Painting with CaTS: Camera-aided Texture Synthesis

Ticha Sethapakdi

CMU-CS-18-117
May 2018

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
James McCann, Chair
Nancy Pollard
Jessica Hodgins

Submitted in partial fulfillment of the requirements for the Degree of Master of Science

Keywords: Texture Synthesis; Creativity Support; Human-Computer Interaction; Interfaces; Digital Painting

Abstract

Digital painting is a popular approach to creating visual art. Much like in traditional media, digital painting programs equip artists with a multitude of tools and brushes so they can produce diverse paintings. It is often the case that artists want to make their own assets and textures to produce specific results. Unfortunately, creating custom assets can be cumbersome and time-consuming as artists often need to go through the process of scanning, importing, and adjusting before they can use their own textures for painting.

We present CaTS, a painting system that synthesizes textures from live video in real-time. The system interfaces with a simple capture rig which facilitates bimanual manipulation, allowing users to manipulate the exemplar object with one hand while painting with the other. Through the close integration of the capture device and painting method, CaTS builds upon the concept of texture synthesis-based painting and augments it with more artistic freedom. This allows digital artists to easily create custom brushes which they can use to produce texture-rich paintings. Furthermore, CaTS fosters an exploratory approach to painting that is not easily achievable through purely digital or traditional means. We demonstrate the expressiveness of our system through paintings produced by artists with varying artistic styles and evaluate its effectiveness through user feedback.

Acknowledgments

Huge thanks to my advisor Jim McCann for working on this exciting project with me and being an all-around cool advisor – I am glad I can go to you for technical advice, life advice, and DDR advice alike (sorry for being bad at DDR).

I would like to thank Jessica Hodgins and Nancy Pollard for taking the time to serve on my committee and give me feedback. None of this would have been possible without your support!

I would also like to give a shout-out to Dave Eckhardt for being a great academic/life advisor and Tracy Farbacher for keeping everything in order for us Masters students. You guys are amazing and make me feel at home in this awesome community.

Last but not least, I'd like to thank all my friends and family for supporting me through thick and thin – I don't know what I did to deserve all of you, but I am glad to have you in my life.

To my family, friends, mentors, and collegues, for all their love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

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

Figure 1-1: A comparison of a painting made with (b) and without (a) the use of texture. Closeups of the textureless and textured results are shown in (c) and (d) respectively.

A popular approach to creating visual art is digital painting. Digital painting interfaces have arguably evolved significantly since their inception in the 1970s from programs like MacPaint [2] and Kid Pix [11]. Much like in traditional media, digital painting programs now equip artists with a diverse set of tools and brushes so they can produce expressive paintings. The most used tool in a digital painting program is, of course, the paintbrush – which is designed to be highly customizable to allow artists to create a variety of effects. Textured brushes in particular can be used to give digital paintings more life and expressiveness (Fig. 1-1), and there exists multiple ways to customize a brush's texture. One common method is to change the brush shape, which affects the appearance of stroke boundaries (Fig.1-2). Another way is to use "pattern stamping", which uses a pattern as the brush's color (Fig. 1-3).

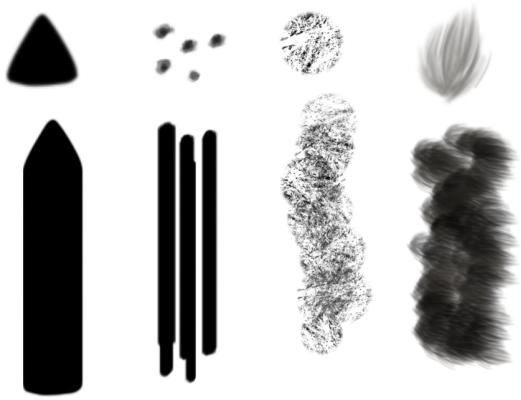


Figure 1-2: Some examples of brush shapes and the strokes produced by them.

1.1 Motivations

While there exists a wealth of built-in brush presets and online assets, artists often want to make their own brushes to produce specific results. With traditional media, artists have the freedom to easily incorporate almost any material they can find into their work. But with digital media, artists must undergo the multi-step process of scanning, cropping, and importing before they can use their own textures to paint. Applications like Adobe Capture [1] attempt to simplify this workflow by allowing users to use their phone camera to quickly make assets.

Additionally existing painting systems use a "stamping" approach to painting, which stamps the image of the brush along the path the user painted. Stamping has the

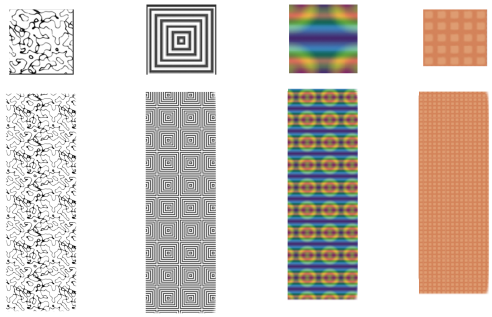


Figure 1-3: Some examples of pattern stamps and the strokes produced by them.

advantage of producing easily controllable and predictable results. At the same time, unpredictability has the potential to produce desirable outcomes users might not have thought of themselves. This characteristic is evident in traditional painting, where control is given in exchange for the possibility of "happy accidents".

At the same time, the amount of control artists have over the progression of their paintings gives digital painting a significant advantage. For example, if a traditional painter were to commit a stroke to the canvas it cannot easily be undone. In digital painting, strokes can easily be modified or undone without destructively affecting the other parts of the painting. Furthermore the multilayer system in digital media allows artists to develop their work in a nonlinear fashion.

1.2 Contributions

With CaTS, we aim to create a texture-based painting system that combines the strengths of digital and traditional approaches. Our contributions are as follows:

1. We integrate *live capture* into our painting system to cut down the texture-creation workflow.

2. We *synthesize texture from live video in real-time*, allowing artists to use their custom brushes immediately after they are created. The texture synthesis method makes use of randomization to encourage "happy accidents". Strokes also have the property of content-awareness, meaning that new strokes have an awareness of the surrounding textures. This ensures cohesiveness between new and existing strokes.

3. We design the system to support *bimanual manipulation*, enabling users to manipulate the texture exemplar while painting.

4. We incorporate a *paint-drying* mechanism in which strokes are 'live' for a short period of time, giving users a small window to make adjustments to the stroke texture after it is painted.

Chapter 4 describes how these solutions are achieved in more detail.

Chapter 2

Related Work

Our work spans research in painting interfaces, texture synthesis, and capture. This chapter discusses related work in these fields.

2.1 Painting with Capture

There are various examples of interface research that try to incorporate captured objects in digital painting. Adobe Capture [1] is a mobile app that allows users to convert pictures into various assets such as palettes, patterns, and brush shapes, which they can later use in their creative applications on their desktop. This greatly simplifies the asset creation workflow because it enables artists to produce custom assets on-the-go. With CaTS we wish to simplify the workflow even further by allowing artists to immediately paint with the textures they select.

I/O Brush is a tangible user interface that allows users to use objects as ink using a paintbrush-like device [18]. FingerDraw is a mobile application that allows users to paint using colors and textures picked from a finger-worn device [10]. UnicrePaint is another tangible user interface that enables users to physically use objects as a paintbrush [12]. These applications demonstrate the power of using real world objects to make the painting experience more engaging. We wish to expand on these ideas by incorporating texture synthesis to allow users to produce richer paintings.

2.2 Painting with Texture

Texture synthesis has long been explored in graphics research. There is a variety of work that applies texture synthesis to simulate natural media in digital painting. Simulation-based approaches such as those used by MoXi [8] and Lei et al. [15] use physics simulations to model the interactions between the paint media and paint surface, which can produce realistic and convincing results. Data-driven approaches like IMPaSTo [6], the Markov Pen [13], and RealBrush [16] use samples of textures to synthesize natural media – with the latter being a pure data-driven approach that does not rely on a physical model or procedural rules.

2.3 Efficient Texture Synthesis

To provide feedback to users as quickly as possible, it is important to optimize the performance of synthesis tasks. Computing correspondences between two images is one approach to texture synthesis. PatchMatch [4] is a patch-based, randomized image correspondence algorithm that works at interactive speeds. A generalized version of PatchMatch [5] was later developed to support finding k nearest neighbors across multiple scales and rotations. As PatchMatch is able to quickly produce high quality results it has been the basis of various computer vision and image processing applications, including Content-Aware Fill in Adobe Photoshop CS5 [3].

Lefebvre et al. [14] presents an efficient parallel texture synthesis solution that allows for intuitive user control over texture variability. To achieve this they use the notion of *coordinate jitter*, which perturbs exemplar coordinates at each level of synthesis to attain variation. Our texture synthesis solution aims to produce high-quality results efficiently by leveraging the framework of Lefebvre et al. approach, with refinements made in the style of PatchMatch.

Chapter 3

User Interface

In this chapter we review the CaTS interface and setup. Section 3.1 shows the available functions in the program. Section 3.2 presents hardware setup for CaTS and the different capture modalities supported by the system.

3.1 User Controls

The user has control over the following parameters:

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

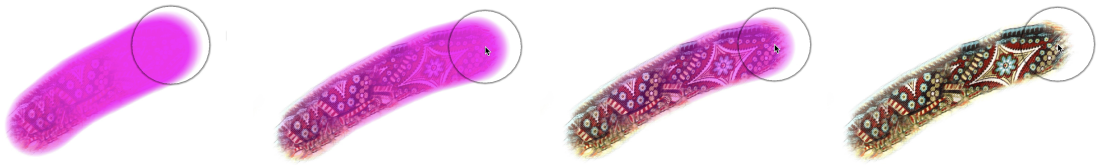
Figure 3-1: Demonstration of the effect of brush size on texture. The exemplar is shown in (a). (b), (c), and (d) show strokes painted using a small, medium, and large brush size.

Brush size Providing variability for brush size is an important feature in painting applications. The size of the paint strokes can be adjusted using either the buttons on the tablet or through pen pressure.

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

Figure 3-2: Demonstration of cropping. (a) and (b) show the exemplar and stroke before cropping; (c) and (d) show the exemplar and stroke after cropping to the desired region of interest.

Region of interest There may be situations where the user wants to paint with a specific region of an exemplar. One way to do this would be to manually zoom the camera in to that region, but sometimes the user wants to preserve the form factor. Thus we give the user the ability to crop the video to a desired region of interest.



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

Figure 3-3: Demonstration of the progression of paint drying. The bright pink color is an indicator of the wetness of the stroke. The stroke starts off as completely wet (a), and gradually becomes completely dry (d).

Stroke wetness Strokes have a 'wetness' parameter that determines how quickly the stroke is committed to the canvas. As a stroke is 'drying' it continues to re-synthesize the texture from the video, allowing the user to make changes to the texture using the exemplar. Once a stroke is completely dry, it is committed to canvas. Thus a brush with a wetness value of 0 is immediately committed to the canvas.

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

Figure 3-4: Demonstration of jitter: the exemplar (a) and the synthesized strokes made with jitter value 0 (b), jitter value 1.5 (c), and jitter value 3 (d).

Stroke jitter Strokes also have a jitter parameter that affects the variability of the texture. Jitter can also be thought of as the controllability of the texture synthesis, with higher levels producing more irregular results.

Erase/Undo These essential painting functions are particularly advantageous for our system due to the randomized nature of our texture synthesis method. They give users the ability to correct themselves when the texture synthesis does not produce desired results.

Loading/Saving Users can save their paintings as an image file or load an image file to use as a painting base (Fig. 6-2).

3.2 Hardware Setup

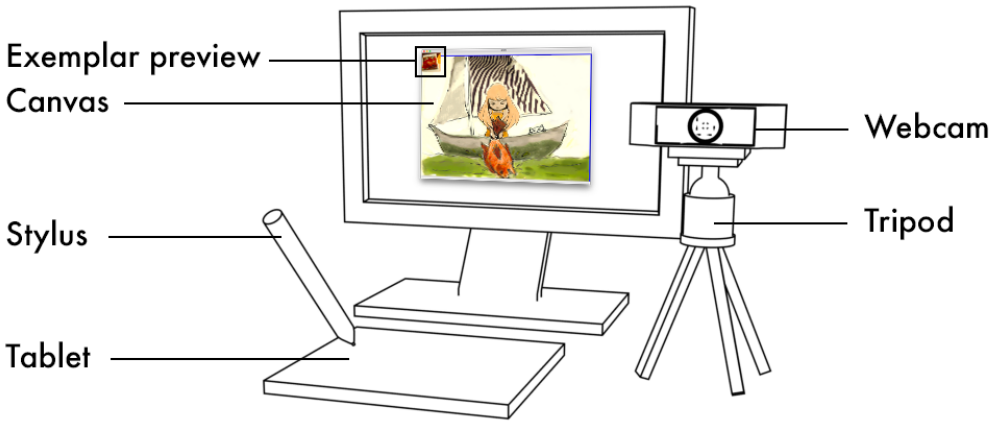
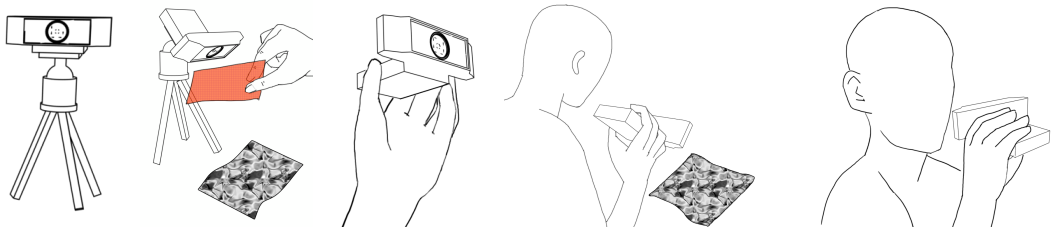


Figure 3-5: The main components for CaTS



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

Figure 3-6: The mounting (a) and handheld (c) modalities with some example use cases: applying a filter (b), pointing at a texture from far away (d), and capturing the user's face (e)

The capture rig is a USB-connected Logitech webcam mounted on a mini tripod. The mini tripod has flexible legs which can be securely wrapped around other objects. The camera feed is displayed on the upper left corner of the interface so users can see what is being captured as they paint. Painting is facilitated by a Wacom drawing tablet and stylus. Since the capture rig operates independently from the tablet input users can paint while manipulating (e.g. moving, rotating) the capture exemplar.

CaTS supports 2 different capture modalities:

1. **Mounting modality**

   In the mounting modality the camera is fixed on a tripod (Fig. 3-6a). The tripod legs can be bent to latch the camera onto an object while the mounting bracket can be rotated to reorient the camera. The user can then either place the exemplar in front of the camera or hold the exemplar in their hands. This modality is appropriate for capturing smaller exemplars or painting techniques that do not involve a lot of camera movements. This modality also makes it easy for the user to put filters over the camera (Fig. 3-6b).

2. **Handheld modality**

   In the handheld modality the camera is unmounted from the tripod, allowing the user to hold it (Fig. 3-6c). This modality is appropriate for capturing larger exemplars (Fig. 3-6d) or painting techniques that involve a lot of camera movements. Users may also use this modality to capture their faces (Fig. 3-6e).

Chapter 4

Method

When the user paints a stroke they are creating a *hole* in the canvas. A hole is any portion of the canvas that has less than 100% opacity. We call the process of filling the hole with synthesized texture from the exemplar *hole filling*. At a high level, the objective of hole filling is to make the inside of the hole look like the exemplar (webcam feed) and the outside of the hole look like the canvas. The hole is always eventually filled to 100% opacity, but the speed at which the hole is filled is affected by the *wetness* of the paint. Holes created using wetter paint are filled more slowly, which is similar to how wet paint takes a longer time to dry in natural media. We call this process *paint-drying*. So long as there is paint that is drying on the canvas, the program continues to resynthesize the texture from the webcam feed. This mechanism is what enables users to make adjustments to the stroke texture after strokes are painted.

Section 4.1 describes the details of the hole-filling[1] procedure for CaTS. Section 4.2 then discusses the mechanism for paint-drying.

[1] Hole filling is synonymous to texture synthesis

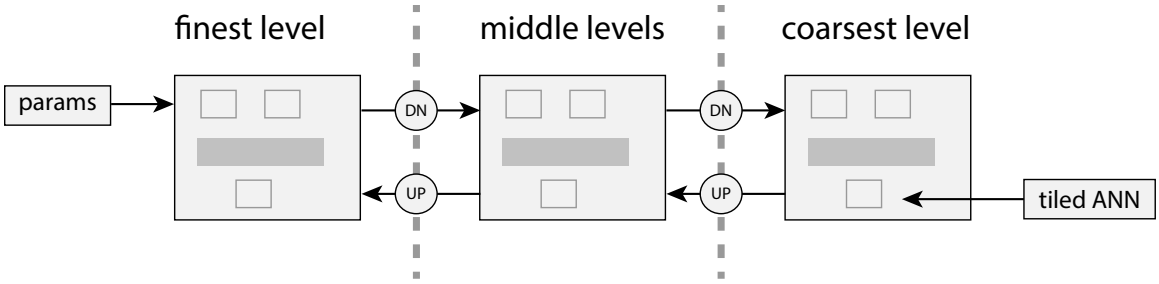
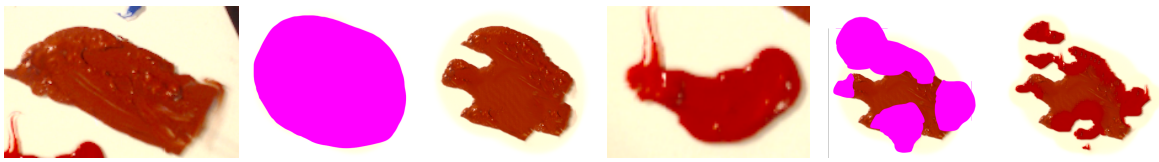


Figure 4-1: Overview of the hole filling procedure. The parameters to the hole filing function are the inputs to the finest level. The ANN of the coarsest level is initialized with some tiling offsets. Downsampling is used to provide canvas and exemplar inputs to the coarse levels from the fine levels, while upsampling is used to initialize the fine levels' ANN from the coarse levels'. The coarsest level's ANN gets some tiling coordinates to start with. Figure 4-3 shows a zoomed-in view of the process flow for an individual level.

4.1 Hole Filling

Hole filling is performed in real time using a combination of parallel controllable texture synthesis [14] and PatchMatch [4] approaches. Our method also adapts the hole-filling procedure described in [19] and [9]. Because PatchMatch is used in the refinement phase each painted stroke has awareness of the area around it. This allows for more visual coherence between new and existing strokes, which is not achievable through stamping-based approaches (Fig. 4-2).

A *patch* is defined to be a small square region of pixels of a fixed size. For a texture to 'look like' another texture, it must be the case that any patch inside one texture can be found in the other. Thus to synthesize a target texture that looks like the exemplar inside the hole, it is necessary to build a data structure that keeps track of how exemplar patches are



(a) (b) (c) (d) (e) (f)

Figure 4-2: A demonstration of the texture synthesis and matching behavior. An exemplar (a) is chosen to fill in a target region (b) and the resulting stroke (c) is produced. Stroke boundary effects arise naturally because the edges of the exemplar match with the white canvas. A second exemplar (d) is chosen to fill in target regions (e) of the texture in (c), producing a new texture (f). The melded effect is a result of the underlying image correspondence algorithm used for texture synthesis.

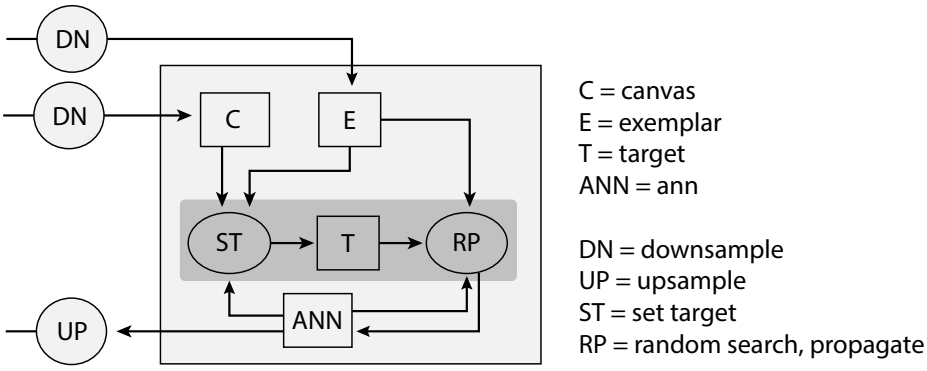


Figure 4-3: Illustration of the hole filling procedure for a given level. Objects are represented by rectangles while functions are represented by circles. Arrows show the direction of data flow.

arranged in the target. This data structure is called the *approximate nearest neighbor field* (ANN). An *approximate nearest neighbor* is the closest matching patch in the exemplar for a given patch in the target. An approximate nearest neighbor field is simply a structure that stores pointers to approximate nearest neighbors. The hole filling process synthesizes the target texture through building and refining the approximate nearest neighbor field.

Algorithm 1 and Figure 4-1 present an overview of the hole filling process. We split hole filling into 2 phases: initialization followed by improvement. Sections 4.1.1 and 4.1.2 describe the initialization and improvement phases, respectively.

4.1.1 Initialization

The initialization phase prepares the data for the improvement phase. The canvas and exemplar are first downsampled using a box filter and passed as inputs to a smaller instance of the hole-filling method, if it exists. Once the solution to the smaller instance is obtained, the method initializes the current instance's ANN to the upsampled ANN of the smaller instance. Jitter is applied at each upsampling step to produce more variability in the target texture. The ANN of the smallest instance is initialized with random tiling offsets to have some variety to start with.

Algorithm 1: Fill Hole

Input : The canvas, exemplar, jitter amount, and the current depth
Output: Fills the hole in the canvas texture using the exemplar texture and returns the ANN

```
/* initialization phase                                                    */
patchsize = 5 · 5;
if canvas.size, exemplar.size > 4 · patchsize then
    coarseCanvas = downsample(canvas);
    coarseExemplar = downsample(exemplar);
    coarseANN = fillHole(coarseCanvas, coarseExemplar, jitter, depth + 1);
    ANN = upsample(coarseANN, jitter);
end
else
    ANN = setTiledAnn();
end
bestDists = initDists();
/* improvement phase                                                       */
iters ← depth < 2 ? 1 : 5;
for k ∈ [0, iters] do
    targetTex ← setTarget(canvas.texture, exemplar.texture, ANN);
    ANN′ ← randomSearch(targetTex, exemplar.texture, ANN, bestDists);
    ANN″ ← propagate(targetTex, exemplar.texture, ANN′, bestDists);
end
return ANN″;
```

Algorithm 2: Upsample

Input : The coarse ANN to upsample, jitter amount
Output: Creates fine ANN from coarse ANN, plus some jitter

```
for (x, y) ∈ fine do
    src ← min(0.5 · (x, y), coarse.size) ;    /* location in coarse level */
    ann ← 2 · coarseANN[src] ;   /* get nearest neighbor and upsample */
    ann ← ann + (x, y) − 2 · src;
    ann ← ann + jitter;
    ann ← max(0, min(ann.size − 1, ann));
    fineANN[x, y] ← ann;
end
return fineANN;
```

Algorithm 3: Downsample

Input : The image to downsample
Output: Scales down the input image by half and applies a box filter

for $(x, y) \in \text{img}'$ **do**
  $\text{src} \leftarrow \left\lfloor \frac{(x,y)}{2} \right\rfloor$; `/* location in fine level */`
  $\text{img}'[x, y] \leftarrow \frac{1}{4} \cdot (\text{img}[\text{src}] + \text{img}[\text{src} + (1, 0)] + \text{img}[\text{src} + (0, 1)] + \text{img}[\text{src} + (1, 1)])$;
end
return img′;

4.1.2 Improvement

In the improvement phase our method tries to improve the ANN by gradually synthesizing the texture from the coarsest to the finest level. For levels in coarse to fine order *set target*, *random search ANN*, and *propagate ANN* are performed for 5 iterations. To save computation time the number of iterations are reduced at the finer levels.

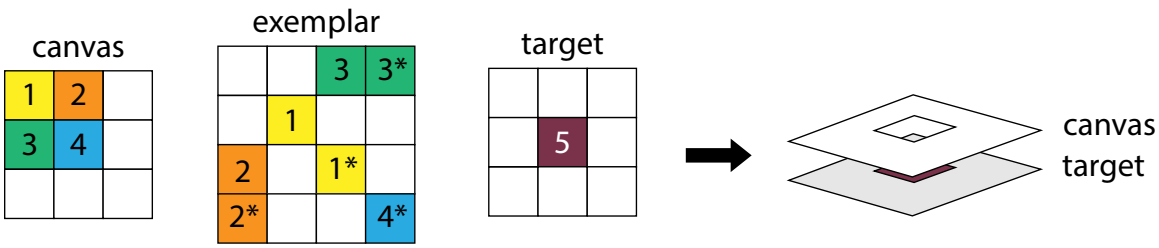


Figure 4-4: Illustration of set target, assuming a patch size of 2×2. The current pixel is marked by square 4 in the canvas. Squares 1, 2, 3, and 4 mark the locations of patches overlapping 4. The nearest neighbors for the overlaps are labeled accordingly in the exemplar, and the squares contributing to the target color are marked with *. The color of target square 5 is computed as the alpha blend between the canvas color and the averages: $\frac{1}{4}(\boxed{1^*} + \boxed{2^*} + \boxed{3^*} + \boxed{4^*}) \cdot (1 - \boxed{4}_A) + \boxed{4}_{RGB} \cdot \boxed{4}_A$.

Set Target. The level's target texture is set by blending exemplar patches from the current ANN field (Fig. 4-4). For each pixel p in the canvas, the set of overlapping patches O is obtained. The ANN is used to retrieve the set of nearest neighbors N for all patches in O. The RGB average is computed for the pixels in N overlapping t. This value is then alpha blended with p to produce the final result.

Algorithm 4: Set Target

Input : The canvas texture, exemplar texture, and ANN
Output: Set level's target texture by blending exemplar patches from current ANN field

```
1  for (x, y) ∈ canvas do
2      sum ← 0;
       /* offsets for all patches overlapping (x,y)                    */
3      offsets ← getOverlapOffsets(x, y);
4      for (x′, y′) ∈ offsets do
           /* nearest neighbor for patch overlapping (x,y)             */
5          overlapann ← ANN[x − x′, y − y′];
6          sum ← sum + exemplar[overlapann + (x′, y′)].rgb;
7      end
8      overlapAvg ← sum/size(offsets(x, y));
9      target[x, y] ← overlapAvg · (1 − (x, y).a) + (x, y).rgb · (x, y).a;
10 end
11 return target
```

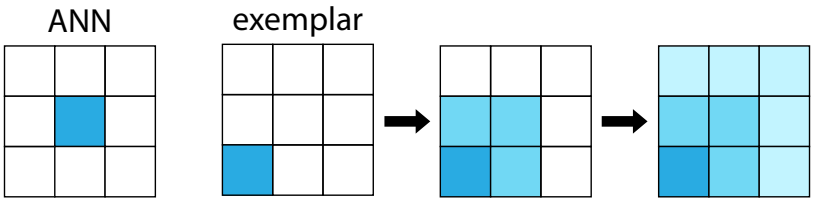


Figure 4-5: Illustration of random search. For each pixel in the target, the random search step randomly samples from successively larger boxes around the current nearest neighbor. In the diagram we see that the search space, shown in blue, expands at each step. The darker colored regions in the exemplar indicate a higher probability of selecting within that region.

Random Search ANN. The method tries to improve ANN(x, y) by randomly sampling successively larger regions around ANN(x, y). If the RGB distance between the randomly chosen patch and the target patch is less than the current best distance, that means a better match was found and the ANN and best distance are updated accordingly.

Algorithm 5: Random Search ANN

Input : The target, exemplar, ANN, best distances
Output: Improves the ANN using several iterations of Random Search

```
1  for (x, y) ∈ target do
2      ann ← ANN[x, y];
3      r ← 1;
4      for i ∈ [1, 7] do
           /* get random patch from exemplar located within radius r of
              ann                                                      */
5          rpatch ← randomSample(exemplar, ann, r);
           /* grab portion of target from x to x+5 and y to y+5        */
6          patch ← target[x : x + 5, y : y + 5];
7          if dist(patch, rpatch) < bestDistances[x, y] then
8              ANN[x, y] ← rpatch.xy;
9              bestDistances[x, y] ← dist(patch, rpatch);
10         end
11         r ← r · 2;
12     end
13 end
```

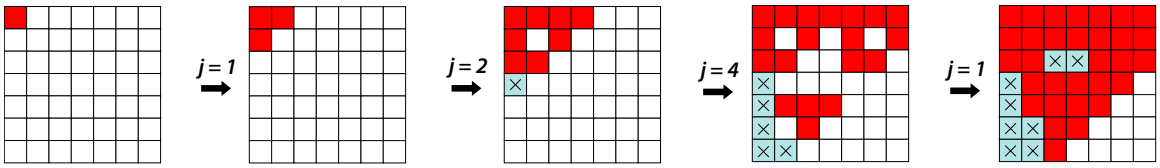


Figure 4-6: Illustration of how jump-flooding is used in content propagation. The content being propagated is the nearest neighbor location. Content is propagated horizontally and vertically from red squares at varying step sizes, j. Propagation is only successful when the nearest neighbor of the propagator is better than that of the propagatee. Red squares represent pixels that had content propagated to them while squares marked with an 'X' represent pixels that ignored content propagation.

Propagate ANN. Our method uses the jump-flooding procedure described in [14] for content propagation (Fig. 4-6). The method tries to improve ANN(x, y) using the known values of ANN$(x + \text{step}, y)$, ANN$(x - \text{step}, y)$, ANN$(x, y + \text{step})$, and ANN$(x, y - \text{step})$, where step follows some sequence of integers. The method looks at the exemplar patch at ANN$(x+a, y+b)-(a, b)$ and checks whether the RGB distance is less than the current best distance, updating the ANN and best distance when appropriate.

Algorithm 6: Propagate ANN

Input : The target, exemplar, ANN, best distances
Output: Improves the ANN using jump-flooding based propagation

```
steps = {1, 2, 4, 1};                          /* steps for jump flooding */
for step ∈ steps do
    for (x, y) ∈ target do
        /* left, right, up, down                                          */
        offsets = { (−step, 0), (step, 0), (0, step), (0, −step)};
        for offset in offsets do
            (x_e, y_e) ← ANN[x + offset.x, y + offset.y] − (offset.x, offset.y);
            epatch ← exemplar[x_e : x_e + 5, y_e : y_e + 5];
            patch ← target[x : x + 5, y : y + 5];
            if dist(patch, epatch) < bestDistances[x, y] then
                ANN[x, y] ← (x_e, y_e);
                bestDistances[x, y] ← dist(patch, epatch);
            end
        end
    end
end
```

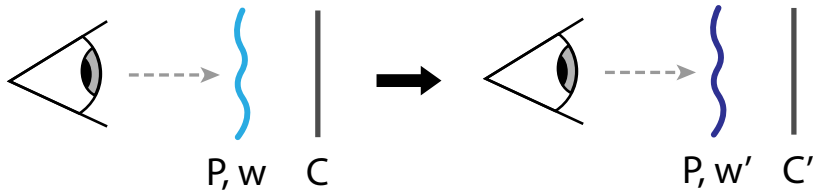


Figure 4-7: Illustration of paint drying. The user initially sees paint with color P and wetness w on canvas C. As the paint dries the wetness of the stroke changes from w to w'. C' represents the new state of the canvas as the drying paint is being gradually committed.

4.2 Paint-drying

Paint-drying is a mechanism in CaTS that shows the uncommitted synthesized strokes on the canvas for a short period of time. Uncommitted strokes possess *liveness*, which means that their texture is continually resynthesized from the exemplar. This allows users to adjust brush texture after strokes are painted. Let C be the canvas color, P be the color of the paint, and w be the wetness of the paint. Then the paint-drying program applies function f to C, P, and w such that

$$C' = f(C, P, w, w') \tag{4.1}$$

where C' is the new canvas color and $w' = w - \Delta w$. In paint-drying we wish to ensure that the color of the paint does not change as it is being dried. This can be expressed as

$$w \cdot P + (1 - w) \cdot C = w' \cdot P + (1 - w') \cdot C' \tag{4.2}$$

To determine the function f it suffices to solve for C' in Equation 4.2:

$$
\begin{aligned}
w \cdot P + (1 - w) \cdot C &= w' \cdot P + (1 - w') \cdot C' \\
(1 - w') \cdot C' &= w \cdot P + (1 - w) \cdot C - w' \cdot P \\
C' &= \frac{w \cdot P + (1 - w) \cdot C - w' \cdot P}{1 - w'} \\
C' &= \frac{(w - w') \cdot P + (1 - w) \cdot C}{1 - w'}
\end{aligned}
$$

$$
\therefore f(C, P, w, w') = \frac{(w - w') \cdot P + (1 - w) \cdot C}{1 - w'}
$$

Chapter 5

System Evaluation

This chapter presents an evaluation of the performance and quality of the CaTS system. Section 5.1 describes the time complexity of the hole-filling procedure used in texture synthesis. Section 5.2 discusses the performance of CaTS and provides timings for the texture synthesis calls. Finally, the quality of the texture synthesis is evaluated in Section 5.3, which presents statistics about the ANN distances.

5.1 Time Complexity of Hole Filling

The hole filling procedure scales down the exemplar and canvas width by a factor of 2 at each level, with the lowest level having a width and height of at least $2\cdot$patch width. Let C be the number of pixels in the canvas, E be the number of pixels in the exemplar, and P be the number of pixels in a patch. The computational cost of operators for a given level of hole filling is $O((C+E)\cdot P)$. For all levels the cost can be expressed as

$$O\left(\sum_{\ell=0}^{k}\frac{1}{4^{\ell}}\Big(C+E\Big)\cdot P\right) \tag{5.1}$$

The value of k depends on the sizes of the canvas and exemplar. Let c be the canvas width, e be the exemplar width, and p be the patch width. If $s=\min(c,e)$ then

$$\begin{aligned}\frac{s}{2^{k+1}} &\geq 2p\\ 2^{k+1} &\leq \frac{s}{2p}\\ k+1 &\leq \log_2\left(\frac{s}{2p}\right)\\ k &\leq \log_2\left(\frac{s}{2p}\right)-1\end{aligned}$$

Since Equation 5.1 is the sum of a geometric sequence, the overall work of hole-filling is $O((C+E)\cdot P)$.

5.2 Texture Synthesis Performance

As the performance of texture synthesis is largely dependent on the performance of PatchMatch, we initially tried different methods to optimize the speed of PatchMatch functions. One approach was to take advantage of SSE intrinsic functions in compute-heavy tasks. The distance function, in particular, greatly benefited from the use of SSE intrinsics because of the highly parallel nature of the distance comparisons. In the end we found that migrating the system to the GPU performed texture synthesis at a fraction of the time it took using the original method. To confirm our complexity analysis we ran the hole filling procedure on empty canvases of varying sizes and timed the results over 100 iterations. Figure 5-1 plots the canvas area (or number of pixels in the canvas) against the minimum synthesis time. The linear trend confirms the linear relationship between canvas area and synthesis time in the complexity analysis.

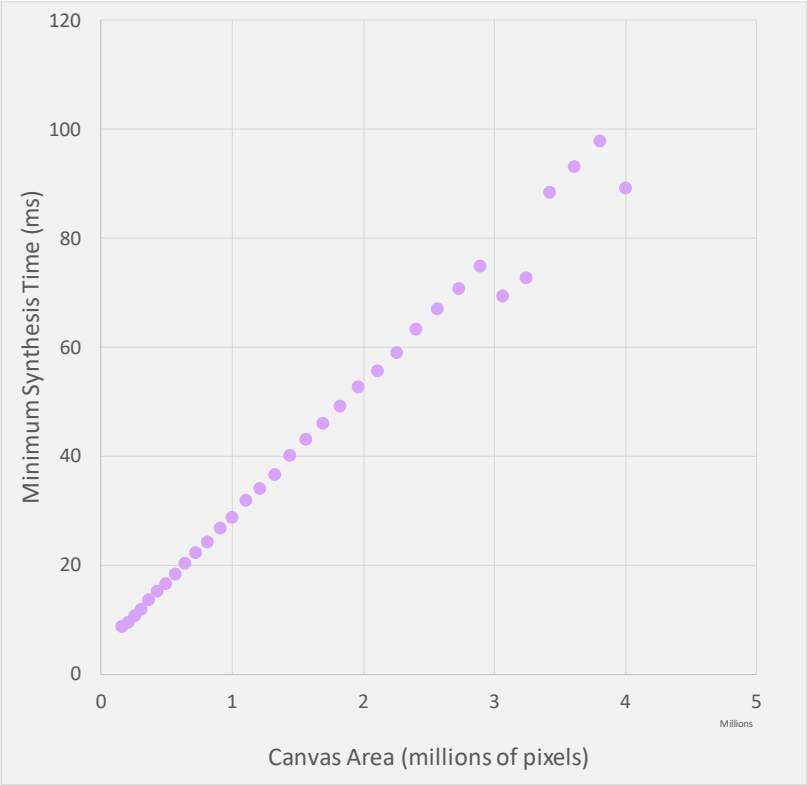


Figure 5-1: A plot correlating canvas area in millions of pixels against minimum synthesis time in milliseconds. The minimum synthesis times were calculated over 100 iterations.

5.3 Texture Synthesis Quality

The RGB distance between the target patches and their approximate nearest neighbors is directly related to the quality of matches: the shorter the distance, the better the match. To evaluate the quality of matches we ran the hole filling procedure on an empty 1 megapixel canvas and computed the average distance over the nearest neighbor field for 200 trials. Figure 5-2 shows a histogram of the results. A distance value of 0 denotes a perfect match, while values closer to 1 are indications of poor matches. The average distance value over all the averages is 0.00577, which indicates that the method produces high quality matches.

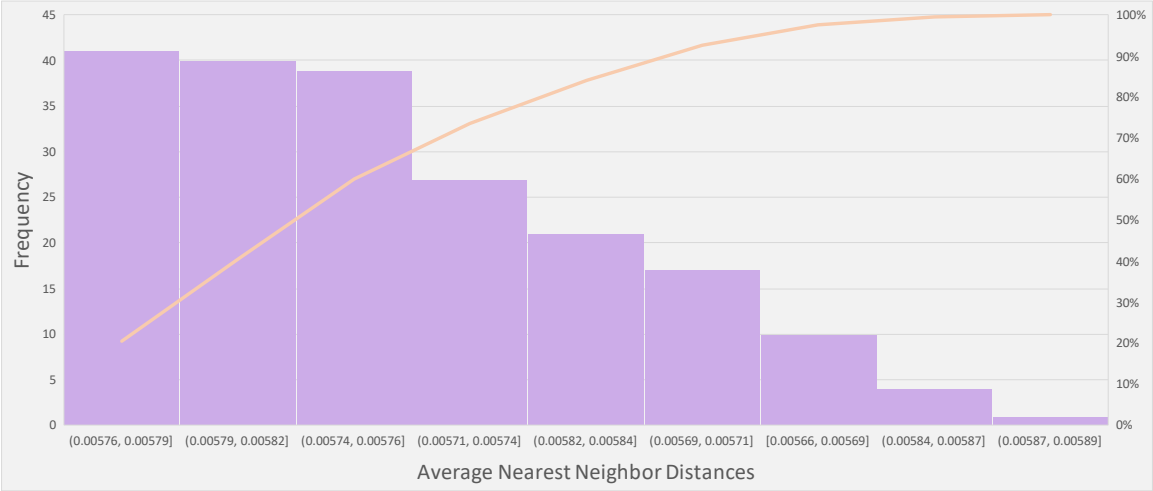


Figure 5-2: A histogram showing the distribution of the average nearest neighbor distances over 200 trials. The CDF is represented by the orange line.

Chapter 6

User Evaluation

This chapter evaluates the effectiveness of the CaTS system from a user perspective. Section 6.1 presents user paintings and feedback from an earlier version of the CaTS system and discusses how their comments informed the design decisions for the new version. Section 6.2 presents the user paintings and feedback from the current version of the CaTS system. Finally, Section 6.3 reviews different painting techniques that can be used with the system.

6.1 Pilot Testing

An earlier version of the system was tested with three artists in an informal user study. In that version of the system, texture synthesis was not working in real-time – instead, intermediate synthesis results were shown as strokes were being synthesized. The undo, cropping, and paint-drying functions were also not present in that version of the system. In that study artists were simply instructed to experiment with the system through freeform paintings and provide feedback on their experience. Through this preliminary user study we were able to identify the weak points of the interface and get a better sense of how typical users would use the system.

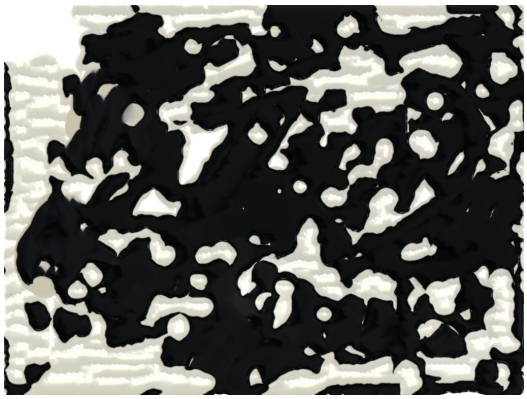
	**Comments from Artist 1** The system blends out the texture too much The system makes a lot of assumptions about what color the stroke should be The smoothing of the texture feels somewhat limiting Likes the artifacts produced by the texture synthesis The system works well with high contrast textures
	**Comments from Artist 2** The system takes some time to get used to Likes the randomness Really compatible with their workflow, which involves allowing the image to organically take shape Wishes there was a way to meld strokes Wants control over blending Wants some sort of save/autosave feature Wishes there was a way to adjust the matching area Wishes there was a soft brush Texture synthesis could be faster
	**Comments from Artist 3** Takes a while to get used to Fun to play around with Super portable Likes the artifacts produced by the texture synthesis Maybe a high level description of the algorithm would be super motivating for people to see how they can utilize it Should be a faster way to change the brush size like a slider on screen–so they can quickly paint the whole background in a few strokes then quickly decrease the size to work out the details A motivating story of how it can be useful would be cool, like someone who's traveling all around the world and creates a drawing for each place using the textures of that place

Table 6.1: Painting results and comments from pilot testing.

6.1.1 Summary of Feedback from Pilot Testing

Table 6.1 shows some of the paintings and comments made by three artists who participated in the pilot study. In general, the feedback received from the early version of the program was promising; while there was a consensus that it takes some time to get used to the system, artists found the painting tool enjoyable to use and liked the random artifacts that would result from the texture synthesis. The main critical comments were directed towards the texture synthesis being too unpredictable. Artists also expressed wanting the ability to specify which region of the video they would like to use for texture synthesis. They also wanted to have erase and undo functions to correct their mistakes – especially given how they had limited control over the synthesis results. Finally some artists wanted to see faster texture synthesis speeds.

Much of the user feedback was integrated into the revised version of the system. The additions made in response to feedback were the erasing, cropping and undoing functions. Back-end changes included porting the implementation to the GPU to support faster synthesis calls and using tiling to add more predictability in the synthesis results.

6.2 Beta Testing

A second pilot study was conducted with the revised version of the system. Testing was conducted with two artists – one who previously participated in the pilot testing phase and one who is less familiar with the system. Similar to the pilot study, we asked the artists to make freeform paintings and comment on the experience. In general, the results and feedback were encouraging.

6.2.1 Summary of Feedback from Beta Testing

Table 6.2 shows the paintings and comments made by the beta testers. Based on the artists' feedback we were able to identify some limitations to our new system. The

	**Comments from Artist 1** Easier to use than the previous version Paint drying effect is interesting Would like to have some way to zoom into the image as otherwise it is difficult to do detailed work Would like to have a layering system so that more complex paintings can be made
	**Comments from Artist 2** Enjoyable to use Feels different from normal painting Easy to make shading by putting hand over the exemplar Can achieve lots of effects that are difficult to get with traditional/digital means Once you have the focus of the camera properly set it isn't too hard to control Sometimes boundary matching can do things you weren't expecting Would be nice to have a color correction system Would be nice to have a simple color picker so that users can lay down rough colors as a guide before applying textures Would like to see a layer clipping system
	

Table 6.2: Painting results and comments from beta testing.

learning curve for the system still exists, but is reported to be less steep than the previous version due to the improved texture synthesis. The artist who participated in pilot testing said that the new version of the system is an noticeable improvement from the previous version, and found the undo function to be very useful. They felt that the system was less unpredictable, though they thought the tiling of the texture gave the strokes too much regularity. They also thought it would be nice if the program could maintain a 'texture history', because having to constantly switch between different textures can be cumbersome. A texture history would allow users to quickly select from a palette of familiar textures – though in this case, instead of sampling from a dynamic webcam feed the system would be using a static exemplar.

The second artist greatly enjoyed making paintings with the system. They liked how some of the results produced by CaTS are not easily achievable through purely digital or traditional methods. They commented on how the design of CaTS encourages them to use unexpected textures, simply because they wanted to see what the texture synthesis would do – for example the hair in the third image of Table 6.2 was made using a picture of a skunk. They also appreciated how the program makes it easy to paint textures that are normally difficult to paint, such as skin.

One feature that both artists would like to see is a layering system. Currently, CaTS only allows users to paint on one layer, making the workflow similar to painting traditionally. With a multi-layered system artists can modify specific parts of the painting without affecting the others and develop their painting in a non-linear fashion. This system would be particularly useful if the artist wished to separate lineart from color.

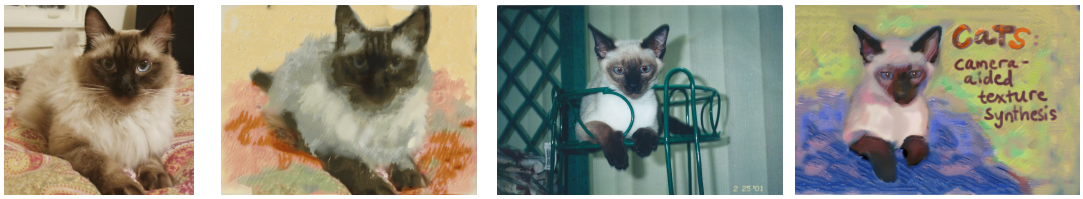
6.3 Painting Techniques and Strategies

After continued use of the system and through observation of the testers we compiled a list of some painting techniques and strategies that can be used with the system.

Filters and distortions Rather than select from a menu of effects, the user can put physical filters, such as colored film, over the camera to achieve the desired outcome.

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

Figure 6-1: Using natural media exemplars (a,b) to produce result (c). Closeups (d) and (e) show the fidelity of the texture synthesis results.



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

Figure 6-2: Examples of paintings made using an image as the canvas base. Both paintings were painted using oil pastel exemplars.

Lighting and shading Darker versions of textures can be made by casting a shadow over the exemplar, and conversely lighter versions can be made by shining a spotlight onto the exemplar. Gradients can also be created using this method.

Paint-overs By importing an existing image into the canvas users can have a base for their painting without having to start from scratch. This technique is often used to make stylized versions of photographs (Fig. 6-2).

Overlays A texture overlay effect can be achieved by taking advantage of the paint drying mechanism. As a wet stroke is drying on the canvas, the user can quickly switch the camera to a different texture. The second texture will appear as a transparent overlay atop the first texture.

Chapter 7

Conclusion and Future Work

We have presented CaTS, a novel digital painting system that synthesizes texture from live video in real time. By capturing textures from live video, CaTS cuts down the texture creation workflow and encourages an exploratory approach to painting. Informal user testing shows that CaTS can enable artists to create paintings that are not easily achieved through purely digital or traditional means. While the results from the informal user evaluation are promising, future efforts will include evaluating the effectiveness of CaTS through a formal user study (see Appendix B).

An interesting direction to take for the future would be to consider how to support real-time texture-preserving blending. One potential approach would be to incorporate an image melding [9] solution to blend brush strokes. Another interesting future effort would be to generate brush shapes. Currently the system only uses a circular brush which may not be effective for certain textures. It would be interesting to consider how to generate different brush shapes from textures and produce convincing boundary effects, as explored by Ritter et al. [17]. Additionally, CaTS is not a fully featured digital painting tool; adding functions like panning and zooming, a layer hierarchy, and clipping masks would provide users with more flexibility in their painting process. Finally, while we have only explored this system in the context of 2D painting, there is potential for it to be integrated into a 3D modeling workflow. Possible applications include painting textures onto models or quickly creating bump maps.

Appendix A

Painting Gallery

Below is a gallery of more paintings made with CaTS. All paintings were created by the author unless otherwise stated. Images are used with permission from the artists.

Figure A-1: "Catfly"

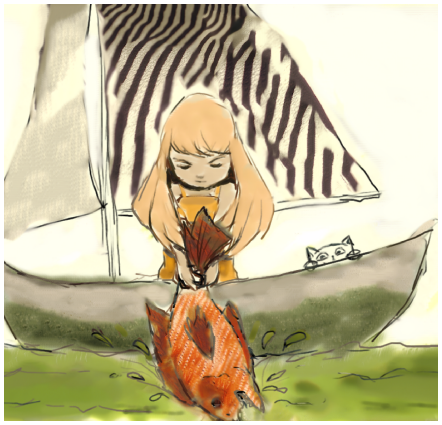


Figure A-2: "Gone Fishing"

Figure A-3: "Wean" by Anonymous

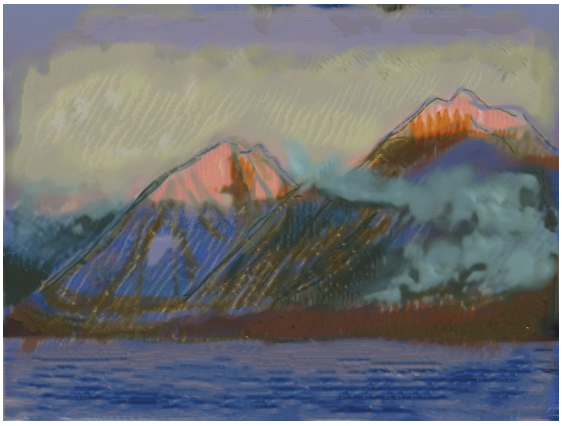


Figure A-4: "Mountain Range"

Figure A-5: "Reclining Lady"

Figure A-6: "Untitled"

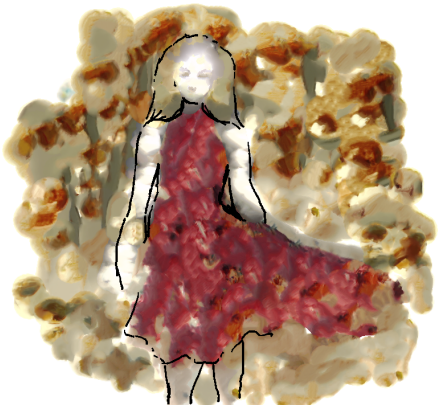


Figure A-7: "Ghost"

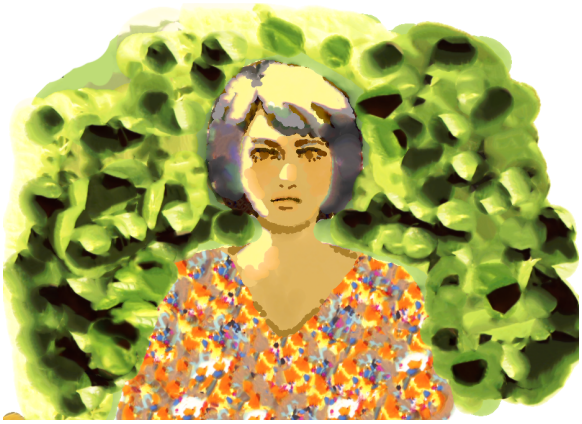


Figure A-8: "Nature"

Figure A-9: "Face Horror"

Figure A-10: "Charcoal Lion"

Appendix B

User Study Design

We will conduct a qualitative user study that aims to evaluate the effectiveness of our digital painting system. The study is directed towards users between 18-60 years old with at least 1 year of digital or traditional painting experience. Two different studies will be conducted: a single-user study and a group user study. The set of participants will be different for each study. In both studies users will create paintings with the system based on a set of guided and freeform tasks.

B.0.1 Single User Study

In the directed single user study participants will create paintings using our interface and we will observe them as they paint. Participants will complete a set of guided and freeform painting tasks and will be asked to think aloud as they are painting. Notes and screen recordings will be taken. After the painting tasks are complete participants will fill out the Creativity Support Index [7] survey to reflect on their experience.

B.0.2 Group User Study

In the group user study we will do the same process as above with a group of participants. The motivation for the group user study is to observe group behaviors and see whether the system fosters collaboration.

Bibliography

[1] Adobe capture. `https://www.adobe.com/products/capture.html`.

[2] Bill Atkinson. Macpaint. `https://archive.org/details/mac_Paint_2`.

[3] Connelly Barnes, Dan B. Goldman, Eli Shechtman, and Adam Finkelstein. The patchmatch randomized matching algorithm for image manipulation. *Commun. ACM*, 54(11):103–110, November 2011.

[4] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 24:1–24:11, New York, NY, USA, 2009. ACM.

[5] Connelly Barnes, Eli Shechtman, Dan B. Goldman, and Adam Finkelstein. The generalized patchmatch correspondence algorithm. In *Proceedings of the 11th European Conference on Computer Vision Conference on Computer Vision: Part III*, ECCV'10, pages 29–43, Berlin, Heidelberg, 2010. Springer-Verlag.

[6] William Baxter, Jeremy Wendt, and Ming C. Lin. Impasto: A realistic, interactive model for paint. In *Proceedings of the 3rd International Symposium on Nonphotorealistic Animation and Rendering*, NPAR '04, pages 45–148, New York, NY, USA, 2004. ACM.

[7] Erin A. Carroll and Celine Latulipe. The creativity support index. In *CHI '09 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '09, pages 4009–4014, New York, NY, USA, 2009. ACM.

[8] Nelson S.-H. Chu and Chiew-Lan Tai. Moxi: Real-time ink dispersion in absorbent paper. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 504–511, New York, NY, USA, 2005. ACM.

[9] Soheil Darabi, Eli Shechtman, Connelly Barnes, Dan B. Goldman, and Pradeep Sen. Image melding: Combining inconsistent images using patch-based synthesis. *ACM Trans. Graph.*, 31(4):82:1–82:10, July 2012.

[10] Anuruddha Hettiarachchi, Suranga Nanayakkara, Kian Peen Yeo, Roy Shilkrot, and Pattie Maes. Fingerdraw: More than a digital paintbrush. In *Proceedings of the 4th Augmented Human International Conference*, AH '13, pages 1–4, New York, NY, USA, 2013. ACM.

[11] Craig Hickman. Kid pix - the early years. http://red-green-blue.com/kid-pix-the-early-years/.

[12] Mami Kosaka and Kaori Fujinami. Unicrepaint: Digital painting through physical objects for unique creative experiences. In *Proceedings of the TEI '16: Tenth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '16, pages 475–481, New York, NY, USA, 2016. ACM.

[13] Katrin Lang and Marc Alexa. The markov pen: Online synthesis of free-hand drawing styles. In *Proceedings of the Workshop on Non-Photorealistic Animation and Rendering*, NPAR '15, pages 203–215, Aire-la-Ville, Switzerland, Switzerland, 2015. Eurographics Association.

[14] Sylvain Lefebvre and Hugues Hoppe. Parallel controllable texture synthesis. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 777–786, New York, NY, USA, 2005. ACM.

[15] Su-Ian Eugene Lei, Ying-Chieh Chen, Hsiang-Ting Chen, and Chun-Fa Chang. Real-time physics-based ink splattering art creation. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '13, pages 191–191, New York, NY, USA, 2013. ACM.

[16] Jingwan Lu, Connelly Barnes, Stephen DiVerdi, and Adam Finkelstein. Realbrush: Painting with examples of physical media. *ACM Trans. Graph.*, 32(4):117:1–117:12, July 2013.

[17] Lincoln Ritter, Wilmot Li, Brian Curless, Maneesh Agrawala, and David Salesin. Painting with texture. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques*, EGSR '06, pages 371–376, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[18] Kimiko Ryokai, Stefan Marti, and Hiroshi Ishii. I/o brush: Drawing with everyday objects as ink. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 303–310, New York, NY, USA, 2004. ACM.

[19] Yonatan Wexler, Eli Shechtman, and Michal Irani. Space-time completion of video. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(3):463–476, March 2007.