# Brush Stroke Synthesis with a Generative Adversarial Network Driven by Physically Based Simulation

Rundong Wu
Cornell University

Zhili Chen
Adobe Research

Zhaowen Wang
Adobe Research

Jimei Yang
Adobe Research

Steve Marschner
Cornell University

(a) Offline simulation (b) Real-time simulation (c) Our system

**Figure 1: Comparison of the same stroke sequence applied in different systems. (a) shows result from an offline simulation with highest fidelity. (b) is from a real-time version of the simulator with a larger step size and a less accurate solver. Discontinuity artifacts appear on the paint surface. (c) is generated in our neural network based system.**

## ABSTRACT

We introduce a novel approach that uses a generative adversarial network (GAN) to synthesize realistic oil painting brush strokes, where the network is trained with data generated by a high-fidelity simulator. Among approaches to digitally synthesizing natural media painting strokes, physically based simulation produces by far the most realistic visual results and allows the most intuitive control of stroke variations. However, accurate physics simulations are known to be computationally expensive and often cannot meet the performance requirements of painting applications.

In our work, we propose to replace the expensive fluid simulation with a neural network. The network takes the existing canvas and a new stroke trajectory as input and produces the height and color of the new stroke as output. We train the network with a dataset generated with a high quality offline simulator. The network is able to produce visual quality comparable to the offline simulator with better performance than the existing real-time oil painting simulator. Finally, we implement a real-time painting system using the trained network.

## CCS CONCEPTS

• **Computing methodologies** → **Physical simulation**; **Non-photorealistic rendering**; **Neural networks**;

## KEYWORDS

Oil painting simulation, Generative adversarial networks

**ACM Reference Format:**
Rundong Wu, Zhili Chen, Zhaowen Wang, Jimei Yang, and Steve Marschner. 2018. Brush Stroke Synthesis with a Generative Adversarial Network Driven by Physically Based Simulation. In *Expressive '18: The Joint Symposium on Computational Aesthetics and Sketch Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering, August 17–19, 2018, Victoria, BC, Canada.* ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3229147.3229150


*Expressive '18, August 17–19, 2018, Victoria, BC, Canada*

ACM ISBN 978-1-4503-5892-7/18/08…$15.00
https://doi.org/10.1145/3229147.3229150

## 1 INTRODUCTION

Over the past few decades we have seen a universal transition from traditional materials to digital tools in the industry of art and design. Digital tools are generally more convenient and efficient but they do not necessarily produce better quality for some art forms. An example is natural media painting. Existing digital painting software still cannot faithfully reproduce the fine details that are found in real paintings. Also, results generated from painting software using pre-defined algorithms usually lack the dynamic variations that artists want to control with the slightest movement of their fingers.

Many research works have been trying to bring the quality of digital painting closer to natural media with various approaches. Earlier works and most commercial software use procedural models that define brush strokes with certain functions along the stroke curve path [Hsu and Lee 1994]. Stamp-based strokes are very common in existing software [Chu et al. 2010; DiVerdi et al. 2010]. These approaches using hand-crafted algorithms are very efficient but cannot achieve the high fidelity of real brush strokes. Example-based methods [Ando and Tsuruno 2010; Kim and Shin 2010; Lu et al. 2013] collect actual photos of stroke samples and generate brush strokes by texture synthesis. Such methods can produce high quality results but do not offer the fine control of stroke variations that artists want. They often produce undesirable repetitive patterns due to limited examples. Simulation based methods [Baxter et al. 2004b; Chen et al. 2015; Chu and Tai 2005] model the interactions between brush and paint material in a physically accurate manner and produce by far the most realistic painting results while offering intuitive controllability of brush and strokes. However, accurate physics simulations are very computationally expensive. In the case of oil painting simulation, a major bottleneck is the expensive fluid solver. The pressure projection needs to be solved at every stamping interval along the stroke path, otherwise discontinuity artifacts as shown in Figure 1(b) may appear. For a long stroke with many sub-steps, even a fully optimized simulation cannot be lag-free with the highest-end consumer hardware available. Due to the strict real-time performance requirements of painting applications, one has to use over-simplified physics models, sacrifice solver accuracy or run the simulation at much lower resolution, which all lead to compromised quality in the simulated results.

In our work, we propose a stroke synthesis method that can achieve visual quality and controllability comparable to an accurate physical oil painting simulator, but at a fraction of its computing cost so that it is usable on common hardware. We use a neural network to directly synthesize the entire segment of a stroke without solving the physics of fluid dynamics. The ideal dataset to train such a network would be a library of sample painting strokes that covers enough variation in brush types, pigments, stroke motions and more. Unfortunately, such a large dataset is very difficult, if not impossible, to build by capturing images of real painting strokes. Moreover, intermediate states related to small-scale paint interactions like individual bristle movement and pressure cannot be captured with regular imaging devices like cameras. Therefore, in our work, we choose to use high quality simulation to generate our training data. We implemented a 3D volumetric oil painting simulator similar to WetBrush [Chen et al. 2015] but used a more accurate fluid solver, more brush bristles, higher canvas resolution, and a smaller time step. The simulation is not real-time any more with these modifications, but it produces much improved visual quality, as shown in Figure 1(a). We feed the offline simulator with randomized strokes recorded from artists' painting sequences to create a large painting sample dataset for network training. The input to the network includes the states of the brush, such as trajectories and pressure, and the current state of the canvas. We still use a traditional mass-spring method to simulate a 3D brush model, which is fast enough for real-time applications. The output of the network contains both the height field representation of the paint surface and the paint color as a result of pigment mixing along the stroke, both of which are then used to render the final stroke appearance.

As far as we know, this is the first work on paint stroke generation with neural networks. The main contributions of our work are:

- a novel approach to synthesizing high-fidelity painting strokes based on convolutional neural networks,
- identification of the necessary state information from physically based simulation as input and output data for neural networks to faithfully recreate realistic results and
- a real-time oil painting system using the trained stroke generator with splitting of long strokes and patch blending.

In this paper, we will show our data preparation process with a painting simulator and the network structure we used for generating both the paint height and color maps for rendering. We will compare different loss functions and design choices and their impact on the quality of generated strokes. For the final painting system, we introduce ways to eliminate discontinuities after breaking down long strokes into overlapping segments for faster network evaluation. In the results, we compare our neural network based system and the real-time simulator in terms of both visual quality and performance and show artworks created in our system by artists.

## 2 RELATED WORK

*Procedural and example-based stroke generation.* Generating brush strokes using procedural algorithms only provides a rough approximation of actual natural media painting. Stamping a brush texture along the stroke path is commonly used in commercial software like Corel Painter and Photoshop. DiVerdi et al. [2010] improved this stamping model with a dynamic brush imprint, and Chu et al. [2010] improved the stamping quality with a better color pickup algorithm. On the other hand, example-based approaches use images of real painting samples to synthesize strokes without explicitly modeling the mechanism of brush-painting interaction. An earlier work by Hsu et al. [1994] deforms an arbitrary image to fill a stroke curve. Schretter [2005] and Ritter et al. [2006] formulate painting with image samples as a texture synthesis problem. Kim et al. [2010] and Ando et al. [2010] use real painting examples to stylize user input lines. Lu et al. [2013] advanced this method to support smudging and mixing behaviors. Lukac et al. [2015] used orientation matching to paint with directional texture samples. Roveri et al. [2015] used a meshless representation to synthesize general repetitive structures in 3D, which could potentially be applied to volumetric oil painting.

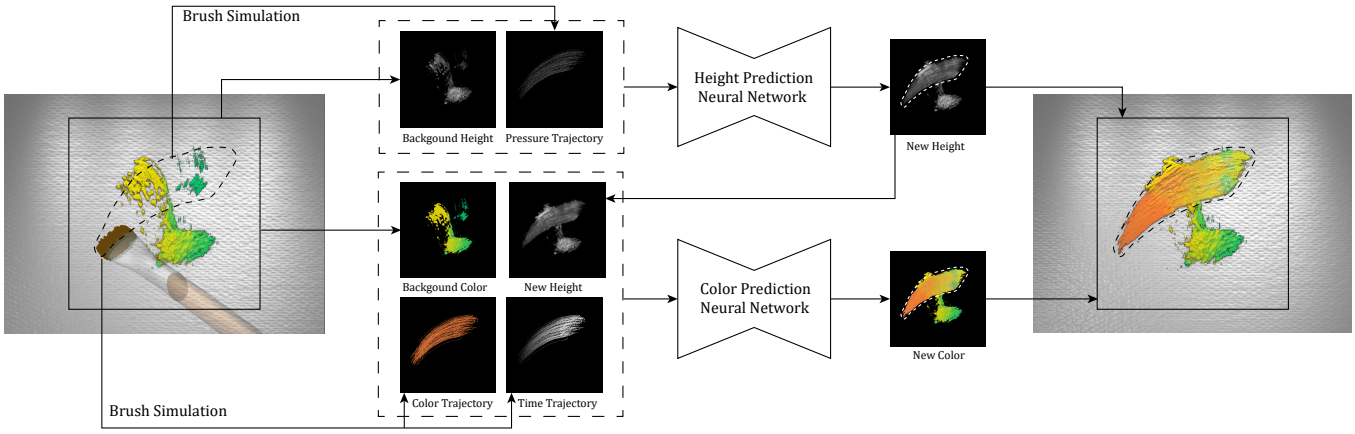



**Figure 2: An overview of our interactive oil painting system. As the user draws a new stroke on the canvas, two neural networks take relevant inputs and generate the new height map and color map respectively. Then the new height map and color map are sent back to the canvas to render the new paint distribution. The height map prediction network takes the background height and the trajectory of the new stroke as input. The color map prediction network takes the background color, the new stroke color trajectory, the new height map and a time channel as input.**

The drawback of example-based approaches comes from the fact that the richness of the painting result is often limited by the texture sample library and the stroke variations resulting from interaction with a dynamic brush model cannot be faithfully reproduced.

*Physically based painting simulation.* Baxter et al. [2004; 2004b] developed one of the earliest simulation-based painting systems by simulating oil paint as a 2.5D height field with the shallow wave equations and modeling the brush with a mass-spring skeleton and a subdivision surface around it. Later they also extended the work to support 3D paint with a volume-of-fluid method [Baxter et al. 2004a], but at very low resolution due to the limited computing power. Chu et al. [2005] simulated realistic watercolor painting based on the lattice Boltzmann equation, and they used a skeleton controlled brush simulated under an energy minimization framework [Chu and Tai 2002]. Chu et al. [2010] also found that a canvas-aligned pickup map and temporary canvas snapshot can be used to improve the sharpness of color streak details. Besides the paint liquid, the brush is also becoming more realistic with physical simulation. Xu et al. [2002] modeled a brush with multiple hair clusters. DiVerdi et al. [2010] modeled individual bristles as discrete elastic rods. Chen et al. [2015] modeled a brush with thousands of bristles and also the interaction among the bristles. They also simulated the two-way coupling of bristles and 3D volumetric oil paint to produce the most realistic oil painting result to date. However, due to extensive simulation workload, their system can only achieve acceptable performance on high-end GPUs. In our work we adapt their brush and paint simulation model as our training data generator.

*Generative models with neural networks.* Goodfellow et al. [2014] proposed generative adversarial networks (GANs) that consist of two networks jointly trained with a minimax objective: one generating samples from noise while the other discriminates generated samples from real samples. Radford et al. [2015] extended GANs using convolutional networks to generate real-world image samples that demonstrate the great potential of GANs for synthesizing photorealistic images and inspired a series of follow-up works. The sharpness and high-frequency detail of images generated by GANs are highly desirable for synthesizing oil paint strokes. Wang and Gupta [2016] attempted to bring image formation into generative networks and proposed a two-layered GAN that first generates a normal map and then generates appearance conditioned on the generated normal map. Liu and Tuzel [2016] proposed a coupled GAN that can simultaneously generate image samples from two different but correlated domains, such as depth maps and color images. Our network generates a height map and a color map in sequence conditioned on input images that describe the states of the canvas and the brush. Dosovitskiy et al. [2015] designed convolutional networks with upsampling layers to generate images from graphics code and it showed that when the input provides sufficient information to define the output image, reconstruction loss (like L1) can be used to generate well-structured images. Yang et al. [2015] and Tatarchenko et al. [2016] trained encoder-decoder networks to generate novel views of 3D objects from single images. Isola et al. [2017] used encoder-decoder networks with skip connections, a.k.a. U-Nets as proposed by Ronneberger et al. [2015], for spatially-aligned image-to-image translation, which are trained with a GAN loss to improve the realism of translated images. Our network also uses a U-Net structure for generating images and is trained with a hybrid L1 and GAN loss.

*Applications of machine learning in rendering and simulation.* Recently there is a trend in applying machine learning techniques in rendering and simulation to reduce the expensive computational cost. Ren et al. [2013] trained a neural network to represent the radiance regression function, and used it to render global illumination for scenes with dynamic light sources in real-time. Kalantari et al. [2015] proposed a method that used a neural network to control the parameters of feature-based filters, which are than used

to denoise Monte Carlo rendering images. Ladicky et al. [2015] modeled physically-based fluid simulation as a regression problem to predict the particles' acceleration. They designed a feature vector to model the forces and contraints, and trained a regression forest to approximate the behavior of particles during real-time simulation. Tompson et al. [2017] replaced the expensive pressure projection step in Eulerian fluid simulation with a neural network approximation, and accelerated the simulation pipeline. Chu and Thuerey [2017] proposed a convolutional neural network to learn a feature descriptor for smoke flow, which was trained on a dataset with high resolution smoke simulation results. When creating new results, they simulate at a low resolution and fetch similar data from the dataset based on the neural network descriptor to synthesize a high quality result.

## 3 OVERVIEW

We introduce a real-time oil painting system that replaces expensive fluid simulation with neural networks while the brush model is still physically simulated. The key idea is that in most scenarios an oil painting can be represented and rendered with only height and color maps of the paint surface, both of which will be synthesized with generative adversarial networks.

Figure 2 gives an overview of the stroke synthesis pipeline. We model individual bristles of the brush and simulate them with a traditional mass-spring method, same as in WetBrush [Chen et al. 2015]. During the process of painting, the user controls the brush and draws strokes on the canvas, which may have existing paint. Our system captures relevant information that describes the states of a newly drawn stroke, generates the resulting height and color map of the paint after applying the stroke, and copies the results back onto the canvas. The height and color maps are generated with two separate neural networks. A long stroke is split into smaller overlapping patches for faster neural network evaluation of each patch. Patches of network output are blended and copied back to the canvas as explained in Section 4.4.

Inputs for the height prediction network include the background height map of existing paint and the bristle pressure trajectory of the new stroke, while inputs for the color prediction network include the background color map, the color trajectory and time trajectory of the new stroke, and the height map produced from the height prediction network. We will explain more on why we selected these inputs in Section 4.1. Training and testing data are generated as described in Section 4.2 with an accurate offline volumetric oil painting simulator based on WetBrush [Chen et al. 2015]. The network training process is described in Section 4.3.

## 4 METHOD

We generate the height map and color map of the paint using generative adversarial networks inspired by the pix2pix model [Isola et al. 2017], which is a framework aiming to solve general image-to-image translation tasks that has been successful in applications such as generating photos from sketches. In this section we first discuss what data we use as input to the networks so that they are able to generate accurate height and color maps. We also introduce our process for preparing the training and testing data sets using an offline simulator. Then we explain our network structure, focusing on the modifications to the original pix2pix model needed to accommodate our specific application. Finally, we discuss our implementation of a neural network based painting system.

### 4.1 Input to Network

For a neural network to perform well, the input to the network should encode enough information to sufficiently describe the states of the system and reduce ambiguity. In a physically based oil painting simulation, a stroke result is deterministically controlled by the existing paint on the canvas, the brush motion, and the paint color on the brush. Thus the inputs to the networks should encapsulate the aforementioned information and come in a form similar to the output height/color map.

*Input Data for Height Prediction.* In a simulation based painting system, paint is deposited onto the canvas through the interactions between the brush bristles and the existing paint on the canvas. It is a rational choice to include the height map of the existing paint as the input for the height prediction network. As for the brush, while it is possible to embed a raw brush motion sequence and the pigment color on bristle nodes into one vector as a network input, it imposes extra burden to the network with the task of learning brush simulation. In our case, brush simulation using the mass-spring method is efficient enough, so we don't need to complicate the network. Instead, we rasterize the trajectories of the brush bristles touching the canvas while traveling along the stroke path. Such 2D trajectory maps are fast to generate on the GPU and define the spatial layout of the expected height map output. Naive additive trajectories can reflect the density of bristles sweeping over the canvas, but the actual amount of paint deposited is affected by other parameters as well, most importantly pressure. Lightly touching the canvas and forcefully pressing the brush down should deposit different amounts of paint. We compute the normal pressure force at the bristle tips from the mass-spring simulation to get a pressure-weighted bristle trajectory, which predicts paint height more accurately than the naive additive trajectory map alone, as compared in Figure 3. We also found that the pressure trajectory map provides sufficient information to faithfully recreate paint thickness that feels natural to the end user. The full input and output for the height prediction network are illustrated in Figure 2.

*Input Data for Color Prediction.* Similar to height prediction, in order to predict color we need the color map of the existing paint and the color deposited by the brush. We want to train the network to learn the color pickup process that occurs during a stroke, so we pass the initial color of the brush at the beginning of stroke as input. Similar to how we use the pressure trajectory map for height prediction, we use the color trajectory of the stroke as input to the network instead of embedding color values of bristle nodes, to make our training easier. In addition, the direction in which the new stroke is drawn makes a big difference to the final result: color smearing only happens after the brush picks up the existing paint and redeposits. In order to recreate the smearing effect, we add a time map as one of the inputs for the color prediction network. The time map is generated similarly to the pressure trajectory by rasterizing the trajectory of the bristles. In the time map, the pixel

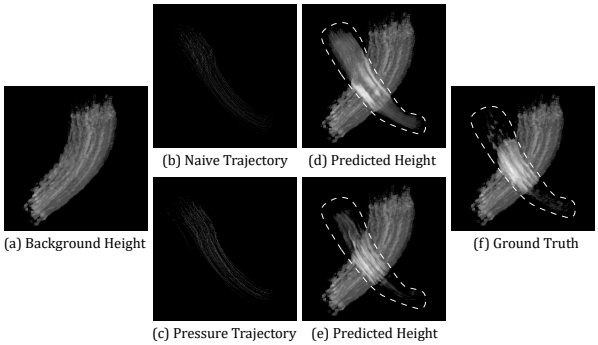



**Figure 3: In this example, the new stroke is drawn so gently through the existing paint (a), that little paint is left where there was no existing paint, as seen in the ground truth (f). In (b), the trajectory of the new stroke is generated by recording the number of times a pixel is swept by a bristle. A network that takes this input inaccurately creates too much paint (d). In (c), the grayscale value represents the pressure of the last bristle that sweeps the pixel. A network that takes this input creates more accurate results (e)**

value records the latest time when the point is swept by a bristle, with 0 representing start of the stroke and 1 representing the end. Without such a time map, the color network may generate incorrect smearing results as shown in Figure 4.

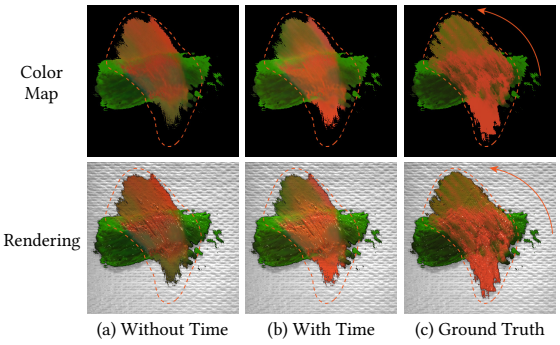



**Figure 4: In this example, an orange stroke (within the dashed line) is drawn on the green and black paint in the direction indicated by the arrow in column (c). Column (a) shows the color map and rendering result generated using a color network without a time input channel, which incorrectly mixes the new stroke with the green tint before it reaches the existing paint. With the time input channel, the network is able to predict mixing on the correct side (column (b)).**

For the best quality final paint rendering, it is important to make sure that the generated height map and color map align with each other. Otherwise, there might be artifacts like the dark region shown at the top-right of Figure 5, due to undefined color in pixels with non-zero height. Since we train the height network and color network separately, it is possible for the two networks to produce

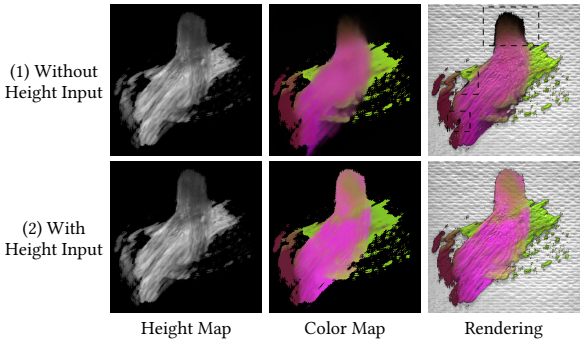



**Figure 5: The columns show the height map and color map generated with the networks, as well as the final rendering respectively. In row (1), the color network does not have the height map as an input channel, and the boundary of the color map is smaller than the height map, which creates some black region in the rendering result as shown in the dash line boxes. In the row (2), the color network has the height map as an input channel, and the boundaries of the two maps are aligned, producing a good rendering result.**

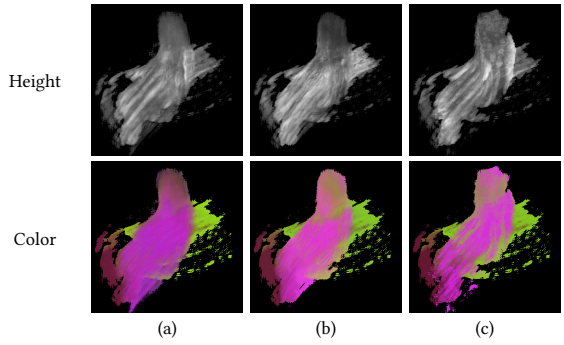



**Figure 6: With the same number of training steps, using a single network to generate both height and color (a) produces more noise and less accurate color than using two separate networks (b). The ground truth is shown in column (c).**

maps with mismatched boundaries due to residual errors. In order to align the color map with the height map, we wire the predicted height map from the output of the height network as an input channel to the color network, as shown in Figure 2. When training the color network, this input channel is fed with the ground truth new height map from simulation, which aligns exactly with the ground truth new color map as the output of the color network. During evaluation, we first run the height network to generate the new height map, then use it as input for the color network. We found that with this strategy the color network is able to learn the alignment requirement between the new height map and color map, and produces artifact-free paint renderings. Figure 5 compares test examples with and without the predicted new height map as input. Note that another option is to stack the height and color maps and train a single network to generate them all together. However, we

found in our experiments that this setting produces much worse results and takes longer to train (Figure 6). We suspect this is because the convolutions across color and height in multiple layers of the network are less meaningful and introduce large errors in training. The full input and output for the color prediction network are illustrated in Figure 2.

Note that the pressure may also influence the color mixing, so it's reasonable to also include it in the input for the color network. Similarly, time trajectory could also be included in the height network's input. However, our experiments show that including these extra inputs does not significantly improve the quality, so we decided to use as few input channels as possible to reduce training and evaluation cost.

## 4.2 Data Preparation

As introduced in Section 1, we implemented an offline physically based oil painting simulator based on WetBrush [Chen et al. 2015] as our data generator. The oil painting simulator represents paint with a 3D volumetric density grid so that it can simulate the dynamics of viscoelastic paint more accurately than a height field model. In our system using neural networks, we use a height map to represent the paint surface produced by the network for several reasons. First, even though a height field cannot model complex 3D shapes like an overhang, it contains enough information for rendering in a painting application where a top-down view of the canvas is primarily used. Secondly, our network is trained with height maps converted from an accurate volumetric fluid simulation, which should contain more accurate 3D surface details than height maps from regular 2D height field fluid simulation. Finally, using a 2D height map instead of 3D for both input and output can greatly reduce network model size and thus yield faster training and evaluation.

To build the training data set, we need to generate height and color maps representing the canvas state before and after drawing the new stroke, as well as relevant information about the new brush stroke as described in the previous section and in Figure 2. We first randomly draw a few strokes on the canvas to be used as existing background paint and then apply a new random stroke to the canvas while recording relevant information such as pressure trajectory map, time map, etc. For best results, the strokes used to generate training data should be similar to what users will apply when using the system. We recorded an artist's stroke sequence working on a painting in the real-time version of oil painting simulator and collected 3741 strokes. These strokes are randomly translated and rotated before applying onto the canvas in the simulator. The canvas size is 512x512 during data generation and the simulator saves the aforementioned maps in this size as grayscale or RGB images. During evaluation, the input patch to the network can be different in size because the network is fully convolutional. We generate input and output pairs from the examples with various numbers of background strokes. A total of 11,000 pairs are generated, which are split into a 10,000-pair training set and a 1,000-pair testing set. Figure 7 shows samples from the data set in which each row shows one complete training example, with existing height and color, information about the stroke, and the resulting new height and color.

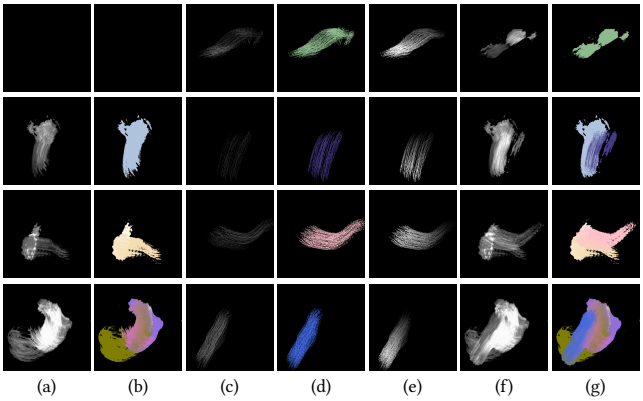



**Figure 7: Examples from our data set. (a) Background height map. (b) Background color map. (c) Pressure trajectory. (d) Color trajectory. (e) Time. (f) New height map. (g) New color map.**

## 4.3 Network Architecture

As both the input and output of our neural networks are represented by spatially aligned images, it is natural to formulate our network as an image-to-image translation problem. Thus we build our network based on the pix2pix model. The whole architecture consists of a generator network $G$ and a discriminator network $D$. The generator is used to translate the input data $x$, which in our case is a stack of 2D images with necessary information such as the background paint and bristle trajectory, to the output result $z = G(x)$, which is either a height map or a color map. The discriminator takes either a result image $z = G(x)$ produced by the generator or a ground truth image $y$ generated with simulation, as well as other necessary input information $x$, and is trained to distinguish ground truth from generated images.

*Generator.* The generator uses a U-Net [Ronneberger et al. 2015] structure, which consists of a number of encoder layers and decoder layers. The encoder layers include strided convolution operations that reduce the spatial resolution of the intermediate feature signals, and the decoder layers are resize convolution operations [Odena et al. 2016] that increase the spatial resolution. Skip connections are added to link mirror encoder and decoder layers, which can improve the sharpness of results by passing low-level features to the output. More details of our implementation of the generator network can be found in the appendix and Figure 15.

In the pix2pix model, the decoder uses transposed convolution, but in our experiments we found that the transposed convolution layers create some checkerboard artifacts in the output. This is likely due to the uneven overlap of the outgoing signals between neighbor pixels when doing transposed convolution operations, as studied by Odena et al. [2016]. Replacing the transposed convolution layers with resize convolution layers removes those patterns as shown in Figure 8. The resize convolution layer first changes the spatial dimensions of the feature map to the target size using nearest neighbor upsampling, and then does a regular convolution operation.

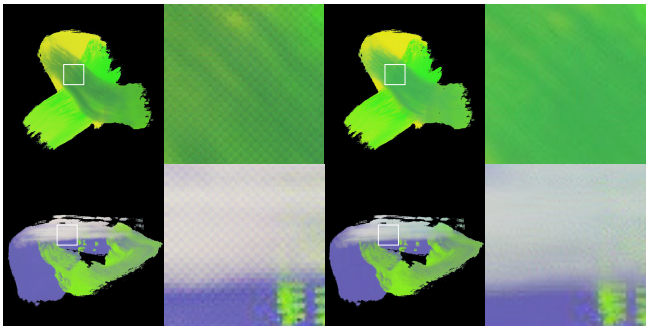


**Figure 8: Two color map examples generated by the network. The left two columns are generated with a network that uses transposed convolution layers, which creates the checkerboard artifacts. The right two columns are generated with a network that uses resized convolution layers, which removes those artifacts.**

We use instance normalization [Ulyanov et al. 2016] between convolution operations in the same way as in the pix2pix model to facilitate training. However, when we split long strokes into separate patches and evaluate each patch individually using the network (explained in Section 4.4), the instance normalization causes discontinuities across the boundaries of the patches even though they have an overlap region (see Section 4.4) that is larger than the receptive field. We suspect that it is because instance normalization normalizes feature maps with their spatial mean and variance independently for each patch at test time. Such a problem is commonly seen in works that use generative neural networks for synthesizing overlapping patches [Iizuka et al. 2017; Li et al. 2017]. Similar to these works, we remove the discontinuity artifacts by applying Poisson image blending [Pérez et al. 2003] in the overlap region of neighboring patches as a post-process. Figure 9 shows a comparison of the results before and after the post-process.

*Discriminator.* We use the same PatchGAN discriminator as the pix2pix model, which is composed of strided convolutional layers. The output of the discriminator is a downscaled 2D map, in which each element represents the likelihood of the corresponding patch in the input being real or fake. The elements are averaged to obtain the ultimate output of the discriminator. During training, the ground truth height or color map $y$ is labeled as real and the image created by the generator $G(x)$ is labeled as fake. See Figure 10 for the conditional GAN architecture for training the height prediction network. A similar structured discriminator is used for training the color network. Details of the discriminator structure can be found in the appendix.

*Objective Function.* During training, a set of input-output example pairs $\mathcal{S} = \{(x, y)\}$ are fed to the network. The training objective function consists of an L1 reconstruction loss and a cGAN loss.

$$\mathcal{L}_{L_1} = \mathbb{E}_{(x,y)\sim\mathcal{S}}\|y - G(x)\|_1$$

$$\mathcal{L}_{cGAN} = \mathbb{E}_x[\log(1 - D(G(x), x))] + \mathbb{E}_{(x,y)\sim\mathcal{S}}[\log(D(y, x))]$$

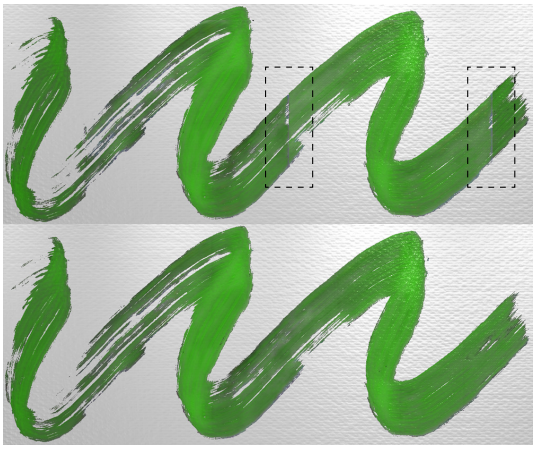


**Figure 9: An example stroke generated with three separate patches. In the top row, there is a clear seam at each patch boundary, seen in the dashed boxes, which is due to the instance normalization operations of the network. In the bottom row we blend the patches in their overlap region and remove the artifact.**

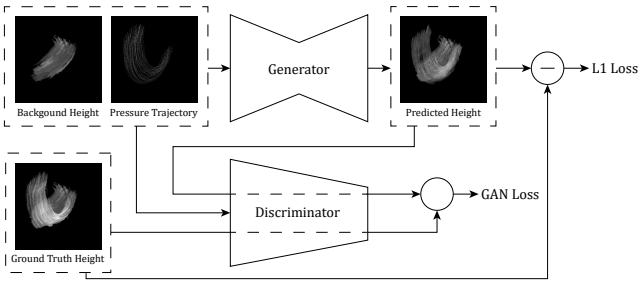



**Figure 10: Conditional GAN architecture.**

The training is a minmax optimization process as follows, in which $\lambda_{L1}$ and $\lambda_{cGAN}$ are the weights for the two losses.

$$\min_G \max_D \lambda_{cGAN}\mathcal{L}_{cGAN} + \lambda_{L1}\mathcal{L}_{L1}$$

The cGAN loss is essential for generating rich textures with high-frequency details, which is desired in oil painting to mimic sharper bristle details and thus to give a more realistic look to the digital paint. However, cGAN loss can lead to inaccurate results compared to the ground truth, such as artifacts and false colors. It has been known that the L1 loss can ensure the correctness of low frequency signals. Therefore, we combine both loss terms and fine tune their weights for a realistic and balanced look. Figure 11 compares the visual characteristics of results generated from the two individual loss terms as well as the fine-tuned result using both loss terms.

## 4.4 Neural Network Based Painting System

We implement a real-time painting system that uses the trained neural networks to predict the height and color map of the paint instead of expensive physically based fluid simulation. We still

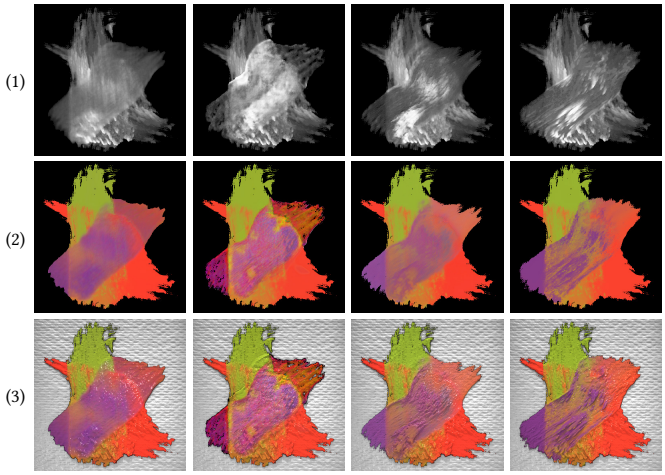



**Figure 11: Loss function comparison. Row (1): height map results. Row (2): color map results. Row (3): rendering results. The columns show the results generated with networks trained with L1 loss only, cGAN loss only, and L1 + cGAN loss, as well as the ground truth for comparison.**

simulate the brush model and generate all the trajectory maps related to brush states in the same way that we generate training data, as discussed in Section 4.2.

When the user uses the brush to lay down a stroke, the system gathers user input and updates the trajectories of the stroke segments applied within the time frame. Once a stroke input is ready, we collect all information needed, such as existing paint and pressure trajectories, within an axis-aligned bounding-box around the stroke and feed it to the networks for stroke synthesis.

For a long and curvy stroke, there might be many empty pixels within its bounding-box so that it is inefficient to evaluate the network on the entire bounding-box. Also for lower-end devices, oftentimes the bounding box can be too large for the network to evaluate as a whole due to GPU memory limitations. Therefore, we have to split the entire bounding box into multiple patches and evaluate them separately. The patches need to have some overlap in order to do the blending we mentioned in Section 4.3. The way we split the bounding box of the stroke is illustrated in Figure 12.

In our current implementation, we run the network evaluation every time a complete stroke is finished. But we could initiate evaluation of a patch immediately upon the brush leaving the patch. This should provide even better responsiveness. Since our current system is lag-free for most strokes in actual painting, we leave this optimization for future development.

Due to residual error, our networks still slightly change the background paint where the new stroke does not contact. When we copy the generated height and color map back to the canvas, the error is not noticeable within just a few strokes, but it can accumulate and become distracting after many strokes. To fix the problem, we create a stroke mask by dilating the trajectory's map and only copy new paint within the masked area back to the canvas.

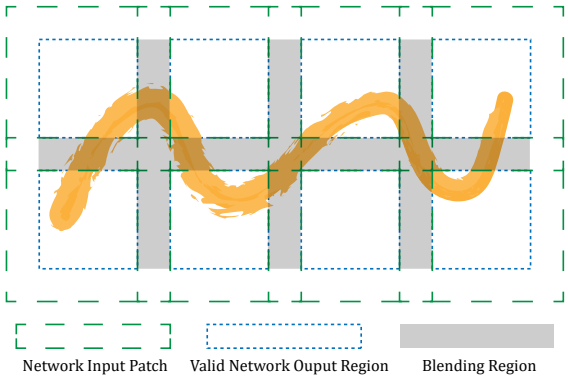



**Figure 12: Patch Split: The bounding box of a stroke is split into multiple patches (shown in dashed green lines). The generated stroke in the valid region (dashed blue lines) of the network output is directly sent back to the canvas. Output of neighboring patches are blended in the overlap region (grey region).**

An important part of good user experience in a painting system is to provide immediate feedback upon stylus action. A typical device with a 60Hz refresh rate requires our system to finish processing the applied stroke segment in 1/60 second. For very long and fast strokes there is no guarantee our system can finish processing all the pixels in time, especially if running on lower-end devices. To solve that, we use a separate thread to generate a preview stroke as a placeholder if the neural network cannot finish prediction in the back-end in time. The preview stroke is generated by extending stroke pressure and color trajectories to fill the height and color channels of the full stroke segment. The computational cost for this method is negligible.

## 5 RESULTS

*Training.* We use an Adam optimizer [Kingma and Ba 2015] to train the networks on the 10,000-pair training set, with learning rate 0.0002, $\beta_1 = 0.5$, $\beta_2 = 0.999$ and batch size 1. Loss weights are $\lambda_{L1} = 10$ and $\lambda_{cGAN} = 1$. The height network is trained for 55 epochs of the training set, which takes 112.6 hours on an NVIDIA GeForce GTX 1080 Ti GPU, and the color network is trained for 45 epochs which takes 113.3 hours. The networks are evaluated on the 1000-pair testing set. The RMSE of network prediction wrt. the ground truth are 0.0827 and 0.0649 for height and color prediction respectively.

*Performance.* Our neural network based painting system can produce realistic oil painting strokes and performs fast enough so that an artist can create high quality pieces of artwork, such as the two paintings shown in Figure 13 created with our system by professional artists. In order to compare the performance of our system and the real-time oil painting simulator, we also record the whole creation process of the paintings and replay the stroke sequence in the real-time simulator. As simulation performance varies under different settings, for a fair comparison we adjusted the stamping interval and solver accuracy of the real-time simulator to find the fastest setup that does not produce noticeable

**Figure 13: Paintings created with our system. ©Jini Kwon**

**Table 1: Statistics for system performance. All numbers are average statistics for generating one stroke. The testing is done on a desktop with an Intel Core i7-5930K @ 3.50GHz and an NIVIDIA GeForce GTX 1080 Ti. Note that both simulation and neural network evaluation are GPU intensive rather than CPU intensive tasks.**

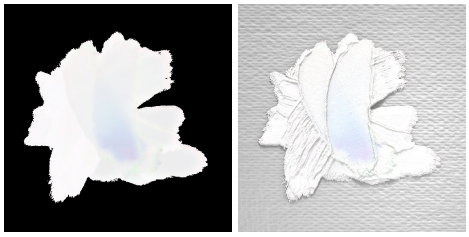
| Painting | Peacock | Gymnast |
|---|---|---|
| time for brush sim and input generation | 1.54ms | 1.66ms |
| time for generating the height map | 8.24ms | 10.34ms |
| time for generating the color map | 8.30ms | 10.17ms |
| total time of our system | 18.08ms | 22.17ms |
| total time of simulation based system | 24.99ms | 57.79ms |

discontinuity artifacts. Table 1 shows a breakdown of time for each step in the generation process. As is shown in the table, for typical paintings, the network based system is about 1.5 to 2 times faster than the simulation based system with comparable quality. Note that the neural network shows more performance advantage in the "Gymnast" painting, which has more background strokes using a larger brush, than the "Peacock" painting.

## 6 DISCUSSION

*Limitations of the color network.* Our neural networks can generate accurate results for a few strokes but the error can accumulate after many strokes at the same location. This is not a huge problem for generating height maps as the users are less likely to detect small errors in paint thickness and they can always adapt to the system's behavior by adjusting their stylus pressure. However, it can be problematic for color generation. The color network sometimes produces noticeably wrong colors, especially when multiple layers of same-colored strokes are laid together and the error accumulates. Figure 14 is an example showing that a purple tint appears in the overlap region when multiple white strokes are drawn. From a user's perspective, this artifact is easy to notice and difficult to fix without using destructive tools like an eraser. Achieving a very high color accuracy with generative neural networks remains challenging and to our knowledge no work has successfully solved the problem yet. According to our experiments, training the color network with only L1 loss produces less but still perceivable false color. A possible alternative is that, instead of creating the final color map, we can use the network to generate an intermediate state that requires less accuracy, such as a mixing weight map, and use it to procedurally generate the final color map without introducing incorrect new colors. We leave this for future exploration.

Also, adopting more complex pigment mixing models such as the Kubelka-Munk model [Lu et al. 2014] instead of linear RGB mixing is another direction for future improvement.



(a) Color Map (b) Rendering

**Figure 14: The error of the color network can accumulate and create wrong colors. In this example, multiple layers of white strokes are laid together, and the network produces a purple tint in the result.**

## 7 CONCLUSION

In this paper we introduce a real-time oil painting system that generates brush strokes using generative adversarial networks. The networks take the representations of the existing paint on the canvas and the new stroke drawn by the user as input, and predict the height map and color map of the new stroke. We train the networks with data generated by a physically based oil painting simulation engine. Our system is capable of generating realistic painting results visually comparable to a high-fidelity simulation engine and has better performance. Even though we only used oil painting simulation in this paper, our framework can be extended to other natural media painting simulations such as watercolor or pastel. We believe our neural network approach to synthesizing brush strokes opens up new opportunities to use even more sophisticated simulations in real-time graphics applications.

## REFERENCES


Ryoichi Ando and Reiji Tsuruno. 2010. Segmental Brush Synthesis with Stroke Images. In *Eurographics 2010 - Short Papers*, H. P. A. Lensch and S. Seipel (Eds.). The Eurographics Association. https://doi.org/10.2312/egsh.20101055

William Baxter and Ming C. Lin. 2004. A Versatile Interactive 3D Brush Model. In *Proceedings of Pacific Graphics*. 319–328.

William Baxter, Yuanxin Liu, and Ming C. Lin. 2004a. A viscous paint model for interactive applications. *Computer Animation and Virtual Worlds (CASA)* 15, 3-4 (2004), 433–441. https://doi.org/10.1002/cav.47

William Baxter, Jeremy Wendt, and Ming C. Lin. 2004b. IMPaSTo: A realistic, interactive model for paint. In *Proceedings of NPAR*. 45–56.

Zhili Chen, Byungmoon Kim, Daichi Ito, and Huamin Wang. 2015. Wetbrush: GPU-based 3D Painting Simulation at the Bristle Level. *ACM Trans. Graph.* 34, 6, Article 200 (Oct. 2015), 11 pages. https://doi.org/10.1145/2816795.2818066

Mengyu Chu and Nils Thuerey. 2017. Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 69.

Nelson Chu, William Baxter, Li-Yi Wei, and Naga Govindaraju. 2010. Detail-preserving Paint Modeling for 3D Brushes. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering (NPAR '10)*. ACM, New York, NY, USA, 27–34. https://doi.org/10.1145/1809939.1809943

Nelson Chu and Chiew-Lan Tai. 2002. An Efficient Brush Model for Physically-Based 3D Painting. In *Proceedings of Pacific Graphics*. 413–421.

Nelson Chu and Chiew-Lan Tai. 2005. MoXi: Real-time Ink Dispersion in Absorbent Paper. *ACM Trans. Graph.* 24, 3 (July 2005), 504–511. https://doi.org/10.1145/1073204.1073221

Stephen DiVerdi, Aravind Krishnaswamy, and Sunil Hadap. 2010. Industrial-strength Painting with a Virtual Bristle Brush. In *Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology (VRST '10)*. ACM, New York, NY, USA, 119–126. https://doi.org/10.1145/1889863.1889889

Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. 2015. Learning to generate chairs with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1538–1546.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2672–2680. http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

Siu Chi Hsu and Irene H. H. Lee. 1994. Drawing and Animation Using Skeletal Strokes. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '94)*. ACM, New York, NY, USA, 109–118. https://doi.org/10.1145/192161.192186

Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. 2017. Globally and Locally Consistent Image Completion. *ACM Trans. Graph.* 36, 4, Article 107 (July 2017), 14 pages. https://doi.org/10.1145/3072959.3073659

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-image translation with conditional adversarial networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017). https://phillipi.github.io/pix2pix

Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. 2015. A machine learning approach for filtering Monte Carlo noise. *ACM Trans. Graph.* 34, 4 (2015), 122–1.

Mikyung Kim and Hyun Joon Shin. 2010. An Example-based Approach to Synthesize Artistic Strokes using Graphs. *Computer Graphics Forum* 29, 7 (2010), 2145–2152. https://doi.org/10.1111/j.1467-8659.2010.01802.x

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *ICIL* (2015).

L'ubor Ladický, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. 2015. Data-driven Fluid Simulations Using Regression Forests. *ACM Trans. Graph.* 34, 6, Article 199 (Oct. 2015), 9 pages. https://doi.org/10.1145/2816795.2818129

Yijun Li, Sifei Liu, Jimei Yang, and Ming-Hsuan Yang. 2017. Generative Face Completion. *IEEE Conference on Computer Vision and Pattern Recognition* (2017).

Ming-Yu Liu and Oncel Tuzel. 2016. Coupled generative adversarial networks. In *Advances in neural information processing systems*. 469–477.

Jingwan Lu, Connelly Barnes, Stephen DiVerdi, and Adam Finkelstein. 2013. RealBrush: Painting with Examples of Physical Media. *ACM Trans. Graph. (SIGGRAPH)* 32, 4 (July 2013).

Jingwan Lu, Stephen DiVerdi, Willa A Chen, Connelly Barnes, and Adam Finkelstein. 2014. RealPigment: Paint compositing by example. In *Proceedings of the Workshop on Non-Photorealistic Animation and Rendering*. ACM, 21–30.

Michal Lukáč, Jakub Fišer, Paul Asente, Jingwan Lu, Eli Shechtman, and Daniel Sỳkora. 2015. Brushables: Example-based Edge-aware Directional Texture Painting. *Computer Graphics Forum* 34, 7 (2015), 257–267. https://doi.org/10.1111/cgf.12764

Augustus Odena, Vincent Dumoulin, and Chris Olah. 2016. Deconvolution and Checkerboard Artifacts. *Distill* (2016). https://doi.org/10.23915/distill.00003

Patrick Pérez, Michel Gangnet, and Andrew Blake. 2003. Poisson Image Editing. *ACM Trans. Graph.* 22, 3 (July 2003), 313–318. https://doi.org/10.1145/882262.882269

Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).

Peiran Ren, Jiaping Wang, Minmin Gong, Stephen Lin, Xin Tong, and Baining Guo. 2013. Global illumination with radiance regression functions. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 130.

Lincoln Ritter, Wilmot Li, Brian Curless, Maneesh Agrawala, and David Salesin. 2006. Painting With Texture. In *Symposium on Rendering*, Tomas Akenine-Moeller and Wolfgang Heidrich (Eds.). The Eurographics Association. https://doi.org/10.2312/EGWR/EGSR06/371-376

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 234–241.

Riccardo Roveri, A. Cengiz ÃŰztireli, Sebastian Martin, Barbara Solenthaler, and Markus Gross. 2015. Example Based Repetitive Structure Synthesis. *Computer Graphics Forum* 34, 5 (2015), 39–52. https://doi.org/10.1111/cgf.12695

Colas Schretter. 2005. A Brush Tool for Interactive Texture Synthesis. *ICGST International Journal on Graphics, Vision and Image Processing* 6 (2005), 55–60.

Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. 2016. Multi-view 3d models from single images with a convolutional network. In *European Conference on Computer Vision*. Springer, 322–337.

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. 2017. Accelerating eulerian fluid simulation with convolutional networks. *ICML* (2017).

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. 2016. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022* (2016).

Xiaolong Wang and Abhinav Gupta. 2016. Generative image modeling using style and structure adversarial networks. In *European Conference on Computer Vision*. Springer, 318–335.

Songhua Xu, Min Tang, Francis Lau, and Yunhe Pan. 2002. A solid model based virtual hairy brush. *Comp. Graph. Forum (Eurographics)* 21, 3 (Sept. 2002), 299–308.

Jimei Yang, Scott E Reed, Ming-Hsuan Yang, and Honglak Lee. 2015. Weakly-supervised disentangling with recurrent transformations for 3d view synthesis. In *Advances in Neural Information Processing Systems*. 1099–1107.


## A NEURAL NETWORK STRUCTURES

Here we describe the details about how the networks are implemented. We use *CnSm* to represent a convolutional layer with *n* filters and stride *m*, *ReLU* as a rectified linear unit operation and *BN* as a batch normalization layer. The encoder part of the generator is *C64S1-ReLU-C128S2-BN-ReLU-C128S1-BN-ReLU-C256S2-BN*. We use *RCnSm* as a resize convolution layer with *n* filters, upscaling factor *m*, and *Tanh* as a tanh operation. The decoder part of the generator is *ReLU-RC128S2-BN-ReLU-RC128S1-BN-ReLU-RC64S2-BN-ReLU-RCkS1-Tanh*, where *k* is 1 for height and 3 for color. Skip links send signals in the encoder layers to the mirror decoder layers, which are concatenated together before being fed to the next layer, as is shown in Figure 15. The discriminator is *C64S1-LReLU-C128S2-BN-LReLU-C256S2-BN-LReLU-C512S1-BN-LReLU-C1S1-Sig* where LReLU represents a leaky ReLU operation and Sig represents a sigmoid operation. The kernel size for all convolution layers is 4.

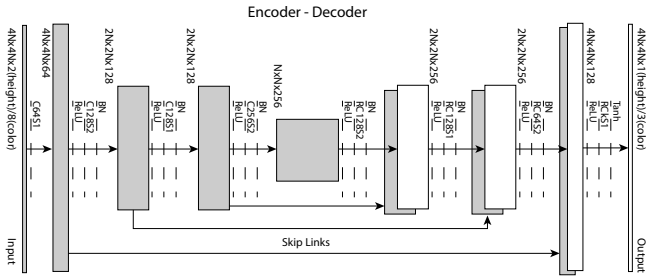



**Figure 15: Structure of the generator. The number of input and output channels is different for height and color prediction. 4N is the size of the patch fed to the network, which can vary for different brush sizes.**