Interactive Image Filtering with Multiple Levels-of-Control on Mobile Devices

Amir Semmo Tobias Dürschmid Matthias Trapp Mandy Klingbeil Jürgen Döllner Sebastian Pasewaldt
Hasso Plattner Institute, University of Potsdam, Germany* Digital Masterpieces GmbH*

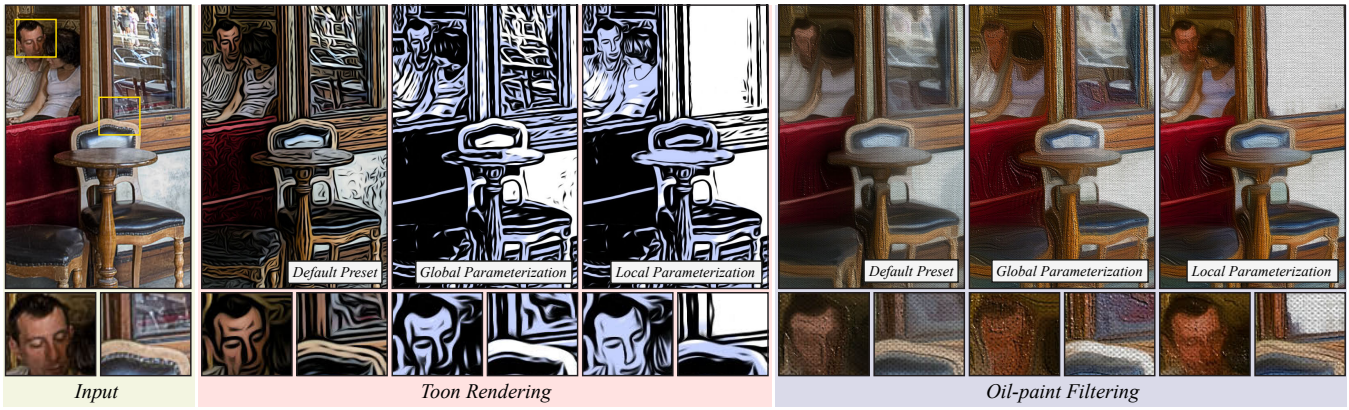


Figure 1: *Results obtained with the proposed framework for image stylization. Users are able to parameterize image filters at three levels of control: presets, global, and local adjustments (here: contour granularity and level of abstraction) enable a creative editing process.*

Abstract

With the continuous development of mobile graphics hardware, interactive high-quality image stylization based on nonlinear filtering is becoming feasible and increasingly used in casual creativity apps. However, these apps often only serve high-level controls to parameterize image filters and generally lack support for low-level (artistic) control, thus automating art creation rather than assisting it. This work presents a GPU-based framework that enables to parameterize image filters at three levels of control: (1) presets followed by (2) global parameter adjustments can be interactively refined by (3) complementary on-screen painting that operates within the filters' parameter spaces for local adjustments. The framework provides a modular XML-based effect scheme to effectively build complex image processing chains—using these interactive filters as building blocks—that can be efficiently processed on mobile devices. Thereby, global and local parameterizations are directed with higher-level algorithmic support to ease the interactive editing process, which is demonstrated by state-of-the-art stylization effects, such as oil-paint filtering and watercolor rendering.

Keywords: mobile, image filtering, NPR, interaction, GPU

Concepts: •**Computing methodologies** → **Image manipulation;**

*http://www.hpi3d.de | http://www.digitalmasterpieces.com

SA '16 Symp on Mobile Graphics and Interactive Applications, December 05-08, 2016, Macao
ISBN: 978-1-4503-4551-4/16/12...$15.00
DOI: http://dx.doi.org/10.1145/2999508.2999521

1 Introduction

Image stylization enjoys a growing popularity on mobile platforms to support casual creativity [Winnemöller 2013]. Today, a tremendous amount of mobile apps exist that implement stylization effects simulating the aesthetic appeal of artistic media [Dev 2013], such as oil, watercolor, and pencil. In particular, image filtering is very popular with the public to transform images into expressive renditions. However, mobile apps that implement image filters typically serve high-level controls such as global parameter adjustments and only limited low-level controls for local adjustments, e.g., via painting, thus automating art creation rather than assisting it.

Increasing the spectrum of interactivity for these (semi-)automatic filters towards multiple levels of control, e.g., by integrating tools for brush-based painting, poses a contemporary field of research [Isenberg 2016] to ease the visual expression for both artists and non-artists [Salesin 2002; Gooch et al. 2010]. With the continuous development of mobile graphics hardware and modalities (e.g., touch with pressure), interactive tools for filter parameterization are becoming feasible but remain challenging in two respects:

1. High-level and low-level controls should be non-mutually exclusive, and permit an iterative parameterization at global and pixel level to support different user skills and needs.

2. Complex stylization effects that require several passes of (non-)linear filtering need to be processed at interactive frame rates to support immediate visual feedback.

Both challenges need to be addressed to enable an interactive editing that fosters the creativity of both "experts" and casual users.

This paper presents an extensible GPU-based framework for mobile devices that addresses these challenges by shading-based filtering that couples higher-level algorithmic support with low-level controls for parameterization. The framework provides a modular XML-based effect scheme to rapidly build interactive image processing chains, e.g., for toon rendering [Winnemöller et al. 2006] or oil-paint filtering [Semmo et al. 2016b]. At this, presets and global adjustments can be complemented with brush-based painting

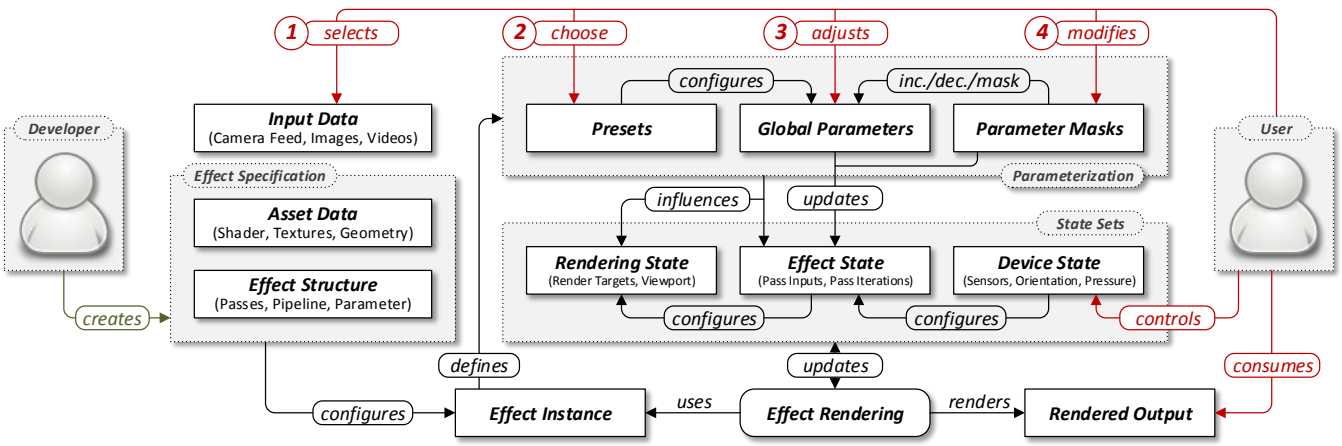


Figure 2: *Overview of the framework from the perspective of the end user (red) and the effect developer (gray).*

that operates within the parameter spaces of the stylization effects, e.g., to locally adjust colors and the level of abstraction (Figure 1). This way, results can be obtained quickly while advanced control is given for low-level adjustments.

To summarize, this paper makes the following contributions:

C1 A framework for effectively building complex image processing chains that can be efficiently processed on mobile devices.

C2 Concepts for interactive per-pixel parameterizations of image filters and their interplay with global parameterizations, which are demonstrated by stylization effects such as toon rendering and oil paint filtering using the proposed framework.

2 Related Work

Image processing constitutes an essential building block for mobile applications such as augmented reality and painterly rendering [Thabet et al. 2014] to facilitate the expression, recognition, and communication of image contents [Dev 2013]. Previous works that deal with image stylization primarily focused on technology transfers and optimizations to address the inherent limitations of mobile graphics hardware such as computing power and memory resources [Capin et al. 2008], while mobile apps primarily explored modalities for interactive parameterization.

Image filtering and stroke-based rendering [Kyprianidis et al. 2013] have been particularly used in mobile expressive rendering [Dev 2013] to simulate popular media and effects such as cartoon [Fischer et al. 2008; Kim and Lin 2012], watercolor [Oh et al. 2012; DiVerdi et al. 2013], and oil paint [Wexler and Dezeustre 2012; Kang and Yoon 2015]. Most of these works implement optimization techniques to enable real-time performance, for instance using separated filter kernels with optimized branching and varying floating-point precision [Singhal et al. 2011], genetic search algorithms [Wexler and Dezeustre 2012], and vector graphics [DiVerdi et al. 2013]. These works, however, typically provide only effect-specific implementations and solutions. By contrast, we propose a framework for authoring complex stylization effects with a uniform XML-based scheme, which provides a modular asset management for rapid prototyping. This includes shading-based effects that support nonlinear filtering, e.g., adapted to local images structures by following the approach of [Kyprianidis and Döllner 2008], to obtain flow-aligned outputs of deliberate, edge-preserving levels of abstraction. Thereby, we demonstrate that current mobile GPUs enable interactive filter-based image stylization that serves equivalent features than the respective desktop versions, such as toon [Winnemöller et al. 2006], oil paint [Semmo et al. 2016b] and watercolor [Bousseau et al. 2006] rendering.

The productization of research systems also led to a number of mobile apps, such as *PencilFX*, *ToonPAINT* [Winnemöller 2013] and *PaintCan* [Benedetti et al. 2014], which typically involves reducing a large number of technical parameters to a few comprehensive parameters that are exposed to users. State-of-the-art "filtering" apps, such as *Waterlogue*[1] and *Brushstroke*[2], typically provide only high-level parameters for effect tuning (e.g., presets, global adjustments). By contrast, we follow the approach of coupling low-level painting with algorithmic support—e.g., as used in *ToonPAINT* and *PaintCan* [Benedetti et al. 2014]—and extend the concept of specialized local parameterizations [Anjyo et al. 2006; Todo et al. 2007] to a generalized brush-based painting within effect-parameter spaces. Thereby, stylization effects are defined as a composition of ordered, parametrizable image filters that may be injected by user-specified motions [Hays and Essa 2004; Olsen et al. 2005] via on-screen painting [Hanrahan and Haeberli 1990]. We show that this approach greatly increases the feasibility of implementing interactive tools that adjust the appearance of filter-based stylization effects at run-time, which poses a contemporary field of research of the NPR community [Isenberg 2016] to help non-artists explore parameter spaces more easily [Lum and Ma 2002] and enable *right-brained* thinking [Gooch et al. 2010]. Previous works showed that this type of "user-centric NPR" [Winnemöller 2013] has potential to assist art creation, e.g., enabling user-defined stroke orientations on desktop systems [Salisbury et al. 1997; Gerl and Isenberg 2013] and mobile devices [Benedetti et al. 2014].

3 Technical Approach

Figure 2 gives an overview of the stages and interfaces—for effect developers and users—of the proposed framework. The stages comprise the effect specification (Section 3.1) and parameterization (Section 3.2), which are described in the following.

3.1 Effect Specification

For effective design and implementation of complex image processing pipelines, a modular approach is used for specifying: (1) rendering resources (e.g., effect shaders, textures, and buffers), (2) control

[1] http://www.tinrocket.com/waterlogue
[2] http://www.codeorgana.com/brushstroke

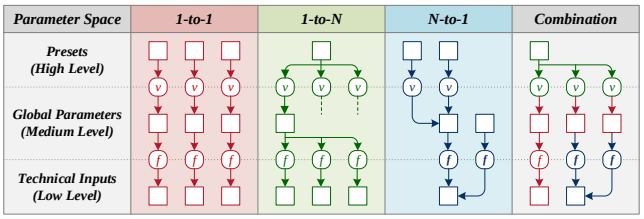


Figure 3: *Mechanisms for combining parameters spaces: 1-to-1 (propagation) supports advanced technical control, 1-to-N (composition) supports casual usages by reduction, N-to-1 (decomposition) supports single-element settings of vector inputs, and their respective combinations (here: indicated by the respective colors).*

flow and state of rendering passes (such as ping-pong rendering or image pyramid synthesis), and (3) parameters for different levels of control using a XML-based format. This way, effect specifications are facilitated that combine implementation-specific asset data with a platform-independent effect structure and parameterization that is required by user interfaces (refer to the supplemental material for an example). The format can be interpreted and instantiated for different rendering frameworks on various platforms, e.g., OpenGL ES with Android, iOS, or WebGL.

Moreover, this modular approach facilitates the implementation of complex pipelines for effect developers of different skill levels. While shader developers may have direct control at implementation level, e.g., shader source code, technical parameters, and rendering state, effect composers are able to create complex effects by re-using, combining or referencing existing effects or only parts thereof, e.g., edge enhancement or color quantization.

The core effect specification consists of (1) shader program declarations that reference fragment and vertex shader files, (2) texture declarations that define OpenGL textures with parameters such as wrap modes, filter modes and an optional bitmap source, and (3) rendering pass specifications that configure pass inputs and outputs for corresponding shader programs. Using this XML format, complex processing chains can be defined by creating a pipeline with multiple parallel or sequential rendering passes connected by pass input and output textures. An example is shown in Listing A1.

3.2 Effect Parameterization

For effect parameterization, the main idea is to decouple complex technical parameters—e.g., defined in the effect shaders as uniform variables—from parameters that are exposed to a user, particularly to hide technical complexity and facilitate ease-of-use. The conceptual overview shown in Figure 3 divides a parameterization into three technical layers. Each layer serves as abstraction of the underlying effect configuration by grouping it semantically and providing a meaningful name. At this, presets may provide high-level control by assigning concrete values v to global parameters (medium-level control). Global parameters map to technical (shader) parameters using a transformation function f that can be adjusted by parameter masks injected by painting to provide low-level control.

3.2.1 Presets (*high-level control*)

A preset comprises a "convincing" configuration of global parameters targeting a characteristic style. A set of presets should reveal the assorted characteristics and variations of an effect—e.g., different hatching styles (Figure 4)—that each serve as a starting point for fine tuning. Presets adjust values of global parameters that map to internal, technical parameters defined by the effect developer.

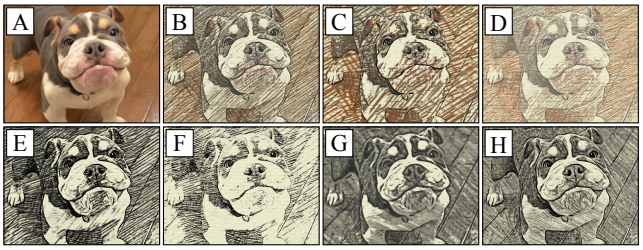


Figure 4: *Overview of presets defined for pencil hatching: (A) input, (B) default, (C) fine liner, (D) fine, (E) strong black and white, (F) lighten black and white, (G) stumpy pencil, (H) sharp pencil.*

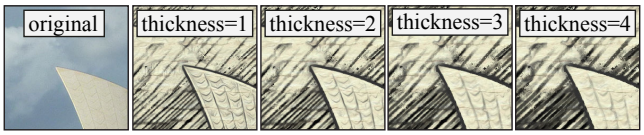


Figure 5: *Adjusting the global parameter "stroke thickness". The thickness and level of abstraction scales with the global parameter.*

Effect developers define presets in the XML effect file with a descriptive name, an icon to be displayed in the GUI and target values for global parameters (Listing A2). End users are then able to select one of the presets by choosing a named icon from a list.

3.2.2 Global Parameters (*medium-level control*)

A global parameter maps a user-definable value to interdependent technical inputs that influence visual attributes of an effect (Figure 5). A global parameter consists of a value of a specific type (e.g., integer, float, color, boolean) within a pre-defined value range.

Effect developers map global parameters to input values of effect shaders using the XML specification. This mapping may include a basic mathematical transformation—e.g., adding or multiplying constants and value parameters, or non-linear functions such as logarithmic (Listing A3, line 10). This way, global parameter values—changed by the end user—are automatically propagated to the effect shaders. Thereby, GUI elements such as value-range sliders, color pickers, or switch buttons can be used to influence visual attributes such as the color brightness, amount of color, contour width, or level of abstraction.

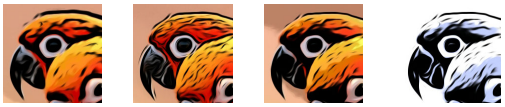
3.2.3 Parameter Masks (*low-level control*)

Parameter painting is a per-pixel adjustment of the functions that transform global parameters to technical inputs. Here, the metaphor of brush-based on-screen painting with touch-enabled inputs is applied (e.g., using fingers, pen) to locally refine the appearance.

Effect developers define a painting parameter as a distinguished global parameter with (1) an additional associated mask texture that is automatically updated by the framework when painting on-screen and is passed to the effect shaders, and (2) a set of masking brushes that can be used for parameter painting (Listing A4). Analogous to [DiVerdi 2013], these masking brush configurations are used to adjust the *painting strength*, *stroke width*, and *falloff* that controls the smoothness at the brush border. A mask is either referenced with an add or subtract operation to adjust or scale a global parameter in correspondence with the XML-based effect specification and effect shaders. Examples for these modes are given in Section 4.

Table 1: *Overview of presets, global parameters and technical inputs defined in the effect structure and asset data for toon rendering. The mapping of global parameters to technical parameters is indicated on the right-hand side, which includes bilateral filtering, color quantization, rendering with a spot color, and XDoG filtering as described in [Winnemöller et al. 2006; Winnemöller et al. 2012]. Each mapping can be locally adjusted by the proposed parameter painting. The color highlights exemplary show categories of parameter mappings (Figure 3).*

Presets				Global Parameters		Technical Inputs											
Comic	Details	Papercut	Kriek	ID	Value Range	σ_r	σ_d	q	φ_q	α	β	σ_e	σ_m	p	φ	ϵ
1.5	3.0	0.5	0.5	Details	$0.0 - 3.0$	x	x	x					x			
10	24	8	64	Colors	$5 - 64$			x								
1.6	2.5	1.0	1.0	Color Blur	$1.0 - 3.0$				x							
1.0	1.0	1.0	0.65	Color Threshold	$0.0 - 1.0$					x						
#00000000	#00000000	#00000000	#99C5D0FF	Spot Color	colors						x					
2.5	1.0	1.1	1.1	Contour Width	$0.4 - 4.0$							x				
0.98	0.99	0.96	0.96	Contour Gran.	$0.8 - 1.0$								x			
0.0	0.0	2.2	2.2	Blackness	$0.0 - 3.0$									x	x	
						BF		Quant.		Color		XDoG				



End users are able to locally adjust a selected parameter in local image regions by touch-enabled inputs such as wiping and dragging. Further, they can choose between the pre-defined brush configurations in order to customize the painting process. This enables to use small hard brushes for fine-granular painting as well as large smooth brushes for speed painting. Furthermore, users can choose between increasing and decreasing parameter values (e.g. lighten vs. darken) which enables to undo a painted process.

Finally, a specialized painting mode—*direction painting*—is introduced that traces the tangent information of the virtual brush stroke. As shown in Figure 6C, this mode enables pencil stroke rotations according to the painting direction by using a three-channel texture containing the x- and y-component of the tangent and the strength-/pressure of the stroke.

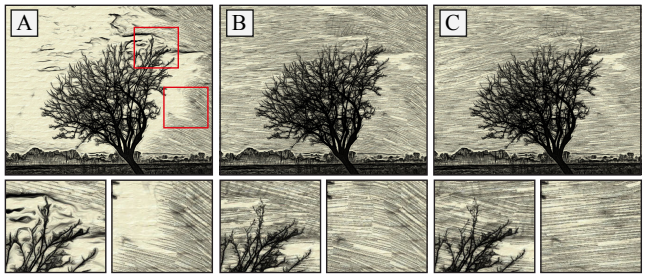


Figure 6: *Parameter painting: (A) global adjustments, (B) locally adjusted contours, (C) unified stroke directions via painting.*

4 Case Studies

The framework and effects were implemented with the Android SDK using Java, OpenGL ES, and the OpenGL ES Shading Language. In addition, the interaction concept was tested on iOS devices. We have implemented a range of state-of-the-art effects that build on image filtering and example-based texturing, and which were originally designed for (semi-)automatic stylization, i.e., to demonstrate the effectiveness of our methods to increase the spectrum of interactivity. The following effects were deployed on a Google™ Pixel C with a NVIDIA® Maxwell 256 core GPU.

Toon Filtering. We have implemented the toon effect (Figure 7) of [Winnemöller et al. 2006], which is based on bilateral filtering, local luminance thresholding, and difference-of-Gaussians (DoG) filtering, and enhanced it by rendering with spot colors [Rosin and Lai 2013]. A this, flow-based filter kernels are used for bilateral and extended DoG (XDoG) filtering—following the approaches described in [Kyprianidis and Döllner 2008; Winnemöller et al. 2012]—that are adapted to local image structures to provide smooth outputs at curved boundaries. In total, eight global parameters enable interactive control (Table 1). On the one hand, the *color amount* and their *transitions* mapping to technical quantization parameters [Winnemöller et al. 2006], the *level of detail* parameterizing the bilateral filter, contour granularity and color amount, and the *spot color* with its *threshold*. On the other hand, the *contour width*, *contour granularity*, and *blackness* map to technical parameters of the XDoG filter [Winnemöller et al. 2012]. Each mapping can be locally adjusted with the proposed parameter-painting concept. XML-specific definitions of this effect are found in the supplemental materials.

Pencil Hatching. The pencil hatching effect shown in Figure 4–6 aligns tonal art maps [Praun et al. 2001; Webb et al. 2002] with the main directions of image features—information that is obtained by an eigenanalysis of the smoothed structure tensor [Brox et al. 2006]. The approach is enhanced by gradations in luminance and complemented by contours derived from a flow-based DoG filter [Kyprianidis and Döllner 2008]. At this, contour enhancement is handled separately from tonal depiction, where a painting value parameter is introduced to locally remove contours. The virtual brush models for the pencil hatching effect vary with the global parameters. In particular, brightness and luminance-related parameters use a small strength and a soft brush to accomplish smooth color transitions, while parameter masks that adjust the contour granularity may be drawn with a stronger brush to instantly remove contours. The painting of the contours has scale semantic, i.e., the impact of the value parameter is reduced by the mask. Therefore, users are able to globally adjust the strength of the contours and locally remove them. The painting of the brightness parameter uses the add and subtract mode—i.e., the painting is interpreted as positive or negative offset around the value of the global parameter—to locally lighten or darken image regions.

Figure 7: *Results of toon and watercolor rendering implemented with our framework. The insets show parameter masks with color-encoded values that were injected by brush-based painting, i.e., blackness for toon rendering and the color threshold for watercolor rendering.*

Oil Paint and Watercolor Rendering. We used our framework to implement the oil-paint filter described in [Semmo et al. 2016c], using flow-based joint bilateral upsampling [Kopf et al. 2007] to work on image pyramids, and thus to be able to reduce the number of texture fetches when decreasing the image and filter kernel sizes [Semmo et al. 2016c]. This multi-scale approach enables interactive local parameterizations at run-time, e.g., of Gaussian filter kernel sizes to adjust the level of abstraction (Figure 1) and the flow direction. In addition, we have implemented watercolor rendering based on the works described in [Bousseau et al. 2006; Wang et al. 2014] that simulates effects such as wobbing, edge darkening, pigment density variation, and wet-in-wet (Figure 7). Each of these effects can be locally parameterized, e.g., to adjust the wetness by varying the filter kernel sizes when scattering noise in gradient direction of feature contours. Both effects provide presets for entry-level parameterizations. Here, the interested reader is referred to the supplemental video.

5 Discussion

We evaluated our framework with respect to performance, user experience, and developer experience, which are described in the following.

Performance. Evaluations of the nonlinear image effects using the test system described in Section 4 indicate that the framework is able to perform at interactive frame rates for images with full HD resolution, i.e., toon rendering runs at 14 frames per second (fps), pencil hatching at 8 fps, oil-paint filtering at 4 fps, and watercolor rendering at 5 fps, primarily due to optimizations such as separated filter kernels [Singhal et al. 2011] and processing of image pyramids [Semmo et al. 2016c].

User Experience. We have implemented the interaction concept with the oil paint and watercolor rendering in the iOS app *BeCasso* [Semmo et al. 2016a] and gathered qualitative feedback from 26 participants—using their smartphone (iPhone) and tablet (iPad)—as part of a public beta testing. Overall, they noted that "many apps currently do this type of image manipulation" but expressed that the proposed (filtering) results are of "higher quality than what is [currently] available". Example outputs using test images of the NPR benchmark of [Mould and Rosin 2016] are shown in Figure 9. However, our tests also exposed significant challenges in usability. These are mainly related to the trade-off in providing an easy-to-learn interface for novices while offering a highly efficient, but hard-to-learn brush-based painting tool for expert users.

The *learnability*—the ease of completing a task the first time exploring the design—and the *efficiency*—the speed of accomplishing a task after learning the design—as two key components of usability as described by [Nielsen 1993] were affected the most. To measure the users' performance, we asked them to apply and edit a specific preset to a given image. First, users should adjust a set of parameters of the preset globally, using the appropriate sliders. As expected, both novices and experts reached a reasonable level of usage proficiency and efficiency within a short time. Secondly, we asked them to adjust a specific parameter locally, using their finger as brush or eraser to increase or decrease the value of the parameter at a specific part of the image. All in all, more than 90% of the sample group needed assistance to complete this task, as they were not able to grasp the connection between a chosen parameter and the brush or eraser. Those results are backed up by our quantitative analysis, exposing the users' behavior and performance through event observations within the app.

We achieved much better results in other key components, including *satisfaction*—the pleasure of using the design—and *memorability*—the ease of reestablishing proficiency after a period of not using the design. Upon understanding the concept, users willingly returned to use the app and highlighted that the "parameter painting function sets the app apart from others". Due to the amount of mistakes made, users only requested a precise undo functionality, recording each individual step of global and local changes. In addition, some users would find it exciting to combine different styles at different spots of their images.

Developer Experience. Our framework is used for teaching image and video processing on mobile devices with OpenGL ES. We had 20 graduate and undergraduate students that developed novel stylization effects such as the proposed pencil hatching and a photographic plate simulation, using our XML-based effect specification and asset data as a construction kit. The students' results (Figure 8) and evaluations indicate that the proposed modular effect compositing is suitable to engage developers in rapid prototyping, help focus on effect design, and address usability concerns such as parameter optimization. Effect-specific features such as *render to mipmap* or *ping-pong render buffers*, however, can be easily integrated with additional programmatic effort.

We anticipate that the proposed methods and the parameterization concept will lead to improvements in user-defined image filtering that aspires casual creativity. Nonetheless, there are open questions that need to be addressed in future studies, such as the range of variation and level of engagement that can be achieved.

Figure 8: *Additional effects implemented by our undergraduate students as part of a seminal project, using the proposed framework for development: (left) photographic plate simulation based on unsharp masking, grayscale rendering and noise, (middle) simulation of the* Sheng Qi *painting style by rendering with spot colors and visualizing paint drops, (right) bloom effect based on lightening image regions.*

6 Conclusions and Future Work

This paper presents a GPU-based framework that enables to deploy and parameterize image filters on mobile devices for stylization effects such as toon, oil paint and watercolor rendering. The key contribution of this framework is an effect parameterization at three levels of control that allows users to rapidly obtain stylized outputs of aesthetic value, while giving creative control for global and local effect adjustments with algorithmic support. Moreover, with the integrated asset-based effect specification, developers are able to rapidly compose multi-stage nonlinear image effects and help focus on the interplay of technical and user-adjustable parameters.

For future work, we plan to resolve usability issues by exposing local effect adjustments as an own set of tools to the users, visually separated from the preset parameters. For example, this could include a pencil to add edges to a local spot, a spray bottle and paper tissues to control the wetness of an image, or brushes to regulate the amount of paint used. These tools resemble real-world objects, making the process less technical, but more authentic. Further, we consider to support multiple granularities of effect specification (e.g., expert modes with 1-to-1 parameter mappings), use mobile sensors for effect parameterization, and employ machine learning to precompute parameter masks according to image contents.

Acknowledgments

This work was funded by the Federal Ministry of Education and Research (BMBF), Germany, for the AVA project 01IS15041B and within the InnoProfile Transfer research group "4D*n*D-Vis" (www.4dndvis.de).

References

ANJYO, K.-I., WEMLER, S., AND BAXTER, W. 2006. Tweakable Light and Shade for Cartoon Animation. In *Proc. NPAR*, 133–139.

BENEDETTI, L., WINNEMÖLLER, H., CORSINI, M., AND SCOPIGNO, R. 2014. Painting with Bob: Assisted Creativity for Novices. In *Proc. ACM UIST*, 419–428.

BOUSSEAU, A., KAPLAN, M., THOLLOT, J., AND SILLION, F. X. 2006. Interactive Watercolor Rendering with Temporal Coherence and Abstraction. In *Proc. NPAR*, 141–149.

BROX, T., WEICKERT, J., BURGETH, B., AND MRÁZEK, P. 2006. Nonlinear structure tensors. *Image and Vision Computing 24*, 1, 41–55.

CAPIN, T., PULLI, K., AND AKENINE-MLLER, T. 2008. The State of the Art in Mobile Graphics Research. *IEEE Computer Graphics and Applications 28*, 4, 74–84.

DEV, K. 2013. Mobile Expressive Renderings: The State of the Art. *IEEE Computer Graphics and Applications 33*, 3, 22–31.

DIVERDI, S., KRISHNASWAMY, A., MECH, R., AND ITO, D. 2013. Painting with Polygons: A Procedural Watercolor Engine. *IEEE Trans. Vis. Comput. Graphics 19*, 5, 723–735.

DIVERDI, S. 2013. A Brush Stroke Synthesis Toolbox. In *Image and Video-Based Artistic Stylisation*. Springer, 23–44.

FISCHER, J., HALLER, M., AND THOMAS, B. 2008. Stylized Depiction in Mixed Reality. *International Journal of Virtual Reality 7*, 4 (December), 71–79.

GERL, M., AND ISENBERG, T. 2013. Interactive Example-based Hatching. *Computers & Graphics 37*, 1–2, 65–80.

GOOCH, A. A., LONG, J., JI, L., ESTEY, A., AND GOOCH, B. S. 2010. Viewing Progress in Non-photorealistic Rendering through Heinlein's Lens. In *Proc. NPAR*, 165–171.

HANRAHAN, P., AND HAEBERLI, P. 1990. Direct WYSIWYG Painting and Texturing on 3D Shapes. *Computer Graphics 24*, 4, 215–223.

HAYS, J., AND ESSA, I. 2004. Image and Video Based Painterly Animation. In *Proc. NPAR*, 113–120.

ISENBERG, T. 2016. Interactive NPAR: What Type of Tools Should We Create? In *Proc. NPAR*, 89–96.

KANG, D., AND YOON, K. 2015. Interactive Painterly Rendering for Mobile Devices. In *Entertainment Computing - ICEC 2015*. Springer International Publishing, 445–450.

KIM, T. H., AND LIN, I., 2012. Real-Time Non-photorealistic Viewfinder on the Tegra 3 Platform. Stanford University, unpublished.

KOPF, J., COHEN, M. F., LISCHINSKI, D., AND UYTTENDAELE, M. 2007. Joint Bilateral Upsampling. *ACM Trans. Graph. 26*, 3.

KYPRIANIDIS, J. E., AND DÖLLNER, J. 2008. Image Abstraction by Structure Adaptive Filtering. In *Proc. EG UK TPCG*, 51–58.

KYPRIANIDIS, J. E., COLLOMOSSE, J., WANG, T., AND ISENBERG, T. 2013. State of the 'Art': A Taxonomy of Artistic Stylization Techniques for Images and Video. *IEEE Trans. Vis. Comput. Graphics 19*, 5, 866–885.

Figure 9: *Results produced with the proposed framework for mobile image stylization: toon, pencil hatching, oil paint and watercolor. The input images are obtained from the benchmark of [Mould and Rosin 2016] and are standardized at a 1024 pixels width (shown cropped).*

LUM, E. B., AND MA, K.-L. 2002. Interactivity is the Key to Expressive Visualization. *SIGGRAPH Comput. Graph. 36*, 3, 5–9.

MOULD, D., AND ROSIN, P. L. 2016. A Benchmark Image Set for Evaluating Stylization. In *Proc. NPAR*, 11–20.

NIELSEN, J. 1993. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 23–48.

OH, J., MAENG, S., AND PARK, J. 2012. Efficient Watercolor Painting on Mobile Devices. *International Journal of Contents 8*, 4, 36–41.

OLSEN, S. C., MAXWELL, B. A., AND GOOCH, B. 2005. Interactive Vector Fields for Painterly Rendering. In *Proc. Graphics Interface*, 241–247.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-Time Hatching. In *Proc. ACM SIGGRAPH*, 581–586.

ROSIN, P. L., AND LAI, Y.-K. 2013. Non-photorealistic Rendering with Spot Colour. In *Proc. CAe*, 67–75.

SALESIN, D. H., 2002. Non-Photorealistic Animation & Rendering: 7 Grand Challenges. Keynote talk at NPAR.

SALISBURY, M. P., WONG, M. T., HUGHES, J. F., AND SALESIN, D. H. 1997. Orientable Textures for Image-based Pen-and-ink Illustration. In *Proc. ACM SIGGRAPH*, 401–406.

SEMMO, A., DÖLLNER, J., AND SCHLEGEL, F. 2016. BeCasso: Image Stylization by Interactive Oil Paint Filtering on Mobile Devices. In *Proc. ACM SIGGRAPH Appy Hour*, 6:1–6:1.

SEMMO, A., LIMBERGER, D., KYPRIANIDIS, J. E., AND DÖLLNER, J. 2016. Image Stylization by Interactive Oil Paint Filtering. *Computers & Graphics 55*, 157–171.

SEMMO, A., TRAPP, M., DÜRSCHMID, T., DÖLLNER, J., AND PASEWALDT, S. 2016. Interactive Multi-scale Oil Paint Filtering on Mobile Devices. In *Proc. ACM SIGGRAPH Posters*, 42:1–42:2.

SINGHAL, N., YOO, J. W., CHOI, H. Y., AND PARK, I. K. 2011. Design and Optimization of Image Processing Algorithms on Mobile GPU. In *ACM SIGGRAPH Posters*, 21:1–21:1.

THABET, R., MAHMOUDI, R., AND BEDOUI, M. H. 2014. Image processing on mobile devices: An overview. In *Proc. IPAS*, 1–8.

TODO, H., ANJYO, K.-I., BAXTER, W., AND IGARASHI, T. 2007. Locally Controllable Stylized Shading. *ACM Trans. Graph. 26*, 3, 17:1–17:7.

WANG, M., WANG, B., FEI, Y., QIAN, K., WANG, W., CHEN, J., AND YONG, J.-H. 2014. Towards Photo Watercolorization with Artistic Verisimilitude. *IEEE Trans. Vis. Comput. Graphics 20*, 10, 1451–1460.

WEBB, M., PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2002. Fine tone control in hardware hatching. In *Proc. NPAR*, 53–58.

WEXLER, D., AND DEZEUSTRE, G. 2012. Intelligent Brush Strokes. In *Proc. ACM SIGGRAPH Talks*, 50:1–50:1.

WINNEMÖLLER, H., OLSEN, S. C., AND GOOCH, B. 2006. Real-Time Video Abstraction. *ACM Trans. Graph. 25*, 3, 1221–1226.

WINNEMÖLLER, H., KYPRIANIDIS, J. E., AND OLSEN, S. C. 2012. XDoG: an extended difference-of-Gaussians compendium including advanced image stylization. *Computers & Graphics 36*, 6, 740–753.

WINNEMÖLLER, H. 2013. NPR in the Wild. In *Image and Video-Based Artistic Stylisation*. Springer, 353–374.

Appendix

Listing A1: *Color quantization and conversion passes defined with the XML format used by the proposed framework. The passes are connected by using a single texture as output buffer of the first pass (line 15) and as input resource of the second pass (line 22).*

```
1  <pass id="colorQuantizationPass" enabled="true"
       shaderprogram="colorQuantizationShaderProgram">
2     <passinputs>
3        <passinput id="u_Texture" type="sampler">
4           <value>colorBilateralPass1Texture</value>
5        </passinput>
6        <passinput id="u_NumBins" type="float">
7           <value>2.0</value>
8        </passinput>
9        <passinput id="u_PhiQ" type="float">
10          <value>3.4</value>
11       </passinput>
12    </passinputs>
13    <passoutputs>
14       <passoutput id="colorQuantizationPassOutput0">
15          <value>colorQuantizationTexture</value>
16       </passoutput>
17    </passoutputs>
18 </pass>
19 <pass id="lab2RgbPass" enabled="true"
       shaderprogram="lab2RgbShaderProgram">
20    <passinputs>
21       <passinput id="u_Texture" type="sampler">
22          <value>colorQuantizationTexture</value>
23       </passinput>
24    </passinputs>
25    <passoutputs>
26       <passoutput id="lab2RgbPassOutput0">
27          <value>colorQuantizationTextureLAB</value>
28       </passoutput>
29    </passoutputs>
30 </pass>
```

Listing A2: *Example of a portrait preset defined with the XML format to adjust the values of referenced global parameters.*

```
1  <preset name="Portrait" icon="fx/toon/preset/portrait.png">
2     <parameter ref="details">3.0</parameter>
3     <parameter ref="colorBlur">2.1</parameter>
4     <parameter ref="colorQuantization">9.0</parameter>
5  </preset>
```

Listing A3: *Example of the definition of a global parameter in our XML format. The pass inputs of the defined passes are referenced.*

```
1  <valueparameter id="colorQuantization" description="Color
       Quantization"
       icon="fx/toon/icon/parameter/color_quantization.png"
       type="float" hidden="false">
2     <default>24</default>
3     <minrange>5</minrange>
4     <maxrange>30</maxrange>
5     <step>1.0</step>
6     <setpassinput pass="colorQuantizationPass"
          passinput="u_NumBins"/>
7     <expressionset pass="colorQuantizationPass"
          passinput="u_PhiQ">
8     <parameterexpressionvariable ref="colorBlur"/>
9        <expression>
10          (colorQuantization * 0.0339) + 0.0135 + (3.0 -
                colorBlur) * (3.0 - colorBlur)
11       </expression>
12    </expressionset>
13 </valueparameter>
```

Listing A4: *XML-based definition of a painting parameter.*

```
1  <paintingvalueparameter description="Paper Structure"
       hidden="false"
       icon="fx/pencilhatching/icon/parameter/paper_structure.png"
       id="paperStructure" type="float">
2     <mask>paperStructureMaskTexture</mask>
3     <brushes>
4        <brush>large</brush>
5        <defaultbrush>strong</defaultbrush>
6     </brushes>
7     <default>5.0</default>
8     <minrange>0.0</minrange>
9     <maxrange>20.0</maxrange>
10    <step>1.0</step>
11    <setpassinput pass="hatchTexturingPass"
          passinput="u_PaperStructure"/>
12 </paintingvalueparameter>
```