Vision, Modeling, and Visualization (2023)
T. Grosch and M. Guthe (Eds.)

Art-directable Stroke-based Rendering on Mobile Devices

Ronja Wagner[1], Sebastian Schulz[1], Max Reimann[1], Amir Semmo[2], Jürgen Döllner[1], and Matthias Trapp[1]

[1]Hasso Plattner Institute for Digital Engineering, University of Potsdam, Germany
[2]Digital Masterpieces GmbH, Potsdam, Germany

Abstract

This paper introduces an art-directable stroke-based rendering technique for transforming photos into painterly renditions on mobile devices. Unlike previous approaches that rely on time-consuming iterative computations and explicit brush-stroke geometry, our method offers a interactive image-based implementation tailored to the capabilities of modern mobile devices. The technique places curved brush strokes in multiple passes, leveraging a texture bombing algorithm. To maintain and highlight essential details for stylization, we incorporate additional information such as image salience, depth, and facial landmarks as parameters. Our technique enables a user to control and manipulate using a wide range of parameters and masks during editing to adjust and refine the stylized image. The result is an interactive painterly stylization tool that supports high-resolution input images, providing users with an immersive and engaging artistic experience on their mobile devices.

CCS Concepts

• ***Computing methodologies*** → ***Image-based rendering; Non-photorealistic rendering; Image processing;***

1. Introduction

Painterly rendering denotes the artistic stylization of two-dimensional visual media (e.g., images and videos) imitating the style of using analog painting tools. According to Kyprianidis et al. [KCWI13], the original approach by Aaron Hertzmann [Her98] is classified as a Stroke-based Rendering (SBR) technique using local, automatic brush stroke placement based on low-level image characteristics. Painterly rendering is among the most popular Non-Photorealistic Rendering (NPR) techniques and has been employed in many applications [HGT13]. With the continuous development of mobile graphics hardware, interactive high-quality image stylization is becoming feasible and increasingly used in casual creativity apps [SDT*16].

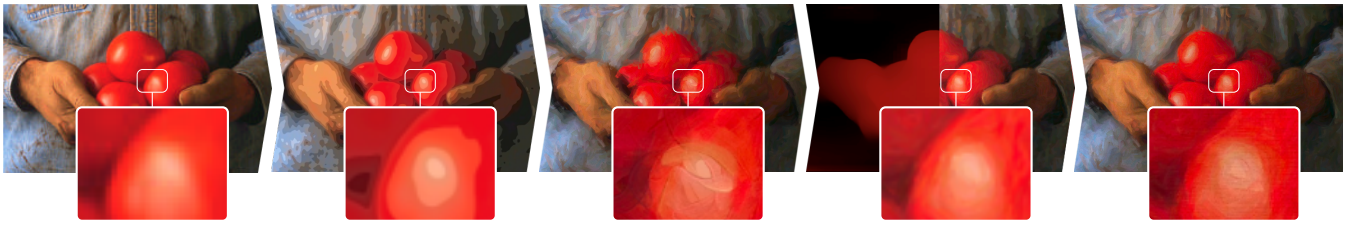
Problem Statement. Most of the previous approaches [Her98, WD12] achieve painterly stylization by rendering and blending textured brushstroke geometry successively until the result sufficiently approximates the input image. The limitations of such automatic approaches are (1) the runtime performance is limited by the number of strokes required to cover an input image at a certain resolution and (2) there is little to no explicit control over design aspects of brush strokes such as their size, density, curvature, or texture. However, "art-direction" of the visual elements and aesthetics of computer-generated imagery, preferably at different levels of control [Ise16], is argued to be of significant value to support a "full design cycle" [Sal02].This involves providing artists, and users not trained in the arts, with interactive tools for algorithmic support, i.e., to bridge the gap between traditional artistic expression and the capabilities of computer-generated imagery [Ise16].

Figure 1: *Mobile application for painterly rendering using curved brushstrokes.*

To achieve this in the context of resource-limited mobile devices, a sufficiently fast method for the synthesis and rendering of brush strokes represents the foundation for editing at multiple levels of control [SDT*16] for high-resolution input images (Fig. 1).

Approach and Contributions. To approach the challenges above, this paper approximates the original method of Hertzmann [Her98] by combining techniques that are suitable for a Graphics Processing Unit (GPU)-aligned brush stroke synthesis on mobile devices, such as texture bombing for stroke placement, texture warping for

DOI: 10.2312/vmv.20231240



(a) Input Image I (b) Background Layer B (c) Base Brush Layer L_B (d) Detail Brush Layer L_D (e) Stylized Output O

Figure 2: *Conceptual stage overview of the proposed painterly rendering technique.*

rendering curved brush strokes, and multi-layer brush stroke rendering. Similar to Lindemeier et al. [LSD16], the results are composed of multiple base layers blended with details layers to preserve the expression of high-frequency details. To accelerate stroke placement, we use a texture bombing approach [Gla04] and apply it at different grid resolutions successively. Texture bombing works by placing strokes in a grid, offset randomly to avoid pattern artifacts – therefore only a constant number of neighboring grid cells need to be taken into account for each fragment, which leads to a runtime independent of the total stroke count. Our approach enables the rendering of high-definition previews at interactive framerates and supports tile-based processing for high-resolution outputs up to 145 Mpx on mobile devices. Furthermore, our approach can be configured by a wide range of design parameters and thus provides an SBR technique that allows to control various aspects of brush strokes such as size, shape, texture, color, and placement to achieve painterly aesthetics in an interactive way. Further, our technique can be spatially parameterized by an importance map computed based on image features such as depth, salience, or facial landmarks [WD12].

2. Related Work

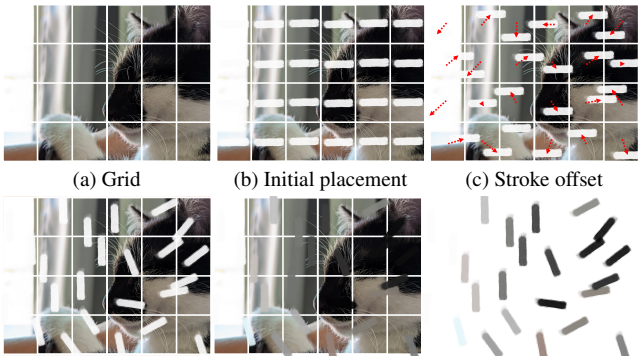
There are various approaches to painterly rendering expressed in multiple styles, such as watercolor or oil painting. The reviews of Hedge et al. [HGT13] and Nolte et al. [NMR22] provide a comprehensive overview of such approaches. Following Hertzmann [Her03] and Nolte et al. [NMR22], painterly rendering algorithms can be categorized by how they make decisions about the placement of strokes. Generally, the problem of SBR can be approached from the perspective of error minimization to the original image or by following a "greedy" heuristic.

Approaches using error minimization require many iterations of optimization using brute-force or random searches [Her01, OH11, KCC06], gradient-descent [Nak19, ZSQ*21, KWHO21], or require training of deep neural networks using supervised approaches [ZJH19, LLH*21] or deep reinforcement learning [GKB*18, HHZ19]. Due to the high number of iterations and runtime needed, these approaches are generally not suited for interactive applications. While recent approaches such as PaintTransformer (PT) [LLH*21] can predict strokes in parallel on a per-layer basis, they require a large amount of GPU memory even at low image resolutions, which is not available on mobile devices.

Heuristics-based approaches directly compute the stroke placement in a single pass using a representation of the image content, provided by edge detection or region segmentation. Low-level edges are obtained from the image gradient commonly used to guide strokes. Early works, such as Haeberli [Hae90], let users place strokes, or position them in a regular grid, such as Litwinowicz [Lit97], and then clip and orient them according to low-level edges. Hertzmann [Her98] proposes the use of longer, curved strokes using B-splines that are oriented along the local gradient orthogonal. The painting process incorporates multiple layers wherein broader strokes are drawn following the large-scale gradients. Smaller, thinner strokes are then layered atop areas where the broad strokes fail to encapsulate all the intricate details. Several approaches use a similar layering strategy while varying the edge placement strategy [HE04, KS04, SPY09, HFL11]. To better mimic the human painting process, semantically more meaningful high-level information can be incorporated. As such, segmentation has been employed to provide levels of abstraction and to guide strokes along object borders [ZZXZ09, LSD16]. Lindemeier et al. [LSD16] propose using hierarchical segmentation of a given input image into several regions, which are then further divided into layers, representing the content from coarse to fine details. We employ a similar strategy of base and detail decomposition, which allows for more precise control over the painting process. Collomosse et al. [CH02] propose using calculated image salience to place smaller strokes in more important regions while Wexler et al. [WD12] use importance sampling to place details along facial landmarks. In contrast to previous approaches, our approach combines image salience, facial landmarks, and additionally scene depth to better separate the foreground-background levels of detail into an importance map to guide stroke placement. The weighting of these components can be interactively adjusted by users to art-direct the level of stroke details in various regions.

Several strategies have been devised to expedite the painterly rendering process. For instance, Hertzmann [HP00] adapts painterly rendering for videos by placing strokes only in those areas of the canvas with a significant change in a new frame. Mukudan et al. [MH08] propose a fast index-table-based region-labeling approach for painterly rendering on mobile devices, while Fischer et al. [FBS05] propose fast pointillistic rendering filters for augmented reality applications. Our approach accelerates the rendering by parallelizing stroke placement on the GPU using brush stroke texture atlases sampled by texture bombing [Gla04].

Various approaches add interactive control mechanisms to painterly rendering. These mechanisms include the integration of hand gestures [GCI08] or touch-based controls [KY15] for finger-painting, as well as the implementation of interactive stroke processes to allow users to interactively modify stroke styles [ZZ11] or simulate brush-canvas interactions [BL04]. Benedetti et al.



(a) Grid (b) Initial placement (c) Stroke offset

(d) Stroke Rotation (e) Stroke Coloring (f) Pass Result

Figure 3: *Illustration of basic texture bombing stages during stroke-based brush rendering.*

[BWCS14] postulate a number of design guidelines for interactive painting applications for novice users, and develop a painting system according to these criteria where users draw strokes on a canvas, and the system provides guidance through a reference image. Similar to theirs, our systems allow novice users to easily create painterly images while at the same time allowing for creative flexibility and achieving high-fidelity results. Specifically, in contrast to most previous approaches, our approach allows interactive control, even at high resolutions, over a range of parameters that influence stroke placement and appearance as well as background rendering.

3. Method

Our approach consists of several conceptual stages, shown in Fig. 2, which are configured by parameters summarized in Tab. 1. We first give an overview of these in the following and then expand on important aspects of the main stages.

Input Image I: The image to be stylized (Fig. 2a). Our approach supports raster images without (RGB) and with associated depth information (RGB-D) that can be taken into account during painterly stylization.

Background Rendering B: The background image (Fig. 2b) is rendered first and represents a basis for the subsequent base and detail brush layers. Since the background may not be completely covered with brush strokes, high image frequencies are removed by applying an aggressive smooth filter followed by color quantization [WOG06] with ψ quantization levels (e.g., 10 in the shown examples).

Base Layer Rendering L_B: This layer represents coarse brush strokes for the stylization (Fig. 2c). During three rendering passes, coarse brush strokes scaled by the grid-size σ_i (Sec. 3.1) are added to the background using texture bombing [Gla04].

Detail Layer Rendering L_D: Similar to the previous layer, detail brush strokes are added to the final base layer using two additional passes. The probability of detail brush stroke occurrence is controlled by an importance map (Sec. 3.2).

Output Compositing O: In the final stage, the resulting layers are blended with canvas textures (Sec. 3.3). Due to the nature of our approach, the result of the painterly stylization process (Fig. 2e) can be of significantly higher resolution than the input image.

Table 1: *Overview of the major parameters provided by our painterly rendering technique.*

Parameter	Domain	Description
β	$[1, \max(w,h)] \subset \mathbb{N}$	Blur radius for background
ψ	$[1,256] \subset \mathbb{N}$	Quantization level for background
κ	$[0,1] \subset \mathbb{R}$	Focus depth for depth processing (M_D)
λ	$[0,1] \subset \mathbb{R}$	Threshold for depth processing (M_D)
ω_S	$[0,1] \subset \mathbb{R}$	Salience strength for important map M_I
ω_D	$[0,1] \subset \mathbb{R}$	Depth strength for important map M_I
ω_F	$[0,1] \subset \mathbb{R}$	Landmark strength for important map M_I
τ	$[0,1] \subset \mathbb{R}$	Brushstroke transparency
μ	$[0,1] \subset \mathbb{R}$	Color variance for brush strokes
γ	$[0,2] \subset \mathbb{R}$	Brushstroke curvature
σ	$[0,1] \subset \mathbb{R}$	Brushstroke size
ξ	$[1,5] \subset \mathbb{R}$	Detail amount for detail layer
ζ	$[0,1] \subset \mathbb{R}$	Canvas strength for compositing
η	$]0,2] \subset \mathbb{R}$	Canvas scale for compositing

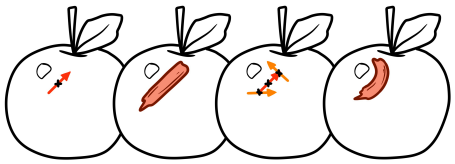


Figure 4: *Curve fitting using the orientation field*

3.1. Base Layer Rendering

Similar to an artist's workflow [HE04], the base layer is represented by coarse strokes layered atop each other to approximate the background of an image. We use three passes of stroke placement using texture bombing, where the brush size decreases in each pass. Fig. 3 illustrates the texture bombing process for one pass: based on a grid resolution, the initially placed brush strokes are offset using a randomized 2D vector per grid cell and rotated according to the direction obtained from a smoothed local orientation estimation [KD08]. The orientation field is sampled again at each end of the rotated brushstroke (Fig. 4). The difference between these orientation samples and the brushstroke orientation then determines the curvature of the brushstroke used in Sec. 3.5.

The grid size is proportional to the brush size parameter σ and adjusted on a per-pass level, yielding σ_i. We found that three passes with grid sizes $\sigma_0 = 0.7\sigma, \sigma_1 = 0.8\sigma, \sigma_2 = 0.95\sigma$ result in sufficient background coverage.

During layer rendering, a random texture from the brush texture atlas (indexed by id_B) is used that can vary per grid cell (Sec. 3.4). The texture coordinates are warped prior to sampling in order to simulate curved brushstrokes (Sec. 3.5), color interpolation with color variation is performed (Sec. 3.4) and the result is blended (τ).

3.2. Detail Layer Rendering

Especially for the overall aesthetics of portraits, it can be important for the user to maintain facial features by using finer details during stylization. For historic reference, Fig. 6 shows an oil painting that uses different Level-of-Detail (LOD) for the background and face as well as the hands of the person. While previous approaches place details implicitly by iterative refinement, our approach allows detail

(a) Input Image I (b) Salience M_S (c) Depth M_D (d) Landmarks M_L (e) Importance M_I (f) Output O

Figure 5: *Constituents of the importance map M_I used to control the level-of-detail during painterly stylization of an input image I to an output image O. M_I is a combination of a saliency mask M_S, a transformed scene depth M_D and an optional face landmarks mask M_L.*

control using an importance map M_I that is generated by combining the following data computed during preprocessing (Fig. 5):

Image Salience M_S: Salience refers to what is noticeable or important in an image. High values of samples s_{M_S} indicate higher importance. A salience map can be computed using approaches reviewed by Borji and Itti [BI13]. In our prototype, we use the attention-based salience mask feature provided by the Apple Vision framework [App23].

Scene Depth M_D: In the case of RGB-only input image, the preprocessing stage uses MiDaS [RLH*22] to compute relational depth information based on color values (Fig. 5c). In case the depth data is provided by the device depth sensors, it is usually of lower spatial resolution than the respective color image. For it, the depth map is up-sampled to the color image resolution using joint-bilateral upsampling [KCLU07].

Landmarks M_F: Samples of the facial landmark mask s_{M_F} indicate regions that should be rendered at highest LOD, such as eyes, mouth, eyebrows, or skin in general. In our prototype, we triangulate and render landmark points provided by the Apple Vision framework.

Given the respective weights ω that can be controlled by the user, the combination of the above data into the total importance map M_I is computed by

$$s_{M_I} = \frac{\omega_D s_{M_D} + \omega_F s_{M_F} + \omega_S s_{M_S}}{\max(0.01, s_{M_D} + s_{M_F} + s_{M_D})}.$$

The values of M_I determine the probability that a stroke is being placed in a grid cell. The grid size computation is similar to the base layer. For the first detail layer, the detail amount ξ is multiplied to σ, and the second detail layer always has 2 times as much detail as the first one.

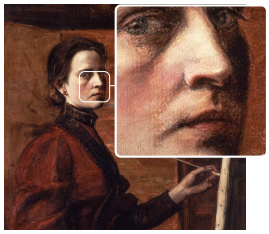


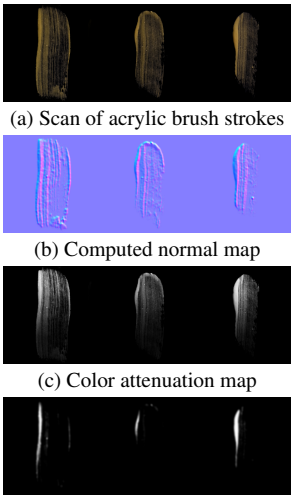
Figure 6: *LOD of face and hands compared to the background (Elizabeth Nourse, "Self Portrait").*

3.3. Tile-based Rendering & Compositing

Mobile devices often use tile-based rendering and processing to optimize performance and power efficiency by dividing the render destination into a grid of smaller regions (tiles) that are processed separately. To account for this, the stylization technique must be computed using a fixed overlap per tile to ensure seamless tile-compositing. Since texture bombing only requires access to a constant number of neighboring cells, it is therefore suited for tile-based rendering. To process some tile A of the original image, the technique needs access to section A itself as well as its original image coordinates and an overlap area A'. A' is a border around A whose width is the grid size for texture bombing multiplied by the number of surrounding cells needed in each direction. Therefore splitting an image into tiles of size A, each with processing size $A+A'$, yields the same stylized result as if processed at once.

For compositing, a user can choose between different background textures that can be randomly tiled for high-resolution output to avoid tiling artifacts [Bur19] and finally blended over the brush layers using a multiply operator.

3.4. Brush Set Texture Atlas Generation



(a) Scan of acrylic brush strokes
(b) Computed normal map
(c) Color attenuation map
(d) Smudge map

Figure 7: *Stages of brush texture atlas generation.*

For effective representation and access, each set of brush stroke variations characteristic for stylization is stored as an individual RGBA texture atlas [Wlo05]. Fig. 7 illustrates the stages of the brush texture generation process. Using scans of real-world acrylic brush strokes (Fig. 7a), a gray-scale brush texture is created. Based on this, normal maps are derived (Fig. 7b, contrast enhanced for visibility) using partial derivatives. Further, a luminance offset map is computed (combining a vertical gradient and contrast enhancement) that models the attenuation of color within a brush stroke (Fig. 7c). Using further level adjustments, a smudge map is created that models the thickness of the applied paint (Fig. 7d). The final texture atlas combines these textures: the first combines normal map (as RGB components) and gray-scale texture (as A component), the second combines luminance offset map (as R component) and smudge map (as G component). The resulting texture atlases and canvas textures are organized using a 2D texture array, in which slices are indexed during sampling (id_B and id_C). The sampling of the indexed brush texture atlas is performed according to Glanville [Gla04].

3.5. Rendering of Curved Brush Strokes

To align individual brush strokes to local image features and enable an organic look, brush strokes are rendered curved according to the smoothed local orientation estimation [KD08] derived from

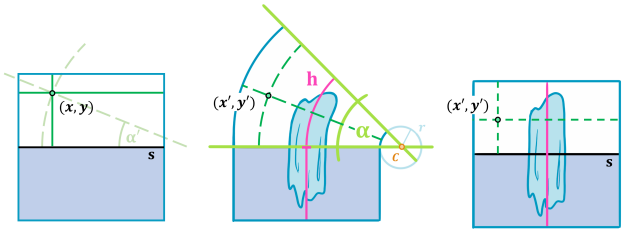


Figure 8: *Coordinate transformation for brush texture warping.*

the input image. Specifically, this can improve accuracy in image areas with curved features. Hertzmann's approach uses Splines for representing curved brushstrokes [Her98]. However, that is not applicable to texture bombing techniques. Instead, texture coordinate warping prior to sampling the brush set textures is applied.

In the following, we describe the warping of the top half of the brush texture – the other follows analogously. The warping function should fulfill three requirements: (1) the length of the brushstroke should not be affected by warping (necessary for texture bombing), (2) the transition between the two curved texture halves should be seamless, i.e., the tangent of the warped brushstroke at the seam s (see Fig. 8) should align with the stroke orientation, and (3) the function is parameterized by the angle α between the brushstroke direction at its tip and the y-axis – i.e., the local orientation estimation at this position. To achieve this, the warping function wraps the texture around a circle whose center C aligns with the texture's horizontal axis of symmetry, transforming vertical lines into arcs. All horizontal lines align with the radii of the circle and remain the same length after the transformation. In order for the center line (magenta in Fig. 8) to remain the same length (1) and the direction of the tip of the brushstroke to be defined by the input α (3), we can then derive the following equations for the radius of the circle: $\alpha/2\pi = h/2\pi(h+r)$, where the length of the arc with angle α is half the texture size h and $h+r$ is the distance between the center line and the circle center, $2\pi(h+r)$ is the full circumference. From that, $r = h/\alpha - h = h((1-\alpha)/\alpha)$ follows.

Using this radius, the transformation of the input texture coordinates (x,y) to (x',y') is computed as follows. Let α' be the angle between the x-axis and the line from (x,y) to the circle center C (see Fig. 8). The relation to y' is $\alpha'/\alpha = h-y'/h$, y' being the length of the arc with angle α' at the center line of the brush stroke. We can use trigonometric functions in the orthogonal triangle defined by (x,y), C, and α' to determine x' by $\sin(\alpha') = h-y/r+2h-x'$, where $h-y$ is the length of the opposite side of the triangle (relative to α') and $r+2h-x'$ the length of the hypotenuse.

3.6. Color Variations and Interpolation

The entire color processing is performed in L*a*b color space. To add plausible variations that would occur during the application of real brush strokes, we perform color variation and interpolation during brush rendering. First, Fig. 9 shows how the color C_O of a single brush is mixed using two colors C_A, C_B sampled from the input image I according to the value of the attenuation map α as follows: $C_O = C_A \cdot (1-\alpha) + C_B \cdot \alpha$.

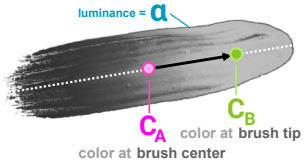


Figure 9: *Color interpolation.*

In addition to that, both of the sampled colors can be varied with respect to their tone. This facilitates the painterly impression, especially in uniform-colored image regions. Fig. 10 shows the impact of applied color variation strength μ controlling the interpolation between the original sampled color C_O and its variant $C_V = C_0 + R$, with R being a random value, thus, $C = C_O + \mu \cdot R$.

Figure 10: *Color variation.*

4. Results and Discussion

We implement our proposed technique based on iPadOS using Swift, UIKit, CoreImage, CoreML, and Metal APIs. However, the implementation method is not device-specific and can be transferred to other high-end mobile devices.

4.1. Exemplary Results

Fig. 11 shows exemplary results obtained using the proposed painterly technique. The average editing time for each image was approx. 1 min. By using different brush stroke textures (e.g., acrylic or pastel chalk) in combination with settings on stroke lengths, curvature, color variations, and blending, a multitude of different stylization can be achieved. Our interactive rendering technique provides immediate visual feedback and enables users to iterate fast on different stylization alternatives as well as to control the overall and local appearance in certain image regions such as background or foreground. By providing an effective user interface suitable for mobile devices, it facilitates casual creativity applications.

4.2. Performance Evaluation

System & Setup. We test the application performance using the following setup on two Apple iPad Pro 11" devices: (1) A1980 (1st Generation, 2018) equipped with an Apple A12X Bionic and 4 GB RAM. and (2) A2377 (3rd Generation, 2020) equipped with an Apple M1 and 8 GB RAM. For the comparison with Hertzmann et al. [Her98] and PT [LLH*21], we use a test machine with an AMD Ryzen Threadripper 1920X 3.5 GHz, a Nvidia RTX 3090, and 48 GB RAM. We perform runtime analysis using images of different resolutions (Tab. 2).

Table 2: *Input image resolutions for runtime performance measurements in pixels (px).*

Resolution	Width	Height	∑ Mpx
HD	1280	720	0.92
FHD	1920	1080	2.07
UHD-1	3840	2160	8.29
UHD-2	7680	4320	33.18
Hi-Res	9600	5400	51.84

Results. Fig. 12 shows the runtime performance results. One can observe that the runtime performance scales with the image resolution. While the overall performance depends on the respective output resolution, thus, the number of overall brush strokes placed,

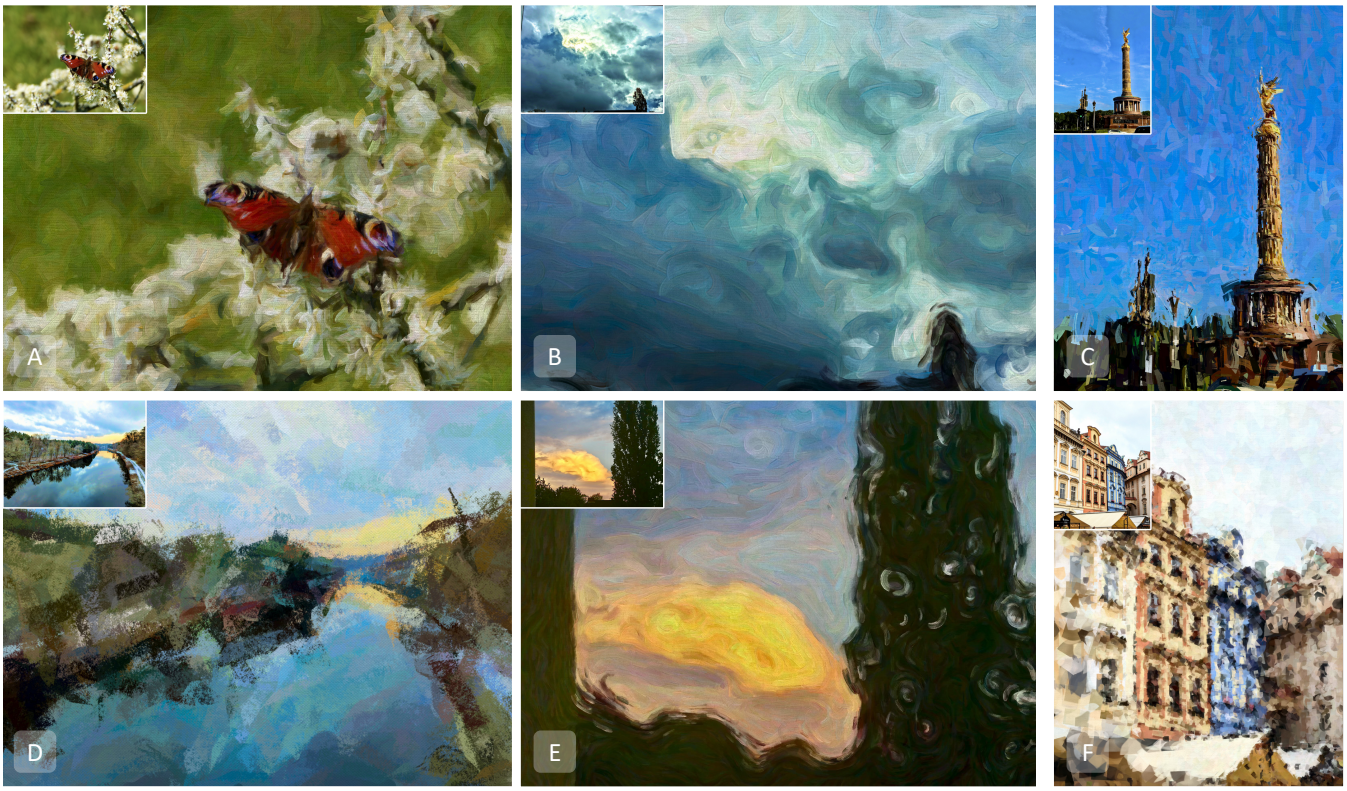
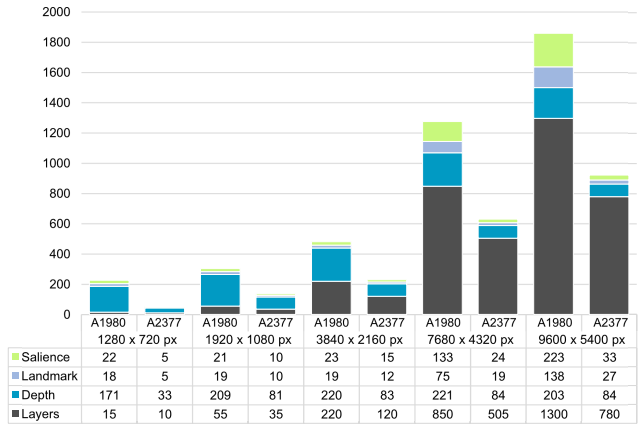


Figure 11: *Exemplary editing results obtained with our system demonstrating the use of different brushstroke textures (e.g., pastel texture (D) and oil color (A, B, E)) and sizes (e.g., coarse (B) and fine (C) strokes), curvature (e.g., low (D), medium (A), high (E)), color variation, as well as blending settings (e.g., semi-transparent (A) to opaque (C)); ranging from detailed (A) to coarse impasto (B, E), more abstract stylizations (D, F). Separate curved brushstrokes are especially noticeable in (E).*



	A1980 1280 x 720 px	A2377 1280 x 720 px	A1980 1920 x 1080 px	A2377 1920 x 1080 px	A1980 3840 x 2160 px	A2377 3840 x 2160 px	A1980 7680 x 4320 px	A2377 7680 x 4320 px	A1980 9600 x 5400 px	A2377 9600 x 5400 px
Salience	22	5	21	10	23	15	133	24	223	33
Landmark	18	5	19	10	19	12	75	19	138	27
Depth	171	33	209	81	220	83	221	84	203	84
Layers	15	10	55	35	220	120	850	505	1300	780

Figure 12: *Runtime performance in milliseconds for two mobile devices (A1980, A2377) and different image resolutions (Tab. 2).*

the impact of computing data for the importance map depends only on the input image resolution. Thus, with larger images, the impact is negligible. However, for small input resolutions, it represents a major part of the total execution time. During editing, our prototype renders at a preview resolution of 1632 px × 918 px, and any intermediate results are cached. Tab. 3 shows a superior runtime performance comparison of our approach (using the A2377 device). For Hertzmann et al., we use four layers and a minimum brush size of four. For PT, we use the slower serial prediction, as with parallel prediction, their approach runs out of GPU memory even for High Definition (HD) resolutions. When adjusting parameters, a preview is rendered that enables immediate visual feedback. The preview rendering performance (computed at approx. HD resolution) significantly improves on the respective runtime performance of the full pipeline (Tab. 3), resulting in rendering times of on average 0.1 s. This performance gain is achieved by caching of computed inputs, in particular of importance map components.

Memory Consumption. The prototypical app has a storage size of 1.23 GB on the iPad. The memory consumption of our prototype scales linearly with the amount of pixels of the input image. For an image of spatial resolution of 1920 px × 1080 px, the memory usage is approx. 330 MB. The final export step increases the memory usage to 650 MB. Thus, the application has a reasonable memory footprint. Overall main memory consumption is a limit-

Table 3: *Runtime comparison, all times in seconds (N/A denotes out-of-GPU memory).*

Resolution	[Her98]	[LLH*21]	Ours
HD	3.96	25.42	0.28
FHD	8.00	25.46	0.33
UHD-1	29.30	N/A	0.79
UHD-2	133.74	N/A	2.75
Hi-Res	153.82	N/A	4.37

(a) Input (b) Curved SBR [Her98] (c) PT [LLH*21] (d) InstructPix2Pix [BHE23] (e) Ours

Figure 13: *Comparison between the curved-brush Stroke-based Rendering (SBR) approach of Hertzmann [Her98], the feedforward stroke-predicting PaintTransformer (PT) by Liu et al. [LLH*21], the Stable-Diffusion [RBL*22]-based InstructPix2Pix [BHE23], and Ours (with default parameters for better comparability).*

ing factor regarding the maximum output resolution, i.e., A1980 achieves approx. 75 Mpx and A2377 approx. 145 Mpx.

4.3. Qualitative Comparisons

We compare our results with the original approach of Hertzmann [Her98] and PaintTransformer (PT) [LLH*21], a state-of-the-art deep-learning-based approach for stroke prediction. Our technique effectively approximates Hertzmann's while introducing a nuanced brush texture that softens its pronounced edges. Similarly to ours, PT differentiates between salient objects and the background using brushstroke size. However, it fails to orient strokes in alignment with local edges, and the superimposition of detail layers may result in grid-like patterns. Compared to Hertzmann [Her98] and PT [LLH*21], our approach may introduce blurring of some fine image features, such as trees or glasses, which can be remedied by tweaking the brush size and detail layer parameters, namely focus depth, depth threshold, and the strengths of each importance map component.

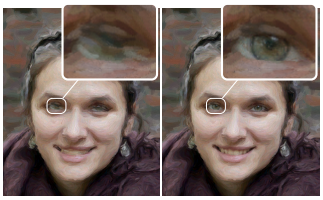
We also compare to InstructPix2Pix [BHE23], a Stable-Diffusion- [RBL*22]-based approach for prompt-based image editing (Fig. 13), and stylize the input according to the instruction "make it an impasto painting with thick strokes". While the results demonstrate a good integration of strokes with the content, creative control remains constrained; for instance, prompts to modify stroke thickness according to image content or preserve the input color scheme is not recognized by the model. Additionally, the resolution is substantially limited (below HD on our desktop test system) due to GPU memory constraints.

4.4. Limitations

A major limitation of our approach represents the brush stroke length that depends on grid resolution per layer. Since stroke placement is performed based on noise, transferring the approach to the video domain will require temporally coherent noise and further studies. Further, one can observe "chaotic" appearance in areas with high flow variance. This impacts especially the appearance of small facial features in portraits, such as eyes or mouth (Fig. 14a). To counterbalance this, an additional detail layer can be applied (Fig. 14b).

5. Conclusions & Future Work

This paper presents an art-directable brushstroke rendering technique for mobile devices that enables interactive control over the stylization by offering a wide range of parameters as well as respecting image features such as saliency, depth, and facial landmarks for level-of-detail control. To achieve interactive rendering, the proposed technique is implemented using the GPU capabilities of modern mobile devices. Our current work can be improved on conceptual and technical levels. Additional control can be achieved by enabling importance and flow maps directly modifiable by users using brush metaphors. To improve overall quality, a more elaborate stroke simulation could be added. Since our approach relies on noise distribution we plan to evaluate the impact of different noise types. Besides potential performance optimization, we plan to transfer our approach to the video domain.



(a) (b)

Figure 14: *Additional detail layer to convey the eyes.*

Acknowledgments

We thank the anonymous reviewers for their feedback to improve the paper and Frank Rupprecht for his technical support. This work was partially funded by the German Federal Ministry of Education and Research (BMBF) through grants 01IS18092 ("mdViPro") and 01IS19006 ("KI-LAB-ITSE").

References

[App23] APPLE: Vision documentation, 2023. Accessed: 2023-09-01. 4

[BHE23] BROOKS T., HOLYNSKI A., EFROS A. A.: Instructpix2pix: Learning to follow image editing instructions. In *Proc. CVPR* (2023), pp. 18392–18402. 7

[BI13] BORJI A., ITTI L.: State-of-the-art in visual attention modeling. *IEEE TPAMI 35*, 1 (2013), 185–207. 4

[BL04] BAXTER W., LIN M.: A versatile interactive 3d brush model. In *12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings.* (2004), pp. 319–328. 2

[Bur19] BURLEY B.: On histogram-preserving blending for randomized texture tiling. *Journal of Computer Graphics Techniques (JCGT) 8*, 4 (2019), 31–53. 4

[BWCS14] BENEDETTI L., WINNEMÖLLER H., CORSINI M., SCOPIGNO R.: Painting with bob: assisted creativity for novices. In *Proc. UIST* (2014), pp. 419–428. 3

[CH02] COLLOMOSSE J., HALL P.: Painterly rendering using image salience. In *Proceedings 20th Eurographics UK Conference* (2002), IEEE, pp. 122–128. 2

[FBS05] FISCHER J., BARTZ D., STRASSER W.: Artistic reality: Fast brush stroke stylization for augmented reality. In *Proc. VRST* (2005), p. 155–158. 2

[GCI08] GRUBERT J., CARPENDALE S., ISENBERG T.: Interactive stroke-based NPR using hand postures on large displays. In *Short Papers at Eurographics 2008* (2008). 2

[GKB*18] GANIN Y., KULKARNI T., BABUSCHKIN I., ESLAMI S. A., VINYALS O.: Synthesizing programs for images using reinforced adversarial learning. In *Proc. ICML* (2018), pp. 1666–1675. 2

[Gla04] GLANVILLE R. S.: Texture bombing. In *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Addison-Wesley Longman, 2004. 2, 3, 4

[Hae90] HAEBERLI P.: Paint by numbers: Abstract image representations. In *SIGGRAPH '90* (1990), pp. 207–214. 2

[HE04] HAYS J., ESSA I. A.: Image and video based painterly animation. In *Proc. NPAR* (2004), pp. 113–120. 2, 3

[Her98] HERTZMANN A.: Painterly rendering with curved brush strokes of multiple sizes. In *SIGGRAPH '98* (1998), p. 453–460. 1, 2, 5, 6, 7

[Her01] HERTZMANN A.: Paint by relaxation. In *Proceedings. Computer Graphics International 2001* (2001), IEEE, pp. 47–54. 2

[Her03] HERTZMANN A.: A survey of stroke-based rendering. In *Comput. Graph. Appl. 23* (2003), IEEE. 2

[HFL11] HUANG H., FU T.-N., LI C.-F.: Painterly rendering with content-dependent natural paint strokes. *The Visual Computer 27* (2011), 861–871. 2

[HGT13] HEGDE S., GATZIDIS C., TIAN F.: Painterly rendering techniques: a state-of-the-art review of current approaches. *Computer Animation and Virtual Worlds 24*, 1 (2013), 43–64. 1, 2

[HHZ19] HUANG Z., HENG W., ZHOU S.: Learning to paint with model-based deep reinforcement learning. In *Proc. ICCV* (2019), pp. 8709–8718. 2

[HP00] HERTZMANN A., PERLIN K.: Painterly rendering for video and interaction. In *Proc. NPAR* (2000), p. 7–12. 2

[Ise16] ISENBERG T.: Interactive NPAR: What Type of Tools Should We Create? In *Proc. NPAR* (2016), Expressive '16, p. 89–96. 1

[KCC06] KANG H. W., CHUI C. K., CHAKRABORTY U. K.: A unified scheme for adaptive stroke-based rendering. *The Visual Computer 22* (2006), 814–824. 2

[KCLU07] KOPF J., COHEN M. F., LISCHINSKI D., UYTTENDAELE M.: Joint bilateral upsampling. *ACM Trans. Graph. 26*, 3 (jul 2007), 96–102. 4

[KCWI13] KYPRIANIDIS J. E., COLLOMOSSE J., WANG T., ISENBERG T.: State of the "art": A taxonomy of artistic stylization techniques for images and video. *IEEE TVCG 19*, 5 (May 2013), 866–885. 1

[KD08] KYPRIANIDIS J. E., DÖLLNER J.: Image Abstraction by Structure Adaptive Filtering. In *Theory and Practice of Computer Graphics* (2008). 3, 4

[KS04] KOVÁCS L., SZIRÁNYI T.: Painterly rendering controlled by multiscale image features. In *Proceedings of the 20th Spring Conference on Computer Graphics* (2004), pp. 177–184. 2

[KWHO21] KOTOVENKO D., WRIGHT M., HEIMBRECHT A., OMMER B.: Rethinking style transfer: From pixels to parameterized brushstrokes. In *Proc. CVPR* (2021), pp. 12196–12205. 2

[KY15] KANG D., YOON K.: Interactive painterly rendering for mobile devices. In *Entertainment Computing - ICEC 2015* (2015), p. 445–450. 2

[Lit97] LITWINOWICZ P.: Processing images and video for an impressionist effect. In *SIGGRAPH '97* (1997), pp. 407–414. 2

[LLH*21] LIU S., LIN T., HE D., LI F., DENG R., LI X., DING E., WANG H.: Paint transformer: Feed forward neural painting with stroke prediction. In *Proc. ICCV* (2021), pp. 6578–6587. 2, 5, 6, 7

[LSD16] LINDEMEIER T., SPICKER M., DEUSSEN O.: Artistic Composition for Painterly Rendering. In *Vision, Modeling & Visualization* (2016). 2

[MH08] MUKUNDAN R., HAN C.: A Fast Algorithm for Painterly Rendering on Mobile Devices. In *Theory and Practice of Computer Graphics* (2008). 2

[Nak19] NAKANO R.: Neural painters: A learned differentiable constraint for generating brushstroke paintings. *arXiv preprint arXiv:1904.08410* (2019). 2

[NMR22] NOLTE F., MELNIK A., RITTER H.: Stroke-based rendering: From heuristics to deep learning. *arXiv preprint arXiv:2302.00595* (2022). 2

[OH11] O'DONOVAN P., HERTZMANN A.: Anipaint: Interactive painterly animation from video. *TVCG 18*, 3 (2011), 475–487. 2

[RBL*22] ROMBACH R., BLATTMANN A., LORENZ D., ESSER P., OMMER B.: High-resolution image synthesis with latent diffusion models. In *Proc. CVPR* (2022), pp. 10684–10695. 7

[RLH*22] RANFTL R., LASINGER K., HAFNER D., SCHINDLER K., KOLTUN V.: Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE TPAMI 44*, 3 (2022), 1623–1637. 4

[Sal02] SALESIN D. H.: Non-photorealistic animation & rendering: 7 grand challenges. *Keynote Talk at NPAR* (2002). 1

[SDT*16] SEMMO A., DÜRSCHMID T., TRAPP M., KLINGBEIL M., DÖLLNER J., PASEWALDT S.: Interactive image filtering with multiple levels-of-control on mobile devices. In *SIGGRAPH ASIA 2016 Mobile Graphics and Interactive Applications* (2016). 1

[SPY09] SEO S., PARK J., YOON K.: A Painterly Rendering Based on Stroke Profile and Database. In *Computational Aesthetics in Graphics, Visualization, and Imaging* (2009). 2

[WD12] WEXLER D., DEZEUSTRE G.: Intelligent brush strokes. In *SIGGRAPH Talks* (2012), p. 50. 1, 2

[Wlo05] WLOKA M.: Improved Batching via Texture Atlases. In *ShaderX3*. Charles River Media, 2005, pp. 155–167. 4

[WOG06] WINNEMÖLLER H., OLSEN S. C., GOOCH B.: Real-time video abstraction. *ACM Trans. Graph. 25*, 3 (jul 2006), 1221–1226. 3

[ZJH19] ZHENG N., JIANG Y., HUANG D.: Strokenet: A neural painting environment. In *Proc. ICLR* (2019). 2

[ZSQ*21] ZOU Z., SHI T., QIU S., YUAN Y., SHI Z.: Stylized neural painting. In *Proc. CVPR* (2021), pp. 15689–15698. 2

[ZZ11] ZHAO M., ZHU S.-C.: Customizing painterly rendering styles using stroke processes. In *Proc. NPAR* (2011), p. 137–146. 2

[ZZXZ09] ZENG K., ZHAO M., XIONG C., ZHU S. C.: From image parsing to painterly rendering. *ACM Trans. Graph. 29*, 1 (2009), 2–1. 2