Painting Algorithms
PhD Thesis Proposal

Manuel Rodriguez Ladron de Guevara
Doctoral Candidate in Computational Design
School of Architecture, Carnegie Mellon University

Advisor:
Ramesh Krishnamurti
Professor, PhD Computational Design Track, School of Architecture, Carnegie Mellon University

Co-advisor:
Daragh Byrne
Associate Teaching Track, School of Architecture, Carnegie Mellon University

Committee members:
Jun-Yan Zhu
Professor, Robotics Institute, Carnegie Mellon University

April 2022
Abstract

This document presents new approaches for painting algorithms based on optimization methods and learning models. My thesis focuses on artistic stylization of images under a stroke-based painting framework. That is, given an image, an algorithm generates a set of strokes that results in a painting. The two main problems that this thesis will address are finding the right combination of strokes for a particular artistic style and, by extension, how such combination follows a painting strategy that leads to certain styles.

The former is not fully resolved under a stroke-based painting framework, as there exists a trade-off between reconstructions of the input image and a controllable stylistic variation. Existing works that address style are still limited in style variations and controllability and principally use different stroke models and textures to output styles. The latter task is unresolved and challenging. State-of-the-art algorithms normally output strokes in an uncontrollable manner without a planned strategy that might help stylization. However, human artists employ painting techniques such as “blocking in”, grouping by semantics or colors, “background-foreground” or “color-then-contours” that help them convey artistic styles. My thesis will aim to disentangle such a complex landscape of painting styles and strategies, and will try to leverage some artistic vision under some perception of art. Finally, my thesis will dissect existing common artistic styles with the use of deep nets which might find their underlying patterns and inform painting models. Ultimately, my thesis will shed some light as to whether painting algorithms can be creative or not, under a definition of creativity, and whether it is possible to model a notion of self-awareness in the painting process.
Contents

1 Introduction 4
  1.1 Thesis Overview ........................................ 5
  1.2 Evolution of Painting from a Technological Perspective .... 6
  1.3 Non-Photorealistic Rendering ........................... 8
  1.4 The Stroke-Based Rendering Model ....................... 9
  1.5 Loss Functions for Gradient-Based Methods .............. 10

2 Survey of Non-Photorealistic Rendering 12
  2.1 Non-Learning-Based Methods .......................... 12
  2.2 Neural-Based Methods .................................. 14

3 Stroke Optimization Methodology 16
  3.1 Introduction ............................................ 16
  3.2 Loss Functions ........................................ 17
  3.3 Differentiable Renderer ................................. 18
  3.4 Methods ................................................ 19
    3.4.1 Naïve Implementation ............................. 20
    3.4.2 Coarse-to-Fine Algorithm ........................ 23
  3.5 Conclusion and Future Thesis Work ..................... 26

4 Neural-based Methodology 28
  4.1 Introduction ............................................ 29
  4.2 Painting Framework .................................... 30
    4.2.1 Model Architecture ............................... 31
    4.2.2 Stroke Parameterization and Renderer .......... 31
    4.2.3 Energy Function and Optimization ............. 32
  4.3 Painterly Styles ....................................... 33
    4.3.1 Comparison with State-of-the-Art Methods ... 37
    4.3.2 Ablation Study .................................... 38
  4.4 Conclusions ............................................. 38

5 Thesis Work 41
  5.1 Conclusion and Discussion of my Recent Work ........ 41
5.2 Future Thesis Work ................................................ 41
  5.2.1 Explorations on Stroke Model .............................. 42
  5.2.2 Painting Strategies ........................................ 42
  5.2.3 Explorations on Style Analysis ............................ 43
  5.2.4 Artistic Styles ............................................ 44
  5.2.5 On Creativity in Painting Algorithms .................... 44
5.3 Timetable .......................................................... 46

A Stroke Optimization ............................................. 53
  A.1 Stroke Optimization ......................................... 53

B Supplemental Material for Chapter 4 ......................... 56
  B.1 Higher Resolution Paintings ................................ 56
  B.2 Ablations on Model Architecture ........................... 56
    B.2.1 Visual Abstraction ...................................... 58
    B.2.2 Visual Abstraction Painting Progress ................. 59
    B.2.3 Artistic Styles .......................................... 60
    B.2.4 Precise Representations on Portraits ................. 60
  B.3 Precise Representations on Landscapes ..................... 61
    B.3.1 Comparison with SOTA on Landscapes ................. 61
    B.3.2 Stroke Budget Ablations ............................... 64
  B.4 Energy Functions ............................................ 64
    B.4.1 WGAN-GP Loss .......................................... 64
    B.4.2 Contrastive Loss ..................................... 65
Chapter 1

Introduction

Representational artwork emerged in prehistory and it is rooted in human’s culture since more than 70,000 years ago, where the first cave paintings are believed to have occurred. Neanderthals used their fingers and red ocher, a natural pigment, and rendered their imagination on stone walls. A large part of these paintings bring figures of animals, plants, and objects with varying degrees of realism. There are also graphic and abstract representations and complex scenes, and many archaeologists and scientists link the meaning of these paintings to rituals to attract good hunting, fertility, ward off danger, or simply give symbolic language to ideas, feelings or everyday life. Whatever this type of communication might be, it is undeniable that art forms a great part of human history and cultural values.

Art has continuously evolved in tandem with scientific and technological innovations, or rather, scientific and technological breakthroughs trigger new art forms. Section 1.2 provides a very brief summary of the evolution of painting, framed around technology, and while it is out of the scope of this document to provide a historical analysis of art, I believe disseminating art through the lens of technology will provide a contextual base for understanding my thesis. The evolution of paintings is tight to the evolution of technology, among other cultural and societal contexts. Technology shapes the way humans communicate their thoughts to the world, and painting is a central part of such communication. New technologies have not only enabled new art forms due to new painting tools and mediums, but act as a catalyst for many new painting styles to emerge.

The research described in this document aims to generate and understand new forms of art through state-of-the-art (SOTA) technological grounds, that of Artificial Intelligence (AI) and optimization. Specifically, my PhD thesis will aim to contribute to solve the problem of stylization in painting algorithms. I will also investigate whether it is possible to model creativity under this context, and under some definition of creativity. I believe it is important to spend time looking at art and understand the creative patterns that make art different from photographs, before trying to create a model that offers such artistic vision besides pure reconstructions. In this sense,
creating artistic versions of an image is a difficult task, as the literature is evolving towards unlocking such potential without using style-transfer like approaches.

This document is structured so that after this introductory chapter and a review of current work in literature, I first describe work on optimization and neural methods that I have already done during the past year. This is written in two different chapters Chapter 3 and Chapter 4. At the end of each chapter, I talk about limitations and work to be done after thesis proposal. The last chapter describes the organization of the work and a tentative thesis timeline, and provide further discussion about art, technology and creativity.

The remainder of this introductory chapter is structured as follows: firstly a provide a brief thesis overview, followed by a historical analysis of art under the lens of technology. Then I situate my work within the non-photorealistic rendering field, and describe common parts that all painting algorithms share, regardless whether they are learning or non-learning based.

1.1 Thesis Overview

This section describes briefly a tentative structure of my PhD thesis. A few potential contributions might be:

- Description of painting frameworks and their components. I classify these frameworks in two main categories: non-learning and learning based. Non-learning based frameworks can be further subdivided into two categories, those based on deterministic and heuristic algorithms, and optimization-based. In general, computational painting has existed since the early 90s, but humans have used computers to make images and art for as long as computers could generate images (see Figure 1.1). While computers were limited at that time, we still have some pioneering and seminal work done by Haeberli and Hertzmann. I will revisit some of these computational approaches and compare them against newer methods.

- Novel methodologies for algorithmic painting via optimization and neural methods. I present in this document work that I have done during the past year that contributes to my PhD thesis. The work that uses optimization methods focuses on developing algorithms that paint at any resolution using different strategies to convey some coarse-to-fine general strategy, and I provide an initial analysis on loss functions that also inform neural methods. The work done using neural methods scratches the surface on style generation, without using style-transfer like techniques. I further discuss future directions from this point onwards until the completion of my PhD thesis.
• Analysis of existing paintings with AI: the motivation behind such analysis is to learn different layers of information from existing works of art. This part might require to train a classification network over artworks using the WikiArt dataset or similar, and to perform some network analysis on layers or neurons like the work done by Bau and colleagues in [3, 74] to understand some underlying differences between artistic styles. This process might be useful for painting models to achieve such styles.

I highlighted the three major parts of my PhD Thesis, and I might expand or contract as needed and after conversations with the committee. The majority of the work is contained in the second bullet. The first bullet point is needed to understand how current AI methods work, and how they are different from older, pre-AI ones. The last bullet is needed to find useful information in deep nets’ inner layers that might help the design of stylization algorithms.

1.2 Evolution of Painting from a Technological Perspective

This section provides a brief description of the evolution of art through the lens of technological advancements, and how the bigger picture shows us how art is to some degree dependent of the technological availability of the moment.

The development of agriculture and social structures replaced rocky walls by small ceramic artifacts as canvases as well as mural paintings to render art. Mural paintings or frescos became the standard form of art during 1500 years, and common styles and scenes in ancient Greece were figurative style mixed with geometric designs in scenes of worship, games, palace ceremonies, animals and landscapes [69]. One of the first major technological breakthroughs happened around the fifth century B.C. in ancient Greece and it was the development of visual perspective and volumetric shading, which gives some depth to representations of complex scenes. As such, paintings became more realistic and human bodies became more idealistic. Even though Hellenistic painters could create an illusion of depth, there is no evidence that they understood the precise mathematical laws which govern perspectival representations [33]. Besides representation technology, Greeks also developed egg-based tempering, wax-based enameling and parietal fresco, using mineral and vegetable pigments in an aqueous medium.

Between the 13th and the 15th century art radically changed with new advancements. For instance, a more naturalistic means of representational painting was invented by Giotto di Bondone in 1306 using depth, perspective and temporal realism to present a single moment in time. In the 15th century, the linear perspective was invented with work by Filippo Brunelleschi, and a treatise on perspective theory by Leon
Battista Alberti. Perspective is a method for depicting the illusion of three-dimensions on a two-dimensional surface [2]. The development of oil paint gave artist a much richer palette of tones and colors than the previous technology of fresco, giving richer visual imagery. With years perfecting this new technology, it was possible to paint highly realistic images. At the time where photograph was not developed yet, the only possible way to see a static representation of the world was to see a painting from a highly talented and skilled artist.

In the 18th century, the development of photography allowed humans to capture static frames of the world, and presented a threat to painters, as a machine, for the first time, could do what painters where doing. This threat became apparent when Louis Daguerre publicly presented his work on photographic process called “Daguerreotype”. In the same year, Paul Delaroche, an influential French painter in the early 19th century stated “From today, painting is dead” [15, 28]. Not only photography became an accepted form of art, but a real transformation of art happened when artists had to rethink their art. Probably, this led to new form of representations that did not exist before photography, or what we know today as modern art. For example, Vincent Van Gogh stated: “Accurate drawing is not the essential thing to aim at, because the reflection of reality in a mirror would not be a picture at all, no more than a photograph” [28].

The advent of computers made possible for artists to find new ways of making art, using digital technology as part of the creative process, and this practice consolidated what we know today as digital art. From late 60s, computational artist Vera Molnar was a pioneer in making combinatorial images; in 1968 she would use a computer to create her first algorithmic drawings. In general, much of the early work focused on geometric forms and on structure, as opposed to content. This was, in part because of the restrictive nature of the available output devices, for example, pen plotter drawings tended to be linear. Some early practitioners deliberately avoided recognisable content in order to concentrate on pure visual form. They considered the computer...
an autonomous machine that would enable them to carry out visual experiments in an objective manner. 'Hommage à Paul Klee 13/9/65 Nr.2', a screenprint of a plotter drawing created by Frieder Nake in 1965, was one of the most complex algorithmic works of its day. An algorithmic work is one that is generated through a set of instructions written by the artist. Nake took his inspiration from an oil-painting by Paul Klee, entitled 'Highroads and Byroads' (1929). Paul Brown used computer-generated drawings in the 70s, and used individual elements that evolve or propagate in accordance with a set of simple rules [68], see Figure 1.1.

More recently, the advent of Artificial intelligence (AI) gave birth to a current of artists that use data-driven approaches and AI algorithms to generate new forms of arts never seen before. For instance, Mario Klingemann, an award-winning artist uses, between other methods, generative neural networks to interrogate aesthetic theories. Anna Ridler creates new and unusual narratives working with collections of information and self generated datasets. Tom White’s artwork investigates how machines see and articulate the world, generating physical abstract prints that are reliably classified by neural networks. Sofia Crespo is interested in the intersection of biological process and machine learning, see Figure 1.2.

1.3 Non-Photorealistic Rendering

Non-photorealistic Rendering (NPR) is the area of computer graphics that focuses on enabling a variety of representational styles of 2D and 3D inputs. Examples of NPR are the environment of a computer-aided design software (CAD) 3D model, or an artistic rendering of an input image. This thesis will focus on 2D NPR (images and videos). In this context, the input to an NPR algorithm is an image or 3D model and the output is typically an artistic rendering of that input imagery, for example, watercolor, painterly or sketched style [27, 70]. Stroke-based rendering (SBR) algorithms, a subset of NPR that focuses on painting-like outputs, create paintings by placing discrete strokes on a canvas [27], and traditionally, NPR focused on procedural heuristic algorithms to
simulate such painting process [41].

Painterly renderings are created by strategically placing brushstrokes on a digital canvas as a sequential process. Generally, artists manage this sequence in different ways to create different styles, and to focus viewers’ attention on specific regions and create the perception of depth through combinations of some stroke parameters such as color, shape, size, or placement. By avoiding too much detail—in computational means, highest detail would correspond to painting each pixel—artists aims to invoke our imagination and allows us to fill in the gaps [65, 26].

SBR algorithms have a particular component of a digital discrete brushstroke, and as such, the painting process is sequential and resembles traditional human painting processes. This is in contrast to some other learning-based methods that can generate art by learning a combination of pixels, but are not sequential in nature, such as [12, 32, 75, 39, 40]. However, there exist some overlaps on how SBR and generative algorithms work. SBR algorithms share some similarities with models like Generative Adversarial Networks (GANs) [19]. Generally, GANs output an image—CxHxW pixels—that is as close as possible to the distribution of training images. Painterly rendering algorithms also learn from a training distribution of images; however, their output is not in pixel space but in brush stroke parameters space—NxK, where N is the number of strokes and K is the number of parameters that define the brush stroke. The next section provides a definition of an SBR model.

1.4 The Stroke-Based Rendering Model

Normally, SBR algorithms are defined by:

- A stroke parameterization model. This is a mathematical model that defines how brushstrokes look. Examples of this are lines, n-Bézier curves, n-splines, or other primitives such as circles, triangles or squares. Figure 1.4 shows a diagram of the control points of a quadratic Bézier curve.
• A canvas model where strokes are rendered sequentially. This canvas is a matrix of numbers which are updated every time a stroke is placed. Except for unusual circumstances, a canvas is normally initialized with a constant background color.

• A rendering module. Rendering is the process in computer graphics of converting vector information on two or more dimensions into 2D images. Traditional rendering frameworks involve an operation called rasterization, which typically is non-differentiable and thus breaks the differentiability of the rendering process. While not a problem for hand-design painting methods, such traditional rendering methods present a caveat for gradient based methods. Deep learning makes possible to make a differentiable renderer and thus breaks previous limitations allowing the backpropagation of gradients.

• A function that determines how the translation between the input photograph and the output painting is performed. This ranges from deterministic to stochastic functions and defines the family of the painting algorithm.

The above points encapsulate common parts that SBR algorithms share. I provide more details about some painting functions in Chapter 2, as it is the most important of the three, and it can be broadly categorized into main sections: early heuristic-based methods, optimization-based methods, and learning-based (AI) methods.

1.5 Loss Functions for Gradient-Based Methods

Loss functions are needed in methods that use gradients to update optimization parameters. Such optimization parameters can be pixels in an image, stroke parameters in a SBR setting, or neural network’s weights. Generally, a combination of loss functions is more effective than using one loss function, although this depends on the nature of the problem. Formally, we define a loss function \( L \) as follows:
\[ L := \beta_1 L_1 + \beta_2 L_2 + \cdots + \beta_N L_N \] (1.1)

where \( \beta_n \) controls the weight or importance of each loss, balancing the overall loss \( L \).

The most common loss is pixel loss, and this measures the distance between pixels in space. Pixel loss is commonly represented by \( L_1 \), also known as Least Absolute Deviation and formally, \( L_1 = |\hat{x} - x| \). \( L_2 \) function, also known as Least Square Errors, and formally \( L_2 = (\hat{x} - x)^2 \), is very popular in machine learning, especially in regression models. Both are used as functions to measure image similarity, but it has been empirically proven that \( L_2 \) tends to produce blurrier results.

Perceptual losses [35, 72] carry semantic information captured by networks that were trained to classify image classes. As such, they compare high-level differences between images, like content or style. They work by extracting features from a pre-trained image classification network at different convolutional layers, and performing \( L_2, L_1 \) or cosine similarity in such extracted features.
Chapter 2

Survey of Non-Photorealistic Rendering

This chapter provides a survey of the different painting methods that exist in the literature, ranging from hand-design rule based methods in the early 90s to state-of-the-art AI-based methods.

2.1 Non-Learning-Based Methods

Within this group, we can further distinguish two main categories: a first one based on heuristics and hand-design rules became popular in the 90s, and a second one based on optimization emerged in the early 2000s.

The earliest painterly stylization algorithms applied hand-designed rules to produce impressionistic effects [23, 43, 30]. These methods generally point sample the image and compute control points of the brush stroke model. Generally, parameters such as stroke width are fixed in advance, and color is selected from the reference image based on the location of the first control point of the brush stroke. The oldest of the cited work, “Paint by Numbers”, is an interactive system in which an user places strokes on a canvas by clicking with a mouse. The direction of the stroke can be fixed, or can be guided dragging the mouse and releasing the button. In this work, Haeberli also indicates different ways to control the shape of the stroke, guided by the gradients of either a secondary image or the same reference image. In [30], Hertzmann also uses the reference image’s gradients to guide a cubic spline stroke model, and implements a coarse-to-fine method by scanning the image with steps sizes, which correspond to stroke widths, and by extension, the amount of detail to paint at each level. Figure 2.1 (b) uses this type of algorithm.

Optimization-based algorithms are fundamentally different than hand-design based algorithms in that there is an iteration procedure to search for an “optimal” solution. The setup changes in the way that the rendering procedure has to be differentiable in order to backpropagate the gradients of the error of a loss function to the parameters.
that are being optimized. In general, we formulate this problem as:

$$\mathcal{L} : x^* = \arg\min \mathcal{L}(\hat{I}, I)$$  \hspace{1cm} (2.1)

Where $\mathcal{L}$ is a loss function that measures the similarity between a canvas $\hat{I}$ and a reference image $I$, and $\hat{x}$ are the brushstroke parameters to be optimized. We use gradient descent to update the stroke parameters as follows:

$$\hat{x} \leftarrow \hat{x} - \mu \frac{\partial \mathcal{L}(\hat{I}, I)}{\partial \hat{x}}$$  \hspace{1cm} (2.2)

The first optimization-based painting method [29] produced precise stroke placements, but with cumbersome optimization heuristics (see Figure 2.1). Expectation-Maximization (EM)-like packing algorithms have been used for non-overlapping stroke primitives [27, 57], such as stipple [61] and tile arrangements [36]. A variety of related methods can be used to stylize 3D models, e.g., see [20, 4].

A few recent methods have explored direct stroke parameter optimization. DiffVG [42] and Stylized Neural Painters (SNP) [76] directly optimize the stroke arrangement with gradient-based optimization. SNP is the state-of-the-art of stroke optimization. These methods do not use neural networks to choose stroke placement, instead they pseudo-randomly initialize a fixed number of strokes that are optimized at each iteration in the optimization procedure. Such methods produce attractive outputs, and SNP produces style variation through different stroke parameterization and textures. Generally, these methods generate thousands of strokes per image, with either a more impressionistic effects or using thousands of tiny strokes. CLIPDraw [16] generates intriguing results using a text-based prompt rather than a target image.

Painterly stylization is a high-dimensional optimization problem, with challenging local minima, and each of these of methods may struggle to efficiently achieve precise results with a small number of strokes. SOTA techniques employ strategies that alleviate the difficulties of optimizing strokes, such as sequentially working on small
overlapping patches instead of finding the stroke parameters that produce the whole painting. Moreover, direct optimization can be slow for each output image.

2.2 Neural-Based Methods

In neural methods, a network is trained to produce a set of strokes from an input image. These methods optimize the neural network’s weights with gradients that come from the error of a function that measures the difference between the painting and a reference photograph. Generally, there is no stroke supervision nor the reference images are paintings. Neural approaches use either a pre-trained neural renderer, or a non-neural differentiable renderer, and often use perceptual losses \([L_1]\) rather than common pixel losses such as \(L_1\) or \(L_2\). One of the main benefits of using a deep-learning based method is that they can generalize to unseen images. Stroke optimization based methods use one image at a time, and a new set of parameters needs to be optimized every time a new painting is needed. Deep-learning based methods, however, tend to generalize to unseen images, generally of the same type or distribution than the training set, without further training.

Reinforcement Learning (RL) algorithms train a painting agent [17, 49, 31, 34, 63, 60], responsible for generating a sequence of strokes through interactions with a critic network. The learning signal comes in the form of rewards from the critic network, normally trained using adversarial learning [19]. Some RL methods focus on precise or accurate depiction between painting and reference [31, 63]. Huang et al.’s method [31] can precisely reconstruct images, but requires thousands of tiny strokes to do so, and is stylistically limited, though Schaldenbrand and Oh [60] show some variations on this style for use by a robotic arm. LpaintB [34] is a combination of RL and self-supervised learning that often produces coarser versions of the input image. Singh and Zheng [63] also focus on precise depiction using a semantic guidance pipeline, slightly improving image reconstructions, but also without particularly artistic results.

A few RL methods do create more artistic abstractions. SPIRAL [17] introduced adversarially-trained actor-critic algorithms that learn unconditional and conditional generation and reconstruction on MNIST and OMNIGLOT datasets, but failed to reconstruct CelebA faces, producing blurry results. SPIRAL++ [49] presented several improvements, achieving a very intriguing range of abstracted image styles. However, their method provides little or no interpretable control over the style and level of precision in the reconstruction.

PaintTransformer [44] models the painting problem as stroke prediction, using a CNN-Transformer model without the need for training with off-the-shelf datasets, showing an excellent generalization capability. Both SNP and PaintTransformer share a very similar texturized painting look to the detriment of very precise depictions. None
of these methods address stylistic control beyond applied texture, or abstractions given by limitations of the optimization approach.

Other image translation methods train with painting images to achieve similar outputs [32, 75]. However, these are fundamentally different in that they operate in pixel space while we do so in stroke space.

Recurrent neural networks (RNN)-based methods learn painting or sketching agent using RNNs. Sketch-rnn [22] use a sequence-to-sequence variational autoencoder [38] using a bidirectional RNN as encoder and an autoregressive RNN as decoder. This model works on simple line drawings and does not address RGB images. StrokeNet [73] use a CNN as encoder and RNN as decoder which also reconstructs simple line drawings and sketches, but fails to generalize to color images. More recently, Mo et al. also use a CNN encoder RNN decoder framework and introduce a dynamic window for vector line drawings [51].

Different from the above work and instead of learning a painting agent, Nakano uses pretrained classification CNNs on ImageNet to optimize a set of strokes that, using a differentiable renderer, paint an image which maximally activates a particular neuron in the network [52].

The collection of work in the literature indicates that there seems to be a trade-off between accurate depiction and style variation. Those works that achieve some level of visual abstraction are unable to achieve high precision paintings and vice-versa. In one of the experiments of this thesis, I will demonstrate how, with the right combination of loss functions and optimization strategies, our method is able to output precise reconstruction paintings in far fewer strokes, as well as producing different controllable levels of artistic styles without stroke supervision nor the use of external stylistic images as done in style transfer algorithms [18, 39, 75, 46]. Moreover, we also demonstrate that gaining control over such complex painting process does not require highly complex models or difficult training procedures such as RL.
Chapter 3

Stroke Optimization Methodology

This chapter describes new methodologies for stroke-based optimization. I will explain the contributions of the proposed methodologies, and how such techniques are useful to inform neural-based methods described in the next chapter. Some of the methods presented in this chapter use deep learning techniques in specific parts of the algorithm such as the renderer, but they are not used to determine the position or shape of the stroke. For brevity, I refer to stroke optimization-based painting methodologies as SOBP. I will firstly give a brief introduction to the matter, and then describe essential common parts shared by this type of algorithms. Then I will describe the methodology I implemented and finally I will discuss its limitations and future work to be done for my thesis.

3.1 Introduction

Direct optimization on stroke parameters presents an alternative to hand-design algorithms. The goal of stroke optimization methodologies is to find a set of stroke parameters that best minimize an objective function. However, as we will see, finding an optimal solution is difficult when trying to optimize the total number of strokes at once. I implement some strategies to ease the landscape and help the model find “optimal” solutions. Normally, optimization on stroke parameters is done using one
reference image at a time, that is, we optimize the strokes for just one painting. Each new image that we would like to stylize would require a new optimization process. This is in contrast to neural methods, which they act in batches of images and they can generalize to unseen images. Optimization-based methods do not generalize, because they do not learn from data.

3.2 Loss Functions

As defined in Equation (1.1), a loss function is needed in optimization algorithms to backpropagate its gradients to the stroke parameters that are being optimized. In the following experiments, we use three different loss functions and combinations. Pixel loss, expressed by the $L^1$ distance between pixels of a reference image and the painting, generally works sufficiently well enough to approximate the reference image with the painting.

However, pixel loss alone might not be enough if we want to approximate the target image with high precision or realism. For this, we add a perceptual loss. Let $V_{ij} = \{V_{ij}^1, ..., V_{ij}^k\}$ and $W_{ij} = \{W_{ij}^1, ..., W_{ij}^k\}$ be a set of $k$ feature vectors extracted from image $I$ and canvas $C_T$, respectively. We define perceptual loss as the cosine similarity between the two feature vectors as follows:

$$L_{perc} := \cos \theta = \frac{1}{K} \sum_i \sum_j \frac{V_{ij}^k W_{ij}^k}{\|V_{ij}^k\| \|W_{ij}^k\|}$$

(3.1)

where $i,j$ index the spatial dimensions of the feature maps $V$ and $W$, and $K$ the extracted layers from VGG16 trained on ImageNet [62]. Specifically, we use layers 1, 3, 6, 8, 11, 13, 15, 22 and 29. We find that minimizing perceptual loss between the finalized canvas $C_T$ and image $I$ works well to approximate the image. This loss captures high frequency parts of the image necessary for a high precision painting, and thus providing the network with enough signal to capture edges and fine details.

A third loss I try is another type of perceptual loss given by the CLIP model [55]. CLIP is a very big model trained on 400 million images and text descriptions, and the authors demonstrate its capabilities to classify, caption, translate, and generate. We encode both reference image and painting using CLIP’s image encoder to get meaningful representations, and then minimize their cosine similarity. This CLIP-based loss has recently been proven to be effective in line drawings generation in [8], and other language-driven models also use it for image generation and painting [56, 16, 59, 40, 58].
3.3 Differentiable Renderer

Optimization-based methods require backpropagation of the gradients of a loss function. This means that, from the parameters we are optimizing to the loss function, each component in the algorithm must be made of differentiable and continuous functions. Otherwise there would be a break in the backpropagation of gradients and the optimization signal would not reach the optimizing parameters.

In computer graphics, the process of translating vector information to images is called rasterization, and generally it is a non-differentiable function. The rasterizer (or renderer) is essential for this type of algorithms to work. However, the raise of deep nets made possible to overcome this issue by training a differentiable network that imitates what a non-differentiable renderer does. Recently, work has been done to make differentiable renderers without the use of deep learning [42], and works like [16] use this engine.

In all experiments in this document, unless otherwise specified, I use a differentiable neural renderer provided by [31] that translates quadratic Bézier curve parameters (see Figure 1.4) to a $128 \times 128$ image. The architecture of this network is shown in Figure 3.2. This network is trained for 500000 steps, using Adam optimizer [37] with a learning rate of 0.0001, decreasing it every 200000 iterations, and fixing it after the 400000th iteration. The authors use batch size of 64. At each training step, the algorithm draws stroke parameters from a uniform distribution and uses OpenCV to render the stroke. This is the considered “ground truth” reference image, and the error between the generated painting and such reference image is computed via $L_2$ loss.

Because the painting is made by blending the stroke images output by this renderer, all experiments’ outputs are, to some degree, related to the stroke image size of $128 \times 128$. 
3.4 Methods

I present a collection of painting methods in increasing complexity. This allows us to examine the inner workings of the algorithm and its capabilities in a more manageable way. All methods share a backbone composed by a canvas, differentiable renderer $R$, stroke model, and optimization and objective function. These methods start with a random set of stroke parameters $\hat{x}$ and the objective of minimizing their distance with the optimal solution $x^*$. For all methods described below, the algorithm starts with a fixed number of strokes. For consistency, I call it stroke budget, or simply budget. The final stroke budget does not necessarily need to be the same as the initial budget, as we will see how some algorithms decrease the number of strokes based on how “good” the painting already is at that moment. All methods use a variation of gradient descent, formulated in Equation (2.2), called Adam [37]. For clarity, all experiments presented here use the same stroke model, a quadratic Bézier curve parameterized with 13 variables. Each stroke is formalized as $s_t := x_0, y_0, x_1, y_1, x_2, y_2, r_0, r_2, t_0, t_1, R, G, B$, where $x$ and $y$ are the coordinates of start, middle and end control points of the curve, $r$ corresponds to radius at start and end points and $t$ corresponds to transparency at start and end points. I generally fix transparency parameters to 1, so all strokes are opaque.

The painting setup is as follows. Given a budget of $T$ strokes, we randomly or semirandomly initialize all strokes parameters in the budget $T \times 13$. These parameters are fed to a differentiable renderer $G$ which outputs a sequence of $T \times 128 \times 128$ rasterized alpha strokes. The colored rasterized stroke is obtained with a simple multiplication of the alpha with each RGB value. Each rasterized alpha stroke is blended on a canvas.
in sequence, following the blending equation:

$$\text{canvas}_t = \text{canvas}_{t-1} \ast \text{alpha}_t + \text{stroke}_t$$  \hspace{1cm} (3.2)$$

where $t$ is the time step within a budget $T$, alpha$_t$ corresponds to the alpha or mask channel output by the differentiable renderer $\mathcal{G}$, and canvas$_{t-1}$ corresponds to the previous state of the canvas (before adding a new stroke). For all experiments, I set the background color of the canvas as black. We then compute a loss function $\mathcal{L}$ between the reference image $I$ and the final painting $C_T$, and backpropagate the gradients of the loss function. We repeat this process until the optimization converges.

### 3.4.1 Naïve Implementation

A simple naïve, and suboptimal solution is shown in Algorithm 1. The naïve SOBP algorithm works with a canvas size of 128 × 128 pixels given by the differentiable renderer’s output size. Although such differentiable renderer could be trained at a higher resolution, working with a small size of 128 pixels in both dimensions, height and width, is more efficient. Generally, because we are training stroke parameters, we can output at any resolution by using the non-differentiable renderer after the strokes are optimized. However, as shown in Figure 3.4, the outputs from the two different renderers are different. This is because the differentiable renderer outputs brushstrokes whose edges are not as define, crisp or sharp as the original non-differentiable renderer. The difference between both outputs can decrease via training the differentiable renderer.
Figure 3.5: Difference of quality of paintings at 128x128 between random and grid stroke initialization. Top row corresponds to a grid stroke initialization, bottom row corresponds to random stroke initialization. Columns correspond to clipping the maximum stroke width the model is allowed to paint with: (a) no constraint, (b) maximum width 0.2*canvas size, (c) 0.1*canvas size and (d) 0.04*canvas size. All paintings use 324 strokes, and $L_1$ as pixel loss function. We let the optimization run for 800 iterations.  

for a longer time, however, even training the renderer until convergence, there exists a gap in fidelity between the two. As such, we always work with the differentiable renderer and the naïve algorithm cannot work at higher resolutions.

**Algorithm 1** Naïve SOBP

1: Inputs: ref image $I$, diff. renderer $G$, budget $T$, loss $L$, optimizer ADAM, learning rate $\beta$

2: Initialize: $\hat{x} \in \mathbb{R}^{T \times 13} \sim U(0,1)$

3: while $\hat{x}$ has not converged do

4: $\alpha \in [0,1]$, $S_t \in [0,1]$ ← $G(\hat{x})$  \hspace{1cm} ▷ Alpha mask and colored stroke

5: for $t$ in $[T]$ do

6: $C_t \leftarrow C_{t-1} \ast \alpha_t + S_t$  \hspace{1cm} ▷ Blending function

7: end for

8: $g_k \leftarrow \nabla_{g} L(C_t, I)$

9: $\hat{x} \leftarrow \hat{x} + \beta \cdot \text{ADAM}(\hat{x}, g_k)$

10: end while

The optimality of this algorithm heavily relies upon the initialization of the strokes. In its most naïve implementation, we draw stroke parameters from a uniform distribution in the range $[0,1]$. Figure 3.5 shows some examples of paintings at 128x128 with different stroke initialization and stroke widths. Top row shows four paintings with strokes initialized in a grid and bottom row shows four paintings with strokes initialized randomly. Columns correspond to clipping the maximum stroke width the model is allowed to paint with: (a) no constraint, (b) maximum width $0.2 \times$ canvas size, (c) $0.1 \times$ canvas size and (d) $0.04 \times$ canvas size. Initializing strokes in a grid and controlling their length with gaussians centered around such grid points makes the optimization problem easier (a diagram of these two initialization modes is shown in Figure 3.8 (c)). All paintings shown in this section use 324 strokes, and $L_1$ as pixel loss function. We let the optimization run for 800 iterations. Surprisingly we find that the paintings with random stroke initialization tend to achieve a lower $L_1$ loss, even though visually they are much worse. Specifically, the painting in the bottom row column (b) is arguably
Figure 3.6: Painting progress. Optimization process is shown every 10 iterations on each column for the first 100 iterations. The final painting is shown in last column. One of least visually appealing, and yet it reaches the third lowest loss out of the eight examples.

Figure 3.7: Difference in loss functions. We show the difference in the outcome when using L1, perceptual loss and CLIP loss. A weight of 0.01 seems very low for either perceptual or CLIP to make an impact. However, a weight of 0.1 makes the perceptual loss define some fine details than L1 loss alone cannot (see definition in the toes). CLIP loss introduces some noise in the painting.

Figure 3.6 shows the optimization process every 10 iterations for the first 100 steps. We can see in the bottom row how very long strokes are difficult to optimize, and while color seems to be an easier task to optimized for, these long strokes struggle to shrink accordingly. Figure A.1 in Appendix A.1 shows a comparison of the same set of loss functions with and without a grid stroke initialization.

Surprisingly, in the naïve algorithm, the initialization of strokes has more impact on the output image than any combination of losses. By simply starting with short strokes arrayed in a dense grid, the algorithm achieves much better results.

Loss function, stroke model and number of strokes are responsible for the outcome of the painting. In this experiment, we analyze the impact of different loss function,
Figure 3.8: Canvas composition by patches. (a) Patches are organized into a structured grid without overlaps. (b) Example of an organization of patches in a structured grid with overlaps of 20 pixels. (c) Initialization of strokes before being optimized: strokes are evenly distributed across the x and y axis of the patch (upper part). Strokes are randomly sampled from a uniform distribution (bottom part).

see Figure 3.7. While $L_1$ works fine for reconstruction (top row), it seems unable to capture finer details. Adding a perceptual loss helps alleviate this issue. CLIP loss seems to add some noise to the painting. Second and fourth rows use a small weight for perceptual and CLIP losses, resulting in imperceptible contributions. However scaling up the weight to 0.1 seems to work better for reconstructions, being the perceptual loss in the third row the one that achieves best reconstructions.

### 3.4.2 Coarse-to-Fine Algorithm

The naïve algorithm is limited in resolution size and lacks a strategy to control the stroke placement. An improvement over the naïve algorithm is to follow a coarse to fine approach following Hertzmann’s algorithm in [30]. Hertzmann defines some levels of detail ordered from coarse to fine, and strokes are added to each level with different stroke widths per level such that the coarser level is painted with wider strokes than finer levels. Each refinement phase normally has more strokes than the previous layers. Starting from the coarsest level with fewer strokes, we optimize this set until convergence before moving onto the next level. Examples of such levels of detail in a coarse-to-fine approach are shown in Figure 3.1 and Figure 3.9

#### The Cost of Higher Resolution

So far, our algorithm’s output resolution is tied to the differentiable renderer’s output size. Such low resolution limits the amount of detail in the painting and the practicality of the algorithm. We can, however, overcome this limitation by using a patch strategy: we divide the reference image into a set of $128 \times 128$ patches and perform optimization in batches without paying too much extra cost, at least under a 1024x1024 resolution.
In a nutshell, this approach optimizes and paints in parallel all patches at once and then a blending function composes the final canvas from the set of patches. Figure 3.10 shows an analysis of the cost of increasing the number of patches (left) and increasing the stroke budget (right). Increasing resolution to 4x results in a cost of 1.56x. The bottleneck of SBR algorithms, regardless of whether we use patch-based strategies or not, is the number of strokes because it has an approximately linear relationship. That is, doubling the number of strokes results in double time to compute. This is because this operation can’t be produced in batch and thus strokes have to be rendered sequentially, one after another.

**Painting by Patches using Coarse-to-Fine**

Given a reference image $I$ and a number of patches per side $N_h, N_w$, the patch-based strategy first resizes the input image according to a set of $N_h \times N_w$ equal patches of size $128 \times 128$. The new sizes of $I$ are $I_h = N_h \times 128 - (v \times (N_h - 1))$ and $I_w = N_w \times 128 - (v \times (N_w - 1))$, where $v$ is the amount of overlap between patches (see Figure 3.8). The higher $v$ is, the smaller the overall resolution is. Depending on the
type of blending function, overlap might help mitigate potential visible seems in the final painting.

I apply a coarse-to-fine approach on this algorithm and show the pseudo-code in Algorithm 2. Each level of detail is associated with a budget or number of strokes $T$ and a maximum stroke width. Normally we increase the budget and decrease the maximum stroke width at each level to make the painting feel more natural. At each level, we initialize the corresponding stroke parameters, and render and optimize all patches in batch. The loss function is calculated per patch and not over the entire composition because the differentiability breaks in the pasting operation (unless using a differentiable pasting operation), and because the painting is already well guided by its patches.

Algorithm 2 Coarse-to-fine patch-based SOBP

1: Inputs:
   ref image $I$, diff. renderer $G$, number of patches $N$, list of budgets $[T]$, list of max stroke widths $[R]$, composite $Z$, overlap $v$, loss $L$, optimizer ADAM, learning rate $\beta$

2: $H, W = N \times 128 - (v \times (N - 1))$

3: Initialize:
   $C_{\text{general}} \leftarrow 0_{[N^2 \times W \times H]}$

4: for $r$ in $[R]$ do

5:   Initialize:
       $\tilde{x} \in \mathbb{R}^{N^2 \times T \times 13} \sim U(0, 1)$
       $C_{r0} \leftarrow 0_{[W \times H]}$

6:   while $\tilde{x}$ has not converged do

7:       $\alpha_{T \times [W \times H]}, S_{T \times [W \times H]} \leftarrow G(\tilde{x})$ \hfill $\triangleright$ Alpha mask and colored stroke

8:       for $t$ in $[T]$ do

9:          $C_{rt} \leftarrow C_{r(t-1)} \ast \alpha_t + S_t$ \hfill $\triangleright$ Blending function

10:      end for

11:    $g_r \leftarrow \nabla_{\tilde{x}} L(C_r, I)$

12:    $\tilde{x} \leftarrow \tilde{x} + \beta \cdot \text{ADAM}(\tilde{x}, g_r)$

13: end while

14: $C_{\text{general}} \leftarrow Z(C_{rt})$ \hfill $\triangleright$ Composite function

15: end for

To make the painting look more natural, I setup an option to have a different number of strokes in each patch. To compute this in batch, it’s necessary to pad the different sequence lengths to have a rectangular matrix. After the coarsest level, we compute the difference of each painting patch with each reference image patch. If a patch’s loss is lower than a threshold, then we can reduce the number of strokes. This threshold indicates that we are satisfied with the amount of detail in that region, and no further strokes need to be added. Although the goal behind this implementation is that the painting would look more natural, I do not find a substantial difference between the two methods (see Figure 3.11).
Figure 3.11: Difference between paintings with same and different number of strokes per patch. (a) Original image with overlap of 20 pixels. (b) Painting with same number of strokes per patch at each coarse to fine pass for a total of 400 strokes (the 25 per patch) at the coarsest level, 576 (36 per patch) at the medium level and 1296 (81 per patch) at its finest level for a total of 2272 strokes. (c) Painting with different number of strokes per patch at each coarse to fine pass for a total of 400 strokes (the 25 per patch) at the coarsest level, 363 (36 or fewer per patch) at the medium level and 1023 (81 or fewer per patch) at its finest level for a total of 1786 strokes.

Painting Styles

We can change the style of the paintings using different combinations of number of strokes, stroke widths, stroke model and loss functions. All experiments use the same stroke model. Figure 3.12 shows four different styles. (a) is an abstraction of the input image by reducing the number of strokes. (b) limits the stroke width and paints in two passes instead of three. (c) is a realistic looking painting that uses a combination of $L_1$ and perceptual loss. (d) results in an expressionist painting and uses $L_1$ and CLIP loss weighted with a higher number (1).

The algorithm is limited in producing a more varied and better artistic stylizations. More research in other loss functions and algorithm design explorations will be done during thesis period to achieve such styles.

3.5 Conclusion and Future Thesis Work

The main contribution of this chapter is the adaptation of Hertzmann’s algorithm [30] to a stroke optimization setup. However, the current setup barely models artistic styles as it simply reconstructs a reference image with more or less detail, which it’s a form of abstraction. This is because the model was designed to respond to the problem of solving realistic paintings at any resolution without much extra cost, and little effort was put on understanding and modeling styles.

The explorations in this chapter are informative stepping stones towards neural methods. Both methodologies share common parts, as they are both optimization methods, but direct optimization of strokes is, a priori, an easier problem to address since there is no learning nor generalization involved. In fact, the experiments of this chapter were done after nearly a year of work on neural methods described in the next chapter. It is important to develop this framework to examine which parts could be
adapted to neural methods such as the effect of loss functions. Theoretically, we can expect that they behave similarly.

The objective of my thesis is to contribute to resolve the problem of stylization of paintings, that is, to make the painting feel like it comes from a real artist, with their biases and artistic abilities. Stroke optimization-based algorithms are good for testing strategies and losses, because they take less time to produce a painting than training neural methods, which makes the prototyping and loss testing faster.

While the focus of my thesis will be neural methods, I believe it is important to contribute to solve the stylization problem with optimization methods, and such important step is to be done after my thesis proposal.
Chapter 4

Neural-based Methodology

This chapter describes neural models for painting and the majority of its content was done while working as an intern at Adobe Research with Aaron Hertzmann and Matthew Fisher as mentors. I will describe the elements of neural frameworks, methodologies and contributions, and I will finish with future work done during my thesis.

Specifically, this chapter presents a new approach to stroke optimization for image stylization. Given an input image, our method generates a set of strokes to approximate the input in a variety of artistic styles. A skilled artist can effectively represent a scene with relatively few brush strokes, whereas current methods require large numbers of brush strokes. We aim for good image reconstruction with relatively few brush strokes, as a step toward enabling artistic styles with varying degrees of abstraction. Moreover, we show effective results using much simpler architectures than previous learning-based methods, avoiding the challenges of training control models. Instead, our method learns a direct non-linear mapping from an image to a collection of strokes. Perhaps
surprisingly, this produces higher-precision results than previous methods, in addition to style variations that are fast to train on a single GPU.

4.1 Introduction

A skilled painter can convey the essence of a scene with a few brush strokes (Figure 4.2). In art, this is sometimes called economy: conveying a lot with just a few lines or strokes. The tension between content and media has been hypothesized as a major source of art’s appeal [54], and economy heightens this tension: something that just looks both like a compelling landscape and like a splattering of brush strokes can be more intriguing and appealing than an accurate and detailed painting of the same landscape.

Painterly stylization begins with an input photograph, and generates a set of brush strokes to create a stylized version of the input. Many existing painterly stylization algorithms create an appealing “impressionistic” appearance by placing many scattered paint strokes that roughly approximate the image, e.g., [23, 43, 30, 76, 44]. These methods are not economical; they can capture fine details only by drawing thousands of small strokes.

This chapter proposes neural algorithms for economical painting styles. As in previous optimization methods, we express economy as minimizing a perceptual reconstruction loss with a small number of brush strokes. But previous optimization-based approaches struggle with non-linearities and local minima, as we saw in Chapter 3, and recent deep methods struggle with the difficulties in training recurrent neural agents. Our main technical insight is that we can use much simpler architectures than previous methods, by training a network that directly maps an image to a collection of strokes. Perhaps surprisingly, this produces higher-accuracy results than do agent-based methods, while being straightforward to train in a few hours. Our method produces accurate renderings with far fewer strokes than previous methods.

We demonstrate variations of artistic abstraction of the input image, i.e., very visually similar to the input in one extreme (Figure 4.1(b)), and very approximate, abstracted representations in the other extreme (Figure 4.1(c–f)).
Figure 4.3: Model overview. We train an end-to-end painting network without stroke supervision. (Left) The framework takes an image as input and produces a stylized painting. There are two main components, an encoder function that extracts visual features from an input image, and a projection layer that maps such features into a sequence of brushstroke parameters. Finally, a differentiable renderer sequentially renders the brushstroke parameters onto a canvas. An energy function $\mathcal{L}$ is a combination of loss functions that, in combination with the network type and optimization strategy control the style of the output. (Right) Im2Painting Decoders. We propose two decoders, $D_{FC}$ for visual abstractions, consisting on 2 fully connected layers followed by non-linearities (a) and $D_{LSTM}$ which adds an LSTM for precise reconstructions and artistic styles (b).

We also show variations of artistic style by varying the techniques used for the artistic abstraction, such as by preferring shorter strokes (Figure 4.1(h,j)) or injecting noise into the strokes (Figure 4.6(left, bottom row)). Varying the losses and constraints produces predictable variations in style, e.g., adding more strokes produces more accurate image reconstructions but less visual abstraction.

Though our method could be applied to any class of images, we focus on landscape painting. Landscapes in art history span a range from realism to abstraction, as seen in the work of painters like J. M. W. Turner, Claude Monet, and Richard Diebenkorn, each of whom created increasingly abstract landscapes as they got older. We also show quantitative comparisons on facial portraiture.

4.2 Painting Framework

Given an input image $I \in \mathbb{R}^{3 \times H \times W}$ and a stroke budget $T$, our method outputs a sequence of brush strokes that, when rendered sequentially onto a blank canvas $C_0$, produce a painting representation $C_T$ of $I$, as shown in Figure 4.3. The level of precision of the final canvas as well as the painting style depend on the loss function $\mathcal{L}$, the stroke budget, and the optimization strategy. Except where stated, we use $T = 300$ strokes as the economical stroke budget in our paper.

In contrast to previous work that either achieves good reconstructions but do not control style [31], or focuses on stylization via texture [44, 76] or via brushstroke parameterization and style-transfer technique [76], our goal is to learn a network that produces economical paintings in different and controllable styles, from reconstructions to visual abstractions, by varying loss functions and constraints. That is, our main interest resides in the ability to generate artistic styles given by the concept of
economy in art. We maintain the same stroke parameterization and we do not use external style images to achieve style variations.

4.2.1 Model Architecture

We propose a direct non-linear mapping network \( f : \mathcal{I} \rightarrow \mathcal{S}^{T \times 13} \) from image to brush stroke sequence, in contrast to previous agent models. We split the mapping into an encoder and a decoder (Figure 4.3). The encoder extracts feature maps using the four residual blocks of a Resnet-18 [24], not including the last average pooling layer, to extract a vector of feature maps \( X \in \mathbb{R}^{512 \times 4 \times 4} \) from a given image \( I \). The decoder \( D \) transforms these feature maps into a fixed sequence of stroke parameters \( s = \{s_1, s_2, ..., s_T\} \), that is, a \( T \times 13 \) vector. A differentiable renderer \( g \) renders the stroke parameters onto a canvas, and an energy function \( L \) that evaluates how well the painting fits the desired artistic style.

In this work, we use two different decoder architectures, \( D_{\text{FC}} \) and \( D_{\text{LSTM}} \), as shown in Figure 4.3. The first decoder architecture, \( D_{\text{FC}} \), is a stack of non-linear fully-connected (FC) layers. \( D_{\text{FC}} \) resizes and transforms \( X \) into a fixed sequence of strokes using 2 FC layers, with ReLU after the first layer and a sigmoid function after the last layer. The second decoder, \( D_{\text{LSTM}} \), uses a bidirectional LSTM layer in combination with fully-connected layers. \( D_{\text{LSTM}} \) uses average pooling on \( X \) to get a vector \( H \in \mathbb{R}^{512} \) before feeding it into its first FC layer. The first FC layer expands \( H \) into \( W \in \mathbb{R}^{512 \times T} \), forming the sequence of vectors that are the input to the LSTM layer. A second FC followed by a sigmoid outputs the sequence of brushstroke parameters. We find that \( D_{\text{FC}} \) works better for visual abstractions, whereas \( D_{\text{LSTM}} \) is more effective at producing precise reconstructions and artistic styles (see Appendix B for a thorough comparison, more technical details, and visualizations about the difference between the two decoders).

4.2.2 Stroke Parameterization and Renderer

The stroke model used in this chapter is the same as in the previous one, a quadratic Bézier curve parameterized by 13 variables as described in Section 3.4 and shown earlier in Figure 1.4. By extension, we also use the same differentiable renderer described in Section 3.3, introduced first in [31]. A common fact when working with these stroke models is that they do not have any texture and we do not optimize nor train with textures. Rather, if we want to add, say, an oil painting texture, it is normally done at inference time.

31
Figure 4.4: **512x512 300-stroke economical paintings** These examples illustrate results with higher resolution, and show how our method can provide nuanced, interpretable control over styles through varying losses and other interpretable factors. (b) shows a basic style by smooth visual abstraction. (c) adds constraints on thickness and length that reduces the number of small strokes. (d) adds random noise to strokes resulting in a scratchy style. (e) oil-painting style.

### 4.2.3 Energy Function and Optimization

Training our model with different loss functions, weights and optimization schemes produces different styles. Given an input image $I$, we define the objective as a combination of different loss functions:

$$L_{\text{painting}} = \lambda_1 L_{\text{perc}} + \lambda_2 L_{\text{guidance}} + \lambda_3 L_{\text{style}}$$  \hspace{1cm} (4.1)

where the first term is a perceptual loss comparing the output painting $C_T$ to the input image, the second term is a pixel loss applied to intermediate canvases, the third term is optional style losses on strokes, and the $\lambda$ values are constant weights. We explain the first two terms below, and describe the optional style losses in the next section. For perceptual loss, we use the loss described in Section 3.2 with corresponding equation shown in Equation (3.1).

**Guidance losses.** Previous methods typically apply losses to the output image. However, propagating losses to earlier strokes leads to slow convergence. RL methods partially bypass this with critic functions, but training the critic is itself challenging, leading to very long training times.

Instead, we simply apply a pixelwise loss to every intermediate stage of the canvas to help “guide” the optimization. Let $C_t$ be the painting after the $t$-th stroke is added,
so that $C_0$ is the initial canvas. Then,

$$\mathcal{L}_{\text{guidance}} = \sum_{t=1}^T \mathcal{L}_{\text{pixel}}(I, C_t) \quad (4.2)$$

where $\mathcal{L}_{\text{pixel}}$ computes the pixelwise $L_1$ distance between the input image and $C_t$.

### 4.3 Painterly Styles

In this section, we show a range of artistic styles, from accurate reconstructions or realistic-looking paintings, to variations of visual abstraction given by loosening constraints of our precise network and varying optimization guidance, stroke budget, shape constraints or rapid drawing motions. We show that a limited stroke budget of 300 strokes is enough to achieve all our styles, and we can increase visual abstraction more by further limiting the stroke budget.

We train using a compilation of landscapes to show different artistic styles. Figures 4.4 and 4.8 show examples at 512x512 of such artistic styles. We also train face models using CelebA [45] for precise depiction, as shown in Figure 4.5(left) and Figure 4.6(right, second row).

**Accurate Reconstruction** Our basic style aims for an accurate reconstruction of the input image Figure 4.1(column 2) and Figure 4.5(left) show examples of precise style paintings. To achieve this precise or realistic style, we set $\lambda_2$ to 1 and $\lambda_1$ to 0.1 on landscapes and 0.001 on face portraits, and we do not use any style-specific loss.
term, \( \mathcal{L}_{\text{style}} \): this setting is used in Figure 4.1(b), Figure 4.5(b,d), Figure 4.6(right, second row), and ablations in Appendix B. Landscapes require a higher \( \lambda \) due to a higher variance of the distribution of images. To achieve nearly-perfect reconstructions on landscapes, we add a fine-tuning step consisting on a 100-step optimization of the stroke parameters output by the network. Note that such step is only necessary for high-accurate reconstructions on landscapes, but not necessary for face images, nor any other visual style (see Figure 4.5(left) (b) does not use fine-tune step, (d) uses fine-tune step. Refer to Appendix B for more details).

**Visual Abstraction.** We can achieve stylistic variations and more artistic abstraction by reducing optimality. This is done by varying the optimization process, i.e., backpropagating gradients at different timesteps. We find that using only \( L_1 \) loss reduces visual fidelity, as does applying the loss only on the final canvas \( C_T \) rather than on all canvases as in Equation 4.2. Likewise, we can control the level of abstraction by varying the number of strokes used; more strokes allows the system to create more precise reconstructions. Note that although the number of strokes \( T \) is fixed during training, at inference time we can vary the number of strokes to achieve more abstracted levels by choosing \( K \leq T \) strokes. Examples of these ranges of abstraction are shown in Figure 4.5(right). This figure, arranged top to bottom in increasing abstraction, shows how economical paintings are achieved using different constraints. While in the third row only 100 strokes are used, the last row uses 300 strokes within a more wasteful schema. This is because the loss is applied only in the final frame, and the algorithm essentially “wastes” many strokes in the early stages that do not contribute to the final painting. This is further shown in Appendix B Figure B.4.

**Stroke Styles.** In the above styles, the network is free to paint with any stroke size. In this section, we add optional style losses to penalize stroke parameters that lie outside desired ranges, i.e., penalizing strokes that are too large.

Specifically, let \( S_p \) be some property of stroke shape. We set a shape threshold \( T_s \) to penalize properties that are too large.

\[
\mathcal{L}_{\text{shape}} = \begin{cases} 
0 & \text{if } S_p < T_s \\
||S_p - T_s||_1 & \text{if } S_p \geq T_s
\end{cases}
\]  

(4.3)

The properties we restrict are (1) the arc length of the control polygon \( S_l = ||p_1 - p_2|| + ||p_2 + p_3|| \), where \( p_i = [x_i, y_i]^T \) are the three control point locations, and (2) the center stroke radius \( S_w = (r_1 + r_2)/2 \), where \( r_1 \) and \( r_2 \) are the start and end radii. Theses losses are summed over all canvases, i.e., \( \mathcal{L}_{\text{style}} = \sum_t \mathcal{L}_{\text{shape}}(I, C_t) \) in this case.

Figure 4.6(left) shows artistic styles using these losses. The second row shows a
Figure 4.6: Painting Styles and SOTA Comparison (Left) Top row: input image. Second row: abstraction by length constraint on stroke length. A small penalty encourages the network to approximate the image with short strokes. The network generally chooses a larger stroke thickness to balance the lack of long strokes. Third row: visual abstraction by thickness penalty. The network learns to quickly approximate the upper part of the image with very few strokes that receive a penalty, creating a detail foreground and coarse background effect. Fourth row: a combination of constraints on thickness and length leads to a style of thick and short rounded shapes in a coarse but smooth style. Bottom row: scratchy effect by approximating rapid motions with gaussian noise. The photos here are taken from the training set. We use $D_{LSTM}$ for all styles. (Right) Top row: Original. Second row: Our results using $D_{LSTM}$ Third and fourth rows: Learning-to-Paint [31] limited to 300 strokes and full model, respectively. Bottom row: PaintTransformers [44] limited to 300 strokes.

style using an arc length threshold $T_s = 0.3$. Strokes longer than a third of the canvas are penalized, and so the network compensates with thicker strokes. This results in a smooth image approximation. As shown in the third row, penalizing thick strokes ($T_s = 0.05$) leads to a style with few large strokes in low-detail regions, instead placing much more detail in the foreground. Due to the difficulty in approximating the input image with 300 thin strokes, the network assumes a larger cost in a small number of strokes on low-frequency areas of the input image by choosing thicker strokes, resulting on an interesting contrasting style of thick and thin strokes. The fourth row combines both these constraints. For these experiments, we use $D_{LSTM}$ and we set $\lambda_1$ to 0.01, $\lambda_2$ to 1 and $\lambda_3$ to 0.001.

Noisy Motions. With enough drawing skills and time, a human can accurately paint a precise painting of a target image. However, on a limited time budget, painting becomes less precise and more like a rough sketch. These rapid motions create their own styles in writing [5] and painting.

We approximate the effect of motion dynamics given by a limited time per individual stroke by adding Gaussian noise into the actions. Let $p_{1:3}$ be a vector of brushstroke coordinates produced by the painter network at run-time. We add noise to the control
Figure 4.7: 512x512 300-stroke economical paintings These examples illustrate results with higher resolution, and show how our method can provide nuanced, interpretable control over styles through varying losses and other interpretable factors. (b) shows a basic style by smooth visual abstraction. (c) adds constraints on thickness and length that reduces the number of small strokes. (d) adds random noise to strokes resulting in a scratchy style. (e) oil-painting style.

points as \( p_{1:3} = p_{1:3} + z \) where \( z \sim \mathcal{N}(0, \beta^2 I) \). The minimization objective becomes

\[
\mathcal{L}_{\text{noisy}} = \lambda \mathcal{L}_{\text{perc}}(I, C_T) + \sum_{t=1}^{T} \mathbb{E}_{z_t \sim p_z} \left[ \mathcal{L}_{\text{guidance}}(I, C_t) \right]
\]  

(4.4)

where \( C_t \) is affected by the noise value as described above. In the context of stochastic optimization, we can approximate the expectation with a single random sample, and, further improve optimization by fixing the random seed [66, 53]. We use \( D_{LSTM} \) and set the same lambdas as for the precise network, and we use \( \beta = 0.05 \) and \( \lambda = 0.01 \). We do not apply noise at test time.

Figure 4.6(left, bottom row) shows examples of this style. Because the network learns that movements are noisy, the painter overrides any detail and approximates the main image composition with scratchy strokes.

**Training details.** Different optimization approaches require different settings. For precise depiction painting, we train on 8 Nvidia Tesla V100 16-GB GPU with batch size of 168, and takes approximately 24h to train. For other optimization methods, we train on a single GPU with a batch size of 24, converging after 5h on average. We use Adam optimizer with a learning rate of 0.0002 and betas 0.5 and 0.99. We use 100000 landscapes training images gathered from [9, 64, 75] and 200000 CelebA [45] training images.
Table 4.1: Quantitative results on 500 random faces from the CelebA dataset. We limit each method to 300 strokes the painting algorithms from previous work. We measure $L_1$ loss and perceptual loss using cosine similarity between input image and painting feature maps. The existing methods can achieve lower perceptual loss in their full models, but these models use thousands of tiny strokes and do not produce painterly abstraction. Refer to Table B.2 in Appendix B for quantitative results on landscapes.

4.3.1 Comparison with State-of-the-Art Methods

Simply comparing reconstruction error alone would provide a poor metric for painting algorithms. In this case, the best scores would be achieved by an algorithm that produces thousands of tiny strokes, reproducing the input image almost exactly, but not looking much like a painting. Instead we focus on reconstruction error with a tight stroke budget. How well can an algorithm reconstruct an image with only 300 brush strokes?

Quantitative Comparison. We provide a comparison with existing methods on our precision network, trained on CelebA. Table 4.1 shows our algorithm compared with RL, sequential and optimization methods. To make a fair comparison, in the first two columns we limit the number of strokes of previous methods to match our budget of 300 strokes. We show that our high precision network outperforms previous methods with such limit. Our $D_{LSTM}$ achieves better reconstructions than $D_{FC}$.

Qualitative Comparison. The lack of previous methods that produce visual abstractions and styles without the use of external style images or brush textures, challenges an assessment on styles. We instead compare our precision network with existing methods. Our network is able to produce crisper and more defined faces in far fewer strokes. Learning-to-paint [31] achieves better reconstructions with hundreds of more strokes as seen in Figure 4.6(right). This figure compares our method with RL [31] and Transformers [44]. Our results are shown in the second row. The third and fourth row show the RL approach with 300 and full model, respectively. The last row shows PaintTransformer with 300 strokes. We see that our model produces better defined and higher precision economical paintings at 128x128 resolution than previous methods. Figure B.7 in Appendix B shows a qualitative comparison on landscape images.
4.3.2 Ablation Study

Mapping Function. We run ablations on two types of mapping functions, a direct mapping or projection from CNN feature maps to a sequence of strokes, $D_{\text{FC}}$, and a sequential mapping that consists on a first linear layer to map CNN feature maps to a sequence of hidden vectors, followed by LSTM and FC layers, $D_{\text{LSTM}}$. We show results of both models in Appendix B, and provide insight for the size of such mapping function. We found that for visual abstractions, the LSTM-based architecture does not improve our results. However, we obtain our best results for precise reconstructions, reported on Table 4.1 on CelebA, and styles using our $D_{\text{LSTM}}$ model. $D_{\text{FC}}$, however, is sensible to the number of parameters.

Loss Functions and Optimization Regime. There is a relationship between the effect of a particular loss and the optimization regime. When opting for a greedy optimization, doing perceptual loss at each intermediate canvas becomes computationally expensive. However, by adding perceptual loss at the last time step the paintings achieve a better approximation to the input image. We found that when adopting less strict optimization regimes, such as evaluating the loss at the last frame, the impact of the perceptual loss decreases drastically. This is because the lack of backpropagated guidance disables the network to paint with more detail. We present an extensive ablation on loss functions in Appendix B, where we describe other losses and training approaches such as adversarial training with Wasserstein loss, or contrastive loss.

4.4 Conclusions

This chapter demonstrates that it is possible to control the existing trade-off between accurate reconstruction and abstraction. In contrast to previous methods that are able to produce either accurate representations or abstracted styles, we provide a flexible optimization framework that allows for adjustments in artistic styles without using external style images. From highly precise paintings to very abstract styles, we demonstrate that combining pixel, perceptual and style-specific losses in an energy function, and optimizing under a greedy or a one-off scheme is enough to capture a portion of the spectrum of visual abstraction. Moreover, our method proves a better optimization than previous methods, achieving higher precision paintings in approximately a third of the strokes.

Limitations. Our algorithm still suffers from local minima, and the optimization for the high-precision network converges slowly, resulting in larger training time. Our algorithm maintains a tight relation between output size and number of strokes. We choose to work with a tight stroke budget of 300 strokes to produce economical paint-
nings, which we demonstrate enough for good reconstructions of size 128x128. Since the algorithm outputs stroke parameters, at test time we can substitute the pretrained neural renderer by a non-differentiable renderer and make the painting larger, as seen in Figures 4.4 and 4.8. Even though our best results on precision on CelebA do not need further processing, in order to achieve accurate reconstructions on landscape images we need to do a fine-tuning operation. This is partially because the variance in the distribution of landscapes images is much higher than the CelebA dataset. This step is not needed for any other style.

Another limitation of this work is that it is not designed as object oriented painting. This method struggles with definitions of objects, and this is partially due to the nature of the landscape dataset which barely has any objects. Likewise, artistic styles are still limited and do not completely feel natural; in a way, the paintings still feel “computational-ish”. More work on understanding and modeling how artists transfer photorealism to expressionism or abstraction needs to be done in order for a neural method to work that way.

**Discussion and Future Thesis Work.** Our work scratches the surface for economical painterly stylization. Previous work focuses either on good reconstructions of an input, or on generating some level of abstraction via texture maps or relying on external style images. For instance, Learning to Paint [31] is able to approximate the input with high accuracy with more than a thousand strokes, but does not output any stylistic variation. PaintTransformers [44] and Stylized Neural Painters (SNP) [76] output paintings in a very similar style. They use texture maps in their strokes and SNP achieves style by changing the stroke primitive or doing style transfer. Our work, however, focuses on the generation of variations and styles (given by the concept of economy in art) using different optimization procedures and loss functions, but keeping the same stroke primitive (quadratic Bézier curve). The concept of economy in art opens up new stylistic possibilities, and we design our networks under this scenario. Even though our method generates good reconstructions, we find it less interesting than the explored artistic styles and visual abstractions. We also find it intriguing that we can obtain SOTA results with a direct mapping approach, proving that more complex approaches such as RL, Transformers, or semantic guidance models are not necessary for this problem. Future work comprises a better understanding of a direct mapping from convolutional features, finding a larger spectrum of style variations, and exploring a more controllable path to output strokes.
Figure 4.8: **512x512 300-stroke paintings** These examples illustrate results with higher resolution, and show how our method can provide nuanced, interpretable control over styles through varying losses and other interpretable factors. (b) shows a basic style by smooth visual abstraction. (c) adds constraints on thickness and length that reduces the number of small strokes. (d) adds random noise to strokes resulting in a scratchy style.
Chapter 5

Thesis Work

This brief concluding chapter summarizes the contributions done so far and provides directions to follow until the completion of my PhD thesis.

5.1 Conclusion and Discussion of my Recent Work

The previous two chapters described in detail two different approaches to painting algorithms. Chapter 3 provided a description of algorithms in increasing complexity, which addressed primarily a stroke initialization problem, an up-scaling problem and a coarse-to-fine painting strategy. The upscaling problem was solved via a patching approach, in which we leverage the use of GPUs to make computations in parallel. We saw how there is a mismatch between the two type of renderers given the exact same set of stroke parameters (Figure 3.4), and thus, the batched strategy solves this issue with the correct general canvas compositing algorithm (with or without overlaps). The coarse-to-fine strategy ease the optimization landscape, and also makes the painting look more natural.

Chapter 4 provided a description of the work done collaboratively with Aaron Hertzmann and Matthew Fisher. This work was done before the work presented in Chapter 3. Here, we are primarily concern with generating stylistic outputs, rather than developing a controllable painting strategy. The main contributions of this work are the algorithm based on a direct mapping from a learned latent space to a set of brush strokes and a set of controllable styles. However, this algorithm has a major limitation of resolution when using artistic styles (it does not have this limitation when painting in a realistic fashion thanks to the same patch strategy that we described on Chapter 4). Another limitation is that the variability of painting styles is rather low.

5.2 Future Thesis Work

My PhD thesis will focus on current limitations presented here, which some of them are also present in the literature. From the previous two chapters, Chapter 3 informs
neural methods in that instead of working with an entire canvas, it could be divided into patches and adopt a paint-by-patches approach. Likewise, explorations on loss functions and other painting strategies are transferable to neural methods. Overall, optimization algorithms could be useful as testbed for neural methods. My thesis will aim at contributing to resolving the stylization problem and, by extension, the “planning” problem in SBR algorithms. Both problems have not been solved yet entirely by the literature, as seen in Chapter 2. Please, note that stylization here does not consider style-transfer-like techniques, in which case, the problem is considered solved. Also, stylizing paintings via bitmap texturing has been done in works like [76, 44]. There seems to be a trade-off between reconstruction and stylization. Those algorithms that aim at stylizing an existing image seem to suffer at reconstruction and vice-versa. Regardless of whether aiming at reconstruction or artistic style, there seem to be a lack of control on stroke position. Almost none of the algorithms in the literature can really control the precise location of each stroke, usually because of a lack of “planning” or painting strategy when designing the algorithm.

5.2.1 Explorations on Stroke Model

All experiments used in this document are done with the same stroke model, a quadratic Bézier curve. This is a flexible model that allows the painter to choose from straight to curve lines, but it is a constrained by the amount “wiggleness” and continuity. Since it is quadratic equation, the stroke will always be composed by just one curvature. There are as many models as primitives we can imagine, from straight lines, to n-degree Bézier curves, to square, triangles, circles, ovals, rectangles, higher degree b-splines, blobs, etc. Each one of this will have a big impact in how the painting looks.

Texturizing strokes is important to achieving styles, as a simple oil-paint texture map can convert a plain painterly painting to a rich texturized oil-painting. I will delve into the impact of such textures. It seems useful to have a more flexible framework in which the painting model, be it based on optimization or deep nets, can freely choose between different brushes, as humans would do. So far, all our algorithms work with a fixed stroke model, which constraints stylistic range.

5.2.2 Painting Strategies

There are many different painting strategies that human artists follow when painting. Almost all of them follow a general coarse-to-fine procedure, which consists of a natural increase in the amount of detail as we approach the end of a painting—this is itself a known problem, as finishing a painting is more a personal decision based on many factors like timing, budget, tiredness, satisfaction, etc—but does not necessarily achieve a specific style nor more complex painting strategy.
A few planning strategies that human artists employ are: “blocking in”, which works by blocking in the general colors and shapes on the canvas and whose purpose is to lay down a general composition and color harmony without needing to worry about details; “line-drawing first and coloring later”, which works by firstly laying out the lines or edges of objects in a scene and then proceed to color; “grouping” by colors, shapes or semantics; “background, then foreground”; “foreground, then background”, etc. These painting techniques are a challenge for algorithms to replicate. Current algorithms seem to not model any overarching strategy, probably due to the fact that it is difficult to find objectives that can guide the process in such intentional manner.

My thesis will aim to provide a deeper insight about this and model some of the above techniques. Generally, these techniques are related to a certain stylistic output, which add a layer of complexity or entanglement in the algorithm. From an algorithmic perspective, however, there seems to be three directions that do not encapsulate the above mentioned human painting techniques: painting the totality of the canvas at once, as done in the work presented in Chapter 4, painting by patches, as seen in Chapter 3, or painting by objects or masks. I have not implemented this approach yet, but this strategy normally relies on image segmentation or masks to address each object separately. The most simplest case of this form might be to have a foreground and background masks, and paint accordingly. I have initiated this approach but results were not successful, and I do not show them in this document.

5.2.3 Explorations on Style Analysis

The intention of style analysis is to better leverage the representational power of deep nets. Understanding the relation between network layers and type of information (for instance, patterns in geometry might appear in some intermediate layers while semantic patterns might appear in later layers) seems important to better use them for painting tasks. A simple CNN image classification network trained on WikiArt might disentangle the type of layers we can use in an encoder-decoder framework in a neural painting model. Likewise, this might be a useful exercise to learn the difference between styles. Of course, there are many ways we can train the same model and many different architectures that we can train (CNNs, Transformers, Visual Transformers, RNNs, FCs, etc.). For instance, for the experiments described in Chapter 4, we default to a CNN-based encoder, and we did not use any pre-trained encoder. Using a pretrained network on a specific style to encode reference images might or might not inform the painting better.
5.2.4 Artistic Styles

Understanding artistic styles from an algorithm point of view is difficult, as style is the result of many different factors: brush stroke model(s), color palettes, composition type, texture, type of ink or materials, strategy, time, perception, innate skills of artists, etc. All these variables make this problem in computation extremely challenging, and this might explain why it is still considered unresolved. Algorithms that succeed on outputting artistic styles are mostly based on two elements: stroke model and texture, or style-transfer approaches. The former can output interesting looking paintings, with styles covering oil-painting, watercolor, tape-art or line-drawings. However, there exists a limitation with the number of styles we can generate via stroke model. The latter can theoretically achieve any type of style, and it works by blending or mixing a content and a style image via mathematical operations between feature vectors obtained with a pre-trained image classification neural network. The recent CLIP model [55] has been successfully used to guide NPR algorithms and GANs to achieve certain styles [40, 16, 58, 59].

However, instead of computing style via external images, my thesis will try to algorithmically understand how humans paint and how to deploy more interesting ways to make art. Although challenging, I will try to generate a neural model that can paint in a range of different styles, be it painterly art, expressionism, impressionism, abstraction and their variations.

5.2.5 On Creativity in Painting Algorithms

Assessing whether an algorithm is creative is a highly difficult task partially because it is unclear how to frame and evaluate creativity within AI algorithms. An issue is that there needs to be a meaningful mapping between human creativity and AI creativity. Despite long efforts throughout the late nineteenth century until today trying to consolidate theories of creativity, there seems to be a loose consensus about defining a criterion for creativity [67]

Painting algorithms have arguably been considered a quintessential AI creative process [10]. While painting is an unambiguously creative domain because, among other things, the number of forms used in paintings may be indicative of flexibility and originality [67], my work situates such creativity in between scientific and artistic value. That is, just because an algorithm can contribute to a scientific novelty—an unseen method—to generate paintings, does not mean that the paintings have artistic value. This is the case for recent painting algorithms such as Paint Transformers [44] or learning-to-paint [31]. They both offer novel computational models, but their paintings are reconstructions of input images; in the case of Paint Transformers look like digital oil painting, and in the case of learning-to-paint they look like plain digital paintings.
In my most recent work done at Adobe Research, partial creativity is achieved by (a) generating a novel method to paint and (b) achieving unseen implicit styles. I say partial creativity because styles are fixed at the beginning of the process and because the algorithm models a specific task, in opposition to a greater artificial artist—this is discussed later. In this context, my understanding of creativity, at its highest level, closely follows the definition provided by Boden and Stenberg, and it should be applied in both, scientific—novelty in problem solving—and artistic—novelty in artistic ability—contribution. Next, I will provide more detail as to how to assess creativity and the criteria applied to the method’s outcomes.

A natural approach to achieving an algorithm that can be considered creative might be starting with a psychological creativity theory as inspiration. Among the list of theories presented in [67], the flow theory by Csikszentmihalyi [11], which describes an optimal state of intrinsic motivation, the Geneplore model by Finke and colleagues [14], by which there are two processes of generation and exploration of ideas, the primary and secondary process of cognition supported by Kris, Mednick and Mendelsohn [14, 48, 50], and Martindale’s principle of least effort [47] might be feasible to describe in computational means. In fact, Elgammal et al. [13], loosely build a GAN model based on Martindale’s theory and define style novelty by finding regions in the latent space that deviate from style norms, yet without moving too far away from the distribution of art—to avoid falling into the negative hedonic response [6, 71]. There seems to be a mismatch, however, between trying to model a psychological creativity theory and the mainstream trend in computer vision of modeling a specific task, be it an image-to-image translation, a painting algorithm, or an object detection task. After all, individual tasks do not need complex creativity theories in order to work—I am not denying perceptual, biological, and neurological inspirations of state-of-the-art vision approaches, as it is the case, for instance, in convolutional neural networks and/or attention mechanisms. My most recent work is an example of this: we model a, partially creative, task of painting with implicit novel styles.

A more ambitious work would rely on a theory more heavily, and would not model just a task, but an artificial being or system. This would help have a more interpretable mapping between human and AI creativity, and the creative act would be, at the very least, more complete. Under this scenario, and assessing and evaluating creativity by its process as well as the product, and targeting an H-creativity [7], I would need to model a larger entity, i.e. system, such as a probably incomplete representation of an artificial artist, and not just the task of painting itself. To do this, I would need to model fundamental constituents such as a memory block which might consist of visual and textual information to recreate knowledge, and a self-evaluative mechanism to recreate self-awareness of creativity—one of the traits that creative subjects share [67]. This is
further supported by Colton: “A poet with no critical ability to judge its own work is no poet at all” [10]. While modeling personality is completely out of the question, I might as well imagine modeling some environment that fosters creative behavior, and thus tap into the systems theory by Csikszentmihalyi and working environments by Amabile and Gryskiewicz [1], although this might be difficult. Assuming a successful outcome, that of having novel paintings with respect to its knowledge of the world, the process would be deemed creative and novel, as there is no other algorithm in the literature of painterly rendering that attempts this.

One of many possible ways to assess whether an algorithm is being creative is that artistic novelty should be abundant and diverse, and as Colton mentions in [10], a novel output should not be hand-selected by humans, but rather by the algorithm itself. Another factor to measure creativity is controllability and more importantly, interpretability. Being able to interpret the process of, in this case, neural networks, is fundamental to understanding the inner workings of the creative process and drawing meaningful conclusions between outputs and algorithms—or product and process.

While I am not sure if deploying a successful creative system would realistically fit in the scope of my PhD thesis, I would like to include some explorations in this regard.

5.3 Timetable

![Figure 5.1: Tentative Timetable for PhD Thesis Development](image-url)
Bibliography


Andrew Y Ng and Michael I Jordan. “PEGASUS: A policy search method for large MDPs and POMDPs”. In: UAI. 2000.


[68] Victoria and Albert Museum. A History of Computer Art. URL: http://www.vam.ac.uk/content/articles/a/computer-art-history/.

[69] Wikipedia. History of Art. URL: https://lobopart.com.br/en/the-history-of-painting/#::text=Painting%5C%20emerged%5C%20in%5C%20prehistory%5C%20in%5C%20when%5C%20more%5C%20than%5C%20400%5C%20years%5C%20ago.


Appendix A

Stroke Optimization

A.1 Stroke Optimization

Figure A.1: Comparison between two stroke initialization methods. Left block of images is initialized with a grid of strokes. Right block of images is initialized with random stroke initialization. Random initialization seems to struggle independent from the loss we optimize for.

Figure A.1 shows a comparison of results using different losses between strokes initialized in grid (left) and strokes initialized randomly (right). We see how critical the grid initialization is in order for this algorithm to work. Figures A.2 and A.3 show the entire painting process by frames.
Figure A.2: First 170 painting steps
Figure A.3: Last 154 painting steps
Appendix B

Supplemental Material for
Chapter 4

B.1 Higher Resolution Paintings

We show 512x512 visual abstraction and styles landscapes paintings with 300 strokes (Figures 4.4 and B.1) by using the set of strokes parameters output by the network and rendering with a non-differentiable renderer at a higher resolution. Figures 4.4 and B.1 (b) show smooth abstraction using pixel loss at all intermediate canvases. Figures 4.4 and B.1 (c) show artistic style with constraints on thickness and length. Figures 4.4 and B.1 (d) show artistic style given by rapid motions. Figures 4.4 and B.1 show that our method provides a very fine-grained and interpretable control with the potential to expand the styles more, as opposed to previous work that have no control over nuances and details of their paintings. This is further shown in Figure B.7.

B.2 Ablations on Model Architecture

This section shows ablations on both presented networks: visual abstraction net, based on FC layers in decoder, and precise and style net based on FC and LSTM layers in decoder. For simplicity, we name the former AbstNet (which uses $D_{FC}$) and the latter Precise-Style-Net (which uses $D_{LSTM}$). Note that Precise-Style-Net is used for precise reconstructions (Chapter 4 - Figure 1 (b) and Figure 5 (left)) as well as artistic styles (Chapter 4 - Figure 1(h-j) and Figure 6 (left)), while AbstNet is only used for visual abstractions (Chapter 4 - Figure 1(c-f) and Figure 5 (right)). We first ablate on network sizes and encoder-decoder combinations for our AbstNet.

The left part of Figure B.2 (a-c) shows the full architecture of three abstraction networks, ordered with increasing number of parameters, from left to right. Figure B.2 (d) shows the full architecture of our Precise-Style-Net, which has ~90M learnable parameters. We find a relation between number of learnable parameters and abstraction quality. Smaller networks with ~30M and ~50M parameters (Figure B.2 (a),(b),
Figure B.1: 512x512 300-stroke paintings These examples illustrate results with higher resolution, and show how our method can provide nuanced, interpretable control over styles through varying losses and other interpretable factors. (b) shows a basic style by smooth visual abstraction. (c) adds constraints on thickness and length that reduces the number of small strokes. (d) add random noise to strokes resulting in a scratchy style.

respectively) are able to produce higher quality visual abstractions. We find that increasing the spatial size of the encoded feature map (a) doesn’t affect the output painting significantly. However, reducing the number of parameters of the network increases the abstraction capability, while increasing the number of parameters Figure B.2 (c) (∼100M) results in better approximations to the input image and thus, reducing the level of abstraction. The result of these ablations are reflected in the following sections.
Figure B.2: **Model Ablations.** Ablation on network architectures for visual abstractions for 300 strokes (a–c) and precise-style network (d). (a) Encoder uses the first 3 residual blocks of Resnet-18 and an additional convolutional layer to extract a larger feature map. Decoder is composed by 1 linear layer followed by non-linear activation. Learnable parameters: \( \sim 31M \). (b) Abstract net used in this paper. Encoder uses the first 4 residual blocks of Resnet-18. Decoder has 2 linear layers followed by non-linear activations. Learnable parameters: \( \sim 50M \). (c) Encoder is the same as (b). Decoder is composed by 3 linear layers followed by non-linear activations. Learnable parameters: \( \sim 100M \). (d) Precise-Style network, used for precise reconstructions and styles. Decoder is composed by a linear layer, followed by a bidirectional LSTM, followed by a final linear layer. Learnable parameters: \( \sim 90M \).

### B.2.1 Visual Abstraction

Figure B.3 shows model ablations on the coarsest visual abstraction using objective \( \mathcal{L} = ||I - C_T||_1 \), where \( C_T \) is the canvas at the last time step. All paintings are made with 300 strokes. Except for the bottom row, which uses Precise-Style-Net (Figure B.2 (d)), the other rows use different versions of AbstNet, specifically, second row uses Figure B.2 (a), third row uses Figure B.2 (b), and fourth row uses B.2 (c). Smaller networks produce better (coarser) abstractions than bigger networks, which is desirable for this objective function. We find that the architecture shown in Figure B.2 (b) produces better results for the range of visual abstractions that we present in this work.

<table>
<thead>
<tr>
<th>Method</th>
<th>( \mathcal{L}_1 ) ↓</th>
<th>( \mathcal{L}_{perc} ) ↑</th>
<th>Num Strokes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbstNet ( D_{FC-S} )</td>
<td>0.038</td>
<td>0.738</td>
<td>300</td>
</tr>
<tr>
<td>AbstNet ( D_{FC-L} )</td>
<td>0.043  <strong>0.750</strong></td>
<td></td>
<td>300</td>
</tr>
<tr>
<td>Precise-Style-Net ( D_{LSTM} )</td>
<td><strong>0.035</strong></td>
<td>0.747</td>
<td>300</td>
</tr>
</tbody>
</table>

Table B.1: Quantitative results on same held-out set on CelebA as in Table 1 in main paper, using networks explained in B.6. Precise-Style network achieves a much lower \( L_1 \) score than AbstNet. The latter, however, achieves a slightly better perceptual score. However, as shown in Figure B.6, it produces slightly worse reconstructions.
Figure B.3: **Model ablations on visual abstraction.** All visual abstraction paintings are made with 300 strokes. Top row: original image. Second row: AbstNet using architecture shown in Figure B.2 (a). Third row: AbstNet using architecture shown in Figure B.2 (b). Fourth row: AbstNet using architecture shown in Figure B.2 (c). Bottom row: Precise-Style-Net (Figure B.2 (d)). The goal of this method is to provide a coarse level of abstraction, and we find a relation between network size and level of abstraction. We see how second and third rows are better abstractions than fourth and last rows. There is not a substantial difference between second and third rows, and both are suitable for this abstracted style. For all visual abstractions shown in the paper, we choose to work with the network corresponding to the third row (B.2 (c)).

**B.2.2 Visual Abstraction Painting Progress**

Figure B.4 shows canvases rendered every 25 strokes. Top row shows visual abstraction given by applying $L_1$ loss at the final frame. This corresponds to a one-off optimization schema, which encourages a non-greedy behaviour. As a result, the network wastes the majority of the strokes, achieving a coarse abstraction with the last 50 strokes. Abstraction in middle row is achieved using $\sum_{t=0}^{T/k} ||I - C_{kt}||_1$, where $T/k$ is an integer, $k = 6$, and it corresponds to a sparse optimization schema, backpropagating errors every 50 strokes. Here only the first few strokes are wasted. The bottom row corresponds to a more guided regime which backpropagates the error at every time step.
This results in a greedy behaviour and no strokes are wasted.

B.2.3 Artistic Styles

Figure B.5 shows model ablations on artistic style given by constraining both stroke thickness and length. All paintings are made with 300 strokes. Figure B.5 (b) shows the results of this style using our Precise-Style network. This network produces clearer results, defining object boundaries better than columns (c) and (d). Even though column (b) and (c) use networks with a similar number of parameters, the LSTM layer seems provide a better structured sequence of strokes. This behaviour happens across all artistic styles and for precise reconstructions.

B.2.4 Precise Representations on Portraits

Figure B.6 demonstrates the improvement that Precise-Style net has over AbstNet on precise reconstructions on CelebA. We compare our Precise-Style LSTM-based network with two abstract FC-based networks, one with approximately the same number of parameters than our Precise-Style net (∼100M parameters, Figure B.2 (c)), and a smaller network with ∼30M parameters (Figure B.2 (a)). We see how the precise net outputs sharper paintings (Figure B.6 (d)) because it is able to capture finer details, such as main facial wrinkles, and expressions. This is perhaps more noticeable in rows 3 and 4. The paintings given by the smallest network, shown in Figure B.6 (b), are able to maintain the overall structure and composition, however, produce coarser paintings. Figure B.6 (c) improves upon (b) but fails to accurately eyes and and facial gestures. Metrics corresponding to this study are reported in Table B.1. Even though the larger AbstNet gives a slightly better perceptual score, overall the image is better reconstructed with Precise-Style net.
B.3 Precise Representations on Landscapes

In this chapter, we introduce a fine-tuning step consisting of a 100-step stroke parameter optimization to achieve very accurate representations with 300 strokes. We use Adam optimizer with a learning rate of 0.002. Figure B.7 (b) and (c) shows a comparative between paintings resulting from our precision-style network with and without such a fine-tuning step, respectively. Our Precise-Style network without such fine-tuning step is able to produce a good reconstruction of the input image. However, applying the fine-tuning step results in a more realistic looking image.

B.3.1 Comparison with SOTA on Landscapes

We provide further comparison with previous work on landscapes. Table B.2 shows quantitative results using 300 strokes (left), and 1000 or more strokes (right). Metrics are calculated using 100 samples from a held-out set. We use Precise-Style network, which uses LSTM in the decoder, \(D_{LSTM}\). Our network with a fine-tuning step outperforms previous methods. Figure B.7 shows a comparison of our networks with previous work. Our network without fine-tuning step seems to maintain a more consistent use of colors, and the paintings are slightly less noisy than the RL method by Huang et al. [31] (Figure B.7 (d)). PaintTransformers [44] (Figure B.7 (e)) and Stylized Neural
Figure B.6: Model ablations on reconstructions. This figure compares results on precision on StyleGAN generated faces obtained by using three different architectures using 300 strokes. Column (b) uses AbstNet with one linear layer (Figure B.2 (a)). Column (c) uses AbstNet with 2 linear layers (Figure B.2 (c)). Column (d) uses our Precise-Style-Net with decoder $D_{LSTM}$. All of them capture overall composition and facial gestures. However, (b) produces coarse strokes leaving some facial features such as eyes undefined. Color gradients are not smooth. (c) improves upon (b) but is unable to capture finer details such as facial gestures or eyes. (d) results in a sharper reconstruction, capturing finer details and edges better.

Painters [76] (B.7 (f)) share the same stroke model, resulting in a coarser representations and needing many more strokes to produce accurate representations of the input image.
Table B.2: Quantitative results on 100 random sample images drawn from a held-out set on landscapes. We provide metrics on a 300-stroke model (left), and a 1000-stroke model (right). For [44], total number of strokes is averaged over the held-out set. We measure $L_1$ loss and perceptual loss using cosine similarity between input image and painting feature maps (higher is better). For a fixed number of strokes, our method achieves a lower perceptual loss than previous methods.

Figure B.7: Precise reconstructions comparison with SOTA. All methods use 300 strokes. Our method with a fine-tuning step (a) is able to reconstruct the input image better. Without fine-tuning step (b) our method still outputs smoother paintings than previous work. This is perhaps more visible in second, third, and last rows. Paint Transformers [44] and SNP [76] output very large strokes and they suffer with reduced stroke budgets.
B.3.2 Stroke Budget Ablations

This section shows results a comparison of our precise network with 300, 600 and 1000 strokes in Figure B.8 columns (b), (c) and (d), respectively. Increasing the stroke budget while maintaining the resolution results in blurrier paintings. We report the metrics for the 300-stroke and 1000-stroke models in Table B.2. Metrics show that the 1000-stroke outperforms the 300-stroke model, however, the quality of the 1000-stroke painting is much worse than the 300-stroke painting, which indicates that other metrics might be more suitable for painting algorithms.

![Figure B.8: Stroke budget ablations. Precise-Style-Net using (b) 300 strokes, (c) 600 strokes and (d) 1000 strokes. Adding more strokes produces a blur effect on the painting.](image)

B.4 Energy Functions

Different energy functions in combination with different decoders, \( D_{FC} \) and \( D_{LSTM} \) give different levels of control of the painting, as shown in previous sections. In this section we show two the effect of two other loss functions.

B.4.1 WGAN-GP Loss

Following [49, 17, 31, 44], we also explore the impact of an adversarial loss for precise paintings. Specifically, we add the same discriminator network, \( D_{net} \), as [31], and use WGAN-GP [21] in our energy function. \( D_{net} \) discriminates between real pairs (input image, input image) \( Q \), and fake pairs (input image, painting) \( P \). WGAN-GP loss is:
\[ \mathcal{L}_{WGAN-GP} = \mathbb{E}_{P \sim P}[D_{\text{net}}(P)] - \mathbb{E}_{Q \sim Q}[D_{\text{net}}(Q)] + \lambda \mathbb{E}_{\hat{Y} \sim \hat{Y}}[||\nabla_{\hat{Y}} D_{\text{net}}(\hat{Y})||_2 - 1]^2 \]  

(B.1)

where \( P \) is the pair canvas-input image at the last time step, \( Q \) is the pair input image-input image, and \( \hat{Y} \) is sampled from \( Q \) and \( P \) with \( t \) uniformity sampled between 0 and 1, \( \hat{Y} = tQ + (1 - t)P \) with \( 0 \leq t \leq 1 \). We substitute perceptual by WGAN-GP loss in our general formulation, and thus, our energy function becomes:

\[ \Phi_{\text{precise WGAN-GP}} = \lambda_1 \sum_{t=1}^{T} \mathcal{L}_{\text{pixel}} - \lambda_2 \mathcal{L}_{WGAN-GP} \]  

(B.2)

We set \( \lambda_2 = 0.01 \). We find that adopting an adversarial schema does not improve perceptual losses, as shown in Figure B.9 (c). In some cases, it introduces some artifacts that make the painting a bit noisier, as seen in Figure B.9 bottom row. We hypothesize that this is partially due to the gap between distributions of real images and distributions of 300-stroke paintings.

**B.4.2 Contrastive Loss**

A contrastive approach might seem suitable for precise paintings. Same as in the previous section, we remove perceptual loss and keep pixel loss. The goal of this loss is to minimize the cosine distance of a vector representation of the input image \( \vec{v} \) with a vector representation of the canvas \( C_T \vec{w} \), while maximizing the distance between \( C_T \vec{w} \) and a set of negative samples \( Z = \{ \vec{z}_1, \vec{z}_2, ..., \vec{z}_K \} \). Contrastive loss is calculated as:

\[ \mathcal{L}_{\text{contrastive}} = -\log \frac{\exp(\text{dist}(\vec{w}, \vec{v})/\tau)}{\sum_{k=1}^{K} \exp(\text{dist}(\vec{w}, \vec{z}_k)/\tau)} \]  

(B.3)

We set \( K = 10 \) and randomly sample \( K \) images from the training set where \( k \neq i \), and \( i \) is the index of the vector representation \( \vec{v} \) corresponding to the positive sample. Our energy function becomes:

\[ \Phi_{\text{precise contrastive}} = \lambda_1 \sum_{t=1}^{T} \mathcal{L}_{\text{pixel}} - \lambda_2 \mathcal{L}_{\text{contrastive}} \]  

(B.4)

We set \( \lambda_2 = 0.0001 \) for our ablation. Figure B.9 (d) shows the paintings obtained using Equation (B.4). As we see, this loss does not improve results using perceptual loss, and results are inconclusive. We hypothesize that a more robust contrastive learning approach with more data augmentations or robust architectures as done in [25].
Figure B.9: **Effect of using WGAN-GP and contrastive losses.** We substitute perceptual loss by WGAN-GP (c) and contrastive loss (d) following Equation (B.3).