Exploring Bridges Between Creative Coding and Visual Generative AI

by

Jiaqi Wu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Master of Science
(Information)
in the University of Michigan
2024

Committee:

Professor Eytan Adar, Chair
Professor Steve Oney

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE

LIST OF TABLES

TABLE

ABSTRACT

How to bridge generative procedural art and visual generative artificial intelligence (AI) for visual content creation is an under-explored topic. On one hand, there are many cases where creative programmers can make use of generative AI, including stylizing canvas content and creating new content based on the existing styles of certain procedural art (style learning). On the other hand, existing approaches don't support creative programmers to flexibly leverage visual generative AI methods within the creative coding environment.

In this work, we explore **how to bridge generative procedural art creation and visual generative AI (specifically diffusion models) by programming functionalities integrated into the creative environment**. Specifically, we want to explore methodologies to condition/stylize art content and perform style learning upon procedural art via accessible interactions for artists and programmers.

We proposed two methods: GenP5, a novel p5.js library enabling generative procedural art creation with flexibly stylizing canvas content and conveniently condition art creation with pre-determined patterns; and P52Style, an extended library built upon p5.gui [1] allowing flexible adjustment of art content and leverage of visual generative AI for style learning tasks.

These terms are equivalent in the context of this thesis:

creative coding = generative procedural art = procedural art

generative AI = generative model

creative programmer = generative procedural artist

---

[1] https://bitcraftlab.github.io/p5.gui/

CHAPTER 1

Introduction

1.1 Motivation

The power of visual generative artificial intelligence (AI) can largely reduce human efforts when it cooperates with human judgments in visual content creation. Meanwhile, generative procedural art is the type of art where the artist includes a pre-designed generative or automatic process into the art creation process [5]. Currently, mainstream generative procedural art is based on programming algorithms (creative coding). Various programming languages/creative coding environments including Processing and p5.js [1] already make the creation of generative art efficient.

At the same time, how to bridge these two powerful computational methods for visual content creation is still an under-explored topic. **On one hand, there are many cases where creative programmers can make use of generative AI**. For example, it is not easy for a generative procedural artist who wants an artwork consisting of moving shapes in the style of Jackson Pollock. In this case, integrating generative AI into the procedural art system enables artists to generate moving shapes first, then apply prompt guidance that transforms those shapes into more stylized elements (e.g., "combination of angry acrylic splatters"). **On the other hand, existing approaches don't support creative programmers to flexibly leverage visual generative AI methods within the creative coding environment**. For example, artists have no control over detailed canvas contents when applying visual generative AI for stylization.

In this work, we explore **how to bridge generative procedural art creation and visual generative AI (specifically diffusion models) by programming functionalities integrated into the creative environment**. Specifically, we want to explore methodologies to condition/stylize art content and perform style learning upon procedural art via accessible interactions for artists and programmers.

---

[1] https://p5js.org/

In each chapter of this thesis, we focus on one of the research questions and propose our method and analysis. Specifically, we provide:

- GenP5, a novel p5.js library enabling generative procedural art creation with flexibly stylizing canvas content and conveniently condition art creation with pre-determined patterns

- P52Style, an extended library built upon p5.gui [2] allowing flexible adjustment of art content and leverage of visual generative AI for style learning tasks.

1.2 Background

1.2.1 Generative Procedural Art

Generative procedural art is the type of art where the artist includes a pre-designed generative or automatic process into the art creation process [5]. Sometimes they are also referred to as mathematical art to differ from art created by generative AI. There are three main types of generative/automatic processes [3]: **randomness (random variables, noises, distributions, etc), rules (algorithm instruction, mathematics formula, ecosystem simulation, etc) and natural systems (e.g., growing biological system)**. Though following the same principles, the specific manifestations of generative procedural art can vary [4] from digital to physical, static to dynamic, and shown in all kinds of styles (Fig. 1.1).

In this research, we focus on generative procedural art that is based on computer algorithms (creative coding) focusing on the process of rules and randomness. The growing popularity of the creative coding community has promised generative procedural art to have an important role in new media [18].

1.2.2 P5.js

Processing [5] is a graphics library and integrated development environment based on the Java programming language environment for visually oriented applications with an emphasis on animation and interactions. It has been one of the most important tools for procedural art creation and is popular among the creative coding and digital art communities. Inspired by

---

[2] https://bitcraftlab.github.io/p5.gui/
[3] https://www.amygoodchild.com/blog/what-is-generative-art
[4] https://openprocessing.org/discover/#/trending
[5] https://processing.org/

7102023
by garabatospr

Enfantines II. Enfantillages Pittoresques
by Alejandro

Genuary Day 4 - Next next Fidenza
by Amy Goodchild

Figure 1.1: Generative procedural art examples

and based on the core principles of Processing, p5.js [6] is a JavaScript client-side library that makes creative coding more accessible just as any other JavaScript web application.

A simple p5.js project has the following structure:

index.html: The main HTML document that the web browser loads when accessing the p5.js project. It serves as the container for p5.js sketch and typically includes links to the p5.js library, custom JavaScript code for art content creation (such as sketch.js), and any additional CSS stylesheets (like style.css).

style.css: This optional file contains Cascading Style Sheets (CSS) rules for styling the HTML elements in p5 project.

sketch.js: The sketch.js file contains artists' p5.js code to create art content, including functions that define the behavior of the project. Two key functions in this file are:

- setup(): This function is called once when the program starts. It's used to set up initial properties such as canvas size, background color, and initial environment settings. It runs only once.

- draw(): This function continuously executes the lines of code contained inside its block until the program is stopped or **noLoop()** is called. It's used to draw shapes, handle interactions, and animate objects on the canvas, by creating animations from frames.

1.2.3 Diffusion Models

One of the most popular visual generative AI nowadays is Diffusion Model (DM) (e.g., Diffusion Probabilistic Model [7]) that learns to gradually denoise an image step by step,

---

[6]<https://p5js.org/>

reversing the noise-adding process. Consequently, it is capable of generating images from pure noise. Furthermore, Latent Diffusion Model (LDM), for example, Stable Diffusion (SD) [13] applies the diffusion process over a lower-dimensional compressed representation of images to reduce the memory and compute complexity. By augmenting the underlying UNet backbone with the cross-attention mechanism [16], It is also possible to perform image generation tasks with various conditions, including DM with textual prompts (text2img DM) and DM with depth information (depth2img DM) [7], by encoding these conditional inputs into embeddings and feeding into the denoising process.

Fig. 1.2 shows the general architecture of a LDM, using the example of text2img DM [8]. With a text prompt and a seed as input, the latent seed is used to generate random latent image representations, and the text prompt condition is transformed into text embeddings via a certain encoder (in this example, CLIP's text encoder). Then the U-Net iteratively denoises the random latent image representations to output a noise residual, which is used to compute a denoised latent image representation via a scheduler algorithm. The denoising process is repeated by N times to step-by-step retrieve better latent image representations. Finally, the latent image representation is decoded by the decoder part of the variational auto-encoder (VAE). As discussed earlier, the U-Net, which performs the main denoising task, consists of a number of self-attention and cross-attention blocks [9, 10]. In the cross-attention module, condition embeddings (e.g., prompt text embeddings) serve as the key and value to inject the condition concerning the query features from the previous layer. Meanwhile, in the self-attention module, features with spatial dimensions generated from the previous layer will serve as key, query, and value by themselves [9].

The abilities of these conditional DMs enable many useful visual content-creation tasks. **Image styliztion** task is a common case of image translation, leveraging DM with image condition (img2img DM [10]). Instead of starting the diffusion process from the random latent image, it leverages a input reference image as the initial latent by adding noise upon the latent of the reference image. The noise added is controlled by a strength value (0-1), to adjust the level of influence of the initial image. **Style learning** task leverages the models' ability to learn the features of a customized visual style and use that style in the generation of new images (further discussed in Chapter 3). Other than the previously mentioned DM backbones, these tasks are also enabled by recent development of DM architectures, including fast inference architecture like latent consistency model [12] (LCM) enabling real-time image generation with a few step inference, and various DM style learning methods (further

---

[7]https://huggingface.co/stabilityai/stable-diffusion-2-depth
[8]https://huggingface.co/blog/stable_diffusion
[10]https://huggingface.co/docs/diffusers/v0.27.2/en/api/pipelines/stable_diffusion/
img2img#diffusers.StableDiffusionImg2ImgPipeline
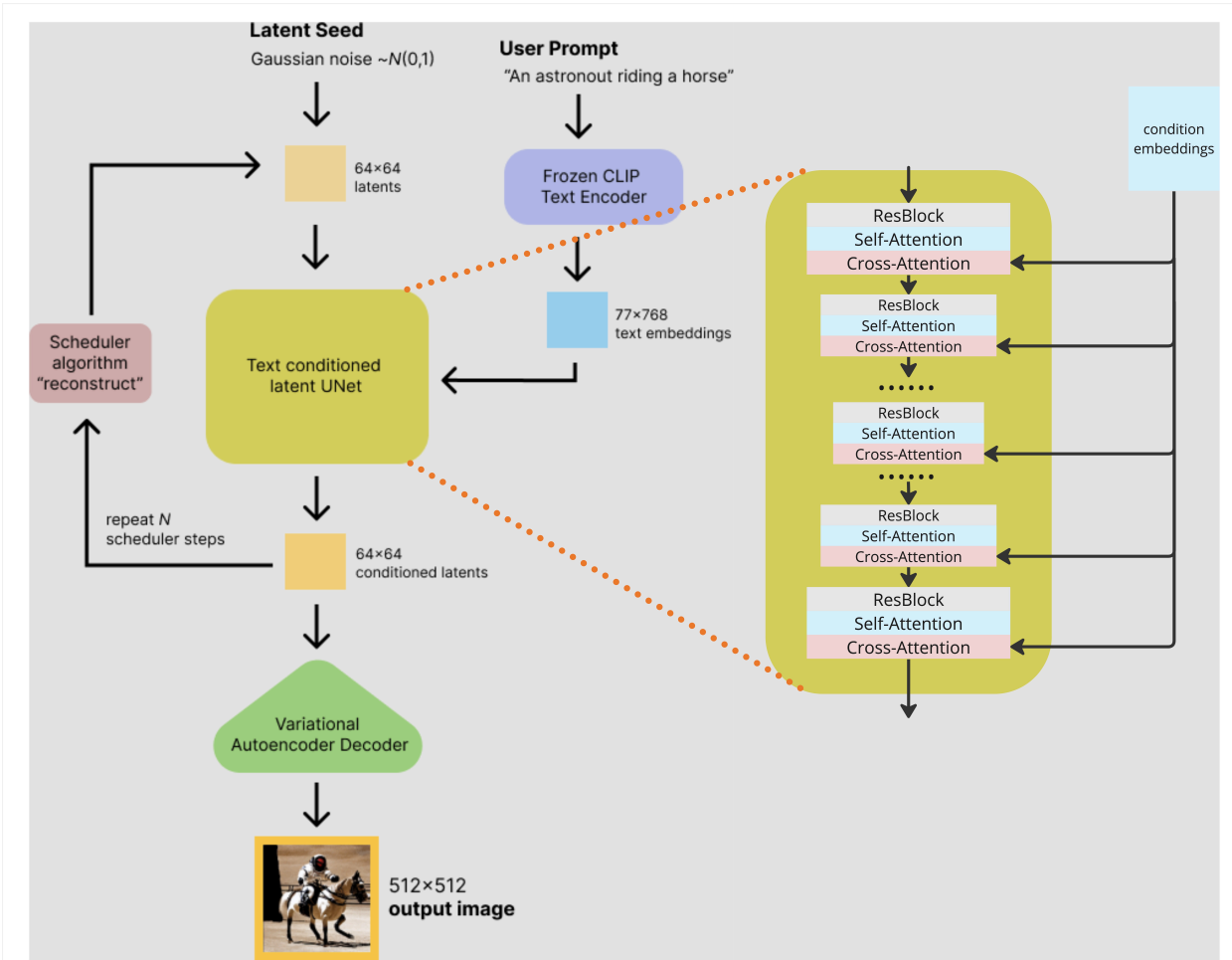
Figure 1.2: Latent Diffusion Model architecture [9]

discussed in Chapter 3).

CHAPTER 2

GenP5

In this chapter, we introduce GenP5, a novel p5.js library enabling generative procedural art creation with flexibly stylizing canvas content and conveniently condition art creation with pre-determined patterns. This enables artists to make full use of both the programmability of code and the stylization ability of DMs.

2.1 Related Work

2.1.1 Assisting the Creation of Generative Procedural Art with Generative AI

Our work GenP5 is one of the approaches using generative AI (specifically, we leverage DMs) to assist the creation of generative procedural art.

Relevant to our works, Liu el. al. [11] used music audio as an input condition, deploying a large language model like GPT4 and a DM like stable diffusion to generate music visualizations. SpellBurst [2] is an authoring tool leveraging large language models to facilitate users to create generative procedural art and explore various possibilities. Various artists, including Takafumi Oyama [1], Roope Rainisto [2], and Brian Jordan [3] used image or video generation models as post-processing step for procedural art to achieve various effects. Artists and technologists including Dae In Chung and Brian Jordan have been integrating generative AI into programming stages of procedural art to create novel design experiences, like programming with conversational text [4] and audio instructions [5].

Integrating DMs into a creative coding environment is still an under-explored area. Recent fast-inference DM approaches like Latent Consistency Model (LCM) [12] enabled real-time

---

[1] https://www.takafm.me/
[2] https://twitter.com/rainisto
[3] https://twitter.com/bcjordan
[4] https://twitter.com/cdaein/status/1756161096778682728
[5] https://twitter.com/cdaein/status/1759306141152985182

canvas stylization applications which are further extended to procedural art environments, by calling DM APIs explicitly or implicitly to send canvas images and retrieve stylized results. For example, Dae In Chung created a plugin for Ssam.js [6] creative coding helper, using Replicate Node.js API to generate SD image from HTML5 Canvas drawing. Meanwhile, it is still not possible to conveniently manipulate any canvas contents for stylization with DMs independently, and integrating DMs functionalities seamlessly into creative coding environments to enable an experience of "Use DM by code". This motivates the creation of our GenP5 library.

2.1.2 Assisting the Creation of Generative Procedural Art with Conditions

Much research has focused on extending the current generative art programming environment for more powerful design assistance and more broad applications. Some work focuses on using input conditions to guide the generated art results. Barile et.al. [3] explored a way of using generative programming to generate animated drawings, with a target input image as a condition. Based on it, Wu [19] used input images as conditions, by designing a system that can detect the saliency map of the input image and then apply it to render and generate an animation consisting of various strokes.

Integrating generative art programming into downstream application tasks can also be a meaningful exploration. Subbaraman and Peek [15] built p5.fab from the creative coding environment p5.js, which provides an interface to control digital fabrication machines from material exploration, fine-tuned control, and workflow iteration. These works further proved that generative art based on algorithmic programming can not only be useful for media and design applications but also applicable in more technical areas like fabrication and materials.

Inspired by all these works, we would like to explore how to condition procedural art with modular and encapsulated functionalities integrated into a p5.js library, to enhance the convenience of creating procedural art with various conditions and improve the whole creative coding experience. This motivates us to extend the GenP5 library with functions to condition canvas contents.

2.2 Method

Our proposed method, GenP5, is a customized JavaScript library that adds to the p5.js core functionality.

---

[6] https://github.com/cdaein/vite-plugin-ssam-replicate

2.2.1 Stylizing Canvas Contents

The first part of GenP5 is the functionalities to stylize canvas contents in real-time.

Overview With the GenP5 framework shown in 2.1. we first introduce some concepts. *original canvas* refer to the p5 canvas that is created by default in any p5.js project. *Stylize buffer* is the new off-screen graphics created that allow contents drawn there to be separated from the original canvas. Specifically, the contents drawn on these buffers will be stylized with DMs later. *Nonstylize buffer* is also the new off-screen graphics created independently, but the contents drawn on these buffers will not be stylized with DMs later. Normally drawn p5.js contents will be shown as rendered p5 animations of all the frames. However, to turn them into processable entities for DMs, we need to capture *frame images* at a certain capture rate. *Original frame images* are those images captured from different *stylize buffers*, with *frame index* as sequence numbers. They will be processed by DMs and result in *stylized frame images* as a one-to-one correspondence. Finally, all the *original frame images* from *original canvas* and all the *stylized frame images* in different *stylize buffers* who share the same *frame index* will be overlayed sequentially to create a single *final frame image* at this *frame index*. Notice that a background-removal algorithm will be applied to all the *stylize buffers* before they are overlayed. Moreover, For the p5 content animations in the *nonstylize buffer*, instead of turning the frames into frame images, we directly store each frame as p5 objects since they do not need to be stylized. In the final step, for all the *final frame images* and stored frames from *nonstylize buffer*, if they share the same *frame index*, they will be overlayed sequentially, to create final frames composing a final animation in a new *final canvas*.

UI components When users use GenP5 library to create p5.js projects, there will also be UI components generated automatically when initiating GenP5 instance. As shown in 2.2, **main p5 container** contains the *original canvas*. **Original frame image container** contains the *original frame images* captured from *stylize buffers*. There will be **N** containers when there are **N** *stylized buffers*. The images will be displayed sequentially by *frame index*. **Stylized frame image container** contains the *stylized frame images*. Similarly, there will be **N** containers when there are **N** *stylized buffers*. An *original frame image* will result in a *stylized frame image* of the same *frame index*. The images will be displayed continuously by *frame index*. **Final frame image container** contains the *final frame images*. There will only be one final image container. *Final frame images* at index **T** will not be created until all the *stylized frame images* at index **T** from different buffers are ready. The images will be displayed continuously by *frame index*. **Final canvas container** contains the final canvas
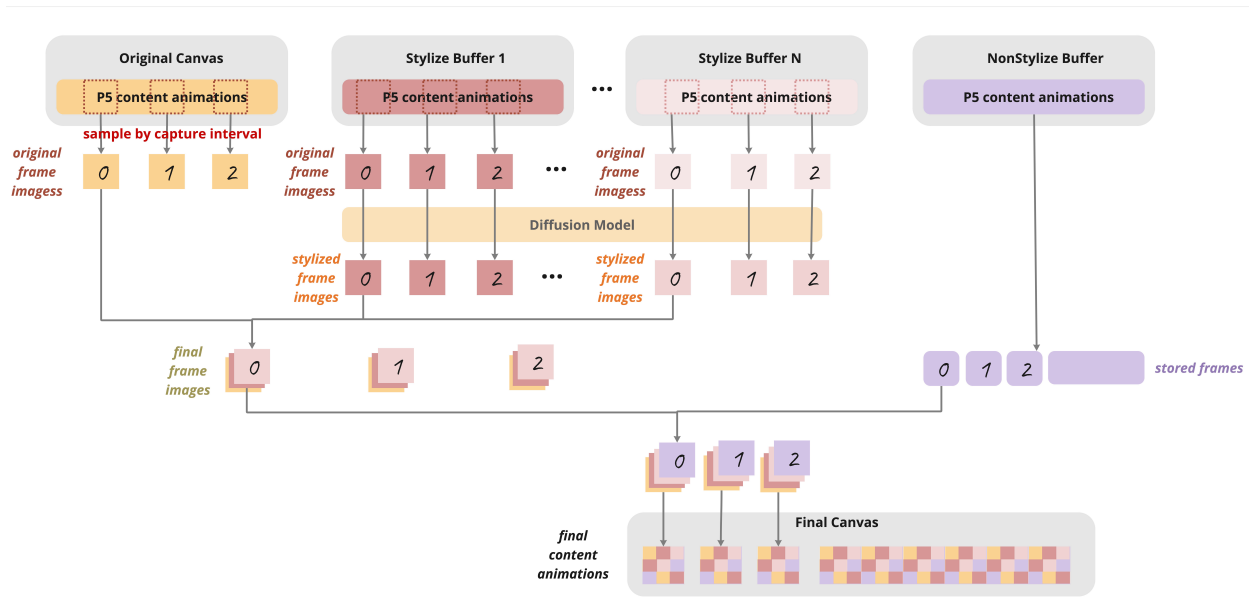
Figure 2.1: GenP5 method overview. numbers indicate *frame index*

and a button. When the button is clicked, a final animation will be displayed by combining stored frames of *nonstylize buffer* and *final frame images*. The speed of the animation is determined by the framerate.

Core Functionalities Specifically, users will directly interact with the following main functionalities shown in 2.1:

The following sketch.js code snippet illustrates a simple project using GenP5 library for content stylization.

```
1  let genP5;
2  var storedframes = [];
3  let captureinterval = 5;
4  let finalframerate = 30;
5  //initiate more variables
6  //......
7
8  function setup() {
9      genP5 = new GenP5(canvas_size, canvas_bgcolor);
10     //initiate genP5 object with canvas size and background color
11
12     [buffer1, buffer2] = genP5.createstylizebuffers(2);
13     //create 2 buffers to draw stylized contents
14
15     [buffer3] = genP5.createnonstylizebuffers(1);
```

10
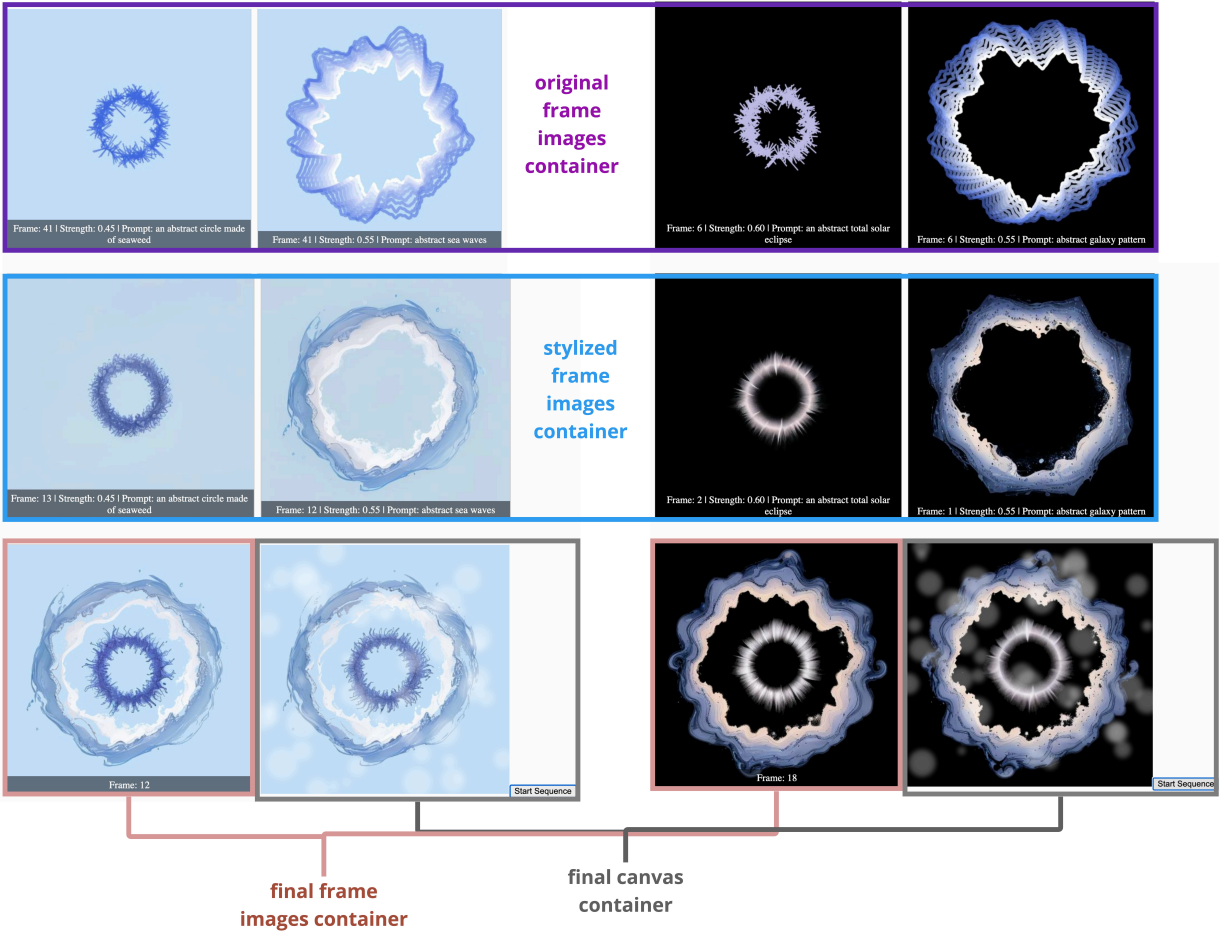
Figure 2.2: UI elements that will be dynamically created when GenP5 library is used. The above example shows two different project examples achieving different effects by applying different prompts on similar p5 contents. In this example, nothing is drawn on the main canvas other than the background color. 2 *stylize buffers* are created each containing a ring-shape animation. 1 *nonstylize buffer* is created containing bubble effect filters.

Function Name	Parameters	Return Value	Introduction
create_stylizebuffers	buffernumber (number)	array of buffer objects	Create *stylize buffers*
create_nonstylizebuffers	buffernumber (number)	array of buffer objects	Create *nonstylize buffers*
clear_stylizebuffercontent	buffer (p5.Graphics)	none	Unlike buffer.clear(), this function clears everything on the buffer except the background color.
stylize_buffers	stylizebufferlist (array), promptlist (array), strengthlist (array), captureinterval (number), canvas (canvas)	none	Continuously capture the *original frame images* from different *stylize buffers* and the *original canvas*, get *stylizes frame images* and finally creates *final frame images* for all the frames.
setup_finalcanvas	finalframerate (number), storedframes (array)	none	Create a final canvas view with a button. When clicked, render the final animation composed of *final frame images* and stored frames from *nonstylize buffer*. The speed of the animation is determined by finalframerate

Table 2.1: GenP5 main function descriptions and details.

```
16      //create 1 buffer to draw not stylized contents
17
18      genP5.setupfinalcanvas(finalframerate, storedframes)
19      //create final canvas frame view + button to render final animation
20  }
21
22  function draw() {
23          genP5.clearstylizebuffercontent(buffer1) //clear contents for next
        frame
24          //draw contents in buffer1
25          //......
26          genP5.clearstylizebuffercontent(buffer2) //clear contents for next
        frame
27          //draw contents in buffer2
28          //......
29          buffer3.clear() //clear contents for next frame
30          //draw contents in buffer3
31          //......
32          storedframes.push(buffer3.get()); // Store everyframe of buffer3
        because it is not stylized buffer
```

```
33
34        promptlist =['prompt for buffer1', 'prompt for buffer2']
35        strengthlist = [strength1, strength2]
36
37        genP5.stylize_buffers([buffer1, buffer2], promptlist, strengthlist
    , 5, canvas);
38        //stylize buffer1, buffer2
39 }
```

Listing 2.1: example

Usage and implementation detail GenP5 uses DM by creating a web socket connection between p5.js project and a server.js in a node.js project. Server.js takes in images and stylizes images by calling LCM API from fal.ai [7]. The server contains queueing, checking, and regenerating logic to make sure the *stylized frame image* of every *frame index* is returned to p5.js project sequentially.

In summary, to use GenP5 for procedural art creation, the user only needs to start server.js either remotely or locally, place library code in the same folder of the p5.js project and declare the library in the HTML file. There are no other things to be changed other than calling the above main function in sketch.js, without any requirements for devices. Here is an outline of the workspace structure:

- (remote or local node.js project folder)

  server.js

- (p5.js project folder)

  genp5lib.js,

  index.html,

  sketch.js

2.2.2 Conditioning Canvas Contents

The second part of GenP5 is the functionalities to condition the art creation with predetermined patterns (e.g., color, shape). These predetermined patterns are provided by external images.

Core Functionalities Specifically, users will directly interact with the following main functionalities shown in 2.2

---

[7]https://fal.ai/

13

Function Name	Parameters	Return Value	Introduction
get_NSampledColors	imagepath(str), numcolors(int)	sampled colors (arrary, promise resolve)	sample colors from an image
load_ContourMap	imagepath(str)	call back function	load the contour map of an input black-white image
find_NearestContour	positionx(number), postiony(number), searchrange(number)	nearest contour position (vector)	Given input coordinate and a search range, return the nearest coordinate that is on the contour map.
Sample_ContourPoints	numpoints(number)	sampled points(arrary)	sample points from the contour
Check_IfOnContour	positionx(number), positiony(number)	True/False	Check if a coordinate is on the contour

Table 2.2: GenP5 main function descriptions and details, cont.

Usage and implementation detail We provide 4 simple examples of procedural art projects using the above functions for conditioning in 2.3. Similarly, users need to place GenP5 library code, and pre-determined image in the same folder of the p5.js project and declare the library in the HTML file. There are no other things to be changed other than calling the above functions in sketch.js, and there are no requirements for devices. Since these functions do not rely on DMs, they don't need the node.js server.

- (p5.js project folder)

  inputimage.png,

  genp5lib.js,

  index.html,

  sketch.js

The following code snippet illustrates how to use GenP5 conditioning functions to create a simple project (line II in 2.3), by controlling the color, initiated location, and movement of art elements. It creates a moving and growing particle system by conditioning the initiation of particles to be on the pre-determined contours and conditioning the movement of particles to be within the contours. The color of the particles is randomly selected from a color palette from the condition of the input color image:

```
1 //initialization
2 let particles = [];
3 maincanvassize = 400
4 let sampledColors = []
```

14
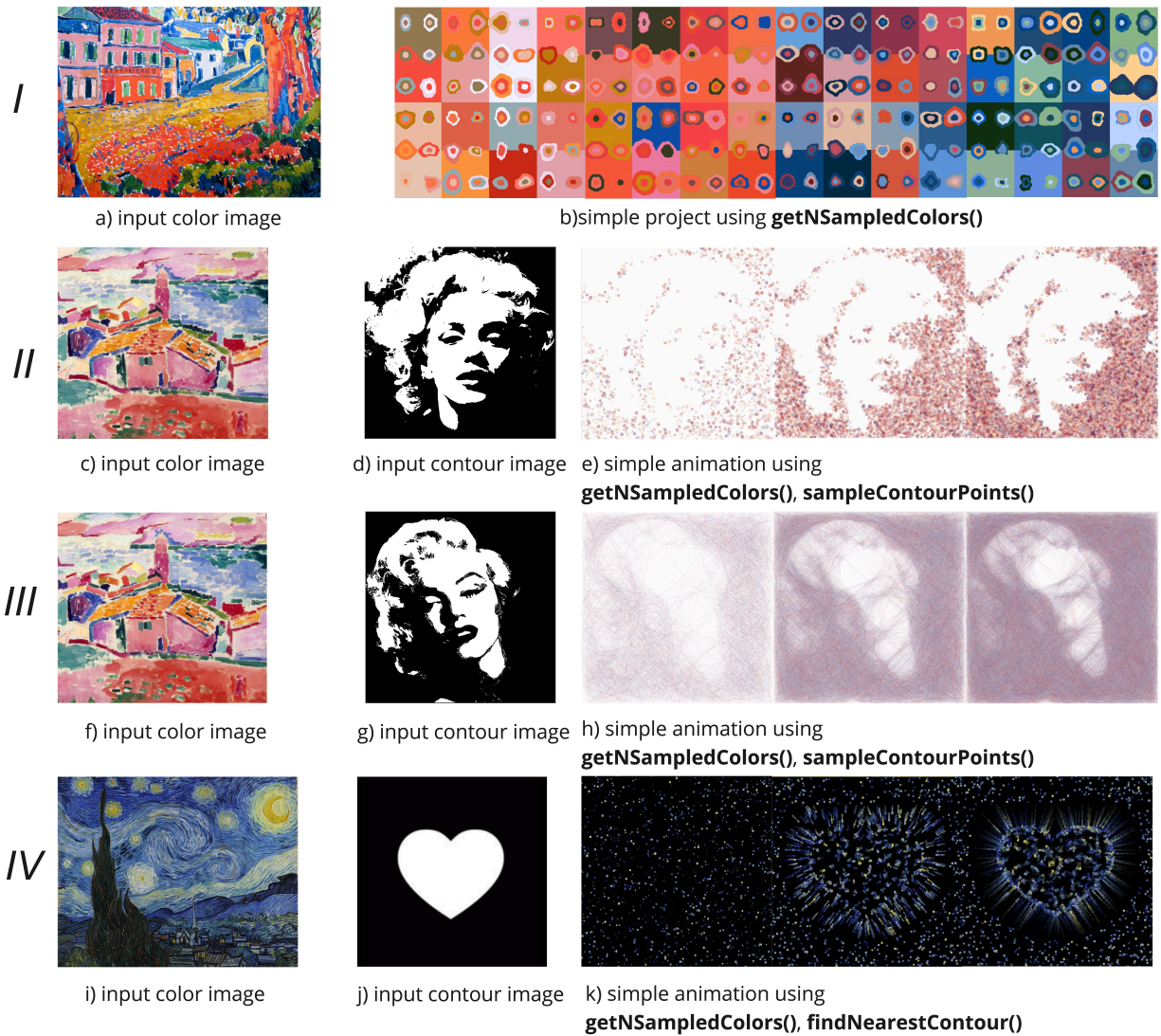
Figure 2.3: 4 simple examples of procedural art project using GenP5 functions for conditioning. The input contour maps are inverted. Input color image source: Restaurant de la Machine à Bougival - Maurice de Vlaminck (row I), View of Collioure - Henri Matisse (row II-III), The Starry Night - Vincent van Gogh(row IV)

```
5
6  function setup() {
7      //initialize GenP5
8      genP5 = new GenP5(maincanvassize, '#FFFCFD');
9      // get the sampled colors from image
10     genP5.getNSampledColors('./colorimage.png', 16).then((colors) => {
11         sampledColors = colors;
12         //load contour map from image
```

15

```
13          genP5.loadContourMap('./contourimage.png', setParticles);
14      }).catch((error) => {
15      console.error("Failed to load or sample colors:", error);
16  });
17
18  }
19
20  function draw() {
21
22      if (frameCount % 10 == 0) {
23          addParticlesBatch(1000);   // Adjust the number of particles per
      batch as needed
24      }
25      clear()
26      for (let p of particles) {
27        p.move();
28        p.update();
29        p.display();
30      }
31  }
32
33  function setParticles() {
34    particles = [];
35    let sampledPoints = genP5.sampleContourPoints(6); // sample particles on
       contour
36    for (let point of sampledPoints) {
37      particles.push(new Particle(point.x, point.y));
38    }
39  }
40
41  function addParticlesBatch(numParticles) {
42      let sampledPoints = genP5.sampleContourPoints(numParticles); // grow
      particles on contour
43      for (let point of sampledPoints) {
44          let color = random(sampledColors); // Select a random color
45          particles.push(new Particle(point.x, point.y, color));
46      }
47  }
48
49  class Particle {
50    constructor(x, y) {
51      this.posX = x;
52      this.posY = y;
```

```
53
54      // Random base size for each particle
55      this.size = random(1, 8);
56
57      // Random speed factors
58      this.xSpeedFactor = random(-5, 5);
59      this.ySpeedFactor = random(-5, 5);
60
61      // Select a random color from a predefined set and add random alpha
     for transparency
62      this.color = random(sampledColors);
63      this.alpha = random(30, 150);  // Adjusted for better visibility of
     the blur effect
64      this.c = color(this.color);
65      this.c.setAlpha(this.alpha);
66    }
67
68    move() {
69      this.posX += random(-this.xSpeedFactor, this.xSpeedFactor);
70      this.posY += random(-this.ySpeedFactor, this.ySpeedFactor);
71
72      //restrict the movement of particles in the contour
73      // Check if the new position is on the contour; if not, re-generate
     the position
74      let onContour = genP5.checkIfOnContour(this.posX, this.posY);
75      while (!onContour) {
76        this.posX = random(width);
77        this.posY = random(height);
78        onContour = genP5.checkIfOnContour(this.posX, this.posY);
79      }
80    }
81
82
83    update() {
84      // keep particles on canvas
85      if (this.posX < 0) this.posX = maincanvassize;
86      if (this.posX > maincanvassize) this.posX = 0;
87      if (this.posY < 0) this.posY = maincanvassize;
88      if (this.posY > maincanvassize) this.posY = 0;
89    }
90
91    display() {
92      // Set the stroke weight to the particle's size for the blurriness
```

```
        effect
 93     strokeWeight(this.size);

 94

 95     // Use the particle's color including its alpha for transparency
 96     stroke(this.c);

 97

 98     // Draw the particle
 99     point(this.posX, this.posY);
100   }
101 }
102 % }
```

Listing 2.2: example

As shown in the code snippet, **getNSampledColors()** and **loadContourMap()** need to be used in line with the asynchronous nature of JavaScript. This ensures the contour map and color palette are loaded properly before other functions.

It is worth notifying that using real images for conditioning is not the ideal way to make use of GenP5 conditioning functions. **GenP5 conditioning functions are designed to leverage input patterns pre-generated by DMs, to enable turning randomness in generative models to manipulatable conditions in p5.js**.

2.3   Evaluation

Since there are no similar existing methods comparable to GenP5, we conduct the evaluations using Cognitive Dimensions of Notations (CDs) [4]. We score each dimension as high, low, mid, and standard (indicating the same level as any other p5.js or programming libraries, the level is acceptable as long as the user has proper knowledge of p5.js programming).

Low *Viscosity: Resistance to Change*. The library is not viscous, and the changes in content code don't require changes in GenP5 function calls. The changes can be easily made by adjusting function parameters or adding/deleting function calls.

Mid *Visibility: Ability to View Components Easily*. The library has simple UI components with little/no labels and captions.

Standard *Premature Commitment: Constraints on the Order of Doing Things*. Like any other p5.js library, initiating instances, including package code, and a certain sequence of executing is necessary.

Low *Hidden Dependencies: Important Links between Entities Are Not Visible*. There are certain dependencies in functionalities, but they are explicit in code information.

High *Role-Expressiveness: The Purpose of an Entity Is Readily Inferred*. The library

18

follows the cognitive habit of users as it has the same structure as any other p5.js library. The purpose of different functions is well elaborated with function names and documentation.

Mid *Error-Proneness: The Notation Invites Mistakes and the System Gives Little Protection.* The library functions have simple and explicit usage, but there might be user mistakes with no clear system feedback.

Standard *Abstraction: Types and Availability of Abstraction Mechanisms.* The library functions have the same abstraction level as other p5.js libraries.

Low *Secondary Notation: Extra Information in Means Other Than Formal Syntax.* The library functions don't require users to record extra information like comments.

High *Closeness of Mapping: Closeness of Representation to Domain.* The purpose of different functions is well elaborated with function names and documentation, whereas the closeness of functions and behavior is standard in programming libraries.

High *Consistency: Similar Semantics Are Expressed in Similar Syntactic Forms.* The library has high consistency, as a feature of programming languages.

Low *Diffuseness: Verbosity of Language.* The function calls have low diffuseness, as a feature of programming languages. For UI components, the label texts are reduced to the minimum.

Standard *Hard Mental Operations: High Demand on Cognitive Resources.* The function calls have a standard level of hardness as programming languages. For UI components, the icons and texts are easy to understand without cognitive burden.

Standard *Provisionality: Degree of Commitment to Actions or Marks.* The library has a standard level of provisionality as any other programming library.

High *Progressive Evaluation: Work-to-Date Can Be Checked at Any Time.* The library has a high progressive evaluation as any other programming library.

In summary, the evaluations by CDs indicate that GenP5 has fair usability in most of the dimensions, meanwhile it requires improvements in Visibility and Error-Proneness. To improve on these limitations, in future work, we hope to increase Visibility and Error Proneness with more detailed UI labels and error messages to help better explain the problem and guide users to find the solution.

2.4   Conclusion

In conclusion, to explore the flexible manipulation of procedural art contents with DMs within the creative coding environment, we proposed GenP5, a novel p5.js library enabling generative procedural art creation with flexibly stylizing canvas content and conveniently conditioned art creation with pre-determined patterns. We also provided detailed documen-

tation and library usage examples, and evaluations based on CDs to show the usability of the library.

2.5 Limitations and Future Work

Due to the time limit, there are certain limitations in GenP5 that we leave to future work.

Firstly, we hope to conduct qualitative analysis and user evaluations through real user studies to test the usability of the tool.

Secondly, current GenP5 only supports overlaying non-stylized contents before and after stylized contents. We would like to further enable a more flexible order of stylized/non-stylized contents.

Thirdly, current approaches of retrieving processed images from DM are restricted by API calls and network traffic, we hope to further reduce latency by optimizing our workflow and enabling local execution of DM.

Lastly, we provide simple principles of conditioning the p5 content with function calls, but in practice, more complex functions for conditioning would be required (e.g., non-binary masks, sample color range, generating skeleton of images other than contours), as well as logics to use DM to retrieve pre-generated images to use as input patterns. We leave these to future development.

CHAPTER 3

P52Style

In this chapter, we introduce P52Style, an extended library built upon p5.gui [1] (a library enabling gui panels for p5 variable adjustment) allowing flexible adjustment of art content and usage of DM for style learning tasks.

3.1 Background

3.1.1 What is Style?

The style of visual content always has a vague definition, with various expectations in different tasks. AesPA-Net [8] concluded that *"The unique pattern repeatability in each image defines its own style"*, indicating that not only the element but also the repeatability of the local patterns is essential to representing style. Inspired by this, and given this thesis's main focus of generative procedural art which is mostly composed of computational art units (e.g., particles, shapes), we propose **our decomposition of style** in this work, which contains:

- Style unit set: it is created by keeping dividing a style reference image into unit pieces and merging similar unit pieces until getting a set that:

  1) Each unit in the set is different in either rough shape, texture, or dominant color. We don't consider small color differences (e.g., same dominant color containing different color degrees or small noise of other colors) or layout direction as a standard to differ style units (e.g., same stroke with different direction in Fig. 3.1 [2](I) are considered the same unit; units with noised color in Fig. 3.1(IV) are not further divided).

---

[1]https://bitcraftlab.github.io/p5.gui/

[2]Style reference image sources:

I:https://openprocessing.org/sketch/2174599

II:https://openprocessing.org/sketch/1983166

III:https://openprocessing.org/sketch/2169008

IV:https://medium.com/red-buffer/johnson-et-al-style-transfer-in-tensorflow-2-0-57cfcba8af36

V:https://github.com/LouieYang/stroke-controllable-fast-style-transfer

2) Each unit in the set is no longer dividable. This means if we further divide a style unit, the resulting style units all have the same rough shape, texture, and dominant color. As demonstrated in Fig. 3.2, set A contains over-divided units with the same shape, texture, and dominant color (stroke, line drawing, blue), and set B doesn't contain the smallest set which can be further divided.

For an image composed of segmentable pieces (e.g., Fig. 3.1 (II-IV)), we divide the image by simply segmenting it. For an image composed of complex elements like vague/continuous transition of color or numerous strokes (e.g., Fig. 3.1 (I)) or realistic photo (e.g., Fig. 3.1 (V)), we divide the image by square patches.

Note that when a potential divide breaks a unit that is repeated in the image for more than N_{rep} [3] times, we do not further divide that unit and keep it as a single style unit (e.g., in V in Fig. 3.1, the stamens and petals can be further divided apart according to previous definition, but kept as one unit because they appear in reference image lots of times).

given shape S, dominant color C, texture T, for a style image with N style units, style unit set $= \{U_1, U_2, U_3, ...U_N\}, U_i = (C_i, S_i, T_i)$

- The composition method of style units. There are various possibilities of composition methods, containing both structural (e.g., far apart, fit, overlay) and functional (e.g., neat overlay, messy overlay) information. This can be interpreted as the mixing method and composition of strokes in traditional painting.

- Four-dimensional freedom $\{F_c, F_s, F_{comp}, F_l\} \in [0, 1)$:

1) The freedom of the color and shape variability of style units beyond reference image. This can be interpreted as the color scheme and shape feature of strokes in traditional painting.

Given the number of colors in the unit set N_c, color freedom $F_c = 1 - \frac{1}{N_c}$. When there is only 1 color in the unit set, $F_s=0$, meaning zero freedom in color variability. When there are a large number of different colors in the unit set, $F_c \to 1$, 1 means maximum freedom in color variability.

 Given the number of shapes in the unit set N_s, shape freedom $F_s = 1 - \frac{1}{N_s}$. When there is only 1 shape in the unit set, $F_s=0$, meaning zero freedom in shape variability. (e.g., in Fig. 3.3, zero freedom in shape variability when there is only 1 shape in the style unit set (circle and rectangle respectively)). When there are a large

---

[3]a big number, like 20

number of different shapes in the unit set, $F_s \to 1$, 1 means maximum freedom in shape variability.

2) The freedom of style unit composition method F_{comp}. This controls the freedom of how the style units are composed. Based on the intuition that the more pattern repetitive an image is, the more fixed the style unit composition method is, we define $F_{comp} = \frac{1}{\alpha_{\mathbf{style}}}$, **where $\alpha_{\mathbf{style}}$ is the pattern repeatability measurement in AesPA-Net [8]**.

3) The freedom of content leak degree. This will only be taken into account when the style reference image has concrete objects (e.g., houses, flowers, stars). Freedom of content leak degree is how much we tolerate the new images to contain objects inside the style reference image and not specified in the prompt.

Given the area of concrete objects in style reference image A_o, the number of concrete objects in the image N_o, and the total area of style reference image A_t, freedom of content leak degree $F_l = \gamma \cdot \frac{A_o}{A_t} + (1-\gamma) \cdot \frac{N_o}{10+N_o}$. γ is hyperparameter that is usually 0.4.

Note that due to the complexity of visual elements, the precise numbers of freedoms are usually hard to calculate, meanwhile, approximate values are good indications of the characteristics of the styles (high or low freedom).

Fig 3.1 shows a simple illustration of the components of our style definition. It is worth noticing that given the complexity of images, **it is usually impossible to visualize the actual style units and composition methods by limited images**, especially if we only sample from the style reference image. For example, example *III* of Fig 3.1 contains style composition method of *"Fit neat to each other"*, and The illustrations for the composition method is only one of the infinite ways the units can fit to each other. Although we proposed general definitions of style components, given the complexity of visual contents, each style component can be task-specific. Instead of acting as the rule for defining each style, our proposed style components are more a way of evaluating and comparing how different images represent specific styles given the specific task and comparing the features of different styles.

3.1.2 Style Learning Task

One of the many popular themes for visual computing is teaching the model to learn a customized style. Since style is often a vague concept [17], there are many different focuses and expectations of style learning. In this work, we define **style learning task** by specifying the following scenario and expectation:
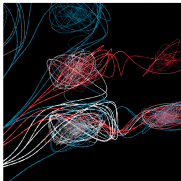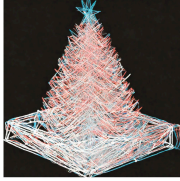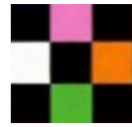
	style reference image	unit set	composition method	unit shape	unit color	composition	content leak	new image using the style
		Style Component 1	**Style Component 2**	**Style Component 3: Freedoms**				
I				Low	Low	Mid	N	a Christmas tree
II				Low	High	Low	N	a man
III				High	High	Mid	N	a cat
IV				High	High	Mid	Low	a car
V				Low	Low	Low	High	a girl

Figure 3.1: Example images and their style components. New images are generated by Visual Style Prompting. **N** indicates no consideration for content leak for the reference image.

| style reference image | candidate style unit set | finite | smallest set |

set A — black background + parallel, radial, messy mixed colorful line scribbles — finite ✓ — smallest set ✗

set B — black background + smooth lines of color white, blue, red, ... — finite ✓ — smallest set ✓

set C — black background + blue line of angle1, angle2, angle3,.......angleN — finite ✗ — smallest set ✗

Figure 3.2: Example correct (B) and incorrect (A, C) style unit sets. Reference image source: Inner Demons by Che-Yu Wu

Paul Signac, *Femmes au Puits*, 1892, showing a detail with constituent colors. Musée d'Orsay, Paris

Piet Mondrian *Composition A*, 1923

Figure 3.3: Two famous art examples with zero freedom of shape variability.

- There are limited style reference images

- The reference image might be semantically and visually far from what we want to model to inference, or contain a complex combination of style units (like examples in Fig. 3.1)

- The model is capable of generating any new content precisely with similar style units, composition methods, and the same levels of freedom as the style reference image.

3.1.3 Style Learning Task using DMs

Style Learning Tasks using DM (DM) have been widely explored in recent years. The naive approach would be simply using the textual description of the style in the prompt, but this yields two main problems: 1)It is hard to precisely describe a style in natural language, especially when the style is abstract; 2)The model would not be able to represent customized style which the model didn't see in the training stage. Many works have addressed the problem in various ways. **Few-shot tuning-based approach** (DreamBooth variants [14]) fine-tune the DMs with reference images in a few-shot manner 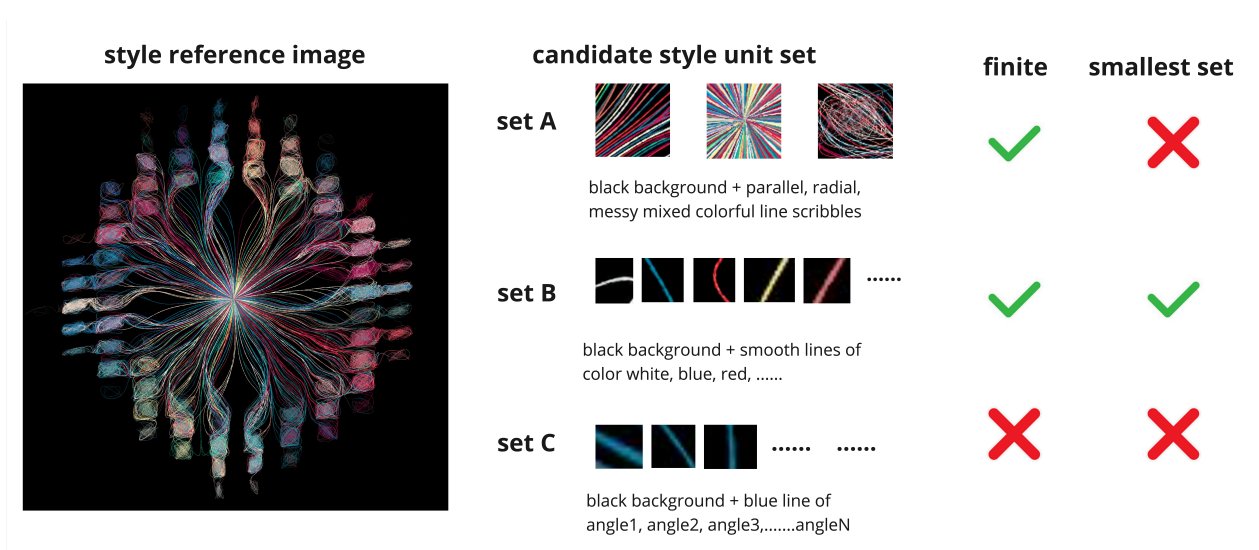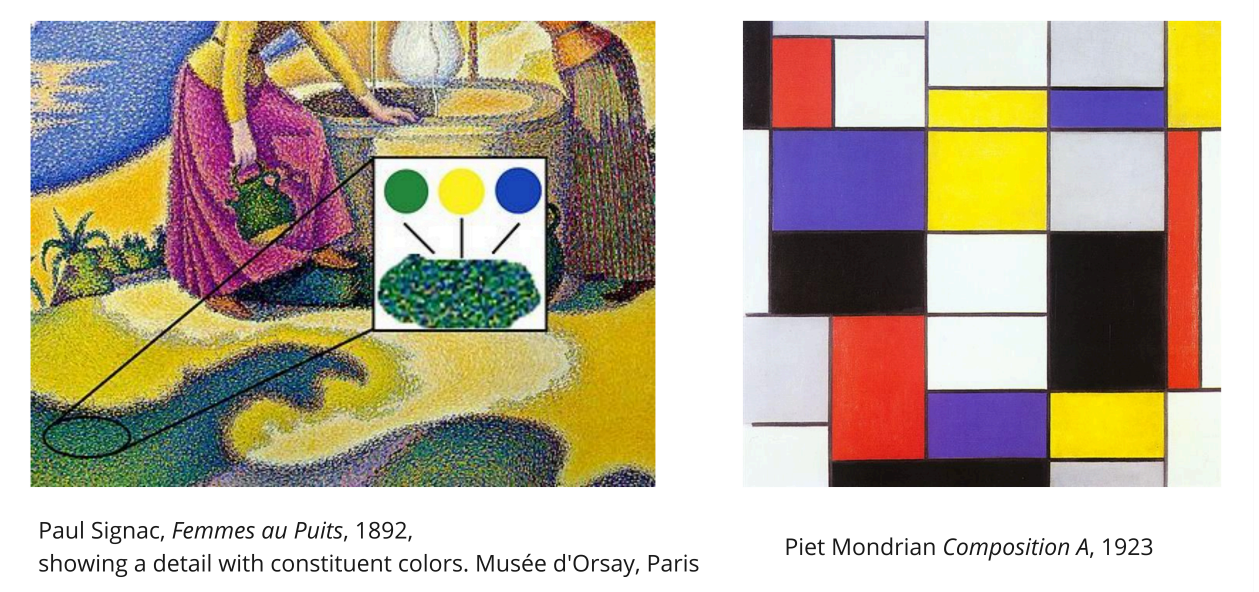with strategies to maintain object precision and content diversity, which can be applied to learn not only objects but also styles. However, this approach is not style-specific and sometimes yields content-leaking problems [9]. **Textual Inverstion-based approach**(Textual Inversion (TI) [6] variants) use off-the-shelf DMs, learning extra text embeddings to represent styles. Specifically, DreamStyler [1] leverages multi-stage TI and Context-Aware Text Prompt to enhance the representation capacity and alleviate the content leaking problem. However, our experiments still show that these approaches sometimes have limited performance when the semantic gap between the reference image and the target image is large. Lastly, **Cross-attention-based approach**(IP-Adapter variants [20] and InstantStyle [17]) manipulates cross-attention layers of DM and inject image features, enabling great generalization ability of DMs with image reference. However, these approaches are often better for image manipulation (e.g., augmentation, variation) and creative visual blending rather than generating new content with a huge semantic gap. Also, all the above methods require certain levels of training (upon large image datasets or style-specific case-by-case) aside from inference.

To solve the limitations on style learning of previous methods, as a **self-attention-based train-free approach**, Visual Style Prompting [9] focuses on self-attention of DMs, created a training-free method by swapping the self-attention of the target image inference guided by the inference of a reference image after inversion (illustrated in Fig. 3.4). Visual Style Prompting showed amazing performance on the previous-defined style learning problem,
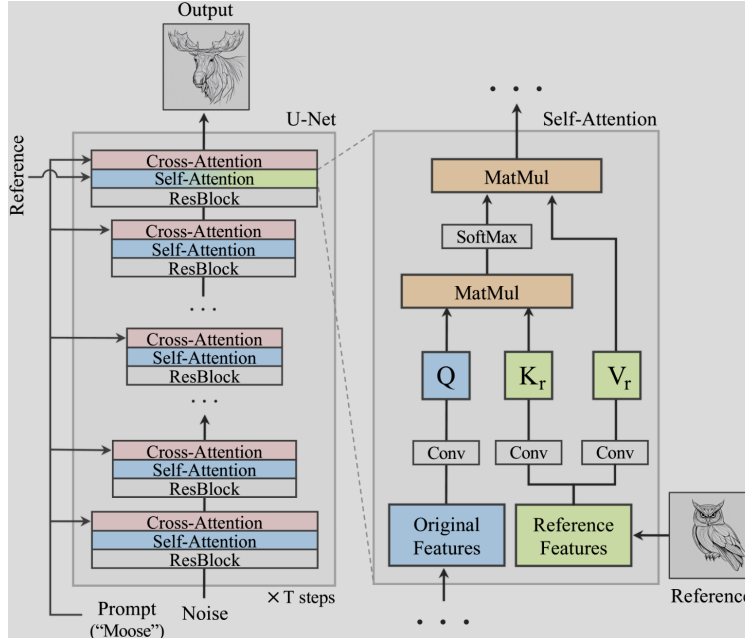
Figure 3.4: Overview of swapping self-attention in visual style prompting, image from original paper [9].

which we find especially useful in the case of **generative procedural art style learning**, where the style is always abstract.

3.1.4 Generative Procedural Art Style Learning

We target facilitating the process of generative procedural art style learning task, where the artist generates new images with the style of existing procedural art as examples. As discussed in the previous section, the key to the style learning task is to teach the model the essence of a style with limited reference images (e.g., the reference image might be far from what we want to model to paint or contain a complex combination of style units). This is especially true for generative procedural art, where the reference images from the rendered creative coding are often abstract, and different image variations for one specific art are often similar. Thus, to learn to paint with the style of procedural art, the reference images from the rendered canvas have limited variety and are often semantically far from what we want to paint. Artists and creative communities have widely explored procedural art style learning. One inspiring example is the *Parametric Swimming* [4] created by Takafumi Oyama [5], where the artist explored the connection between abstract art and real-life images by injecting the style of abstract art into real-world photos. However, since style learning methods require

---

[4]https://www.takafm.me/project/parametricswimming
[5]https://www.takafm.me/

27

reference images as conditions, current approaches require manually adjusting and choosing the reference image as input to style learning models like DM architectures. Motivated by this, we aim to explore how can we augment procedural art style learning using DM by integrating the whole process in a creative programming environment smoothly.

3.1.5 Preliminary Study

Shown in 3.5, we investigated the SOTA DM style learning methods of each approach. Specifically, we experimented with 1) simple, abstract procedural art as style reference images and 2) complex, targeted procedural art as style reference images. For the new image generated by DM, we experimented with both abstract and concrete prompts. Since different methods use different architectures and parameters, it is hard to do a fair comparison; We fix the seed, guidance scale, and inference timestep at the image inference stage, and keep other parameters (e.g., training parameters) as default. For methods that require extra tokens or prompts, we also provide them in detail [6].

Given our setting of style components and style learning task, it is apparent that only InstantStyle and Visual Style Prompting succeed at capturing the style components and generating an image corresponding to the prompt description. Since Visual Style Prompting is more accurate at representing all the units (row 1,2)and composition methods (row 3,4), we choose visual style prompting as our main DM method for style learning.

We also investigated how the parameters and image window of procedural art can affect the style learning result (Fig. 3.6). We took images with different parameters and window sizes over the same procedural art example, then used Visual Style Prompting to generate different images with the same seed and prompt. The results show a huge difference in final images, indicating that choosing image windows and adjusting parameters can be essential for style learning tasks in procedural art.

---

[6]DreamStyler: ref_prompt: a black background with a colorful line in the style of {}
inf_prompt: a cat in the style of {}, abstract angry circle scribbles in the style of {}
ref_prompt: a drawing of a man with long hair in the style of {}
inf_prompt: a car in the style of {},abstract art of mild flow in the style of {}
DreamBooth: instance_prompt: spr generative art
class prompt: generative art
inf_prompt: pcn generative art of a cat, pcn generative art of abstract angry circle scribbles
instance_prompt: pcn generative art
class_prompt: generative art
inf_prompt: pcn generative art of a car, pcn generative art of abstract art of mild flow
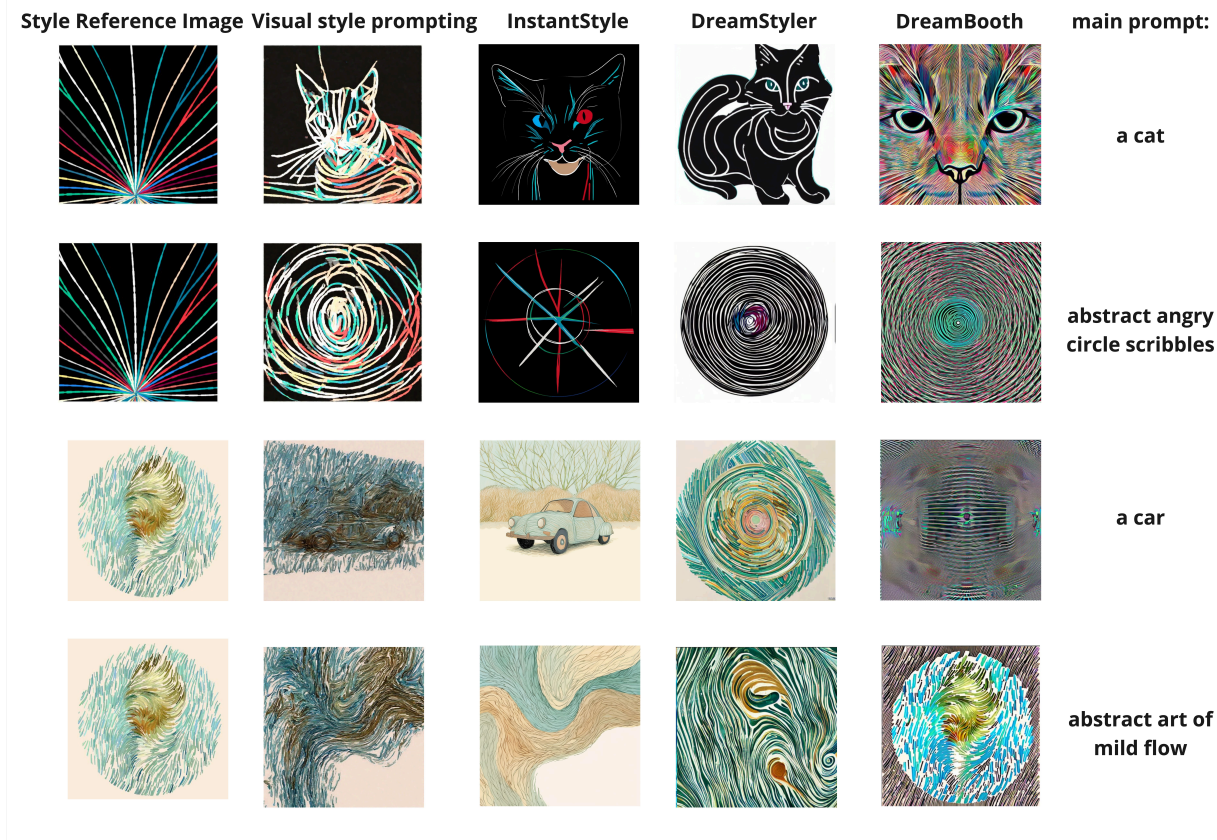
Figure 3.5: Comparison of different approaches. Reference image sources: Inner Demons by Che-Yu Wu (Row I-II), Painting with Circle Noise by Luciana Diaferio (Row III-IV)
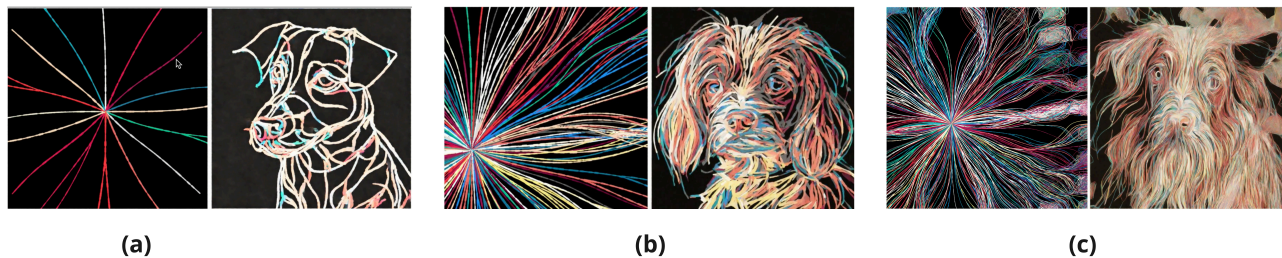

Figure 3.6: Comparison of style learning result of different parameters and windows. Seed: 1024, inference step: 50, guidance scale: 7, prompt: a dog. Reference image sources: Inner Demons by Che-Yu Wu

3.1.6   P5.gui

Since our preliminary study shows that the choice of variable numbers can have a huge influence on the result of the style learning tasks, the first step of building a system to assist generative procedural art style learning is to enable flexible variable adjustment. We found an existing p5.js library—p5.gui [7] which enables the creation of graphical user interface (sliders, color selector, etc) for variables, building upon other existing libraries. This enables real-time adjustments of any variables while the art content is being rendered. However, we observe that although the GUI components enable instant manipulation of variables, it is hard to manipulate the features for animation art (e.g., the animation process that starts after each run of the code), where the variables are not continuously accessed by the draw() function, but only used at the initiation stage. Moreover, there lack of ways to create style reference images for a style learning task, users lack a way to conveniently select frames at different stages evolving through time, and different window patches of the desired frame.

3.2   Method

Overview   As shown in Fig. 3.7, we generalize any p5.js project into 3 parts: function to initialize contents, function to draw contents, and other supporting functions (eliminated in the figure). Thus, we wrap up the function to initialize contents and the function to draw contents separately to be the callback functions of our P52Style library functions. Typically, initialize_gui() adds all the variables as GUI components, and refresh() restarts the whole animation. enable_gui() enables the listeners to restart the whole animation on change of any variables; draw_with_screenshotsquare() helps the drawing process of the p5 contents and enables the frame index selector and screenshot square logic (to be discussed in next paragraph).

UI Component and Usage   When users use P52Style library to create p5.js projects, there will also be UI components are generated automatically when using the library. As shown in Fig 3.8, the core functionalities and UI components or P52Style are:

1) Variable panel: same as original p5.gui, contains all the variables of the art. Any change of the variable will refresh the whole art animation.

2) Framestore counter: every time the art is refreshed, the counter will automatically store N frames and show the storing progress. N is specified by the user in function calls.

3) Frameindex slider: once the framestore counter shows the completion of storing frames,

---

[7]https://bitcraftlab.github.io/p5.gui/
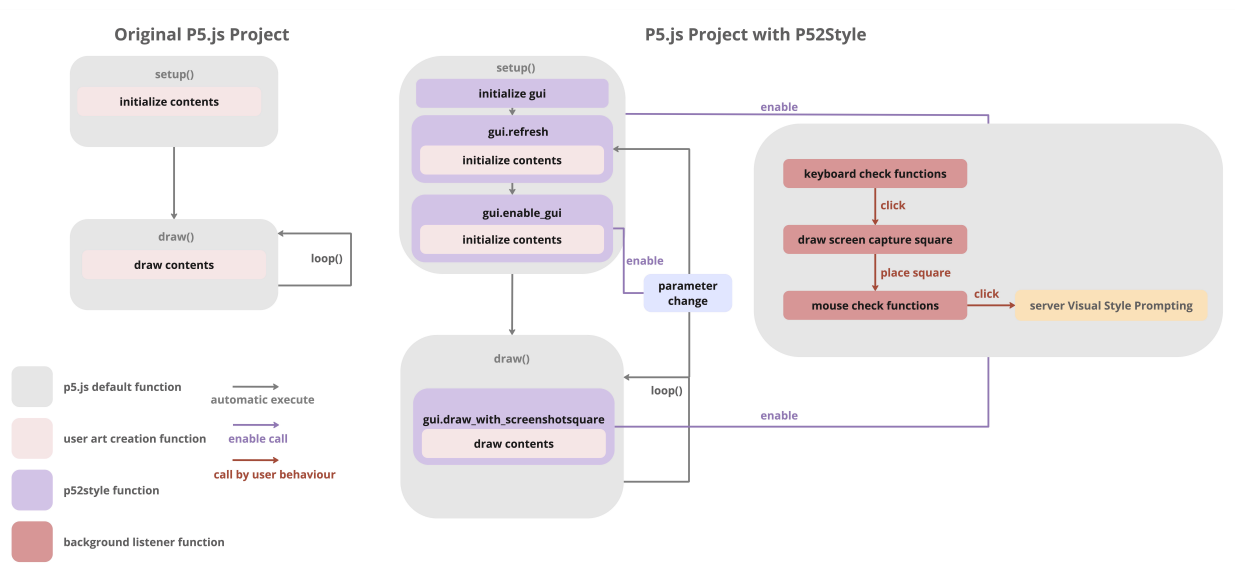
Figure 3.7: Overview of p52style structure.

the slider allows the user to adjust and rewind to any frame number (1-N), and conveniently select any stage of the whole animation.

4) New image prompt: it allows the user to specify the prompt to generate new images for style learning tasks.

5) Image capture square: when the user presses "Tab" of the keyboard, an image capture square will be evoked, allowing the user to select any image patch to use as a reference style image for style learning. Users can adjust the position and size of the square with a keyboard and mouse.

6) New image generator: After the user clicks on the ideal location of the capture square, the system will call Visual Style Prompting with the new image prompt and captured style reference image, showing a new image generation process below the canvas. After the image is generated, it will automatically shown below the canvas and downloaded locally.

Core functionalities Specifically, to leverage P52Style, the user will directly interact with the following core functionalities in Tab. 3.1.

Below is a code snippet illustrating in detail how the user would leverage P52Style:

```
let gui; // For the GUI object

// GUI-controlled variables declared globally
var curlSpan = 75;
var rFacMin = 0.994;
var rFacMax = 0.995;
```

31

Figure 3.8: Overview of p52style UI. Reference image sources: Inner Demons by Che-Yu Wu

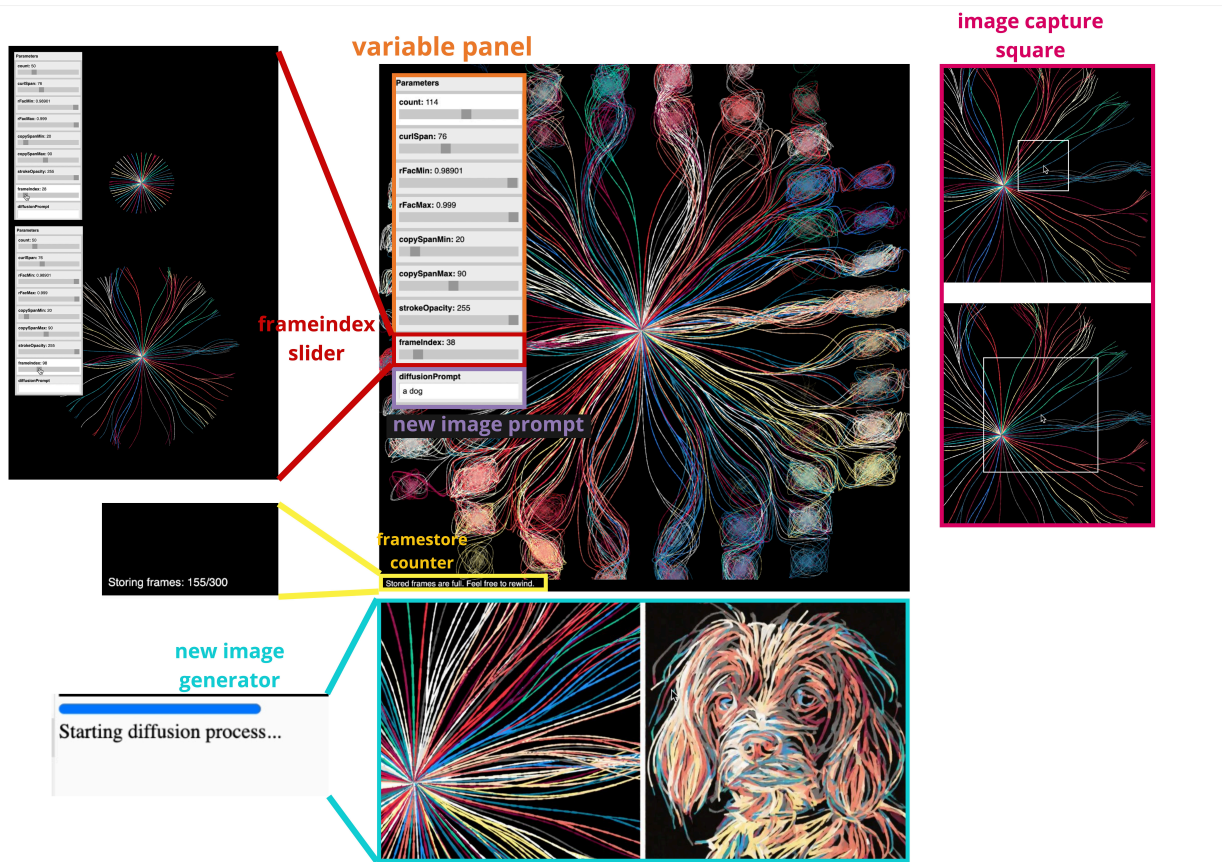| Function Name	Parameters	Return Value	Introduction
createGui	label(str), maxFrames, squareSize	gui instance	create gui class instance
addGlobalsWithRanges	argNames(array) maxValues(array)	none	add numerical variables to the gui panel in a patch. with their maximum adjustable values
refresh	callback(user function to initialize art)	none	refresh/restart the whole art animation process
enable_gui	callback(user function to initialize art)	none	enable the gui panel to keep notice of any change of variables, and refresh the whole art animation when anything changes
draw_with_screenshotsquare	callback(user function to draw art contents)	none	start the loop to draw contents, whole enabling functionalities for framestore counter, frameindex slider and image capture square

Table 3.1: P52Style main function descriptions and details.

```
7  var copySpanMin = 20;
8  var copySpanMax = 90;
9  var strokeOpacity = 255;
10 var count = 50;
11 // GUI-controlled variables declared globally
12 let particles = [];
13 let cnv;
14
15 //url for Visual Style Prompting server
16 url = 'https://bbce-34-143-236-168.ngrok-free.app'
17
18 function setup() {
19     pixelDensity(3);
20     cnv = createCanvas(800,800);
21     background(100);
22
23     // Initialize GUI
24     gui = createGui('Parameters', maxFrames = 300, squareSize = 200);
25
26     // Define arrays for argument names and their corresponding maximum
       slider values
27     var argNames = ['count', 'curlSpan', 'rFacMin', 'rFacMax', '
```

```
       copySpanMin', 'copySpanMax', 'strokeOpacity'];
28      var maxValues = [200, 200, 0.999, 0.999, 200, 200, 255];
29      // Use the new function to add globals with slider rangessss
30      gui.addGlobalsWithRanges(argNames, maxValues);
31
32      gui.refresh(initialize_canvas_particles);
33
34      gui.enable_gui(['count', 'curlSpan', 'rFacMin', 'rFacMax', '
        copySpanMin', 'copySpanMax', 'strokeOpacity'],
        initialize_canvas_particles)
35 }
36
37 function draw() {
38    frameRate(200)
39    gui.draw_with_screenshotsquare(draw_particles)
40
41 }
42
43 //user function to draw contents
44 function draw_particles(){
45      particles = particles.filter(p => p.alive);
46      particles.forEach(obj => {
47          obj.update();
48          obj.draw();
49      });
50 }
51
52 //user function to initialize contents
53 function initialize_canvas_particles(){
54      fill(0);
55      rect(0,0,width,height);
56      particles = [];
57      frames = [];
58      for(let i=0; i<count; i++){
59          particles.push(new Particle({
60              p: createVector(width/2,height/2).add(createVector(-5,5)),
61              v: createVector(0,3.5).rotate(i/count*PI*2)
62          }));
63      }
64 }
65
66 class Particle {
67 //supporting class....
```

```
68  }
69
70  %  }
```

Listing 3.1: example

Implementation Details   P52Style is implemented by adding functionalities in the exist-
ing p5.gui.js library code. For the usage of Visual Style Prompting, we provide Colab code
to set up a remote server using A100 GPU, connecting with p5.js frontend with flask ngrok.
The workspace structure:

- (remote server)

  Colab notebook code

- (p5.js project folder)

  p5.gui.js,

  index.html,

  sketch.js

3.3   Evaluation

Since there are no similar existing methods comparable to P52Style, we conduct the eval-
uations using Cognitive Dimensions of Notations (CDs) [4]. We score each dimension as
high, low, mid, and standard (which means the same level as any other p5.js or program-
ming libraries, the level is acceptable as long as the user has proper knowledge of p5.js
programming).

Low *Viscosity: Resistance to Change.* The library is not viscous, and the changes in
content code don't require changes in P52Style function calls.

High *Visibility: Ability to View Components Easily.* The library has high visibility, with
all the UI components explicitly and adjustable to users.

Standard *Premature Commitment: Constraints on the Order of Doing Things.* Like any
other p5.js library, initiating instances and including package code is necessary.

Mid *Hidden Dependencies: The way the user uses one library function will not affect
other functions implicitly.* Certain user code behaviors concerning the canvas display might
damage the performance of the library, like blend() call.

High *Role-Expressiveness: The Purpose of an Entity Is Readily Inferred.* The library follows the cognitive habit of users as it has the same structure as any other p5.js library. The purpose of different functions is well elaborated with function names and documentation.

Mid *Error-Proneness: The Notation Invites Mistakes and the System Gives Little Protection.* The library functions have simple and explicit usage, but there might be user mistakes with no clear system feedback.

Standard *Abstraction: Types and Availability of Abstraction Mechanisms.* The library functions have the same abstraction level as other p5.js libraries.

Low *Secondary Notation: Extra Information in Means Other Than Formal Syntax.* The library functions don't require users to record extra information like comments.

High *Closeness of Mapping: Closeness of Representation to Domain.* The purpose of different functions is well elaborated with function names and documentation, whereas the closeness of functions and behavior is standard in programming libraries.

High *Consistency: Similar Semantics Are Expressed in Similar Syntactic Forms.* The library has high consistency, as a feature of programming languages.

Low *Diffuseness: Verbosity of Language.* The function calls have low diffuseness, as a feature of programming languages. For UI components, the label texts are reduced to the minimum.

Standard *Hard Mental Operations*: High Demand on Cognitive Resources. The function calls have a standard level of hardness as programming languages. For UI components, the icons and texts are easy to understand without cognitive burden.

Standard *Provisionality: Degree of Commitment to Actions or Marks.* The library has a standard level of provisionality as any other programming library.

High *Progressive Evaluation: Work-to-Date Can Be Checked at Any Time.* The library has a high progressive evaluation as any other programming library.

In conclusion, the analysis indicates that P52Style has fair usability in most of the dimensions, meanwhile requires improvements in Error-Proneness and Hidden Dependencies with more error messages and guidance. To tackle the limitations, we plan to reduce Hidden Dependencies and increase Error Proneness by updating more restrictions in documentation to indicate certain functionalities that might bring errors and include more error messages into the P52Style functionalities to help users tackle errors.

3.4 Conclusion

In conclusion, we conducted preliminary research to propose our definition for the components of style and our problem scope of style learning tasks and studied various trending

DM style learning methods. To explore how to better assist the process of generative procedural art style learning within the creative coding environment, we proposed P52Style, an extended library built upon p5.gui allowing flexible adjustment of art content and leverage of DM for style learning tasks. We also provided detailed documentation and library usage examples, and evaluations based on CDs to show the usability of the library.

3.5   Limitations and Future Work

Due to the time limit, there are certain limitations in P52Style that we leave to future work.

Firstly, we hope to conduct qualitative analysis and user evaluations through real user studies to test the usability of the tool.

Secondly, we would like to extend the library to support flexible manipulation of a large number of variable types including color selection and random variables.

Thirdly, the current approach needs the user to manually pick what variables to add to gui and test their functions in trials. In future work, we would like to integrate methods for facilitating users in understanding the functionalities of variables and their impact on the result of newly generated images in style learning automatically through a user-in-loop workflow.

Lastly, we would like to further reduce latency by storing frames implicitly when there's no need for rendering.

CHAPTER 4

What I Learned

During this thesis, I enhanced my research abilities under the advisory of Prof. Eytan Adar. Specifically, spending the whole summer exploring the interaction of generative procedural art and generative AI and conducting numerous experiments using different tools helps me learn more about how to find a valuable research question. It also tells me that spending time trying out things that might end up being abandoned in the final paper is not a waste of time but sometimes is the necessary way leading to the fulfillment of a project.

Generative procedural art is a really interesting and exciting area bridging computational methods and art creation. I am happy that I gained knowledge and experience through this thesis that allowed me to be part of the creative community and continue to contribute in my unique way. Through this project, I also further deepened my understanding of how DMs work by reading a large number of research papers on various DM approaches focusing on style learning, and emergent APIs and applications enabling endless creativity for everyone. Being my first web application project, building p5.js libraries also enriched my knowledge of JavaScript web application test and development which would be useful in the future.

With the experience of this thesis, I look forward to keeping doing research and making contributions in the domain of Human-Computer Interaction, AI, and Visual Computing.

BIBLIOGRAPHY

[1] Namhyuk Ahn, Junsoo Lee, Chunggi Lee, Kunhee Kim, Daesik Kim, Seung-Hun Nam, and Kibeom Hong. Dreamstyler: Paint by style inversion with text-to-image diffusion models. *arXiv preprint arXiv:2309.06933*, 2023.

[2] Tyler Angert, Miroslav Ivan Suzara, Jenny Han, Christopher Lawrence Pondoc, and Hariharan Subramonyam. Spellburst: A node-based interface for exploratory creative coding with natural language prompts. *UIST*, 2023.

[3] Perry Barile, Vic Ciesielski, Marsha Berry, and Karen Trist. Animated drawings rendered by genetic programming. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 939–946, 2009.

[4] Alan F Blackwell, Carol Britton, Anna Cox, Thomas RG Green, Corin Gurr, Gada Kadoda, Maria S Kutar, Martin Loomes, Chrystopher L Nehaniv, Marian Petre, et al. Cognitive dimensions of notations: Design tools for cognitive technology. In *Cognitive Technology: Instruments of Mind: 4th International Conference, CT 2001 Coventry, UK, August 6–9, 2001 Proceedings*, pages 325–341. Springer, 2001.

[5] Alan Dorin, Jonathan McCabe, Jon McCormack, Gordon Monro, and Mitchell Whitelaw. A framework for understanding generative art. *Digital Creativity*, 23(3-4):239–259, 2012.

[6] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H. Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing text-to-image generation using textual inversion, 2022.

[7] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. 2020.

[8] Kibeom Hong, Seogkyu Jeon, Junsoo Lee, Namhyuk Ahn, Kunhee Kim, Pilhyeon Lee, Daesik Kim, Youngjung Uh, and Hyeran Byun. Aespa-net: Aesthetic pattern-aware style transfer networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22758–22767, 2023.

[9] Jaeseok Jeong, Junho Kim, Yunjey Choi, Gayoung Lee, and Youngjung Uh. Visual style prompting with swapping self-attention, 2024.

[10] Bingyan Liu, Chengyu Wang, Tingfeng Cao, Kui Jia, and Jun Huang. Towards understanding cross and self-attention in stable diffusion for text-guided image editing. *arXiv preprint arXiv:2403.03431*, 2024.

[11] Vivian Liu, Tao Long, Nathan Raw, and Lydia Chilton. Generative disco: Text-to-video generation for music visualization. *arXiv preprint arXiv:2304.08551*, 2023.

[12] Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference. *arXiv preprint arXiv:2310.04378*, 2023.

[13] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. 2022.

[14] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. 2022.

[15] Blair Subbaraman and Nadya Peek. P5. fab: Direct control of digital fabrication machines from a creative coding environment. In *Proceedings of the 2022 ACM Designing Interactive Systems Conference*, pages 1148–1161, 2022.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[17] Haofan Wang, Qixun Wang, Xu Bai, Zekui Qin, and Anthony Chen. Instantstyle: Free lunch towards style-preserving in text-to-image generation. *arXiv preprint arXiv:2404.02733*, 2024.

[18] Iqbal P Wiguna, Adrian P Zen, and Cucu R Yuningsih. Painting with algorithms: The potential for using the p5. js programming language for new media artist. In *Embracing the Future: Creative Industries for Environment and Advanced Society 5.0 in a Post-Pandemic Era*, pages 271–275. Routledge, 2022.

[19] Tao Wu. Saliency-aware generative art. In *Proceedings of the 2018 10th International Conference on Machine Learning and Computing*, pages 198–202, 2018.

[20] Hu Ye, Jun Zhang, Sibo Liu, Xiao Han, and Wei Yang. Ip-adapter: Text compatible image prompt adapter for text-to-image diffusion models. 2023.