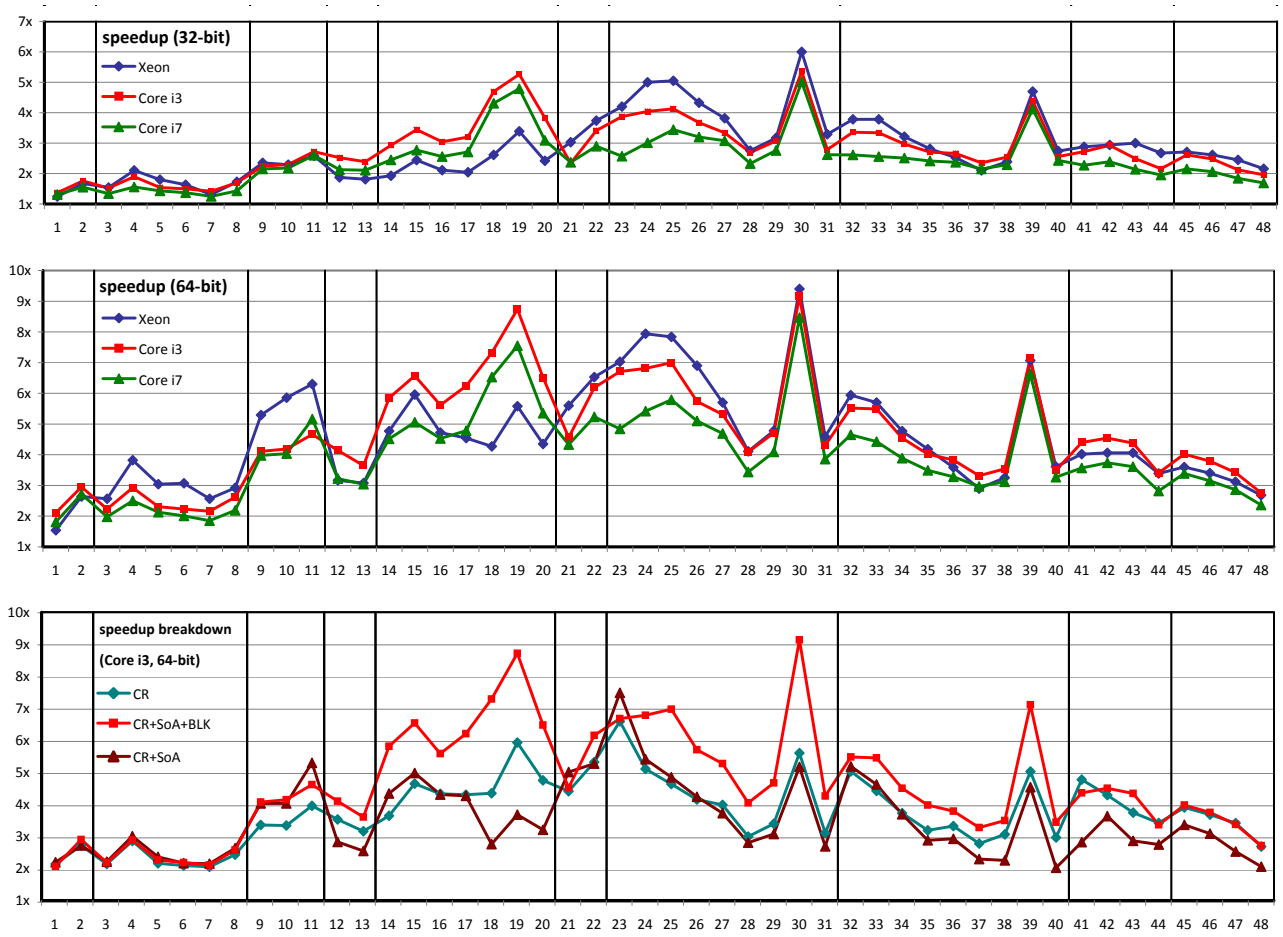

Figure 4: Speedups of OBK with respect to BK in 32-bit (top) & 64-bit (middle) modes and speedup breakdown (bottom).

performance gain is only moderate when such area is so small that it can be fully fitted into the cache (see speedups for small instances 1-8 on all CPUs). Due to same reason cache-efficient memory layout (CR+SoA+BLK, bottom in Fig. 4) brings notable improvements only for larger instances (9-48). It is also interesting to note that the structure splitting technique alone (CR+SoA) typically does not improve the final performance over CR. Only in combination with the blocked layout (BLK) we see notable speedups. Another question is whether cache size influences the speedup. There is not a straightforward dependance, however, from the graphs displaying the difference between Core i3 (3M) and Core i7 (8M) (see Fig. 4) one may deduce that our approach brings better speedups on CPUs with smaller caches.

As OBK brings significant performance gain for a variety of vision and graphics problems we decided to compare it directly with existing state-of-the-art max-flow/min-cut methods. We selected two recent algorithms: (1) voronoi-based pre-flow push (VPP) [2] and (2) incremental breadth-first search (IBFS) [16] which to our best knowledge report best speedups over the original BK. Based on a DLL provided by authors of VPP we performed our own comparison with BK and found that the results shown in [2] were affected by three important factors. Firstly the authors did not set the graph for the BK algorithm correctly and introduced many redundant edges with zero capacities. Secondly, they did not use the latest version of BK, and also did not use full compiler optimizations (`/Ox` in Visual Studio). These points are crucial for the peak performance of BK. With all these issues fixed, VPP performed only comparable or even worse than BK. The performace comparison of VPP with respect to OBK is provided in the supplementary material. Regarding IBFS we used the publicly available implementation[5] and found that on all instances OBK achieves notably better absolute times mainly due to inefficient preprocessing phase used in IBFS. The absolute times were often better even when measuring only the computation of maximum flow (see supplementary material for detailed evaluation).

[5] `http://www.cs.tau.ac.il/~sagihed/ibfs/`

Intel Xeon E5440 @ 2.83 GHz 6M cache | gcc 4.4.0, -O3 -march=native -mtune=generic -DNDEBUG
Intel Core i3 M370 @ 2.40 GHz 3M cache | gcc 4.5.2 (32b) / 4.7.0 (64b), -O3 -march=native -mtune=generic -DNDEBUG
Intel Core i7 950 @ 3.07 GHz 8M cache | gcc 4.5.2 (32b) / 4.7.0 (64b), -O3 -march=native -mtune=generic -DNDEBUG

					Memory		Xeon		Core i3		Core i7	
					32-bit	64-bit	32-bit	64-bit	32-bit	64-bit	32-bit	64-bit
	instance	topo	# nodes	captype	reduction	reduction	speedup	speedup	speedup	speedup	speedup	speedup
1	/mid/denoise/penguin	2D/4C	22k	16-bit	1.71	3.11	1.24	1.54	1.37	2.11	1.31	1.81
2	/mid/denoise/house	2D/4C	66k	32-bit	1.49	2.72	1.67	2.64	1.76	2.95	1.55	2.72
3	/mid/stereo/tsukuba	2D/4C	111k	8-bit	1.91	3.64	1.54	2.57	1.52	2.23	1.34	1.98
4	/uwo/stereo/BVZ-tsukuba	2D/4C	111k	8-bit	1.91	3.64	2.10	3.82	1.90	2.93	1.56	2.50
5	/uwo/stereo/BVZ-sawtooth	2D/4C	165k	8-bit	1.95	3.73	1.80	3.04	1.54	2.31	1.43	2.13
6	/uwo/stereo/BVZ-venus	2D/4C	166k	8-bit	1.93	3.68	1.63	3.07	1.50	2.23	1.37	2.01
7	/mid/stereo/venus	2D/4C	166k	16-bit	1.85	3.37	1.31	2.57	1.42	2.16	1.25	1.85
8	/mid/stereo/teddy	2D/4C	169k	8-bit	1.97	3.76	1.73	2.92	1.70	2.62	1.43	2.19
9	/mid/segment/flower	2D/4C	270k	16-bit	1.87	3.41	2.35	5.29	2.24	4.11	2.15	3.98
10	/mid/segment/person	2D/4C	270k	16-bit	1.87	3.41	2.29	5.86	2.30	4.19	2.18	4.04
11	/mid/segment/sponge	2D/4C	307k	16-bit	1.86	3.40	2.58	6.30	2.72	4.66	2.60	5.16
12	/mid/photomontage/family	2D/4C	426k	32-bit	1.56	2.86	1.87	3.17	2.52	4.14	2.13	3.22
13	/mid/photomontage/panorama	2D/4C	514k	32-bit	1.55	2.83	1.81	3.08	2.39	3.65	2.11	3.04
14	/ctu/lazybrush/footman	2D/4C	593k	16-bit	1.89	3.44	1.93	4.77	2.93	5.85	2.45	4.52
15	/ctu/lazybrush/hmdman	2D/4C	593k	16-bit	1.89	3.44	2.44	5.96	3.44	6.57	2.77	5.06
16	/ctu/lazybrush/mangadinner	2D/4C	593k	16-bit	1.89	3.44	2.11	4.72	3.04	5.62	2.56	4.53
17	/ctu/lazybrush/mangagirl	2D/4C	593k	16-bit	1.89	3.44	2.04	4.54	3.20	6.24	2.71	4.78
18	/ctu/lazybrush/elephant	2D/4C	2370k	16-bit	1.90	3.48	2.61	4.27	4.69	7.32	4.31	6.53
19	/ctu/lazybrush/bird	2D/4C	2372k	16-bit	1.91	3.48	3.39	5.58	5.26	8.74	4.79	7.55
20	/ctu/lazybrush/doctor	2D/4C	2373k	16-bit	1.91	3.48	2.42	4.35	3.83	6.51	3.09	5.35
21	/uwo/shapefit/LB07-bunny-sml	3D/6C	806k	8-bit	2.51	4.86	3.03	5.60	2.34	4.56	2.37	4.33
22	/uwo/shapefit/LB07-bunny-med	3D/6C	6311k	8-bit	2.70	5.21	3.74	6.53	3.41	6.19	2.90	5.23
23	/uwo/seg3d/bone_subxyz_subxy.n6c10	3D/6C	246k	8-bit	2.31	4.32	4.20	7.03	3.87	6.71	2.57	4.84
24	/uwo/seg3d/bone_subxyz_subx.n6c10	3D/6C	492k	8-bit	2.39	4.46	5.00	7.94	4.04	6.81	3.01	5.42
25	/uwo/seg3d/bone_subxyz.n6c10	3D/6C	983k	8-bit	2.46	4.61	5.05	7.84	4.13	7.00	3.44	5.79
26	/uwo/seg3d/bone_subxy.n6c10	3D/6C	1950k	8-bit	2.53	4.73	4.33	6.90	3.67	5.74	3.20	5.10
27	/uwo/seg3d/bone_subx.n6c10	3D/6C	3899k	8-bit	2.57	4.80	3.82	5.70	3.34	5.31	3.08	4.69
28	/uwo/seg3d/liver.n6c10	3D/6C	4162k	8-bit	2.67	4.99	2.75	4.11	2.69	4.09	2.33	3.44
29	/uwo/seg3d/babyface.n6c10	3D/6C	5063k	8-bit	2.66	4.98	3.17	4.78	3.07	4.71	2.76	4.09
30	/uwo/seg3d/bone.n6c10	3D/6C	7799k	8-bit	2.61	4.88	6.00	9.40	5.36	9.16	5.01	8.47
31	/uwo/seg3d/adhead.n6c10	3D/6C	12583k	8-bit	2.67	4.99	3.29	4.59	2.79	4.30	2.62	3.86
32	/uwo/seg3d/bone_subxyz_subxy.n6c100	3D/6C	492k	8-bit	2.39	4.46	3.78	5.94	3.36	5.52	2.61	4.65
33	/uwo/seg3d/bone_subxyz_subx.n6c100	3D/6C	492k	8-bit	2.39	4.46	3.78	5.70	3.34	5.49	2.56	4.43
34	/uwo/seg3d/bone_subxyz.n6c100	3D/6C	983k	8-bit	2.46	4.61	3.21	4.77	2.97	4.54	2.51	3.89
35	/uwo/seg3d/bone_subxy.n6c100	3D/6C	1950k	8-bit	2.53	4.73	2.81	4.18	2.70	4.02	2.41	3.49
36	/uwo/seg3d/bone_subx.n6c100	3D/6C	3899k	8-bit	2.57	4.80	2.52	3.59	2.66	3.83	2.37	3.28
37	/uwo/seg3d/liver.n6c100	3D/6C	4162k	8-bit	2.67	4.99	2.10	2.90	2.35	3.32	2.15	2.97
38	/uwo/seg3d/babyface.n6c100	3D/6C	5063k	8-bit	2.66	4.98	2.39	3.25	2.54	3.54	2.29	3.12
39	/uwo/seg3d/bone.n6c100	3D/6C	7799k	8-bit	2.61	4.88	4.70	7.07	4.39	7.14	4.12	6.63
40	/uwo/seg3d/adhead.n6c100	3D/6C	12583k	8-bit	2.67	4.99	2.74	3.61	2.56	3.49	2.43	3.27
41	/uwo/seg3d/bone_subxyz_subxy.n26c10	3D/26C	246k	8-bit	5.42	10.65	2.88	4.02	2.72	4.40	2.27	3.57
42	/uwo/seg3d/bone_subxyz_subx.n26c10	3D/26C	492k	8-bit	5.59	10.97	2.94	4.06	2.93	4.54	2.39	3.74
43	/uwo/seg3d/bone_subxyz.n26c10	3D/26C	983k	8-bit	5.76	11.30	3.00	4.06	2.49	4.38	2.14	3.61
44	/uwo/seg3d/bone_subxy.n26c10	3D/26C	1950k	8-bit	5.89	11.57	2.67	3.39	2.16	3.41	1.95	2.82
45	/uwo/seg3d/bone_subxyz_subxy.n26c100	3D/26C	246k	8-bit	5.42	10.65	2.71	3.60	2.61	4.02	2.15	3.39
46	/uwo/seg3d/bone_subxyz_subx.n26c100	3D/26C	492k	8-bit	5.59	10.97	2.61	3.40	2.48	3.79	2.06	3.15
47	/uwo/seg3d/bone_subxyz.n26c100	3D/26C	983k	8-bit	5.76	11.30	2.45	3.12	2.12	3.42	1.84	2.86
48	/uwo/seg3d/bone_subxy.n26c100	3D/26C	1950k	8-bit	5.89	11.57	2.16	2.68	1.96	2.76	1.69	2.36
	average:				**2.77**	**5.26**	**2.81**	**4.58**	**2.84**	**4.65**	**2.47**	**3.99**

Table 1: Detailed performance evaluation for different problem instances.

This leads us to the conclusion that for most vision and graphics problems OBK outperforms current state-of-the-art and delivers best single core performance. However, despite this fact we believe that by incorporating our optimizations into IBFS and making the preprocessing phase more efficient one can possibly beat OBK on selected instances. The key limitation of our approach is that it is tailored to structured grid-like graphs with nodes having identical neighborhood connectivity. However, since we can always introduce redundant edges with zero capacities in places where the edge is missing in the original graph we can handle even more general topologies. To measure the efficiency of our approach in these irregular cases we conducted an experiment where we randomly removed a percentage of edges from the original graphs (22 & 30) having full 6-connected neighborhoods. We then build a new irregular graph on which we run BK as well as OBK with redundant edges, and measured the performance gains. Fig. 5 shows a graceful performance degradation with increasing topological irregularities. However, further investigation is needed to verify that behavior in more practical scenarios.

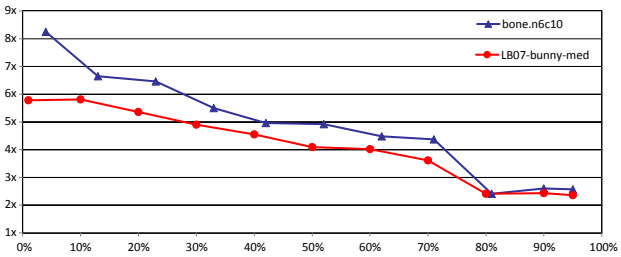

Figure 5: Speedup of OBK with respect to the percentage of removed edges.

5. Conclusions

We have presented a set of cache-efficient optimizations that enable a notable reduction of memory bandwidth when computing graph cuts on structured N-D grids. Our experimental evaluation shows that the proposed optimizations achieve a significant performance gain as well as a reduction of the memory footprint, which renders our method particularly useful for interactive applications and for large problems. Finally, the presented techniques are complementary to other optimizations and can be easily plugged into other graph cut algorithms. Our implementation is publicly available at *http://gridcut.com*.

6. Acknowledgements

We would like to thank all anonymous reviewers for their helpful comments. This work has been supported by the Marie Curie action ERG, No. PERG07-GA-2010-268216 and partially by the Grant Agency of the Czech Technical University in Prague, grant No. SGS10/289/OHK3/3T/13.

References

[1] A. Agarwala, M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin, and M. Cohen. Interactive digital photomontage. *ACM Transactions on Graphics*, 23(3):294–302, 2004. 1

[2] C. Arora, S. Banerjee, P. Kalra, and S. N. Maheshwari. An efficient graph cut algorithm for computer vision problems. In *Proceedings of European Conference on Computer Vision*, pages 552–565, 2010. 5

[3] D. A. Bader and V. Sachdeva. A cache-aware parallel implementation of the push-relabel network flow algorithm and experimental evaluation of the gap relabeling heuristic. In *Proceedings of International Conference on Parallel and Distributed Computing Systems*, pages 41–48, 2005. 2

[4] Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. In *Proceedings of International Conference on Computer Vision*, volume 1, pages 105–112, 2001. 1, 4

[5] Y. Boykov and V. Kolmogorov. Computing geodesics and minimal surfaces via graph cuts. In *Proceedings of International Conference on Computer Vision*, pages 26–33, 2003. 1

[6] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004. 1, 2, 3

[7] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001. 1, 4

[8] B. G. Chandran and D. S. Hochbaum. A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem. *Operations Research*, 57(2):358–376, 2009. 1

[9] T. M. Chilimbi, M. D. Hill, and J. R. Larus. Making pointer-based data structures cache conscious. *IEEE Computer*, 33(12):67–74, 2000. 3

[10] A. Delong and Y. Boykov. A scalable graph-cut algorithm for N-D grids. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2008. 2, 3

[11] B. Fishbain, D. S. Hochbaum, and S. Mueller. Competitive analysis of minimum-cut maximum flow algorithms in vision problems. *The Computing Research Repository*, 2010. 1

[12] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. 1

[13] B. Glocker, N. Komodakis, G. Tziritas, N. Navab, and N. Paragios. Dense image registration through MRFs and efficient linear programming. *Medical Image Analysis*, 12(6):731–741, 2008. 1

[14] A. Goldberg. Two-level push-relabel algorithm for the maximum flow problem. In *Proceedings of International Conference on Algorithmic Aspects in Information and Management*, pages 212–225, 2009. 1

[15] A. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921–940, 1988. 1, 2

[16] A. V. Goldberg, S. Hed, H. Kaplan, R. E. Tarjan, and R. F. F. Werneck. Maximum flows by incremental breadth-first search. In *Proceedings of Annual European Symposium on Algorithms*, pages 457–468, 2011. 1, 2, 5

[17] A. Hornung and L. Kobbelt. Robust reconstruction of watertight 3d models from non-uniformly sampled point clouds without normal information. In *Symposium on Geometry Processing*, pages 41–50, 2006. 1

[18] O. Juan and Y. Y. Boykov. Active graph cuts. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages I: 1023–1029, 2006. 2

[19] O. Juan and Y. Y. Boykov. Capacity scaling for graph cuts in vision. In *Proceedings of IEEE International Conference on Computer Vision*, 2007. 2

[20] P. Kohli and P. H. S. Torr. Dynamic graph cuts and their applications in computer vision. In *Computer Vision: Detection, Recognition and Reconstruction*, pages 51–108. Springer, 2010. 1, 2

[21] V. Kolmogorov. *Graph Based Algorithms for Scene Reconstruction from Two or More Views*. PhD thesis, Cornell University, 2003. 3, 4

[22] V. Lempitsky and Y. Boykov. Global optimization for shape fitting. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2007. 1

[23] V. Lempitsky and D. Ivanov. Seamless mosaicing of image-based texture maps. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2007. 1

[24] Y. Li, J. Sun, and H.-Y. Shum. Video object cut and paste. *ACM Transactions on Graphics*, 24(3):595–600, 2005. 1

[25] J. Liu and J. Sun. Parallel graph-cuts by adaptive bottom-up merging. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2181–2188, 2010. 1, 2, 3

[26] J. Liu, J. Sun, and H.-Y. Shum. Paint selection. *ACM Transactions on Graphics*, 28(3):69, 2009. 2

[27] H. Lombaert, Y. Y. Sun, L. Grady, and C. Y. Xu. A multilevel banded graph cuts method for fast image segmentation. In *Proceedings of IEEE International Conference on Computer Vision*, pages I: 259–265, 2005. 2

[28] C. Rother, V. Kolmogorov, and A. Blake. Grabcut - Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics*, 23(3):309–314, 2004. 1

[29] M. Rubinstein, A. Shamir, and S. Avidan. Improved seam carving for video retargeting. *ACM Transactions on Graphics*, 27(3):16, 2008. 1

[30] A. Shekhovtsov and V. Hlavac. A distributed min-cut/maxflow algorithm combining path augmentation and push-relabel. In *Proceedings of the International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition*, page 14, 2011. 2

[31] P. Strandmark and F. Kahl. Parallel and distributed graph cuts by dual decomposition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2085–2092, 2010. 2

[32] D. Sýkora, J. Dingliana, and S. Collins. LazyBrush: Flexible painting tool for hand-drawn cartoons. *Computer Graphics Forum*, 28(2):599–608, 2009. 1, 4

[33] R. S. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 30(6):1068–1080, 2008. 4

[34] V. Vineet and P. J. Narayanan. CUDA cuts: Fast graph cuts on the GPU. In *Proceedings of Workshop on Visual Computer Vision on GPUs*, 2008. 2, 3

[35] G. Vogiatzis, P. H. S. Torr, and R. Cipolla. Multi-view stereo via volumetric graph-cuts. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 391–398, 2005. 1

Appendix

In this section we present the details of node enumeration and neighbor index computation on the example of 4-connected 2D grid (see supplementary material for other neighborhood systems). In the blocked layout with 8×8 blocks, the index u of a node with grid coordinates x and y is evaluated using the formula

$$u = (x\&7) + ((y\&7) \ll 3) + ((x\&\sim 7) \ll 3) + W \cdot (y\&\sim 7),$$

where W is width of the padded grid. To compute the indices of a node's neighbors, we make use of the fact that in a 4-connected grid the index of each neighbor can differ from the current node's index by only two possible additive constants, depending on whether the two nodes are in the same or different blocks. In the 8×8 blocked layout, we can decide between the two situations by examining the six least significant bits of the node indices. The bits have a specific pattern on the block's boundary. For instance, the lower three bits are always 000 on the left boundary and the higher three bits are always 111 on the bottom boundary. To compute a neighbor's index, we first check whether the node lies on the block's boundary by comparing the relevant bits and pick one of the constants accordingly. To compute the left, right, top and bottom neighbor of a node with index u, we use the functions

$$\begin{aligned}
left(u) &= u \ \&\ 000111_b \ ?\ u-1 \ :\ u-57 \\
right(u) &= (\sim u) \ \&\ 000111_b \ ?\ u+1 \ :\ u+57 \\
top(u) &= u \ \&\ 111000_b \ ?\ u-8 \ :\ u-Y_{ofs} \\
bottom(u) &= (\sim u) \ \&\ 111000_b \ ?\ u+8 \ :\ u+Y_{ofs},
\end{aligned}$$

where $Y_{ofs} = 8 \cdot (W - 8 + 1)$. The use of the ternary operator ? hints the compiler to generate conditional moves instead of branches. This is beneficial as it avoids latencies incurred by branch mispredictions.

Appendix I

Painting by Feature: Texture Boundaries for Example-based Image Creation

M. Lukáč, J. Fišer, J.-C. Bazin, O. Jamriška, A. Sorkine-Hornung, D. Sýkora: Painting by Feature: Texture Boundaries for Example-based Image Creation. ACM Transactions on Graphics, vol. 32, no. 4, art. no. 116, July 2013. ISSN 0730-0301. **IF=3.361**

Painting by Feature: Texture Boundaries for Example-based Image Creation

Michal Lukáč[1]* Jakub Fišer[1] Jean-Charles Bazin[2] Ondřej Jamriška[1] Alexander Sorkine-Hornung[3] Daniel Sýkora[1]
[1]CTU in Prague, FEE [2]ETH Zurich [3]Disney Research Zurich

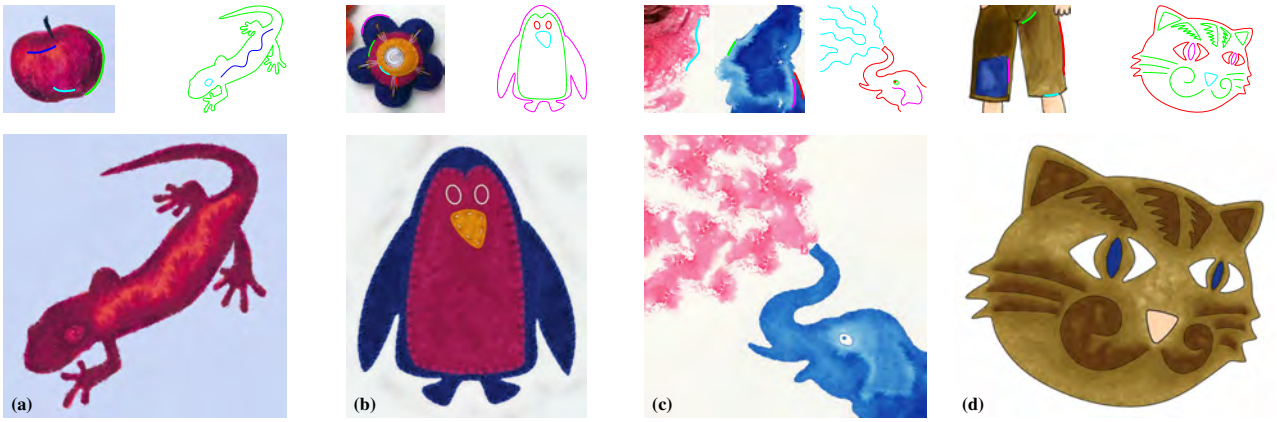

Figure 1: *Representative results generated by our proposed example-based painting framework. The user selects line features in a reference image (colored lines in the top left images, see also area features in the supplementary material) which are then immediately available as brushes for applications such as real-time painting or vector image stylization. The respective top right images depict the user's painted strokes in order to create the images in the bottom row. These demonstrate various use cases of our method: (a) complex paintings from a few input strokes, (b) painting detailed, structured boundaries, (c) watercolor, and (d) diffusion curve effects. Source credits: (a)* Sarah G *via flickr,* fzap *via OpenClipArt; (b)* Pavla Sýkorová*, clipartsy; (c)* bittbox *via flickr,* papapishu *via OpenClipArt; (d)* Anifilm*,* Pavla Sýkorová

Abstract

In this paper we propose a reinterpretation of the *brush* and the *fill* tools for digital image painting. The core idea is to provide an intuitive approach that allows users to paint in the visual style of arbitrary example images. Rather than a static library of colors, brushes, or fill patterns, we offer users entire images as their palette, from which they can select arbitrary contours or textures as their brush or fill tool in their own creations. Compared to previous example-based techniques related to the painting-by-numbers paradigm we propose a new strategy where users can generate salient texture boundaries by our randomized graph-traversal algorithm and apply a content-aware fill to transfer textures into the delimited regions. This workflow allows users of our system to intuitively create visually appealing images that better preserve the visual richness and fluidity of arbitrary example images. We demonstrate the potential of our approach in various applications including interactive image creation, editing and vector image stylization.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation I.3.4 [Computer Graphics]: Graphics Utilities—Paint systems I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

Keywords: example-based painting, stroke synthesis, painting-by-numbers, vector image stylization, non-photorealistic rendering

Links: DL PDF WEB

*e-mail:michal.lukac@fel.cvut.cz

1 Introduction

Strokes and lines are the most elementary primitives in painting, both digital and physical. The concept of drawing shapes by first sketching and developing object outlines seems to be so natural and intuitive that small children employ it just as artists and designers. Any existing image editor implements the basic *pencil* and/or *brush* tools, and various attempts have been made to enhance their expressive power, such as the calligraphic brush or the textured stroke. Similarly, vector-based image editors use *paths* as their most fundamental primitive for defining object boundaries.

Despite their importance for sketching the essential structures in an image, basic brush- or path-based tools are generally less suitable for creating a clean, richly textured image such as the ones shown in Figure 1. Researchers have long been aware of this gap between a sketch and production quality artwork, and proposed various ideas for converting simple sketches into richer and more expressive images [Ashikhmin 2001; Hertzmann et al. 2001; Ritter et al. 2006; Orzan et al. 2008].

Unfortunately, existing approaches often face difficulties when synthesizing images with significant structure, as the underlying algorithms generally focus on synthesizing 2D textured areas, without explicitly enforcing consistency to the boundaries of a shape. Due to the sensitivity of human vision to the contours of a shape [De-

Figure 2: *Comparison of results from different approaches given the input picture (a) as the reference image or* feature palette. *Result of (b) Image Analogies [Hertzmann et al. 2001], (c) Painting with Texture [Ritter et al. 2006], (d) Synthesizing Natural Textures [Ashikhmin 2001], (e) our approach. Source credit:* Wednesday Elf – Mountainside Crochet *via flickr*

Carlo et al. 2003], such artifacts become immediately apparent (see comparison in Figure 2).

This paper addresses these issues by modeling an image as a set of two classes of features. The first class corresponds to 1D *line features*, such as important contours, boundaries of textured regions, or salient strokes which are used to define the basic structure of the image. The second class corresponds to 2D *area features*, which represent regions filled with a nearly-stationary texture. For defining the visual style of an image, we introduce the metaphor of a *feature palette*, which is simply one or more example images of a desired visual style, in which the user selects line and area features with which to paint.

Our main technical contribution is a novel algorithm for interactive synthesis of line features (*brush* tool) which utilizes a randomized graph traversal mechanism with multi-level blending to seamlessly synthesize long, non-repetitive, textured strokes sampled from shorter exemplars located in the input image. For the transfer of area features (*fill* tool) we use a state-of-the-art texture synthesis algorithm [Wexler et al. 2007] which avoids visible discontinuities between painted line features and textured areas while preserving the richness of the original exemplar. Both tools provide immediate real-time feedback, making their use as intuitive and easy as an ordinary brush or fill tool. Creating complex, visually appealing drawings with our system requires similar effort as creating a simple contour sketch in standard drawing systems.

2 Related Work

One of the first works for example-based visual style transfer between images is the Image Analogies approach by Hertzmann et al. [2001]. They discuss the possibility of example-based painting using the texture-by-numbers paradigm where an input image is first segmented into multiple regions denoted by color labels, and then these labels are painted to form a new segmentation from which an output image is generated using their texture synthesis algorithm. While this approach provides a high degree of freedom in defining the output result, it is not clear how to support the concept of 1D structure elements such as contours. Moreover, the algorithm complexity prohibits an interactive implementation. A representative result is shown in Figure 2b.

Ritter et al. [2006] further extended Hertzmann et al.'s framework and created a nearly interactive texture-by-numbers painting program where boundary pixels are refined automatically thanks to an additional energy term which takes similarity of source and target boundaries into account. However, a key limitation is the lack of user control in the boundary forming process and the technique is inherently 2D, i.e., it does not preserve 1D structure of more complex boundaries. Although pixels are transferred from locations with a similar boundary shape, there is no guarantee that they will produce a 1D, visually continuous strip since the source pixels can be located on different parts of the boundary. Hence, the method produces convincing results only for textures which have a nearly constant cross-section profile along the boundary, producing artifacts otherwise (see Figures 2c and 9).

Similar considerations apply to other types of texture synthesis algorithms [Ashikhmin 2001; Efros and Freeman 2001; Kwatra et al. 2003] which partially also provide support for user constraints [Kwatra et al. 2005; Lefebvre and Hoppe 2005], or to matching based image manipulation and morphing techniques [Barnes et al. 2009; Shechtman et al. 2010; Darabi et al. 2012; Yücer et al. 2012]. All these methods provide very flexible and powerful tools for filling or transforming image areas with plausible and visually rich textures, but at the same time they are inherently 2D without support for user-controlled, real-time 1D structure transfer from a reference image. See Figure 2d for an exemplary result with the method of Ashikhmin et al. [2001]. The synthesis step of the above result took 130 seconds, whereas our approach provides instantaneous feedback.

Recently, other example-based content generation techniques have been proposed, which create new images from a user-provided set of examples [Risser et al. 2010; Assa and Cohen-Or 2012]. However, these techniques are non-interactive, global approaches which specialize in rapid generation of a large number of variations of the input image. Currently, the only way to influence this process is by providing a different choice of input images.

Sun et al. [2005] demonstrated the benefit of giving the user control over structural features in the context of image inpainting. They apply a constrained patch-based synthesis on the user-provided line features and then perform inpainting on the remaining areas that is consistent with the previously synthesized structures. A restriction of this approach is, however, that the employed energy minimization provides no guarantees that the global scale visual appearance of the synthesized line feature is consistent with its appearance in the respective source image. In texture synthesis, this problem is generally avoided by multi-scale synthesis, but this is not feasible for linear features, as they eventually disappear on lower resolutions. Using basic energy optimization without a sufficiently expressive model of a feature's global scale, artifacts are often perceptible as a periodic repetition of a pattern along the output path. On a related note, another example for the benefits of contour-based editing is the work of Fang et al. [2007] for detail preserving shape deformation in images.

For vector graphics editing Orzan et al. [2008] presented a technique for creating smooth color transitions between spline paths using Poisson interpolation. Due to the purely vector-based representation this approach is not suitable for style and texture transfer

between images. McCann and Pollard [2008] broke new ground by introducing a set of gradient-painting tools, designed to be fully interactive and directly controlled by the user. Notably, they introduce an edge brush tool which allows the user to select a path in a source image and map it to a path in the result using gradient-domain blending. Their approach, however, targets image editing, and their simple copying procedure offers no variation during feature synthesis, resulting in clearly visible periodicity when the output path is much longer than the source path of the respective feature. In our approach, we utilize a workflow similar to theirs for image creation. However, we introduce a generative line feature model to enable an indefinite extension of a source path without such artifacts.

Related to our algorithm for feature transfer is the work on video textures [Schödl et al. 2000]. They developed a feature model capable of extending a video in the temporal domain, where the frames are represented as graph nodes and the edge weights represent a measure of similarity between two frames. Thanks to this representation a permutation of video frames can be expressed as a low-cost traversal through the graph. Their approach served as an inspiration for our generative model for line feature synthesis. However, similar to the work of Sun et al. [2005] and McCann and Pollard [2008], a direct application of their loop-based synthesis algorithm would result in obvious periodic artifacts. Part of our contribution is a synthesis algorithm that resolves these issues.

3 Our Approach

As briefly outlined in the introduction, our proposed approach is based on three central concepts. The first two of them are the two different types of features and their corresponding tools:

- A **Line Feature** is a one-dimensional feature representing an arbitrary curvilinear structure, such as an edge or contour in an image. It typically represents a boundary between two textured regions, but can also represent other structures such as open curves. The corresponding tool for painting line features is the *Brush* tool.
- An **Area Feature** is a two-dimensional image region which has the semantics of a stationary texture rather than that of a one-dimensional structural element. It typically represents the interior of a region, but can also be a changing gradient or any other area sample. Its corresponding tool is the *Fill* tool.

Both tools consist of two parts, namely a *selection* component which allows the user to define a desired line or area feature, and a *synthesis* component which efficiently renders the corresponding output according to the user's drawing.

The third central concept is the **Feature Palette** and it concerns the feature selection process. Rather than requiring the user to define features in a cumbersome manual way, the basic idea is to regard an arbitrary set of input images as a palette for painting. The user may simply pick one or more input images that reflect a desired visual style, and our algorithm provides the selection tools to intuitively and efficiently define line features as well as area features. Hence, any image can be used as a palette for defining features.

These concepts are fundamentally different from merely building a static database of strokes and fill textures, as commonly done in vector image editors. In our process, the reference image(s) used as the feature palette permit effortless definition of a dynamically changing library of brushes and textures on-the-fly. This facilitates the replication of the desired visual characteristics of the reference images in one's own creation. The user directly benefits from the rich visual details that are typically present in paintings, drawings, or photographs. Just as a painter can efficiently mix colors on a physical color palette, our concept allows the user to intuitively and interactively modify and refine a feature with instant feedback while painting.

In the following section we describe how the respective selection and synthesis components of both the brush and the fill tools for line and area features are implemented.

3.1 Brush

Given a feature palette in the form of one or more input images, selecting a line feature such as an object contour requires the user to simply draw a path (the width of which can be manually adjusted) approximately along the desired feature. Since precise drawing of such a path would be tedious, our algorithm supports an assisted selection that refines the user's approximate path and aligns the selection closely to the actual line feature in the image. We found that a relatively simple gradient-based approach is reasonable in order to provide an active support for the user at a sufficient accuracy for our algorithm, hence we based our path selection on an Active Contours approach [Kass et al. 1988]. A considerable advantage of this approach is that it runs in real-time and gives an instant result, which is an important requirement for a responsive and intuitive user interface.

Once the user has defined a path over a line feature we require a real-time algorithm that synthesizes a corresponding line feature in the output image as the user paints. In the field of texture synthesis it has long been understood that synthesizing a larger texture simply by tiling a smaller example texture produces sub-optimal results. Thus, in order to avoid periodicity, some texture synthesis techniques [Lefebvre and Hoppe 2005] deliberately introduce a degree of randomness instead of tiling the texture. Likewise, our goal is to reproduce the local visual characteristics, i.e., the *look and feel* of a line feature, without introducing noticeable artifacts on a larger scale. Approaches such as [McCann and Pollard 2008] exhibit periodicity and cannot explicitly avoid visible discontinuities when stroke endpoints meet. We present a new algorithm for randomized line feature synthesis based on a graph model of the input feature to resolve such issues.

As line features such as object contours are one-dimensional and oriented, we found the graph formalism introduced by Schödl et al. [2000] for manipulating video over time to be an excellent basis for feature synthesis. We sample an input path at equidistant points and consider the direction of the feature to be equal to the direction of the user's stroke. Treating these samples as graph nodes and using the direction of the feature for ordering, we define a complete oriented graph, where the weight $w(i, j)$ of an oriented edge between nodes i and j is given by a dissimilarity measure. Specifically, we define $w(i, j) = SSD(p(i), p(j - 1))$, where $p(i)$ is a square image patch centered on the i^{th} sample and aligned with the path direction, and SSD denotes the sum of squared differences between patches (see Figure 4). We use $SSD(p(i), p(j - 1))$ rather than $SSD(p(i), p(j))$ because traversing to a consecutive sample on the original feature should be free, and thus $w(i, i + 1)$ should be equal to zero. The size of the patch is a user-configurable parameter which is intuitively equivalent to brush width and can be adjusted interactively. A walk in such a graph represents a permutation (with repetition) of input samples, which, if transferred to equidistant samples on a different path and rendered, would yield a variation of the source feature. The total cost of this walk is then representative for the amount of discontinuities in the output.

Given this representation the main concern is how exactly to generate a walk through this graph to satisfy all of our requirements and

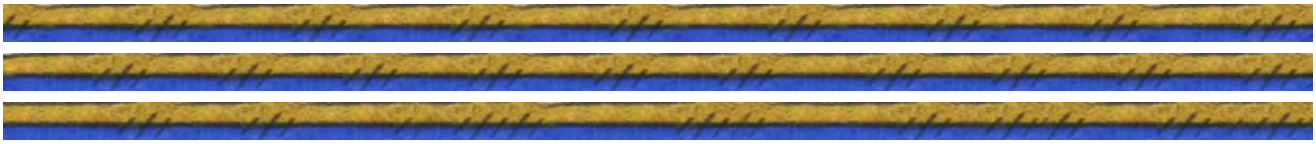

Figure 3: *A comparison of different walk synthesis approaches. Top to bottom: looping, dynamic programming and our randomized graph traversal. Note that finding the cheapest walk of a given length by dynamic programming provides the optimal result with respect to discontinuity cost, but it does so by finding the cheapest loop in the graph and thus introduces periodicity. A randomized approach, though not optimal with respect to the global cost, provides a more natural, varied look without noticeable visual discontinuities.*

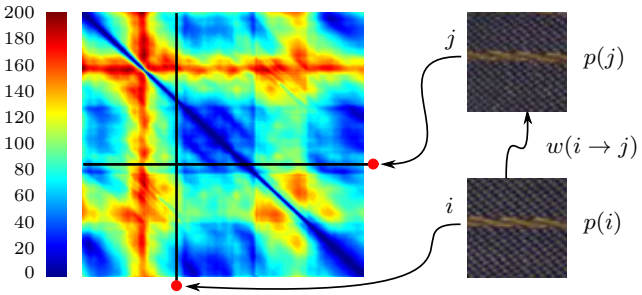

Figure 4: *A feature graph in matrix form, with color coded weights of the similarity between two patches $p(i)$ and $p(j)$. Blue represents a low matching error and hence a high similarity, while red represents a low patch similarity. Note that the matrix is not square, as there can be no edges into node 0, nor is it symmetric, as $w(i,j) = SSD(i, j-1)$ rather than $SSD(i,j)$.*

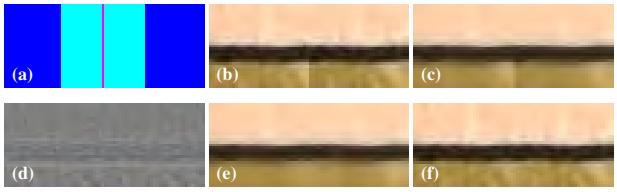

Figure 5: *An illustration of blending on jumps. (a) A colorized indication of the jump strip and blending area. (b) Synthesized feature without blending. (c) Synthesized base layer without blending. (d) Synthesized detail layer. (e) Synthesized base layer with extrapolation-blending. (f) Synthesized base layer with extrapolation blending and added detail.*

constraints. A potential solution could be to employ path optimization techniques that are capable of minimizing discontinuity along the entire path, e.g., by dynamic programming or belief propagation [Sun et al. 2005]. However, in our application such an approach is not suitable for several reasons. First, if the desired length of the walk is long compared to the input path provided by the user, the optimal solution degenerates to simply cycling the cheapest loop, as illustrated in Figure 3. Furthermore, it is not guaranteed that, when the user changes the desired length of the walk, the new optimal solution will have the previous one as a prefix. However, when a user draws a path with the brush tool, this corresponds to a permanent modification of the walk length. Failing to consider this inevitably causes the output stroke to flicker during interactive painting, as it would have to be re-rendered to remain optimal under changing stroke length (see supplementary video). In contrast, a random walk, such as the one employed by Schödl et al. [2000], can generate a randomized solution, but provides no guarantees on the discontinuity cost and assigns a non-zero probability to the highest-discontinuity edges.

So rather than finding a globally optimal walk of a given length or randomly traversing the graph, our synthesis algorithm generates a randomized, low-discontinuity walk of *at least* a given length. Instead of randomly picking the next outgoing edge to traverse, we pick the next goal node to visit. To minimize the discontinuity cost, we do not automatically traverse the edge connecting the two nodes, but instead apply Dijkstra's algorithm [Dijkstra 1959] to rapidly find the optimal path connecting the two nodes and append this path to the current walk. We repeat this process, starting from the previous goal node, until the desired length is achieved.

Although the length of the path, as measured in the number of nodes traversed, is not easily predictable, we may simply continue connecting paths until we obtain a walk of at least the desired length, picking each next goal node randomly. This ensures that any visual element present in the input will be rendered from time to time, without enforcing any particular ordering and keeping the visible discontinuities to a reasonable minimum. One could also conceivably bias the walk to a certain sub-portion of the feature by a more sophisticated selection of goal nodes, although we have not found this necessary for our application.

In order to render the selected feature onto a user-provided path, we sample the output path at equidistant intervals, generate a walk and assign to each of the output samples an input sample represented by the node at the given position in the walk. Having thus established correspondences between output and input samples, we use a simple piecewise-rigid mapping based on the Voronoi diagram of the output samples to determine the output pixel values for pixels within the stroke width of the sketched path. The process is illustrated in Figure 6.

Discontinuities in the synthesized path may occur when an edge with greater cost has to be traversed. To mask these without sacrificing fine details, we employ a decomposition-blending approach inspired by Burt and Adelson [1983]. Whenever consecutive output samples are created by a jump between non-consecutive input samples we perform local blending. To that end, we use a bilateral filter to decompose the source image into a base layer and a detail layer, as proposed by Durand and Dorsey [2002]. We then extrapolate the base layer values for each of the consecutive sub-sequences around the jump point and blend them, re-applying detail immediately thereafter, as illustrated in Figure 5.

3.2 Fill

The second tool, which we provide for efficient filling of image areas between line features, is essentially a paint bucket tool as present in all common image editors. However, analogous to the brush tool, our concept is to provide a fill tool that fills image areas with texture selected by the user from the image serving as the feature palette, maintaining consistency with the existing line features. Selection of area features is more straightforward than for line features as no specific structural properties have to be observed

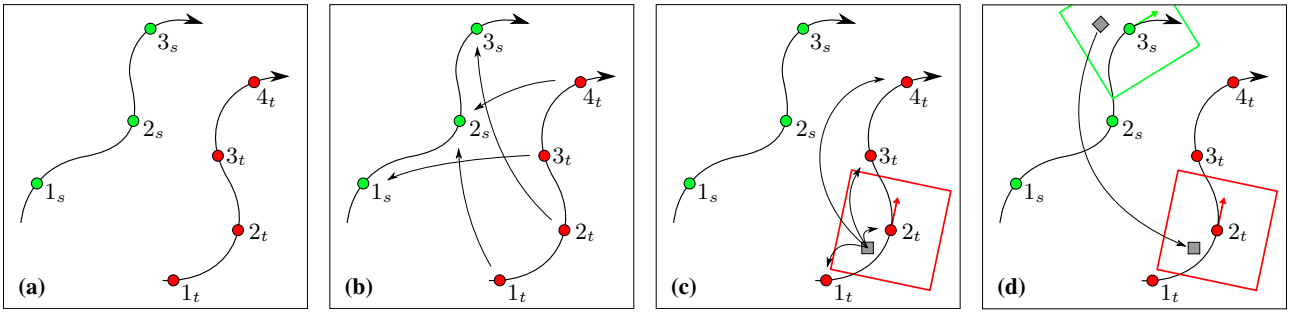

Figure 6: *Line feature mapping process. (a) Both the source path and the target path are sampled (respectively, the green and the red circles) at equal intervals. (b) We map the walk to the target path, determining for each target sample the corresponding source sample. (c) We determine the color of a pixel (gray square) in the target by finding the nearest target sample and (d) taking the value at the same relative position in the corresponding source patch (colored squares, arrows denote patch orientation).*

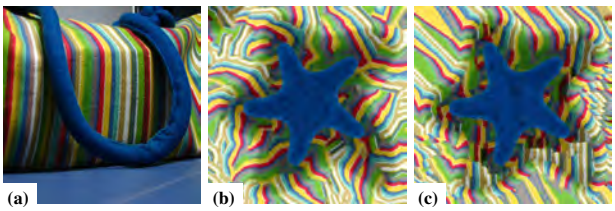

Figure 7: *Given a source image (a), results of the fill tool with (b) and without (c) pre-rotation. Note how the orientation of the stripes on the synthesized linear features guides the orientation of the generated texture in (b). When rotation is not taken into account visible artifacts might appear, as shown in (c). Source credit:* Hrishikesh Premkumar *via flickr*

during the selection. Hence, in our implementation the user can simply specify any arbitrary region in an image and use it as an area feature.

Unlike a simple flood fill tool, we have to consider the boundary conditions of the region being filled to avoid inconsistencies with existing image content like line features. Thus, rather than formulating the task of the fill tool as a simple texture synthesis problem, we treat this step as a content-aware fill which respects boundaries of the filled area and implement the method of Wexler et al. [2007] in combination with PatchMatch [Barnes et al. 2009] for fast nearest-neighbor search.

A multi-scale optimization approach [Wexler et al. 2007] is critical for our purpose, since the areas to be filled span over the majority of the canvas and treating the fill synthesis locally would lead to undesired artifacts and would furthermore be prone to introducing unwanted repetitions in the generated texture. To improve the quality and visual appearance of the result, we also perform the nearest-neighbor search across a limited range of rotations (see Figure 7). However, rather than computing the transformed source patches on the fly, we found that the combination of pre-rotating the source selection and performing the nearest-neighbor search using only translations to be significantly faster, which is crucial for instant results and direct visual feedback to the user.

4 Applications and Results

An overview of our image creation workflow is illustrated in Figure 8. Due to its generality, our approach can be utilized in several applications. One of our primary applications is vector image styl-

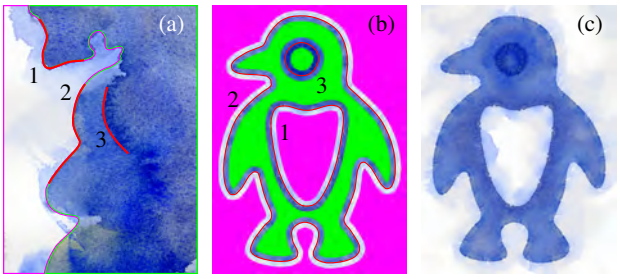

Figure 8: *Image creation workflow overview. (a) Annotated source image: two area features delimited by the pink and green outlines, and three line features indicated by the red curves and the numbers. (b) Line feature synthesis along user-specified paths from the corresponding numbered line features of (a). The pink and green areas represent the parts to be filled in by the corresponding area features of (a). (c) Final result after texture transfer by the fill tool. Source credits:* Alessandro Andreuccetti *via deviantART,* mrjive *via OpenClipArt*

ization: the user selects line and area features in an example image and then assigns them to paths and fill shapes of a vector image, respectively. Figure 1 shows representative results created using our framework. Note that, unlike previous texture-by-numbers approaches, we can handle open paths and strokes. The result images are visually consistent on a local as well as a global scale and represent the visual style of the respective reference image (see comparison with previous texture-by-numbers approaches in Figures 2 and 9). Figure 1a illustrates that even a simple vector image composed of a very limited number of input strokes (three in this example) can lead to richly textured image. In Figure 1b please note the quality of the knitting stitches generated by our line feature synthesis at the boundaries of the different pieces of the penguin. Our approach can also be applied for watercolor painting, as shown in Figure 1c. This challenging task usually requires sophisticated techniques [Curtis et al. 1997; DiVerdi et al. 2013], whereas our approach can solve it without additional specific tools.

An interesting characteristic of our approach is that when paths incident to a region have different "inside colors", the region inpainting algorithm attempts to diffuse the difference between their colors over the intermediate region, producing results similar to Diffusion Curves [Orzan et al. 2008], with no additional creative effort on the artist's part. For example, in Figure 1d, note the diffusion effect along the cat's whiskers and mustache (see also a more complex example of texture transitions in Figure 15). A comparison

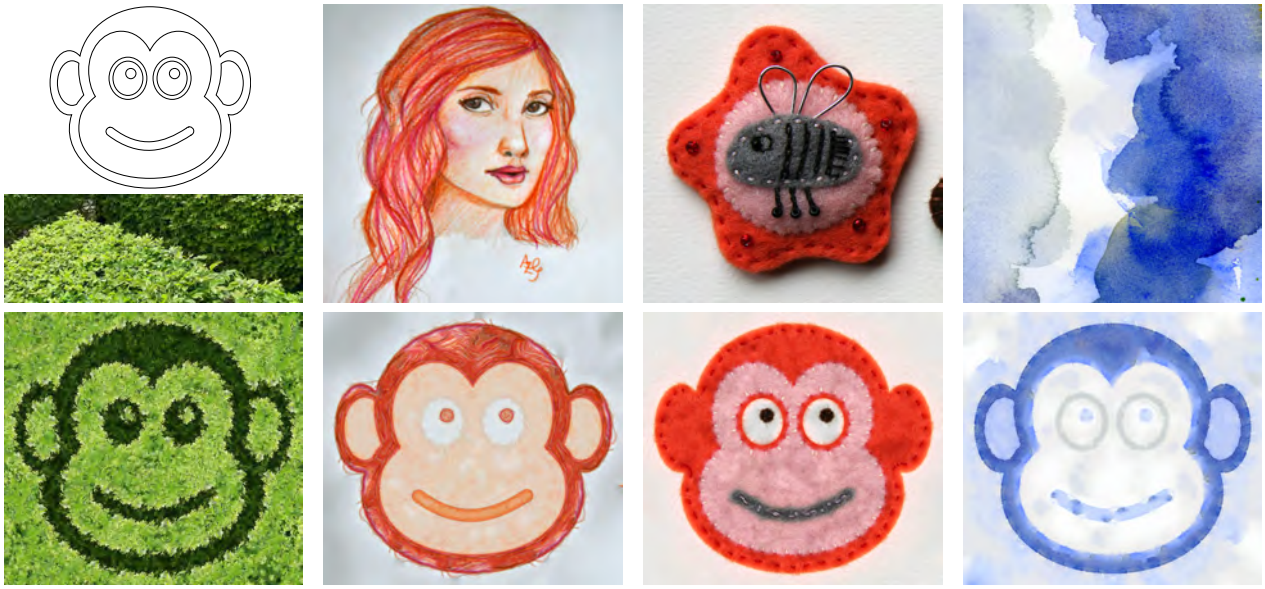

Figure 11: *Example of different stylizations with the same stroke input (top left). Source credits (left–right):* Martouf *via OpenClipArt;* Joe Shlabotnik *via flickr;* Andrea Garcia *via flickr;* Pavla Sýkorová*;* Alessandro Andreuccetti *via deviantART*

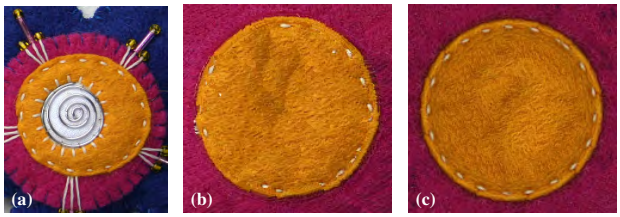

Figure 9: *Comparison with Painting with Texture [Ritter et al. 2006]: (a) source image, (b) result of Painting with Texture, (c) our approach. Note how our method better preserves the important visual charcteristics of the source image thanks to explicit treatment of line and area features. Source credit:* Pavla Sýkorová

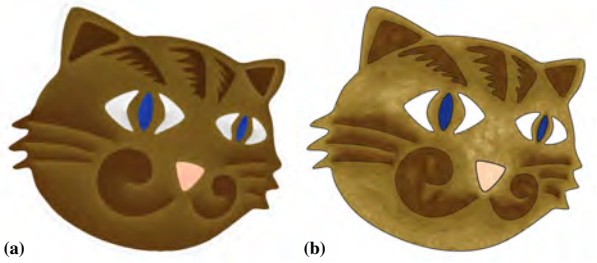

Figure 10: *Comparison of vector stylization techniques. (a) Result of Diffusion Curves [Orzan et al. 2008]. (b) Our result. Source credit:* Pavla Sýkorová

with Diffusion Curves is available in Figure 10. Both approaches took a comparable amount of artistic effort to produce, however, our method enables the transfer of the visual style and richness in terms of texture from a reference image (see Figure 1d). Additional examples of different stylizations given a single user-drawn sketch are shown in Figure 11. In this stylization scenario, the user simply needs to select the line and area features they would like to incorporate in the result image.

Another exciting application is interactive example-based painting. We have developed a painting program which implements just the two tools we introduce in this paper, deliberately leaving out extra functionality of sophisticated image editors, in order to show that our painting-by-feature approach alone enables the creation of appealing results. In our paint program the user may select features from source images and transfer them to manually indicated positions, using the same mode of interaction as with the common brush tool and fill tool known from consumer image editors. A representative interaction with our application is shown in Figure 8 as well as in the supplementary video. It demonstrates that our application is simple to use, and that the user can create and edit paintings interactively with instantaneous feedback. Visually appealing results can be created in a short time, typical editing session for the results shown here were in the order of 1–3 minutes depending on the level of detail the user wishes to incorporate. Additional results are shown in Figure 12. Further potential applications of our method include image editing scenarios such as inpainting. We refer readers to the supplementary material for a representative result.

4.1 Limitations

While our approach has proven suitable for its intended applications and produces high quality results, some limitations do apply.

We do not explicitly handle possible intersections and junctions of line features which may produce visually disturbing transitions in the output image (see Figure 13). These artifacts can partially be alleviated by proper reordering of strokes or using some sort of blending, e.g., min/max-blending (`GL_MIN` or `GL_MAX` blending mode in OpenGL) or decomposition-blending described in Section 3.1. Nevertheless, in future work one may consider to incorporate support for intersections and junctions directly into the synthesis algorithm to automatically produce seamless output.

We also deliberately do not check for consistency of the selected features to give the user full control and artistic freedom. As a consequence the user can select a line feature that is not aligned

Figure 12: *Additional results of interactive example-based painting. Left: source image, Right: result obtained by our approach. Source credits:* Paul Cézanne *(top);* Vincent Van Gogh*,* Kaldari *via Wikimedia Commons (bottom)*

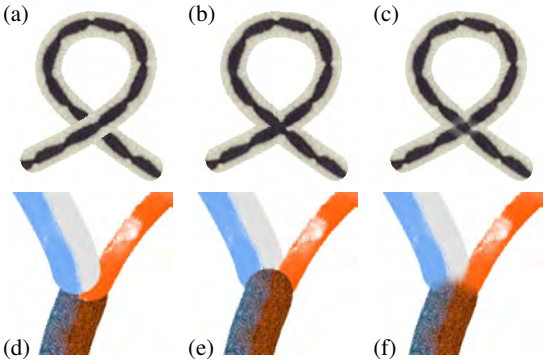

Figure 13: *Self intersections of a brush stroke (a) or junctions of multiple linear features (d) may produce visible discontinuities. These can be alleviated by proper stroke reordering (e), min/max-blending (b) or decomposition-blending (c,f).*

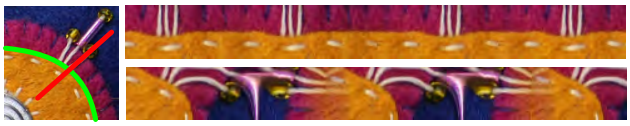

Figure 14: *The user can select linear features which may not be fully in line with our requirements on 1D structure (red and green curves in the left inset), potentially producing unintended results: the green curve generates white vertical sewings (top) and the red curve yileds a completely erronous result (bottom).*

with an actual linear structure in the input image or one that is composed of incompatible structural elements. In these cases our algorithm might produce visually displeasing transitions (see Figure 14). Similarly, selection of an area feature which is incompatible with already drawn line features may also lead to an erroneous result (see Figure 15). An alternative scenario to investigate in future work is that the feature selection process could be assisted by interactive image segmentation tools [Li et al. 2004], or by identification and removal of inconsistent sub-elements, e.g., by texture analysis [Todorovic and Ahuja 2009].

To prevent periodicity, our approach runs a randomized graph walk (see Section 3.1). The disadvantage is that variations might exist between results for a same source image and input sketch. Additional results available in the supplementary video (elephant sequence) show that these variations are very limited, on a local level and visually consistent with the other results on a global level, which is sufficient for our target applications.

5 Conclusion and Future Work

We have presented a feature-based image creation model, useful for vector image stylization as well as manual image creation and image editing. Our flexible example-based stylization approach blurs the traditional border between the vector- and pixel-worlds, allowing us to create and manipulate images while preserving the visual richness of a chosen artistic style. We eagerly anticipate the new possibilities in artwork creation that this approach opens to artists, and are curious about the results which may be achieved by combining this simple, yet powerful basic approach with other existing creation and editing tools.

An interesting direction for future work is the automation of the entire process of vector image stylization. This could be achieved by automatically detecting features in a source image and assigning them to paths and regions of a vector image based, e.g., on similarity of fill and stroke colors to the colors in the feature.

We could also modify our algorithm to automatically synthesize the fill for areas between user-defined curves while they are being drawn, producing an example-based variant of the Diffusion Curves by [Orzan et al. 2008]. However, even though line feature synthesis is fast enough for interactive editing, this would require a real-time fill synthesis algorithm (even with PatchMatch, [Wexler et al. 2007] is too slow to permit this) and a similarly rapid image analysis tool, which would determine source areas for output regions based on the input strokes and other features present in the input in order to keep the output visually consistent.

Similarly, used in conjunction with an automated image decomposition algorithm such as [Guo et al. 2007], one could reduce an input image into a sketch representation and a representative subset of features in order to re-synthesize the original image at a later time. Thus one could facilitate image compression with a configurable loss of information (see supplementary material for an example of image decomposition based on our method).

For the brush tool, it might be possible to investigate whether the input line feature contains any underlying dimensionality (such as texture orientation), and modify our formulation so that the output is constrained by this underlying parameter, determined, e.g., by pen pressure. Similarly, the introduction of control maps for area features could play a role for synthesis.

Acknowledgements

We would like to thank Gioacchino Noris and all anonymous reviewers for their constructive comments. This research has been supported by the Technology Agency of the Czech Republic under the research program TE01020415 (V3C – Visual Computing Competence Center) and partially by the Grant Agency of the Czech Technical University in Prague, grant No. SGS13/214/OHK3/3T/13 (Research of Progressive Computer Graphics Methods).

Figure 15: *An example output of our fill tool when synthesized strokes contain incompatible structures (a). When the whole source image (b) is taken as an example (red and blue rectangles) the fill tool produces pleasing transitions (c). However, when an incompatible portion (red and blue areas) of the source image is selected (d), the algorithm can produce erroneous results (e). Source credit:* Carl Wycoff *via flickr*

References

ASHIKHMIN, M. 2001. Synthesizing natural textures. In *Proceedings of Symposium on Interactive 3D graphics*, 217–226.

ASSA, J., AND COHEN-OR, D. 2012. More of the same: Synthesizing a variety by structural layering. *Computers & Graphics 36*, 4, 250–256.

BARNES, C., SHECHTMAN, E., FINKELSTEIN, A., AND GOLDMAN, D. B. 2009. PatchMatch: a randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics 28*, 3, 24:1–24:11.

BURT, J. R., AND ADELSON, E. H. 1983. A multiresolution spline with application to image mosaics. *ACM Transactions on Graphics 2*, 4, 217–236.

CURTIS, C. J., ANDERSON, S. E., SEIMS, J. E., FLEISCHER, K. W., AND SALESIN, D. H. 1997. Computer-generated watercolor. In *Proceedings of SIGGRAPH 97*, 421–430.

DARABI, S., SHECHTMAN, E., BARNES, C., GOLDMAN, D. B., AND SEN, P. 2012. Image melding: Combining inconsistent images using patch-based synthesis. *ACM Transactions on Graphics 31*, 4, 82:1–82:10.

DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive contours for conveying shape. *ACM Transactions on Graphics 22*, 3, 848–855.

DIJKSTRA, E. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 5, 269–271.

DIVERDI, S., KRISHNASWAMY, A., MECH, R., AND ITO, D. 2013. Painting with polygons: A procedural watercolor engine. *IEEE Transactions on Visualization and Computer Graphics 19*, 5, 723–735.

DURAND, F., AND DORSEY, J. 2002. Fast bilateral filtering for the display of high-dynamic-range images. *ACM Transactions on Graphics 21*, 3, 257–266.

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. In *Proceedings of SIGGRAPH 2001*, 341–346.

FANG, H., AND HART, J. C. 2007. Detail preserving shape deformation in image editing. *ACM Transactions on Graphics 26*, 3, 12:1–12:5.

GUO, C.-E., ZHU, S.-C., AND WU, Y. N. 2007. Primal sketch: Integrating structure and texture. *Computer Vision and Image Understanding 106*, 1, 5–19.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *Proceedings of SIGGRAPH 2001*, 327–340.

KASS, M., WITKIN, A., AND TERZOPOULOS, D. 1988. Snakes: Active contour models. *International Journal of Computer Vision 1*, 4, 321–331.

KWATRA, V., SCHÖDL, A., ESSA, I. A., TURK, G., AND BOBICK, A. F. 2003. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics 22*, 3, 277–286.

KWATRA, V., ESSA, I. A., BOBICK, A. F., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. *ACM Transactions on Graphics 24*, 3, 795–802.

LEFEBVRE, S., AND HOPPE, H. 2005. Parallel controllable texture synthesis. *ACM Transactions on Graphics 24*, 3, 777–786.

LI, Y., SUN, J., TANG, C.-K., AND SHUM, H.-Y. 2004. Lazy snapping. *ACM Transactions on Graphics 23*, 3, 303–308.

MCCANN, J., AND POLLARD, N. S. 2008. Real-time gradient-domain painting. *ACM Transactions on Graphics 27*, 3, 93:1–93:7.

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, 92:1–92:8.

RISSER, E., HAN, C., DAHYOT, R., AND GRINSPUN, E. 2010. Synthesizing structured image hybrids. *ACM Transactions on Graphics 29*, 4, 85:1–85:6.

RITTER, L., LI, W., CURLESS, B., AGRAWALA, M., AND SALESIN, D. 2006. Painting with texture. In *Proceedings of Eurographics Symposium on Rendering*, 371–376.

SCHÖDL, A., SZELISKI, R., SALESIN, D. H., AND ESSA, I. 2000. Video textures. In *Proceedings of SIGGRAPH 2000*, 489–498.

SHECHTMAN, E., RAV-ACHA, A., IRANI, M., AND SEITZ, S. M. 2010. Regenerative morphing. In *IEEE Conference on Computer Vision and Pattern Recognition*, 615–622.

SUN, J., YUAN, L., JIA, J., AND SHUM, H.-Y. 2005. Image completion with structure propagation. *ACM Transactions on Graphics 24*, 3, 861–868.

TODOROVIC, S., AND AHUJA, N. 2009. Texel-based texture segmentation. In *IEEE International Conference on Computer Vision*, 841–848.

WEXLER, Y., SHECHTMAN, E., AND IRANI, M. 2007. Space-time completion of video. *IEEE Transactions on Pattern Analysis and Machine Intelligence 29*, 3, 463–476.

YÜCER, K., JACOBSON, A., HORNUNG, A., AND SORKINE, O. 2012. Transfusive image manipulation. *ACM Transactions on Graphics 31*, 6, 176:1–176:9.