Vision, Modeling, and Visualization (2018)
F. Beck, C. Dachsbacher, and F. Sadlo (Eds.)

Painterly Rendering using Limited Paint Color Palettes

T. Lindemeier[1] and J. M. Gülzow[2] and O. Deussen[2]

[1]Daimler Protics GmbH
[2]University of Konstanz

(a) **(b)** **(c)** **(d)**

Figure 1: *Input image*[†] *(a), paint color palette on black and white surface, extracted from the image and mixed from a set of base pigments by our method (b), painting generated by our software renderer using the paint color palette (c), painting generated by our painting machine and the given paint color palette (d).*

Abstract

We present a painterly rendering method for digital painting systems as well as visual feedback based painting machines that automatically extracts color palettes from images and computes mixture recipes for these from a set of real base paint colors based on the Kubelka-Munk theory. In addition, we present a new algorithm for distributing stroke candidates, which creates paintings with sharp details and contrasts. Our system is able to predict dry compositing of thinned or thick paint colors using an evaluation scheme based on example data collected from a calibration step and optical blending. We show results generated using a software stroke-based renderer and a painting machine.

CCS Concepts

• ***Computing methodologies*** → ***Non-photorealistic rendering;*** *Image manipulation;* • ***Applied computing*** → *Performing arts;*

1. Introduction

A variety of painterly rendering methods exist that use the full spectrum of the RGB color space as color palettes to render an image [ZZXZ09, SLKD16, Her98, ZZ10]. Many of such digital methods neglect complex color interaction and limit color composition to the additive RGB color space, which is not applicable to painting machines that use real paints, whose compositing is much more complex. Other methods which use a more appropriate color compositing do exist and are either based on the Kubelka-Munk theory [BWL04, CAS*97] or particle systems [CKIW15]. However, none of these methods have been applied to painting machines. Lindemeier et al. [LMPD15] proposed a painting machine that is able to use paint colors, but only used a simple prediction model to evaluate the effect of brush strokes based on RGB alpha compositing. This works well as long as paint colors mix linearly, which is only true for a small subset of paint colors. To handle arbitrary paint color palettes, it is necessary to integrate a more advanced paint color compositing model.

We present a system that generates paintings from input images and is applicable to digital painting systems as well as visual feedback based painting machines. The method either extracts a color palette from input images or suggests mixture recipes for manually defined colors, which can be used to mix the image palette from predefined base paints. We use the Kubelka-Munk theory of dif-

[†] Photo from https://commons.wikimedia.org/wiki/File:Sea_shore.jpg, accessed April 4th, 2018.

fuse reflection [KM31, Kub48] to predict the effect of brush strokes of a certain color, which is useful to compute the quality of brush stroke candidates. The system is updated by intermediate calibration during the painting process to adapt its compositing prediction model using data from the visual feedback.

We additionally present a new stroke distribution technique that reduces overpainting and maintains sharp object borders using an adaptive super pixel segmentation based on the SLICO algorithm by Achanta et al. [ASS*12]. This technique is also able to handle the evaluation of brush strokes created with thinned and opaque paint by using optical blending based on extracted regions.

Our contributions can be summarized as follows:

- A new brush stroke distribution method that uses an extended super pixel segmentation.
- A new brush stroke evaluation method for limited color palettes and painting machines that handles compositing of thin and thick layers of paint, based on an adaptive paint compositing prediction model and the Kubelka-Munk theory.

2. Related Work

Stroke-based Rendering: Stroke-based rendering was firstly introduced by Haeberli [Hae90] in 1990. His system makes it possible to create artistic pictures using the mouse to guide a virtual brush. A general scheme for stroke-based rendering was later published by Hertzmann [Her98]. His method starts out with large brushes for initial coarse painterly rendering of an image and add details by decreasing brush stroke size in subsequent steps. More recent painterly rendering methods [ZZXZ09, ZZ10, ZZ11] segment input images into foreground and background, as well as different object classes using a semi automatic approach. A brush texture dictionary was collected from artists and was used to select textures for the rendering process according to object classes. An overview of painterly rendering and stroke-based approaches is given in [Her03, HGT13, KCWI13].

Paint Simulation: Curtis et al. [CAS*97] developed a digital painting system simulating the effect of watercolor. They used the Kubelka-Munk theory of diffuse reflectance to simulate the composition of pigments. Baxter et al. [BWL04] proposed a digital painting system that offers users to manipulate a digital canvas with virtual brushes and oil-based paint colors. They estimated scattering and absorption of base color pigments using a spectrometer and solving a least squares problem. The interaction of the paint colors is computed in real-time making use of graphics hardware. Chen et al. [CKIW15] propose a particle system that was fully implemented with CUDA, which models wet-wet mixture of paint quite realistically in real-time. Recently, some researchers proposed approaches that decompose images in pigment and thickness channels [TDLG17, AMSL17], for later recoloring and further image editing. An image color palette is extracted from an input image and each color present in the image is decomposed into a weighted combination of base pigments. Gatys et. al. [GEB15, GEB16] developed a machine learning approach that uses pretained convolutional nerual networks to transfer the artistic style of one image to a photography, while preserving the original content. All of the above mentioned methods are only used in digital painting systems that still require a user to create the artistic images, and none of them applies the color interaction to real systems, such as painting machines.

Painting Machines: Jean Tinguely (1925-1991, see [Wik17d]) was one of the first artists to build painting machines to create complex but mostly random patterns. He followed a tradition from the 19th century, when people were fascinated by mechanical apparatuses. Harold Cohen [Coh17] built a plotter with the ability to paint abstract paintings. His project, known as "AARON" is regarded as the most important painting machine in contemporary art. Other early artists such as Frieder Nake [Wik17b] used the upcoming pen plotters in the 1960s to create artistic graphics. Wanner [Wan11] built a plotter based on LEGO Mindstorms and explored the question if such a setup is capable of producing aesthetic results that go beyond the control of the programmer. Today, a number of artists use painting machines, Ben Grosser [Wik17a] and Holger Baer [Bär17], Ken Goldberg, [Wik17c] for creating abstract paintings. Specialized plotters such as Vangobot [KM12] are able to create colorful paintings today, but none of them uses a feedback mechanism. Vangobot uses a sophisticated color mixing machine and applies paint directly on a canvas like an inkjet printer. Tresset and Fol Leymarie [TL12, TFL13] created a robotic installation that is able to create portrait sketches of people. Tresset built several versions of Paul, which are shown at many exhibitions. Visitors are drawn by up to five robots in parallel from different perspectives. While this system creates sketches that have an own artistic style, it is however limited to sketches and the focus of the project lies more on the performance and exhibition. Since 2016, van Arman has been developing multiple systems under the name cloudpainter [vA16], which use machine learning algorithms to generate paintings after input photos. Cameras are used to analyze the current state of a painting. The first visual feedback based painting system, e-David [DLPT12, LPD13, LMPD15, LSD16], consists of an industrial robotic arm and creates paintings and drawings of input images using visual optimization. It can adapt to varying painting styles, tools, surfaces and paints. A variety of painterly rendering algorithms have been adapted to work with the feedback loop approach, in which the robot periodically takes a photograph of its progress and determines which actions to take based on that feedback.

3. Overview

Our system consists of multiple parts. First, our system extracts RGB color palettes from images based on an altered version of the approach proposed by Aharoni-Mack et al. [AMSL17]. Our mixing algorithm then computed mixture recipes for target colors in the RGB palette. It outputs mixture weights that can be used to manually mix a paint color from base pigments that are measured in a previous step. This is done for all colors in the RGB palette, resulting in a paint color palette that is passed to the painting algorithm.

This painting algorithm, which is applicable to software or hardware painting systems, uses the paint color palette to painterly render a given input image based on visual feedback optimization. The underlying stroke placement method identifies brush stroke candidates using an adaptive super pixel segmentation. An evaluation

scheme based on optical blending and the segmentation is used to compute the quality of the identified brush stroke candidates. This is done by using data retrieved from a calibration process, that estimates the effect of brushes and paint colors of each color in the palette.

To generate paintings for our results we use two different setups. We use the system proposed by Lindemeier et al. [LMPD15]. Their system consists of an industrial robot wielding brushes, has up to 24 pots for paint colors, 5 brush slots and is connected to a camera that takes images of the canvas that are color corrected. This system offers a general framework for visual feedback based painting algorithms and we are able to directly apply our method. Since the visual feedback system uses polarization filters to reduce specular highlights, which is needed for our paint color compositing model based on the Kubelka-Munk theory.

As software renderer we use an approach that is inspired by the work of Zeng et al. [ZZXZ09], where previously collected brush textures are tinted with a target color and mapped to the canvas along a spline curve, interacting with the canvas through alpha compositing. We extend this approach to Kubelka-Munk compositing, where brush color and canvas color are blended using Equation 1. We acquired brush stroke textures by using a simplified approach of the method of Lu et al. [LBDF13]. We let the painting machine paint randomly generated strokes with black paint and different brushes on a white canvas. We then took pictures of the strokes and cut out and parameterize the brush strokes manually. For each of the strokes, the brush type used, the resulting maximum width, and the original stroke path are stored. Stroke spine and texture coordinates are calculated according to Lu et al. [LBDF13]. We then built a library similar to Zeng et al. [ZZXZ09] that offers a nearest neighbor search to select brush stroke textures according to a given brush size and shape of the stroke path.

4. Estimating Base Paint Colors

In order to produce high quality renderings of images based on real paint, it is necessary the create a system that is able to extract and compute mixtures of paint and predict the interaction of paint when composed. Since this is highly nonlinear and difficult to achieve, we need to identify algorithms and methods that address the interaction of paint. First, we need to identify base paint colors that the system can use as a foundation to compute new mixtures, since this is also what artists usually do. Before artists create palettes, they define a set of base pigments [CAS*97, AMSL17, TDLG17], which are later used to mix new colors. We therefore need to measure paint pigments and we use a model that is widely used to represent paint compositing.

The Kubelka-Munk diffuse reflection theory [Kub48, KM31] is commonly used to predict the diffuse reflectance of paint on a background substrate. Since our camera system minimizes the amount of specular reflectance we can use this theory to predict the result of a paint layer applied to a given surface. The interaction is usually computed for multiple wavelengths, but since we are limited to a RGB-camera, we only consider red, green and blue reflectance. The reflectance of a single layer of paint composed on a background can be expressed by:

$$R_{KM}(K,S,d,R_0) = \frac{1 - R_0(a - b\coth(bSd))}{a - R_0 + b\coth(bSd)}, \qquad (1)$$
$$\text{where } a = 1 + \tfrac{K}{S} \text{ and } b = \sqrt{a^2 - 1}$$

with R_0 as the reflectance of the background substrate and K and S as absorption and scattering of the paint mixture. The mixture of paint can be modeled as a linear combination of their scattering and absorption coefficients [Dun40]:

$$K_{mix} = \sum_{i=1}^{n} c_i K_i, \qquad S_{mix} = \sum_{i=1}^{n} c_i S_i. \qquad (2)$$

Estimating paint colors is usually done by applying layers of known and constant thickness of paint on a perfect black and white background. The scattering and absorption is then computed using the measured reflectance values of each sample composed on the black and white surface and an inversion of the Kubelka-Munk equations [LÏ3, CAS*97]. A problem with this approach is that it assumes uniform thickness across the measured surface to simplify the equations. Due to the nature of brushes used in our painting setup, it is not possible to apply paint of uniform thickness everywhere. An additional problem is that we cannot create perfect black or white surfaces, which are crucial for correctly measuring absorption and scattering coefficients. Another approach is to apply thick layers of each paint and mixtures of them, where the mixing ratio is known, that completely block the reflectance of the underlying substrate and then measure the resulting reflectance using a spectrometer [Cen13, BWL04]. However, this method is difficult to use with our setup and hardware. Many samples of various mixtures are needed and their mixing ratio needs to be measured exactly in order to robustly solve the least squares problem.

We therefore use a different approach to measure the coefficients. We cover a surface using all the paints from the base pigment set to create a spectrum of color spanning most of the spectrum of visible colors. A photo is taken using the color calibrated feedback camera and used as the background reflectance R_0 (see Figure 2 (a)). We then apply brush strokes with constant pressure profiles using each paint from the base pigment set and take another photo as resulting reflectance R_1 (see Figure 2 (b)). These pairs of before and after reflectances can then be used to build and solve a nonlinear least squares problem. We define a set of basis pigments $B = \{B_1, ..., B_{14}\}$ and each pigment B_i consist of its absorption K_i and scattering S_i. In order to measure each K_i and S_i when applying paint, we can use Equation 1. The goal is to find those values of K, S and d that minimize the color difference of the samples in R_1 and composed reflectance computed using Equation 1. For each pigment in the base palette, we solve the following optimization problem:

$$\begin{aligned} \underset{K_i,S_i,d_i}{\arg\min} \quad & \sum_p \left\| R_{KM}(K_i,S_i,d_{i,p},R_{0,p}) - R_{1,p} \right\|^2 \\ \text{subject to} \quad & 0 < K_i \le t_{upper}, \\ & 0 < S_i \le t_{upper}, \\ & d_i > 0 \end{aligned} \qquad (3)$$

with $t_{upper} = 5$ as upper boundary constraint, to avoid large values for S and K and p as the index of a currently visited pixel. We

use the ceres library [AMO] to solve this and all other optimization problems discussed in the paper. A result of this optimization can be seen in Figure 2. The solver takes about 10 seconds to find a solution for the problem on a desktop PC (Core i7 with 32GB RAM). A photo of the canvas before the optimization yielding the background reflectance R_0 is shown in (a). The base pigments applied on the background resulting in the reflectance values R_1 in (b). The reflectance values computed using K_i, S_i and d_i as estimated by the solver and Equation 1 can be seen in (c) and the base pigment palette is visualized in (d).

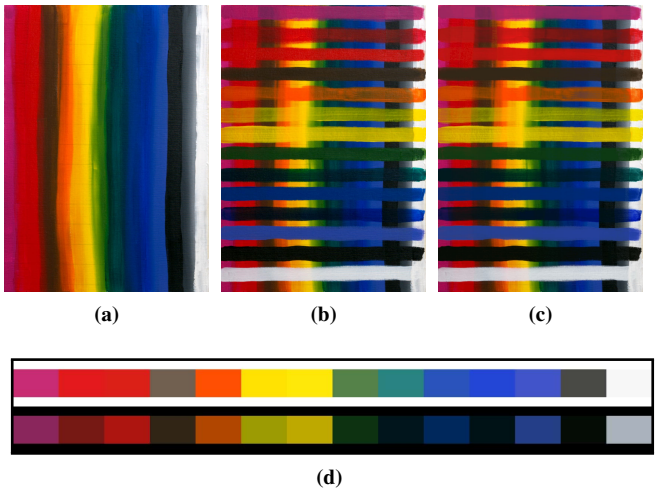


Figure 2: *Example of base pigment estimation. The background reflectance R_0 (a), the 14 paints composed on the background resulting in target reflectance R_1 (b), composition of the measured paint on the background (a) using the values of K and S estimated by the paint coefficient solver and d, the thickness values as estimated by the paint thickness solver (c). The root mean square error (in CIELab color space) of all samples in (b) and estimated samples in (c) is about 2.20. The estimated color palette is visualized in (d) composed on white and black background using $d = 1.0$ as layer thickness and Equation 1.*

Table 1 shows the estimated values for each pigment. The left column lists the paints used, second and third row list the estimates of absorption K and scattering S for the red, green and blue channels. The average root mean square error, computed in CIELab color space, of all samples is 2.20, which falls under the Just Noticeable Difference $\Delta^*_{ab} = 2.3$ [MT11] and is good enough for our needs.

4.1. Extracting and Mixing Palettes

In order to extract dominant colors from images, we use the approach by Aharoni-Mack et al. [AMSL17]. They firstly project the colors from the input image onto the ab-Plane in the CIELab color space. Then, they iteratively remove colors from the convex hull of the colors using the Douglas-Peucker algorithm [DP11]. Colors are removed as long as the desired number of colors is not reached. In contrast to their original method we add the darkest and brightest color to the palette before extracting $k-2$ colors.

Paint color	K_r	K_g	K_b	S_r	S_g	S_b	RMS
Primary Magenta	0.30	2.65	1.16	0.47	0.11	0.34	2.48
Carmine Red	0.14	3.31	2.72	0.25	0.06	0.04	2.49
Cad. Red	0.18	2.50	2.69	0.94	0.04	0.03	1.50
Raw Umber	0.96	1.14	1.32	0.07	0.05	0.02	2.19
Cad. Orange	0.00	1.83	4.68	0.75	0.27	0.00	2.17
Cad. Yellow	0.00	0.14	5.00	0.49	0.59	0.00	2.22
Primary Yellow	0.00	0.11	3.07	1.10	0.77	0.00	2.29
Leaf Green	1.22	0.78	1.40	0.01	0.07	0.02	2.15
Phthalo Green	1.93	0.75	0.77	0.00	0.02	0.02	3.12
Cobalt Blue	1.92	1.29	0.37	0.00	0.06	0.18	1.90
Ultramarine Blue	2.05	1.44	0.19	0.00	0.02	0.01	2.02
Lilac	1.61	1.47	0.29	0.06	0.17	0.43	1.57
Lamp Black	1.37	1.39	1.43	0.01	0.01	0.01	2.47
Titanium White	0.04	0.04	0.04	0.95	0.94	0.99	2.22

Table 1: *Estimated values of K, S for each measured paint. The last column shows the root mean square error of estimated values composed on R_0 and the original rgb data in R_1 in CIELab color space. ("Cad" abbreviates Cadmium)*

After having extracted dominant colors, we need to identify the paints from our base pigments that can be mixed to create the color. The mixture of paint is modeled as a linear combination of their scattering and absorption coefficients using Equation 2. If the set of weights is known, a paint can be mixed by combining the base pigments weighted by their respective concentration. The more common case is that a reflectance and transparency of a color is given and the corresponding weights for the mixture from the base pigments have to be found. In order to find mixing weights for a given target paint, a nonlinear optimization problem has to be solved. Recently, two similar methods were introduced that find mixture weights for pigments to create a target RGB color [AMSL17, TDLG17]. The corresponding mixture weights for each of those paint colors is found by solving an optimization problem that estimates the weights by composing on a white background and comparing the result to the target. We extend these approaches by introducing additional constraints that makes it suitable for our needs.

Again, the goal is to find the weights that mix a target paint from the base pigments and a thickness parameter d. These can be estimated by minimizing the distance of target reflectance R_1 and the effect of the current estimate using Equation 1 with K_w, S_w, resulting from the mixture of w and composition on R_0 using d. This optimization problem can then be defined similar to [AMSL17, TDLG17] as:

$$\begin{aligned} \underset{w,d}{\arg\min} \quad & \|(R_{KM}(K_w, S_w, d, R_0) - R_1\| \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i = 1, \\ & 0 \le w \le 1, \\ & d > 0 \end{aligned} \tag{4}$$

The nonlinear equality constraint $\sum_{i=1}^{n} w_i = 1$ can be introduced to Equation 4 as a term E_{sum} penalizing solutions that create sums of weight far from 1:

$$E_{sum} = \|w\|_{L1} - 1 \tag{5}$$

An additional term penalizes dense solution vectors of w. This helps reducing the numbers of paints involved in a mixture:

$$E_{sp} = 1 - \frac{\sqrt{n} - \frac{\|w\|_{L1}}{\|w\|_{L2}}}{\sqrt{n} - 1}, \tag{6}$$

Putting Equation 5 and 6 in 4 results in:

$$\begin{aligned} \underset{w,d}{\arg\min} \quad & \|(R_{KM}(K_w, S_w, d, R_0) - R_1\| + a_{sum}E_{sum} + a_{sp}E_{sp} \\ \text{subject to} \quad & 0 \leq w \leq 1, \\ & d > 0 \end{aligned} \tag{7}$$

with $a_{sum} = 0.5$ and $a_{sp} = 0.1$. We usually set $R_0 = 1$, which can be seen as white background. Having estimated the mixture recipe, we mix the paint palette using the base paints. We use syringes to measure the volumes and then mix the colors manually, if we use the painting machine. An example of an extracted palette can be seen in Figure 1 (b).

5. Brush and Thickness Calibration

The width of brush stroke marks can be varied by applying different levels of pressure with the brush on the canvas. Therefore, the width of brush strokes can be described as a function of pressure. The pressure is to be understood as the distance that the brush would be dipped into the canvas if it were permeable. We initially select four different brush sizes (12mm, 8mm, 4mm, 2mm) as our set of brushes used for painting. For each brush, the maximum pressure is determined in advance by hand. Minimum pressure is defined as the point where the brush bristles just touch the canvas. For each brush, the pressure to width function is determined by painting several samples of different applied pressure. We chose a sampling rate of 3mm, which resulted in about 10 stroke samples, depending on the size of the brush. The measured values now express the brush width as a function of pressure. In order to convert width into pressure information later, we fit a spline to the manually measured values and sample it at the desired location. A special width value, where the brush produces the optimal brush stroke mark, is given by the nominal size of the brush type. We define this width/pressure as b_{opt}, which is used as the main brush stroke width throughout this paper.

Before a painting process is started, we need to calibrate the effect of brushes and paints when applied on already painted layers in order to evaluate brush stroke candidates. Since the paint palette is mixed from known pigment mixtures, we do know K_l and S_l of each the respective palette colors $L = (l_0, ..., l_n)$, but not the layer thickness $d_{b,l}$ that is also needed to compute the reflectance using Equation 1. We estimate the layer thickness for each color by applying a stroke on a separate canvas and determine the layer thickness per pixel using the reflectances from before and after images R_0, R_1. The thickness map $d_{b,l}$ of a layer can be computed by solving Equation 1 for d:

$$d = \frac{acoth\left(\frac{RR_0 - aR - aR_0 + 1}{bR - bR_0}\right)}{bS}, \tag{8}$$

$$\text{where } a = 1 + \frac{K}{S} \text{ and } b = \sqrt{a^2 - 1}.$$

We solve this equation for each pixel of the before and after images of a certain brush stroke sample. We exclude values where $\Delta(R_0, R_1) < e_{min}$ where e_{min} is usually set to 0.005. This masks out regions that were not covered by the brush stroke or differences that are too small to compute and therefore negligible. It turned out that estimating the thickness map only by solving Equation 8 produced noisy results. We therefore solve a similar optimization problem to Problem 3. We fix $K = K_l$ and S_l to the corresponding palette color and only solve for the thickness map $d_{b,l}$. We initialize $d_{b,l}$ to the solution given by Equation 8. The resulting thickness maps can then later be used to simulate the effect of strokes with a certain brush and paint, in order to evaluate brush stroke candidates. An example of a thickness map estimation can be seen in Figure 3. A brush stroke was applied onto a white surface and a feedback picture I_1 is taken (a). The extracted thickness map can be seen in (b) and a simulated brush strokes applied onto a background of same reflectances as I_0 using the extracted thickness map and paint (c).

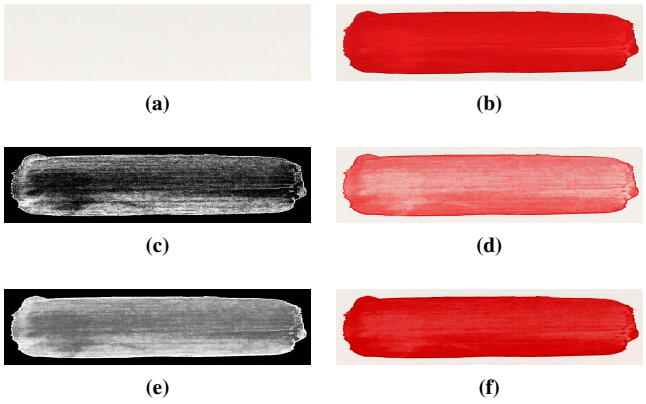


Figure 3: *Estimation of brush thickness maps. Reflectances R_0 before applying the brush strokes (a), Reflectances R_1 after applying the brush strokes (b), thickness map computed using Equation 8 (c) and the reconstructed brush stroke using the map (d), thickness map computed using our optimization approach (e) and the reconstructed brush stroke using the map (f). The results of the optimizer are much smoother and more accurate. The thickness map images are contrast enhanced and normalized for display.*

6. Painting Layers

As it is often done by artists, we paint by starting with bigger brushes and finalizing the painting with smaller brushes [LSD16]. In other words, we start with the biggest brush of our set and paint until we can achieve no improvement with this brush and then switch to the next. The input for this step consists of the current brush b and the input image that is filtered using the Normalized

Convolution Filter of Gastal and Oliveira [GO11] to get similar colored regions while sharpening strong image edges. We choose the kernel sizes according to optimal brush size b_{opt} and set the spatial kernel size $\sigma_s = f_s * b_{opt}$ and color kernel size $\sigma_c = 8.0$. The resulting image represents the target image G_b and is used for all computations involved in painting a brush layer. The orientation field that guides the brush strokes paths is computed similar to the method proposed by Kyprianidis et al. [KK11]. We apply the same smoothing as for the input picture to the resulting structure tensor field to create smooth orientations.

We repeatedly extract regions of sizes according to the optimal brush size b_{opt} using information derived from feedback and target image and generate brush strokes based on that information that are then later realized by the renderer. The regions are used to identify locations for brush stroke candidates and to evaluate their quality using optical blending by comparing the mean colors of regions. Regions are also used to find locations on the canvas that are already very close to the target picture and do not need further processing. This can also be used to determine if the painting process should be terminated or if the next brush should be selected. We discard all regions whose mean difference of color is below 0.8 and exclude them from the painting process. They will be marked as invalid for brush stroke generation. If no region is marked for painting, the system will switch to the next smaller brush.

6.1. Feedback Difference

At every iteration i of brush layers, we firstly query a feedback image from the the used renderer. We then compute the distance map $D_{b,i}$ from the resulting feedback image $F_{b,i}$ to the given target image G_b by:

$$D_{b,i} = \Delta(F_{b,i}, G_b) \tag{9}$$

using the CIEDE2000 color difference Δ^*_{00} [SWD04]:

$$\Delta(p_0, p_1) = \begin{cases} \frac{\Delta^*_{00}(p_0,p_1)}{D_0} & \text{if } \Delta^*_{00}(p_0,p_1) < D_0 \\ 1 & \text{else} \end{cases} \tag{10}$$

with $D_0 = 100$, which brings the color difference in the range from 0 to 1 and cuts of high differences where Δ^*_{00} is too large. The difference image indicates regions where more brush strokes need to be applied and where the current state of the canvas is close to the target image.

6.2. Clustering Regions

We use the SLICO super pixel segmentation, which is a parameter free extension of the simple linear iterative clustering, called SLIC algorithm [ASS*12]. This is useful to evaluate local regions on their current difference to canvas and to help to decide what paint should be used in what area. We extend the algorithm by incorporating an additional distance weight used by the clustering to group regions of similar color difference. The parameter free algorithm SLICO [ASS*12] in its original form is based on the distance function:

$$D_{SLICO} = \sqrt{\left(\frac{d_c}{m_c}\right)^2 + \left(\frac{d_s}{m_s}\right)^2} \tag{11}$$

with d_c as the Euclidean distance of the pixel colors in CIELab color space, d_s as the Euclidean distance of the pixel positions and their corresponding maximal distances m_c, m_s of the previous iteration inside the super pixel, to normalize the distance function. This decreases the compactness of the segmentation in regions where the color distance varies significantly, mostly in textured, detailed regions.

This method can be used to partition the image in smaller regions, but the algorithm will always result in the same segmentation since only the target image is included in the process. This results in strokes that will be painted on top of the strokes of previous painting iterations and results in a lot of overpainting. To prevent this, we include the difference map $D_{b,i}$ of canvas and target to force the segmentation to group locations of similar color in the target image $G_{b,i}$ as well as similar values in the difference image $D_{b,i}$. In addition, we replace the initial distribution of the spatial means based on a grid by weighted Poisson disc sampling. The weights in that case are the values of the difference map $D_{b,i}$. This ensures that the regions are initialized with regions of high differences. The new distance function for the clustering including the difference map can be defined as follows:

$$D^* = \sqrt{\left(\frac{d_c}{m_c}\right)^2 + \left(\frac{d_s}{m_s}\right)^2 + \left(\frac{d_e}{m_e}\right)^2} \tag{12}$$

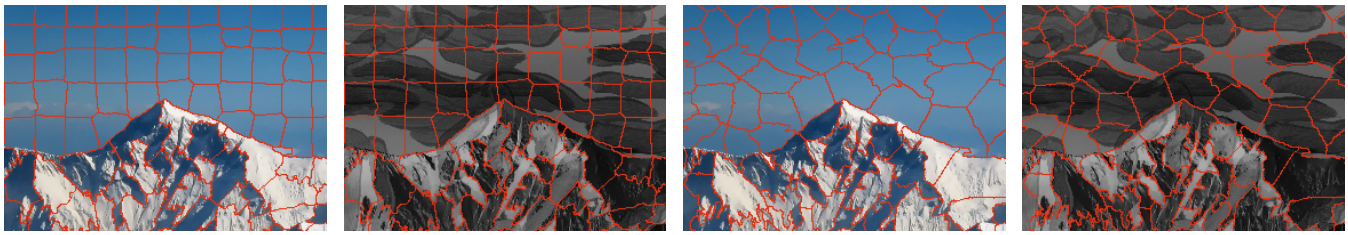
with d_e as the difference of the value of the difference map being currently processed and the mean difference value of the corresponding super pixel and m_e as the maximum difference in the super pixel from the previous iteration.

Including the difference term into the cluster distance function gives the advantage that the segmentation tries to cluster regions of similar difference, where strokes need to be placed, while aligning the clusters to image edges. This means that regions will always group according to the distance of canvas and target, but will be punished when grouping over image edges. An example segmentation can be seen in Figure 4. The segmentation of the target image G_b, i (a) and difference map $D_{b,i}$ (b) using the standard SLICO method. Our approach can be seen in (c) and (d). Our method groups regions of similar differences while aligning also to image edges. The mean area of each region can be controlled by choosing a corresponding number of regions or seed points k used to initialize the clusters using $k = s_b \frac{N}{b_{opt}}$, with N as the number of pixels of the whole image and b_{opt} as the optimal brush size as described in Section 5 and s_b as scaling parameter that can be controlled by the user to scale regions as wanted. This parameter can be used to increase the size of the neighborhood of brush strokes for the optical blending evaluation described in Section 6.3.1. We set $s_b = 1.5$ for all results created using thinned down paint colors.

6.3. Generating Brush Strokes

For all the regions we identified in the previous step, we now need to generate strokes that improve each region. In other words, we are looking for the brush stroke configuration that brings the canvas in this region closer to the target image. For this we have to determine the optimal length, width and paint color of the brush strokes.

The brush width used when drawing a stroke must be selected



(a) (b) (c) (d)

Figure 4: *Segmentation of the target image (a) and difference image (b) using the original SLICO algorithm. Segmentation of the target image (c) and difference image (d) using our adapted version. Our approach groups regions of similar color and similar difference values.*

according to the size and shape of the cell. Brush strokes should paint over as few adjacent regions as possible orthogonally to the line direction. This ensures that contrasts at the segment boundaries are maintained and enhanced. It also minimizes overpainting of objects smaller than the minimum possible stroke width of the currently selected brush. This is particularly noticeable in thin segments such as branches of trees.

Based on the distance transform proposed by Felzenszwalb and Huttenlocher [FH12], we determine the point within the region which has the largest distance to the edge of the cell. This point defines the center of the maximum inscribing circle. The radius is determined by locating the point at the edge of the cell closest to the center. The calculated radius of the brush stroke is now converted to pressure information by the pressure width function described in Section 5.

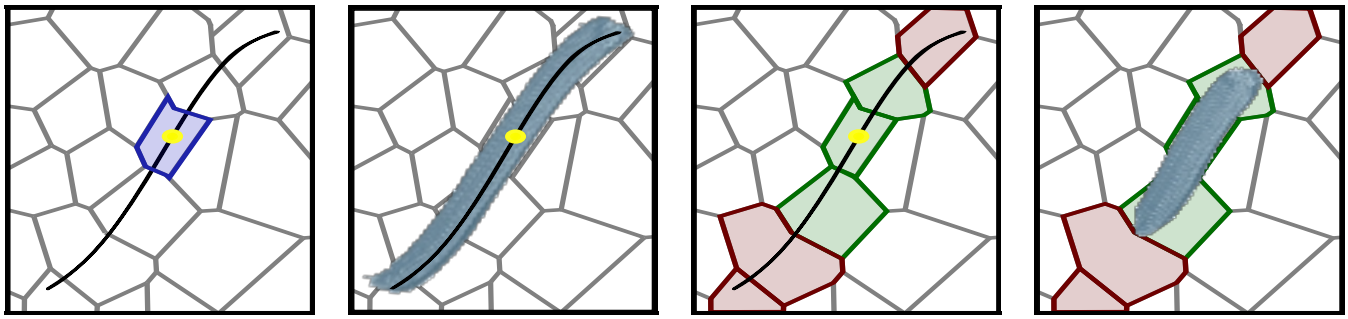
6.3.1. Evaluation of Brush Stroke Candidates

After calculating the seed point and width of the stroke, we use the orientation field to determine a path of a maximum length of $l_{max} = 30 * b_{opt}$ and step size of b_{opt} using the vector field integration following the approach by Hertzmann [Her98]. We stop the path prematurely if we land in a region already visited or marked as converged and if, in addition, a minimum line length $l_{min} = 5 * b_{opt}$ is exceeded.

Having generated a path of maximal length (Figure 5 (a)), we now need to determine the paint color, which reduces the distance from canvas to target. For this we can use the brush effect from Section 5. For each paint color in our palette, we virtually render the stroke using the corresponding thickness map onto a copy of the current canvas reflectances resulting from the feedback picture using a digital renderer instance as described in Section 3. We then identify the best color from the current region by comparing the differences of the mean region colors from target, original canvas and the simulated renderings of each color. We use Equation 10 to compute the differences. The best paint color is the one whose mean color is closest to the average mean color of the region in the target image (Figure 5 (b)). In order to estimate the length of the stroke, we iterate through all regions affected by the stroke path. As long as the paint color, resulting from the evaluation in the previous step for the seeding region, is improving the canvas, we continue. If the selected color is not improving, we stop the path at the current region (c). The final stroke left after this evaluation can be seen in (d) and is added to the strokes to be realized later. All regions affected by the final stroke path are excluded from the evaluation of other regions and strokes. Using this evaluation scheme, which computes the quality of brush strokes through optical blending by averaging regions painted using example data during the evaluation step, it is possible to handle opaque and thinned paints without further adjustments.

7. Results

Figure 6 shows two paintings generated from the input image shown in Figure 1 (a). The color palette was extracted from the image and can be seen in Figure 1 (b). The paint color palette was manually mixed according to the mixture recipe (Table 2) computed by our mixing algorithm. We used 9 parts thinning medium and one part pigments to thin down each paint. The result generated using our software renderer is shown in Figure 6 (top). The result created by our painting machine (bottom) comes close to the quality of our software rendered version. It took about 20 hours to finish the painting. We used the same parameterization for rendering both versions, the only difference lies in the rendering device. The slight differences in colors at the mountain and sky result from inaccuracies in manually mixing the paint for the robot and brushes, that are not properly cleaned during the painting process and therefore taints other colors in the paint pots used by the robot. This result shows that our method is able to create new colors from the palette colors by dry-compositing semi-transparent layers of paint on top of other layers. Two examples using different thickness of paint colors are shown in Figure 7. We manually selected a RGB palette (b) consisting only of white, black and red, green, blue pigment mixtures that we have estimated using our mixing algorithm. The input image (a) is painterly rendered using the palette by our software renderer and is shown in (c). Rendering took about 5 minutes. A result generated by our painting machine is shown in (d). We used the same parameterization as for the software result with the painting machine. The painting process lasted for about 5 hours. Our region-based stroke placement and evaluation strategy is still able to achieve results that resemble the input closely using the optical blending approach. We set the region area scale $s_b = 5$ for the result shown in (d) to increase the area of effect of the optical blending in the brush stroke evaluation. Although one might argue about the aesthetics of both results, they still resemble the input picture quite well when viewed from the right distance, although they



(a) (b) (c) (d)

Figure 5: *Estimation of brush stroke quality and length: current processed region (blue), corresponding seed point (yellow) and generated stroke path (black) (a), selecting best brush stroke paint color regarding the current region (b), identifying those regions, where the chosen color improves each affected region (green), in a forwards and backwards manner, starting from the seed region (c), the final stroke left after discarding regions (red) where the chosen color does not improve the canvas (d).*

Primary Magenta	0.20	0.03	0	0
Carmine Red	0.28	0.01	0	0.01
Cad. Red	0.32	0	0	0
Raw Umber	0	0	0	0.06
Cad. Orange	0.13	0	0	0
Cad. Yellow	0	0.21	0	0
Primary Yellow	0	0.70	0	0
Leaf Green	0	0	0	0.13
Phthalo Green	0.01	0	0	0.23
Cobalt Blue	0.00	0.01	0	0.09
Ultramarine Blue	0.01	0.04	0	0.20
Lamp Black	0.04	0	0	0.28
Titanium White	0	0	1.00	0

Table 2: *Mixing recipe for the pure paint color palette used to generate the paintings shown in Figure 1 and Figure 6.*

have been created with only 6 colors without further mixing. Some differences can be seen, especially at the right cheek. These differences result from slight variations in paint color or brush marks, as well as the change of the used paint colors during the painting process, due to unintended mixing on the canvas or bad brush cleaning. However, the painting still comes close to its goal due to the nature of the visual feedback optimization that incorporates these effects implicitly.

Another result by our painting machine is shown in Figure 8. Here, we just used Lamp Black, Titanium White and thinning medium to mix the palette. Although we did not use the smallest available brush, our algorithm was still able to achieve details at the gripper or the eyes by varying the applied pressure according to our pressure width function, resulting in thinner brush strokes.

8. Conclusion

We proposed a system that generates paintings, extracts the color palette and determines the appropriate mixing ratios to mix the palette from predefined base pigments. In addition, we presented a stroke distribution technique that maintains contrast by adaptively segmenting regions, which reduces over painting, and allows for a novel evaluation of stroke candidate quality. Furthermore, sample data is generated for stroke evaluation, which can be used for the evaluation of stroke candidates on the basis of the Kubelka-Munk theory.

Our least squares solver that estimates thickness values of brush strokes in the calibration step (see Section 5) tends to compute varying values if the background is not homogeneous. We can avoid that by only using single colored surfaces for calibration. Our painting machine is currently not able to adapt to change of paint color over time. Brushes are sometimes not cleaned well enough and mix in other colors when dipped into corresponding pots. We tried to recompute the absorption and scattering coefficients for each palette paint color during the painting process using the painting's reflectance data as captured by the feedback camera, but our estimator (see Section 4) suffers if there are not enough samples spanning a wide range of different reflectances.

Future works will encompass incorporating wet-wet paint interaction in addition to dry compositing presented in this paper. This will be a very challenging problem and can be addressed in various ways, but we think that this results in much more aesthetic paintings and new opportunities of creating artistic imagery. We also like to investigate machine learning techniques for predicting paint compositing effects and addressing the paint's change over time.

References

[AMO] AGARWAL S., MIERLE K., OTHERS: Ceres solver. http://ceres-solver.org. 4

[AMSL17] AHARONI-MACK E., SHAMBIK Y., LISCHINSKI D.: Pigment-based recoloring of watercolor paintings. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering* (New York, NY, USA, 2017), NPAR '17, ACM, pp. 1:1–1:11. 2, 3, 4

[ASS*12] ACHANTA R., SHAJI A., SMITH K., LUCCHI A., FUA P., SUSSTRUNK S.: Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. Pattern Anal. Mach. Intell. 34*, 11 (Nov. 2012), 2274–2282. 2, 6

[Bär17] BÄR H.: Holger Bär Digital Painting. http://www.holgerbaer.com/, 2017. [Online; accessed 05-July-2017]. 2

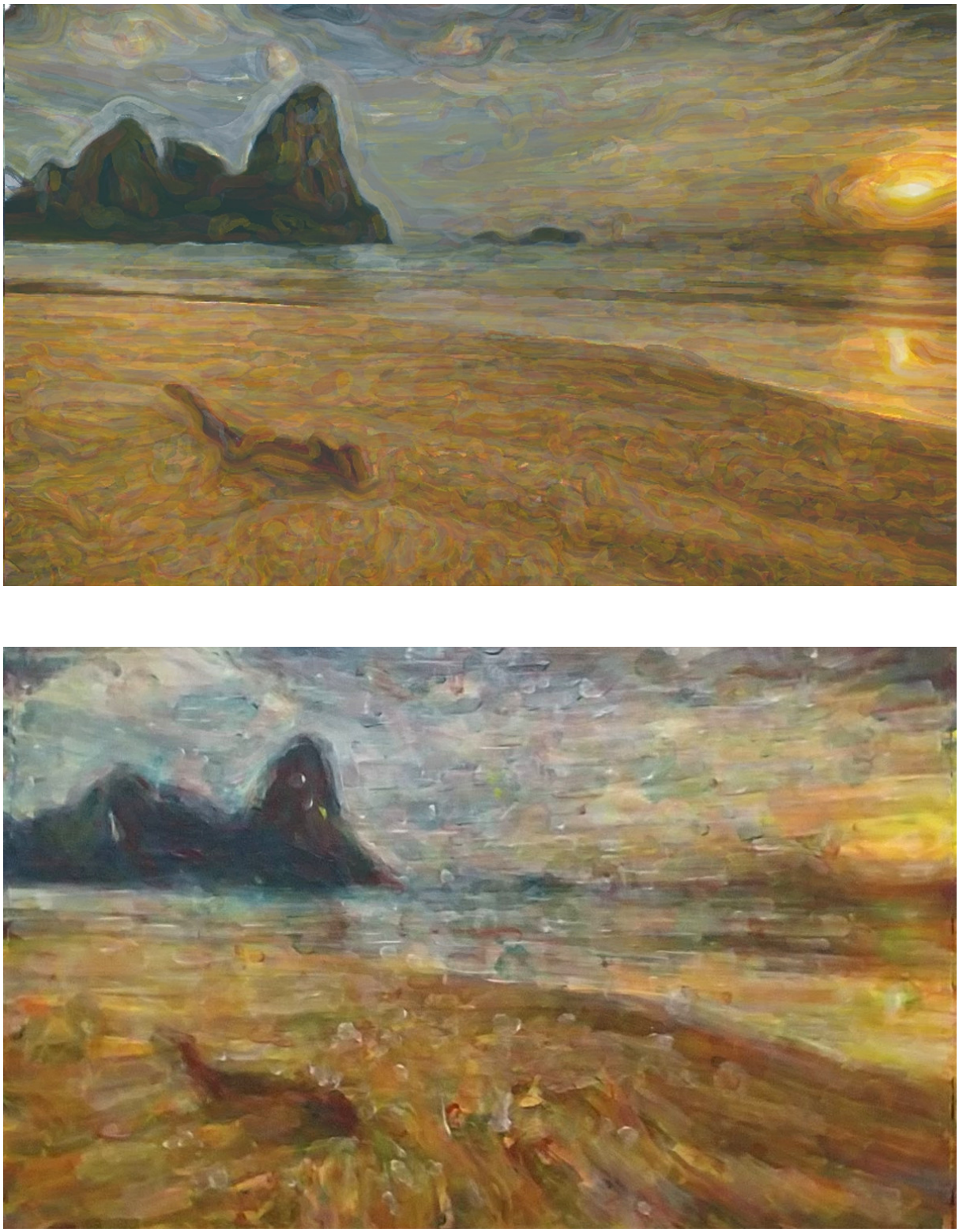


Figure 6: *Painting generated after the input picture and palette shown in Figure 1. Result from the software renderer (top). Result generated by our painting machine (bottom).*

(a) (b) (c) (d)

Figure 7: *Paintings created by our system: input picture (http://www.dannyst.com/gallery/portraits-of-strangers/, accessed April 18th, 2017) (a), manually selected palette (b), result created by the software renderer based on a the original palette and small brushes using optical blending (c), painting generated using the same settings and our painting machine (d).*

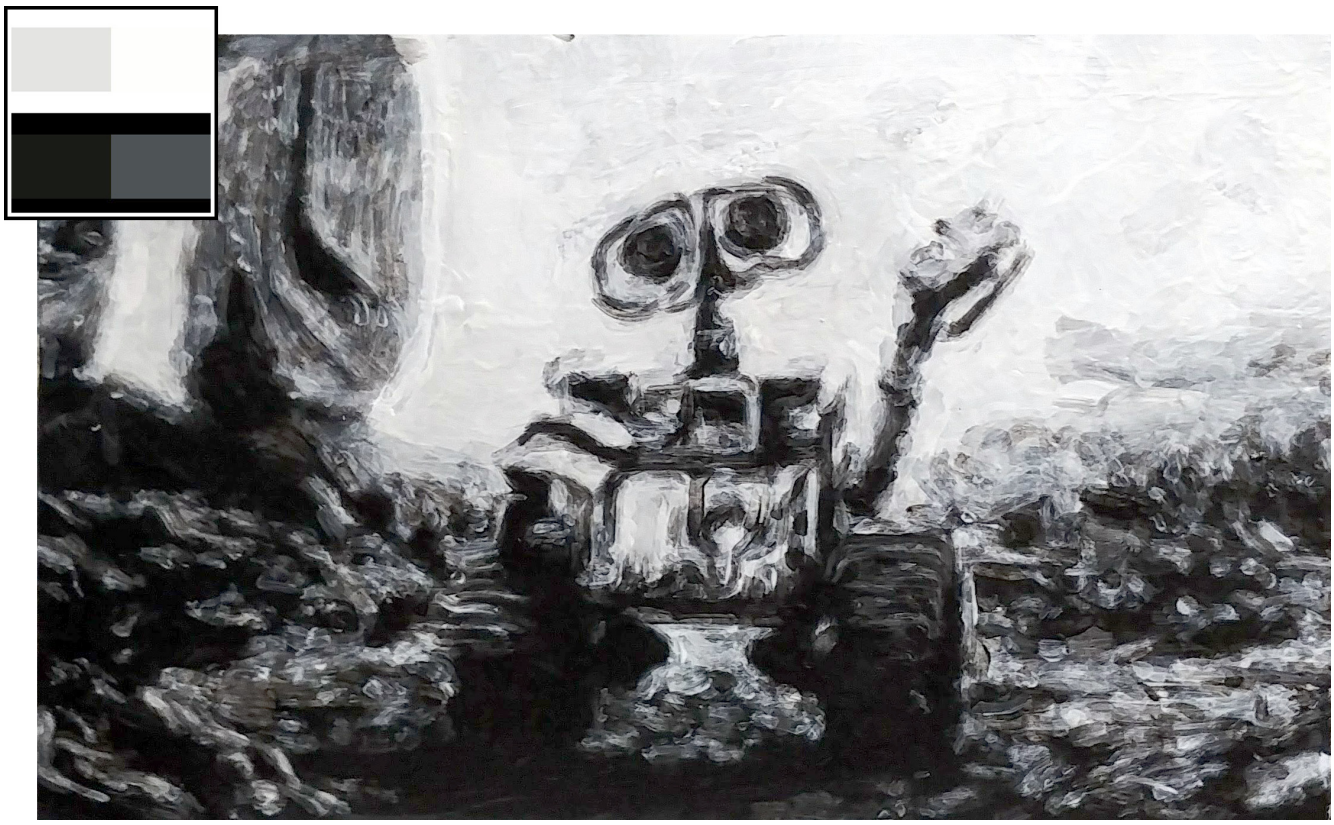


Figure 8: *Painting created by our painting machine from an input picture of "wall-e". We used a manually selected palette consisting of titanium white and lamp black (top left). We added matte medium to thin down the paint.*

[BWL04] BAXTER W., WENDT J., LIN M. C.: Impasto: A realistic, interactive model for paint. *Proceedings of the Third International Symposium on Non-Photorealistic Animation and Rendering (NPAR 2004) 1*, 212 (2004), 45–56. 1, 2, 3

[CAS*97] CURTIS C. J., ANDERSON S. E., SEIMS J. E., FLEISCHER K. W., SALESIN D. H.: Computer-generated watercolor. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97* (1997), 421–430. 1, 2, 3

[Cen13] CENTORE P.: Perceptual reflectance weighting for kubelka-munk estimation, 2013. 3

[CKIW15] CHEN Z., KIM B., ITO D., WANG H.: Wetbrush: Gpu-based 3d painting simulation at the bristle level. *ACM Trans. Graph. 34*, 6 (Oct. 2015), 200:1–200:11. 1, 2

[Coh17] COHEN H.: Aaron. http://www.aaronshome.com/aaron/aaron/index.html, 2017. [Online; accessed 05-July-2017]. 2

[DLPT12] DEUSSEN O., LINDEMEIER T., PIRK S., TAUTZENBERGER M.: Feedback-guided stroke placement for a painting machine. In *Proceedings of the Eighth Annual Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging* (Goslar Germany, Germany, 2012), CAe '12, Eurographics Association, pp. 25–33. 2

[DP11] DOUGLAS D. H., PEUCKER T. K.: *Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature*. Wiley-Blackwell, 2011, ch. 2, pp. 15–28. 4

[Dun40] DUNCAN D. R.: The colour of pigment mixtures. *Proceedings of the Physical Society 52*, 3 (1940), 390. 3

[FH12] FELZENSZWALB P. F., HUTTENLOCHER D. P.: Distance transforms of sampled functions. *Theory of Computing 8*, 19 (2012), 415–428. 7

[GEB15] GATYS L. A., ECKER A. S., BETHGE M.: A neural algorithm of artistic style. *CoRR abs/1508.06576* (2015). 2

[GEB16] GATYS L. A., ECKER A. S., BETHGE M.: Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2016), pp. 2414–2423. 2

[GO11] GASTAL E. S. L., OLIVEIRA M. M.: Domain transform for edge-aware image and video processing. *ACM Trans. Graph. 30*, 4 (July 2011), 69:1–69:12. 6

[Hae90] HAEBERLI P.: Paint by numbers: abstract image representations. *ACM SIGGRAPH Computer Graphics 24*, 4 (Sept. 1990), 207–214. 2

[Her98] HERTZMANN A.: Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 453–460. 1, 2, 7

[Her03] HERTZMANN A.: A survey of stroke-based rendering. *IEEE Computer Graphics and Applications*, 4 (2003), 70–81. 2

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

[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 Transactions on Visualization and Computer Graphics 19*, 5 (May 2013), 866–885. 2

[KK11] KYPRIANIDIS J. E., KANG H.: Image and video abstraction by coherence-enhancing filtering. *Computer Graphics Forum 30*, 2 (Apr. 2011), 593–602. 6

[KM31] KUBELKA P., MUNK F.: An article on optics of paint layers. *Zeitschrift für Technische Physik 12*, 5 (1931), 593–601. 2, 3

[KM12] KELLY L., MARX D.: The Vangobot Project. http://vangobot.com, 2012. [Online; accessed 30-April-2012]. 2

[Kub48] KUBELKA P.: New contributions to the optics of intensely light-scattering materials. part i. *J. Opt. Soc. Am. 38*, 5 (May 1948), 448–457. 2, 3

[LÏ3] LÜBBE E.: *Farbempfindung, Farbbeschreibung und Farbmessung*. Springer, 2013. 3

[LBDF13] LU J., BARNES C., DIVERDI S., FINKELSTEIN A.: Realbrush: Painting with examples of physical media. *ACM Trans. Graph. 32*, 4 (July 2013), 117:1–117:12. 3

[LMPD15] LINDEMEIER T., METZNER J., POLLAK L., DEUSSEN O.: Hardware-based non-photorealistic rendering using a painting robot. *Computer Graphics Forum 34*, 2 (2015), 311–323. 1, 2, 3

[LPD13] LINDEMEIER T., PIRK S., DEUSSEN O.: Image stylization with a painting machine using semantic hints. *Computers & Graphics 37*, 5 (Aug. 2013), 293–301. 2

[LSD16] LINDEMEIER T., SPICKER M., DEUSSEN O.: Artistic composition for painterly rendering. In *Vision, Modeling & Visualization* (2016), Hullin M., Stamminger M., Weinkauf T., (Eds.), The Eurographics Association. 2, 5

[MT11] MOKRZYCKI W., TATOL M.: Color difference delta e - a survey. *Machine Graphics and Vision 20* (04 2011), 383–412. 4

[SLKD16] SEMMO A., LIMBERGER D., KYPRIANIDIS J. E., DÖLLNER J.: Image stylization by interactive oil paint filtering. *Computers & Graphics*, 1665 (2016), 1–15. 1

[SWD04] SHARMA G., WU W., DALAL E. N.: The ciede2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. *Color Research & Application 30*, 1 (2004), 21–30. 6

[TDLG17] TAN J., DIVERDI S., LU J., GINGOLD Y. I.: Pigmento: Pigment-based image analysis and editing. *CoRR abs/1707.08323* (2017). 2, 3, 4

[TFL13] TRESSET P. A., FOL LEYMARIE F.: Portrait drawing by paul the robot. *Computers & Graphics 37*, 5 (2013), 348 – 363. 2

[TL12] TRESSET P. A., LEYMARIE F. F.: Sketches by paul the robot. In *Proceedings of the Eighth Annual Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging* (Goslar Germany, Germany, 2012), CAe '12, Eurographics Association, pp. 17–24. 2

[vA16] VAN ARMAN P.: cloudpainter. http://www.cloudpainter.com, 2016. Accessed: 2018-01-22. 2

[Wan11] WANNER A.: Building "the plotter": An aesthetic exploration with drawing robots. In *Proceedings of the International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging* (New York, NY, USA, 2011), CAe '11, ACM, pp. 39–46. 2

[Wik17a] WIKIPEDIA:. https://en.wikipedia.org/wiki/Ken_Goldberg, 2017. [Online; accessed 05-July-2017]. 2

[Wik17b] WIKIPEDIA: Frieder Nake. https://en.wikipedia.org/wiki/Frieder_Nake, 2017. [Online; accessed 05-July-2017]. 2

[Wik17c] WIKIPEDIA: Ken Goldberg. https://en.wikipedia.org/wiki/Ken_Goldberg, 2017. [Online; accessed 05-July-2017]. 2

[Wik17d] WIKIPEDIA: Tinguely art machines. https://en.wikipedia.org/wiki/Jean_Tinguely, 2017. [Online; accessed 05-July-2017]. 2

[ZZ10] ZHAO M., ZHU S.-C.: Sisley the abstract painter. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering* (New York, NY, USA, 2010), NPAR '10, ACM, pp. 99–107. 1, 2

[ZZ11] ZHAO M., ZHU S.-C.: Customizing painterly rendering styles using stroke processes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering* (New York, NY, USA, 2011), NPAR '11, ACM, pp. 137–146. 2

[ZZXZ09] ZENG K., ZHAO M., XIONG C., ZHU S.-C.: From image parsing to painterly rendering. *ACM Transactions on Graphics 29*, 1 (Dec. 2009), 1–11. 1, 2, 3