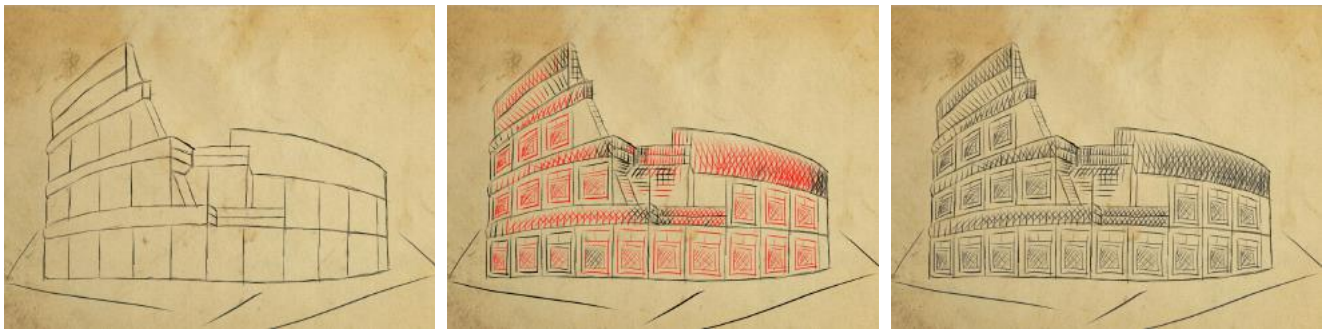
# Autocomplete Painting Repetitions

Jun Xing
Univ. Hong Kong

Hsiang-Ting Chen
Hasso Plattner Institute

Li-Yi Wei
Univ. Hong Kong



**(a)** *initial outline* **(b)** *progress visualization* **(c)** *final outcome*

**Figure 1:** *Reducing manual repetitions by predicting what the user might draw based on past workflows. After sketching the outlines in (a), the user starts to fill in details which require tedious manual repetitions. Our system can help generate such repetitions by predicting what the user might want to draw based on past sketches. (b) visualizes the user-drawn content in black (376 outline strokes and 248 interior strokes), and auto-completed content in red (897 strokes). The final outcome is shown in (c). Our system works for both individual strokes (e.g. the detailed hatches) and structured elements (e.g. the windows). Due to the spatial and temporal variations, these sketches are very difficult, if not impossible, to be generated via prior sequential clone systems.*

## Abstract

Painting is a major form of content creation, offering unlimited control and freedom of expression. However, it can involve tedious manual repetitions, such as stippling large regions or hatching complex contours. Thus, a central goal in digital painting research is to automate tedious repetitions while allowing user control. Existing methods impose a sequential order, in which a small exemplar is prepared and then cloned through additional gestures. Such sequential mode may break the continuous, spontaneous flow of painting. Moreover, it is more suitable for homogeneous areas than nuanced variations common in real paintings.

We present an interactive digital painting system that autocompletes tedious repetitions while preserving nuanced variations and maintaining natural flows. Specifically, users paint as usual, while our system records and analyzes their workflows. When potential repetition is detected, our system predicts what the user might want to draw and offers auto-completes that adjust to the existing shape-color context. Our method eliminates the need for sequential creation-cloning and better adapts to the local painting contexts. Furthermore, users can choose to accept, ignore, or modify those predictions and thus maintain full control. Our method can be considered as the painting analogy of auto-completes in common typing and IDE systems. We demonstrate the quality and usability of our system through painting results and a pilot user study.

**CR Categories:** I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques; H.5.2 [Information Interfaces and Presentation (e.g., HCI)]: User Interfaces—Graphical user interfaces (GUI);

**Keywords:** autocomplete, digital painting, repetition, user interface, workflow, editing history, context, texture, analysis, synthesis

**Links:** DL PDF

## 1 Introduction

Painting is a common practice for content creation, often involving repetitive implements such as stipples, hatches, or brushes. Such repetitions are an integral part of artistic styles and compositions. Yet, they can be quite tedious for manual labor and too versatile for fully automatic generation.

A central goal of digital painting research is to automate such repetitions while allowing sufficient control and expressive power for users [Winkenbach and Salesin 1994]. This has been greatly advanced in recent data driven methods (e.g. [Kazi et al. 2012; Lu et al. 2013; Lukáč et al. 2013]), but they impose a batch/sequential order, in which the desired repetitions have to be prepared first (usually in the form of small exemplars) and then cloned to the desired output regions through various control gestures. Such sequential mode may break the continuous and spontaneous nature of painting flows. For example, users might not know a priori the desired repetitions and would prefer to experiment on the go. Moreover, such batch control is most suitable for large, homogeneous patterned regions. More nuanced or detailed variations, as common in real paintings, can require an excessive amount of exemplars and gestures. Thus, it can be more natural and effective for users to draw as usual, while having the system automatically detect and assist potential repetitions.

We present an UI system that uses past workflows to facilitate interactive authoring of future repetitions (Figure 1). Users can paint with our system as usual similar to ordinary digital painting tools. Meanwhile, our system gradually records their past workflows and analyzes their structure and color relationships. With enough repetition detected, it can predict and auto-complete what the users might want to draw in the near future (temporal ordering), around the current drawing region (spatial proximity), or across similarly colored or structured regions (contextual similarity). Our method

**ACM Reference Format**
Xing, J., Chen, H., Wei, L. 2014. Autocomplete Painting Repetitions. ACM Trans. Graph. 33, 6, Article 172 (November 2014), 11 pages. DOI = 10.1145/2661229.2661247 http://doi.acm.org/10.1145/2661229.2661247.


0730-0301/14/11-ART172 $15.00.
DOI: http://dx.doi.org/10.1145/2661229.2661247

eliminates the need for sequential clone and better adapts to local painting variations. Furthermore, users can accept, ignore, or modify the auto-completes and thus maintain full control.

Autocomplete has been a core feature for common text editing and programming IDE systems, for which our system can be considered as a painting analogy. Our key idea is to treat workflow repetition as a form of texture, and extend texture analysis and synthesis methods [Wei et al. 2009; Ma et al. 2011] to provide a predictive interface for sketching [Lee et al. 2011; Fernquist et al. 2011]. Workflows have been found useful for a variety of applications [Grossman et al. 2010; Denning et al. 2011; Chen et al. 2011; Zitnick 2013]. In our method, we use workflows for both predicting future drawings and adapting them to the surrounding color and shape contexts.

The key challenge is to extend and integrate these methods for quality results and fluid interactions. We perform contextual analysis [Lu et al. 2007; Guerrero et al. 2014] on the shape and color relationships among prior painting operations, from which we derive predictions for potential future ones. These predictions serve as extra constraints in a texture optimization framework [Ma et al. 2011; Ma et al. 2013], and help maintain nuanced contextual relationships in synthesis. Moreover, these analysis and synthesis are conducted incrementally around the spatial-temporal vicinity of the current operation, and thus sufficiently efficient for interactive painting. Unlike prior constrained drawing systems (e.g. [Maulsby et al. 1989; Gleicher and Witkin 1994]) that require explicit user specification, our method automatically detects potential relationships in the background.

We evaluate our method via drawing results and a pilot user study.

In sum, our paper has the following contributions:

- A system to facilitate interactive painting of common repetitions such as stipples and hatches;
- A predictive user interface design that observes ordinary painting flows while reducing tedious manual labor;
- Algorithms to analyze painting workflows and to synthesize new results with high quality and speed.

## 2 Previous Work

**Suggestive sketching** Drawing is a popular and yet challenging activity, and thus significant research efforts have been devoted to design guided or suggestive drawing systems that use various forms of data. For example, portrait sketching can be assisted by analyzed face data [Dixon et al. 2010] or crowd-sourced sketches [Limpaecher et al. 2013]. To help users draw a larger collection of objects, Lee et al. [2011] display shadows extracted from web images to interactively guide user progress, while Iarussi et al. [2013] provide structural guides based on artistic principles. In addition to static images or drawings, it is also possible to guide novices through recorded workflows of experienced painters [Fernquist et al. 2011].

Our method follows this line of work, but uses past drawing workflows from the same user to guide future repetitions.

**Data-driven sketching** While the guided systems can help users draw, they still have to deal with manual repetitions. To help ameliorate such tedious process, many systems have been designed to automate the creation of repetitive drawings via data driven computation. One prime example is detailed textures or patterns [Lukáč et al. 2013; Lu et al. 2013; Lu et al. 2014; Cheema et al. 2014] whose creation can fit well with the traditional copy-and-paste interface scenario. Kazi et al. [2012; 2014] provide friendly gestures to help create both static and dynamic elements.

Our method also follows a data-driven approach to help authoring repetitions, but uses dynamic workflows instead of static patterns or animated sprites.

**Using workflows** Chimera [Kurlander 1993] is an early system for recording and editing graphical histories, including repetitive operations. Recent years have seen the rise of methods that utilize workflows in various forms, such as exploration [Bonanni et al. 2009], visualization [Callahan et al. 2006; Grossman et al. 2010; Denning et al. 2011; Chen et al. 2014], tutorial [Kong et al. 2012; Lafreniere et al. 2013], revision [Chen et al. 2011], stylization [Fernquist et al. 2011], and beautification [Zitnick 2013]. These methods often rely on pre-recorded workflows, which, when not available, may be recovered through a certain extent via analysis [Fu et al. 2011; Denning and Pellacini 2013; Hu et al. 2013; Doboš et al. 2014]. Nancel et al. [2014] provides a comprehensive survey of different conceptual models for workflow analysis.

We follow this line of work, but focus on the analysis and synthesis of dynamic workflows for predictive painting repetitions.

## 3 User Interface

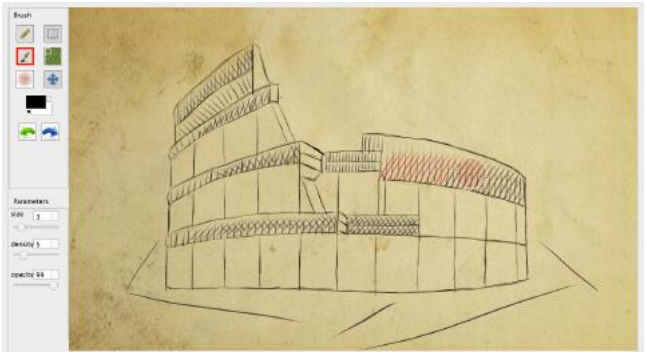


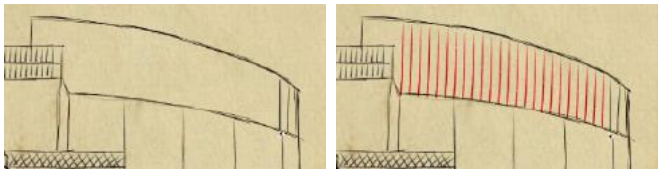
**Figure 2:** *User interface of our system. Similar to ordinary digital painting tools, our system provides widgets (left pane) and a main drawing canvas. However, our system can predict and auto-complete what the user might want to draw in the future based on the detected repetitions in recorded past workflows. Such predictions are shown directly in the main canvas (red strokes), and the user can accept or reject via a simple hot key (via the non-drawing hand) or brush-select specific regions. Please refer to the accompanying video for live actions.*

The user interface of our prototype system (Figure 2) shares the similar visual design to modern sketching systems. Users can sketch as usual while our system automatically records and analyzes the painting workflows in the background. The UI provides two simple functions: *auto-complete* which predicts what the user might want to draw next based on detected repetitions (Section 3.1), and *workflow clone* which allows the manual copy and paste of workflows (Section 3.2). Both functions are context-aware, i.e. the synthesized workflows adapt to the shapes and colors of existing drawings.

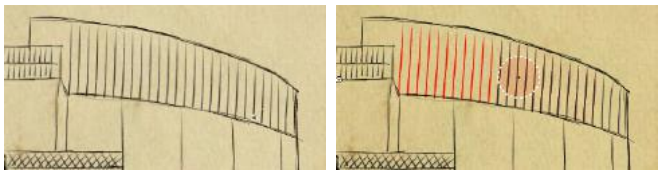
### 3.1 Auto-complete

Inspired by the *auto-complete* function in programming IDE tools, our system automatically analyzes user's painting workflows on the fly and synthesizes what the user might draw in the future. The *auto-complete* function is automatically invoked when the repetitions are detected without requiring additional gestures such as specifying source or target regions.

Figure 3 shows an example where the user is hatching a coliseum sketch for shading effects. As the more hatches are added, our sys-

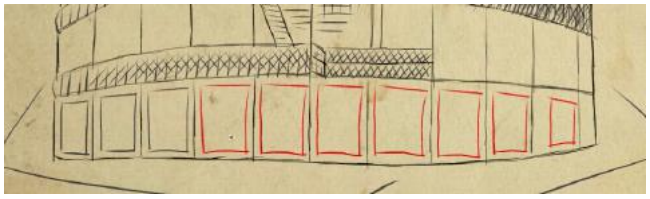


**(a)** *before* **(b)** *prediction*

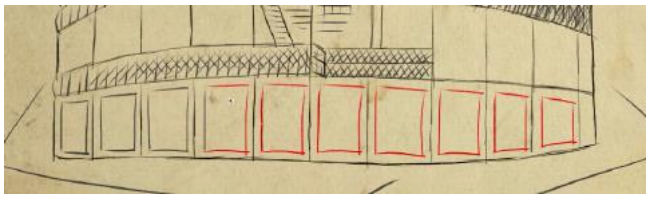


**(c)** *accept all prediction* **(d)** *brush select*

**Figure 3:** *Auto complete. When the user sketches the upper-right pane in (a), our system detects the potential repetition and provides prediction visualized in red color (b). The user can ignore the prediction and continue drawing, accept all prediction via a hot key (c), or brush-select parts of the prediction (d) (pink circle).*



**(a)** *original prediction*



**(b)** *updated prediction*

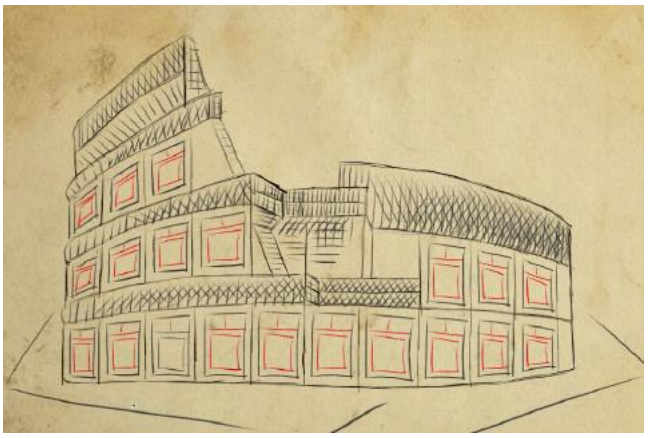
**Figure 4:** *Online prediction update. The initial prediction, shown in (a), does not meet user's expectation, e.g. the fourth window on the bottom row does not fit the frame well. The user can ignore the prediction and continue drawing with a better left edge of the same window, and the system will update the prediction accordingly in real-time (b).*

tem automatically detects the repetition and predicts what the user might want to draw following the current hatching direction and density (Figure 3b). The user can ignore the prediction and continue drawing, accept the prediction (via a hot key similar to IDE), or partially accept the prediction with the selection-brush (Figure 3d).

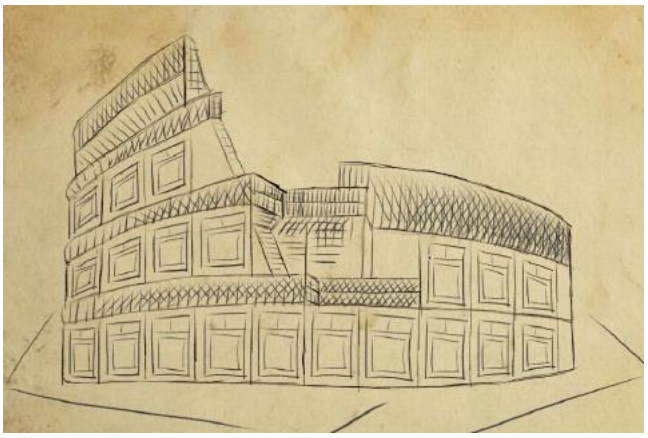
The prediction is being continuously updated in real-time as the user sketches. Figure 4 shows an example where the user is not satisfied with the initial prediction and continue drawing until the system provides satisfactory prediction.

This *auto-complete* function also works for more structured elements by considering color-shape contexts. For example, in Figure 5, the user adds decorative interiors to the windows. Based on the detected repetition among existing window frames, our system automatically propagates the newly added interior sketches to other similar windows with individual adaptions.

By default, all predictions are drawn in the final colors. To help visualization, our system also allows customizing the colors for predictions, such as red and green for gray-scale sketches.



**(a)** *before + prediction*



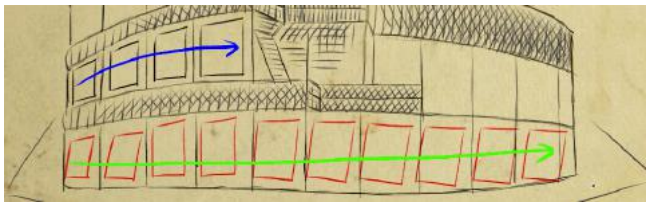
**(b)** *after*

**Figure 5:** *Contextual prediction. When the user adds an interior structure to one window in (a) (the black one in the bottom row, third from the left), our system detects repetitions in the rectangular window frames and predicts the potential interior structures for other windows (shown in red). Similar to Figure 3, the user can accept, reject, or brush-select the prediction. Our auto complete adjusts to the shape context; notice that each interior structure fits the surrounding window frame with slightly different sizes and shapes in Figure 5b.*

### 3.2 Workflow clone

Our autocomplete function does not predict all potential user intentions, especially at areas without sufficient existing repetitions or contexts. We thus provide an optional *workflow clone* function similar to the clone brush in the Photoshop, where the user specifies both the source region to copy from and the target region to clone to. Similar to previous systems [Kazi et al. 2012; Lukáč et al. 2013], our system offers control in the form of 1D paths and 2D regions as visualized in Figure 6 and Figure 15.

Our *workflow clone* has two unique aspects. First, the cloning is conducted *in situ* within the same painting instead of from another source or exemplar and thus is more efficient and can achieve higher predictability on the synthesized outcomes. Second, instead of final pixels, our system clones the fine-grained workflows, which contain rich information that allows the synthesis of new drawings adapting to the contexts of the target areas.



**(a)** *traditional clone*

**(b)** *workflow clone*

**Figure 6:** *Context-aware workflow clone. In both examples, the user specifies the source (via the blue stroke) and targets (via the green stroke). Prior methods, without knowing the workflows, cannot automatically produce new elements fitting the contexts such as window frames (a). Our system, with the knowledge of detailed workflows, can automatically synthesize outputs adaptive to the context in terms of properties such as position, orientation, and shape, as shown in (b).*

## 4 Method Overview

Our main idea is to treat repetitive painting operations as a form of texture [Wei et al. 2009], and extend the sample-based methods in [Ma et al. 2011; Ma et al. 2013] for representation, analysis, and synthesis. We define each painting operation as a continuous user stroke without lifting the pen, such as a stipple, a hatch, or a brush stroke. Below, we first briefly summarize [Ma et al. 2011], followed by an overview of our method.

The method in [Ma et al. 2011] synthesizes geometric elements via data driven texturing. The input consists of a small set of elements (e.g. a small pile of pebbles), and the output is a potentially larger set of elements that resemble the input locally. The output also obeys additional user controls such as global shape and orientation. Each element is represented as a collection of point samples. An energy optimization method is used in [Ma et al. 2011] to synthesize outputs that satisfy both global user controls and local texture similarity to the inputs. Prior methods such as [Kazi et al. 2012; Kazi et al. 2014] have successfully adopted [Ma et al. 2011; Ma et al. 2013] for interactive painting.

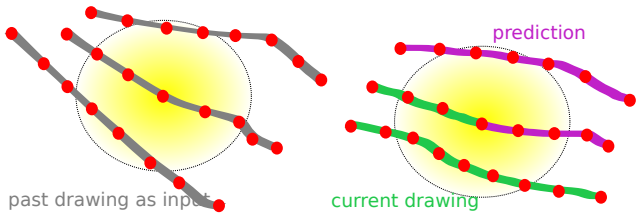



**Figure 7:** *Illustration of our algorithm via a few user sketches. Using past drawings as input (left region), our method predicts what might be drawn next based on partial matching of the current progress (right region). In particular, the green strokes indicate the just completed drawing, and are used to find similar prior drawn strokes in gray (with only the most similar subset shown for clarity). The upper-right part of the matching gray strokes is then used to predict what the user might draw next (the purple strokes). The yellow circles indicate matching neighborhoods.*

Our method follows a similar strategy, but deals with dynamic operations instead of just static appearances, as well as online prediction instead of just batch synthesis. The basic ideas are illustrated in Figure 7. We represent each painting operation (short stroke in this particular example) as a collection of samples that record position, appearance, time stamp – relevant information for reproducing the original painting workflow.

For workflow clone (Section 3.2), we use the recorded painting operations as the input exemplar and synthesize the output operations by treating our operation samples as the geometry samples in [Ma et al. 2011]. The main differences lie in the dynamic nature of our workflows and the need of automatic adaption of painting contexts.

For auto-complete (Section 3.1), we use current user drawings as partial neighborhoods to find potential candidates. In other words, all predictions are based on the similarity match between new and past user operations. We use a small recent temporal window of past operations as input exemplars, for both efficiency and guarantee the use of most recent history for synthesis. We display the most similar match as prediction under our UI canvas.

We present more details of our method in Section 5 (representations and measures) and Section 6 (analysis and synthesis).

## 5 Workflow Texture

We consider workflow repetitions as a form of texture [Wei et al. 2009] with the common assumption of locality within context:

**Locality** – each unit (e.g. image pixel, mesh vertex, element sample) is characterized by its local neighborhood. The locality assumption greatly simplifies the complexity of computation and yet satisfies the repetitive nature of textures.

**Context** – the local repetitions are controlled by high level contexts in the form of maps [Lu et al. 2007; Wei et al. 2008] or structures [Kazi et al. 2012; Lukáč et al. 2013].

We follow a similar convention for defining *workflow texture* - dynamic workflow consisting of a collection of repetitive operation units. However, their formations can be progressively variant, influenced by the surrounding context.

With this definition, the goal of *workflow texture* is to interactively generate output operations that are locally similar to the past inputs while adapting to the output context. Below, we first describe our representation of *workflow texture*, followed by the core concepts of neighborhood and similarity measure.

### 5.1 Representation

**Operation** A *workflow texture* is composed of basic painting operations. Each operation is a continuous user gesture without lifting the pen, e.g. a stipple, a hatch, a brush stroke, etc. Certain operations, e.g. long strokes that usually correspond to skeletons or outlines, are treated as a special kind of *context* operations to capture the control structure of the painting.

We sample each operation *op* uniformly (3-pixel spacing in our current implementation), and for each sample $s \in op$, we store its (1) spatial parameters $\mathbf{p}(s)$ that record the position and direction at $s$, (2) appearance parameters $\mathbf{a}(s)$ such as color, pressure and size, and (3) temporal parameters $\mathbf{t}(s)$ that include the global time stamp and a sample-id for the relative position within *op*. For simplicity, we normalize the sample-id to be a scale value within $[0, 1]$, where 0 and 1 represents the starting and ending positions of *op*, respectively.

In sum, we represent each operation *op* as a collection of samples

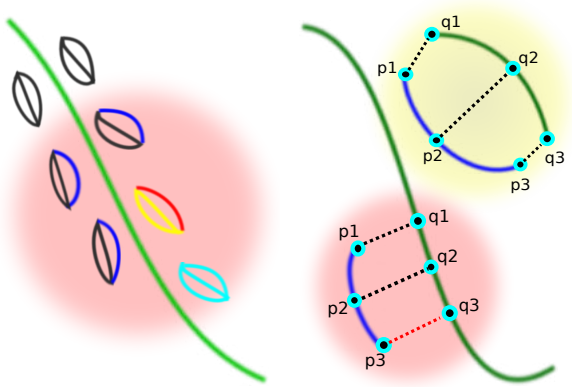
$\{s\}$, and represent each sample $s$ as:

$$\mathbf{u}(s) = (\mathbf{p}(s), \mathbf{a}(s), \mathbf{t}(s)) \tag{1}$$

**Neighborhood** Similar to [Ma et al. 2011; Ma et al. 2013], we define the neighborhood $\mathbf{n}(s)$ around each sample $s$ as the set of all samples within its spatial-temporal neighborhood with user specified size. The neighborhood of an operation $op$ is the union of its sample neighborhoods:

$$\mathbf{n}(op) = \bigcup_{s \in op} \mathbf{n}(s) \tag{2}$$

Our algorithm uses this operation- instead of sample-centric neighborhoods as such element-based representation can work better for more complicated structures as shown in [Landes et al. 2013]. $\mathbf{n}(op)$ can capture relationships within and across elements as well as with contextual operations. See Figure 8a for illustration.




**(a)** *neighborhoods* **(b)** *measurements*

**Figure 8:** *Neighborhoods and measurements. The leaves in (a) are drawn along the green branch from top-left to bottom-right. Each leaf can be considered as a logic unit or element. We use different colors to illustrates different types of neighborhood operations around the central red operation op: across other elements in similar positions (right-most strokes, shown in blue), within the same element (shown in yellow), and the context (shown in green). Operations (cyan) drawn after op are not part of the neighborhood due to temporal causality. (b) shows the measurement of neighborhood similarity. Specifically, we pair samples from different operations with nearest distance to each other (as visualized by the dashed lines).*

### 5.2 Similarity measure

Analogous to [Ma et al. 2011; Landes et al. 2013], the analysis and synthesis of workflow textures hinge on the core formulation of neighborhood similarity. Below, we define similarity in the order of samples, operations, and neighborhoods.

**Sample similarity** We measure the difference between two samples $s'$ and $s$ as:

$$\widehat{\mathbf{u}}\,(s', s) = \left(\widehat{\mathbf{p}}(s', s), \alpha\widehat{\mathbf{a}}(s', s), \beta\widehat{\mathbf{t}}(s', s)\right) \tag{3}$$

which includes their difference in structure $\mathbf{p}$, appearance $\mathbf{a}$, and time-stamp $\mathbf{t}$. $\alpha$ and $\beta$ are two weighting parameters which will be discussed in Section 7. Similar to [Ma et al. 2011; Ma et al. 2013], Equation (3) is computed with respect to the local coordinate frame of $s$. This local frame is sketched by the user under our workflow clone function, and inferred automatically by our system under our auto complete function (to be discussed in Section 6.2).

**Operation similarity** Since each operation is characterized by its samples, we can represent the difference between two operations $op'$ and $op$ via their constituent samples:

$$\widehat{\mathbf{u}}(op', op) = \{\widehat{\mathbf{u}}(s', s) | s' \in op', s \in op\} \tag{4}$$

We match the sample pairs $s \in op$ and $s' \in op'$ to minimize the total distance. Thus, given one sample $s \in op$, its corresponding sample $s' \in op'$ is uniquely determined by finding the one with nearest distance. Via different relative weighting in Equation (3), it is possible to emphasize specific aspects of the operation attributes. An example is illustrated in Figure 8b, in which large spatial weighting emphasizes local difference (red region) while large sample-id weighting emphasizes global difference (yellow region). More details about weighting are in Section 7.

**Neighborhood similarity measure** We measure the dis-similarity between two neighborhoods $\mathbf{n}(op_o)$ and $\mathbf{n}(op_i)$ by summing up the distances of themselves as well as their neighborhood operations:

$$|\mathbf{n}(op_o) - \mathbf{n}(op_i)|^2 = |\widehat{\mathbf{u}}(op_o, s_o) - \widehat{\mathbf{u}}(op_i, s_i)|^2 + \sum_{op'_o \in \mathbf{n}(op_o), op'_i \in \mathbf{n}(op_i)} |\widehat{\mathbf{u}}(op'_o, op_o) - \widehat{\mathbf{u}}(op'_i, op_i)|^2 \tag{5}$$

In particular, the first term measures the dis-similarity between the two operations with respect to their central samples:

$$\widehat{\mathbf{u}}(op, s) = \{\widehat{\mathbf{u}}(s', s) | s' \in op\} \tag{6}$$

And the second term measures dis-similarity of neighborhood operations. Both terms can be expanded into aggregated dissimilarities between sample-pairs.

To observe operation topology, we match sample-pairs with the nearest sample-id. For example, each sample-pair $\widehat{\mathbf{u}}(s'_o, s_o) \in \widehat{\mathbf{u}}(op'_o, op_o)$ matches sample-pair $\widehat{\mathbf{u}}(s'_i, s_i) \in \widehat{\mathbf{u}}(op'_i, op_i)$, where $s_i \in op_i$ has the closest sample-id to $s_o \in op_o$ for all samples in $op_i$, and both $s'_o \in op'_o$ and $s'_i \in op'_i$ are determined via Equation (4). The matching pairs $op'_o, op'_i$ are computed by extending the usage of the Hungarian method [Kuhn 1955] in [Ma et al. 2013] from samples to operations.

## 6 Workflow Computation

Given the dynamic and interactive nature of our system, we extend texture optimization [Kwatra et al. 2005; Ma et al. 2013] for incremental analysis and synthesis. Let $\mathcal{I}$ be the input workflow and $\mathcal{O}$ the incrementally synthesized output, both sequences of operations ordered by their time-stamps. We compute the next output unit $op_o \in \mathcal{O}$ via the following formulation:

$$E(op_o; \mathcal{I}) = \min_{op_i \in \mathcal{I}} |\mathbf{n}(op_o) - \mathbf{n}(op_i)|^2 + \Theta(op_o) \tag{7}$$

where $op_i$ indicates the corresponding input unit with similar neighborhood to $op_o$. The first energy term above measures the neighborhood similarity between $op_i$ and $op_o$ as defined in Equation (5). The second constraint term, $\Theta(op_o)$, corresponds to the prediction $\mathbf{u}(op_o)$ computed from our analysis stage in Section 6.2.

### 6.1 Workflow synthesis

Our synthesis method is built upon the synthesis framework of [Ma et al. 2011]. However (1) instead of specifying the constraints explicitly, we derive them from our analysis in Section 6.2, and (2) instead of batch optimizing all outputs, we incrementally synthesize each output. For each output we start with multiple initializations, and pick the most plausible one via search and assignment steps.

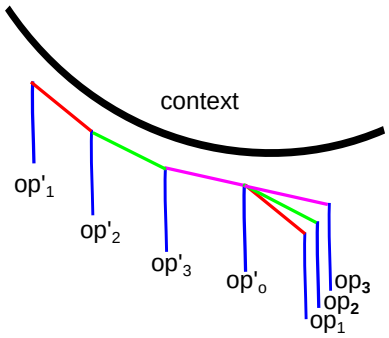



**Figure 9:** *Synthesis initialization. Each $op_i$ is initialized from $op_i'$ for $i = 1, 2, 3$.*

#### 6.1.1 Initialization

We initialize next operation based on local similarity within the past workflow. As inspired by k-coherence for texture synthesis [Tong et al. 2002; Wei et al. 2009], we initialize its next operation $op_o$ of the (last drawn) operation $op_o'$ via:

$$\widehat{\mathbf{u}}(op_o, op_o') = \widehat{\mathbf{u}}(op_i, op_i') \tag{8}$$

where $op_i'$ is a matching operation of $op_o'$, and $op_i$ is the next operation of $op_i'$, as illustrated in Figure 9. The potential initialization $op_o$ can be derived by minimizing the following energy equation:

$$E(op_o) = \sum_{s_o \in op_o} \sum_{s_o' \in op_o'} w(s_i, s_i') \left|\mathbf{u}(s_o) - \mathbf{u}(s_o') - \widehat{\mathbf{u}}(s_i, s_i')\right|^2 \tag{9}$$

where $w(s_i, s_i')$ is a weighting parameter discuss below. For each initialization $op_o$, we go through the search and assignment steps below for further optimization, and select the most plausible one, i.e. with lowest energy in Equation (7).

#### 6.1.2 Search step

During this step, we search for multiple matching operations $\{op_i\}$ within a local spatial-temporal window of $op_o$ whose neighborhoods are similar to $\mathbf{n}(op_o)$. Specifically, for each $op_i$, we first calculate the neighborhood similarity $|\mathbf{n}(op_o) - \mathbf{n}(op_i)|^2$ via Equation (5). We then select all other matching operations whose dissimilarity is smaller than $2\,|\mathbf{n}(op_o) - \mathbf{n}(op_{min})|^2$, where $op_{min}$ is the one with smallest dissimilarity value.

#### 6.1.3 Assignment step

Our assignment step resembles [Ma et al. 2011] except that we assign an operation (a group of samples) rather than a sample. For clarify of presentation, we describe how to deal with the two terms in Equation (7) individually, and sum everything up in the end.

**Neighborhood term** For each matching neighborhood operation pair $(op_i, op_i')$ and $(op_o, op_o')$ from initialization, $\widehat{\mathbf{u}}(op_i, op_i')$ provides a prediction for $\widehat{\mathbf{u}}(op_o, op_o')$:

$$\widehat{\mathbf{u}}(op_o, op_o') = \widehat{\mathbf{u}}(op_i, op_i') \tag{10}$$

which can be expanded into the constituent samples:

$$\widehat{\mathbf{u}}(s_o, s_o') = \widehat{\mathbf{u}}(s_i, s_i') \tag{11}$$

We can expand the first term in Equation (7) as follows:

$$E_n(\{s_o\}) = \sum_{s_o \in op_o} \sum_{op_o' \in \mathbf{n}(op_o)} \sum_{s_o' \in op_o'} w(s_i, s_i') \left|\mathbf{u}(s_o) - \mathbf{u}(s_o') - \widehat{\mathbf{u}}(s_i, s_i')\right|^2 \tag{12}$$

where $op_o'$ runs through every operation within $\mathbf{n}(op_o)$, and $op_i'$ is the matching neighborhood operation of $op_o'$ within $\mathbf{n}(op_i)$. And for each sample $s_o \in op_o$, $s_o'$ runs through every sample within $op_o'$, and $(s_i, s_i')$ is the matching sample-pair of $(s_o, s_o')$ between $(op_i, op_i')$. $w(s_i, s_i')$ is a weighting parameter that will be discussed below.

**Constraint term** By conducting global analysis and local analysis on the similarity operations $\{op_i\}$ extracted in the search step, we can get a prediction $\overline{\mathbf{u}}(op_o)$ (Section 6.2) for the properties of $op_o$. We use the prediction as an additional constraint for synthesis. Similar to the neighborhood term above, we can expand the second term in Equation (7) as follows:

$$E_c(\{s_o\}) = \sum_{s_o \in op_o} \left|\mathbf{u}(s_o) - \overline{\mathbf{u}}(s_o)\right|^2 \tag{13}$$

**Sum up** Thus, by summing all the energy functions above, we can calculate the next operation by minimizing:

$$E(op_o) = E_n(\{s_o\}) + E_c(\{s_o\}) \tag{14}$$

#### 6.1.4 Weighting

For both the search and assignment steps above, we modulate each sample-pair with a weighting parameter. This is because different sample-pairs may capture structures with different topological or geometric importance. For example, close sample pairs can represent joint structures which are important for characterizing the semantic shape/structure of an object, e.g. the window edges in Figure 4, and thus need to be well maintained during the synthesis. Following the smooth synthesis in [Ma et al. 2013], we assign a similar Gaussian falloff parameter to weight each sample-pair, but we only consider the spatial domain in the Gaussian kernel:

$$w(s_i, s_i') = \exp\left(-\frac{|\mathbf{p}(s_i) - \mathbf{p}(s_i')|^2}{\sigma}\right) \tag{15}$$

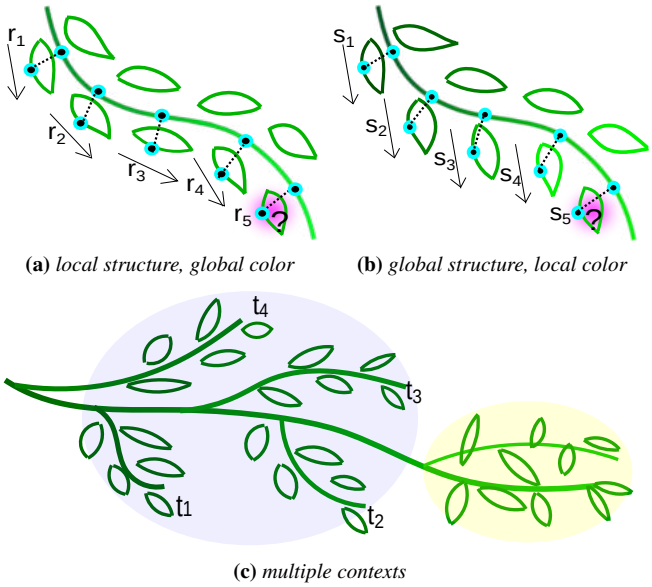
where $|\mathbf{p}(s_i) - \mathbf{p}(s_i')|$ is the distance between sample $s_i$ and $s_i'$, and $\sigma$ is a normalize parameter set to 10 in our implementation.

### 6.2 Workflow analysis

As illustrated in Figure 10, operations can relate to one another via (shared or individual) contexts, can be globally aligned, or can be random and unpredictable. Our system needs to be able to figure out such potential relationships to provide good predictions and to adapt the synthesis to the local context variation.

Intuitively, the way to predict a new operation $op_o$ (including properties like position, direction, color, etc.) is to apply the drawing styles of similar operations drawn before. To do that, we perform contextual analysis on the set of operations similar to $op_o$, to derive how they are drawn in their own contexts and utilize such information to predict the properties of $op_o$ in a new context. The prediction $\mathbf{u}(op_o)$ will be the input to our synthesis stage (Section 6.1). In particular, we need to predict the properties $\mathbf{u}(op_o)$ as the collection of $\mathbf{u}(s_o)$ for each $s_o \in op_o$, including spatial, temporal, and appearance information as in Equation (1). Based on the observations illustrated in Figure 10, we consider two potential frames for each operation: local (as determined by a nearby context operation), and global (as determined by the default global coordinate system). Thus our analysis is performed under two frames separately.

**Local analysis** The goal of local analysis is to determine how a prediction $op_o$ should be constrained by its context operations $\{op_o'\}$, which are determined as the long operations within the




**(a)** *local structure, global color* **(b)** *global structure, local color*

**(c)** *multiple contexts*

**Figure 10:** *Context analysis examples. (a) the directions of leaves ($r_1$ to $r_4$) align to a shared (long) context stroke, and the colors are globally consistent. $r_5$ indicates a resulting prediction. (b) the directions of leaves ($s_1$ to $s_4$) are globally aligned, and the colors are locally constrained by the long context branch. $s_5$ indicates a resulting prediction. (c) the leaves in the blue region align to the context strokes ($t_1$ to $t_4$), while are random in the yellow region.*

neighborhood of $op_o$. By long, we mean at least twice the length of $op_o$. If no such long operations are around, we consider the absence of local context and resort to the global analysis below.

The local analysis is conducted for each $op_o^{'} \in \{op_o^{'}\}$. In order to know how $op_o$ should be constrained by a specific $op_o^{'}$ (e.g. the strokes of leaf $r_5$ relative to the long context stroke in Figure 10a), we perform statistical analysis among the matching set $\{\widehat{\mathbf{u}}(op_i^{'}, op_i)\}$, where each $op_i$ is a matching operation of $op_o$ (e.g. the leaves $r_1$-$r_4$ in Figure 10a) and $\widehat{\mathbf{u}}(op_i^{'}, op_i)$ is the matching pair of $\widehat{\mathbf{u}}(op_o^{'}, op_o)$. Note that the context operations can be individual (e.g. the short branches in Figure 10c) or shared (e.g. the long branch in Figure 10a).

Specifically, for each sample-pair $\widehat{\mathbf{u}}(s_o^{'}, s_o) \in \widehat{\mathbf{u}}(op_o^{'}, op_o)$, we extract its matching sample-pairs $\{\widehat{\mathbf{u}}(s_i^{'}, s_i)\}$ from $\{\widehat{\mathbf{u}}(op_i^{'}, op_i)\}$ (as visualized by dash lines in Figure 10a and Figure 10b). Statistically, $\{\widehat{\mathbf{u}}(s_i^{'}, s_i)\}$ can be characterized as:

$$(\overline{\mathbf{u}}(s_i^{'}, s_i), \sigma(s_i^{'}, s_i)) \tag{16}$$

where $\overline{\mathbf{u}}(s_i^{'}, s_i)$ and $\sigma(s_i^{'}, s_i)$ represent the mean and deviation of set $\{\widehat{\mathbf{u}}(s_i^{'}, s_i)\}$. We use $\overline{\mathbf{u}}(s_i^{'}, s_i)$ as the prediction for $\widehat{\mathbf{u}}(s_o^{'}, s_o)$ and $\sigma(s_i^{'}, s_i)$ as the plausibility of the prediction.

**Global analysis** The computation for global analysis is similar to the local analysis, except that there is no context operation. Specifically, for each sample $s_o \in op_o$, we extract its matching samples $\{s_i\}$ among its matching operations $\{op_i\}$, and compute the mean and deviation of the set $\{\mathbf{u}(s_i)\}$:

$$(\overline{\mathbf{u}}(s_i), \sigma(s_i)) \tag{17}$$

Similar to local analysis above, $\overline{\mathbf{u}}(s_i)$ provides a prediction for $\mathbf{u}(s_o)$, and $\sigma(s_i)$ indicates the plausibility of the prediction.

**Prediction quality** From local and global analysis, we can derive two separate predictions. However, as exemplified in Figure 10, the prediction quality may vary from case to case depending on whether the contexts are local or global and if structure, color, or some other combinations of properties are involved.

Accordingly to above observation, we consider each property (e.g. direction and color) separately. For a particular parameter $\mathbf{u}_i(s_o) \in \mathbf{u}(s_o)$, we formulate the quality of its prediction $\overline{\mathbf{u}}_i(s_o)$ (both for local and global analysis) as:

$$\mathbf{Q}(\overline{\mathbf{u}}_i(s_o)) = \exp(-\frac{\sigma(\mathbf{u}_i(s_o))}{\sigma_1}) \tag{18}$$

where $\sigma_1$ is parameter dependent on the specific $\mathbf{u}_i(s_o) \in \mathbf{u}_i(s_o)$ (discussed in Section 7). In particular, if the size of matching set ($\{\widehat{\mathbf{u}}(op_i^{'}, op_i)\}$ in local analysis and $\{\mathbf{u}(s_i)\}$ in global analysis) is small (less than 3), we think such prediction is not reliable, and set each $\mathbf{Q}(\overline{\mathbf{u}}_i(s_o))$ to be 0.

**Prediction** By combining all predictions (local prediction for each context operation and global prediction), we can calculate the final prediction by minimizing the following energy function:

$$\begin{aligned} E(\mathbf{u}(op_o)) = \sum_{s_o \in op_o} \sum_{op_o^{'} \in \{op_o^{'}\}} & \\ \mathbf{Q}(\overline{\mathbf{u}}(s_o)) \left|\mathbf{u}(s_o) - \overline{\mathbf{u}}(s_o)\right|^2 & \\ + \mathbf{Q}(\overline{\mathbf{u}}(s_o, s_o^{'})) \left|\mathbf{u}(s_o) - \mathbf{u}(s_o^{'}) - \overline{\mathbf{u}}(s_o, s_o^{'})\right|^2 & \end{aligned} \tag{19}$$

where $s_o^{'} \in op_o^{'}$ is the matching sample of $s_o \in op_o$ determined by Equation (4), and each prediction is weighed by their prediction quality.

## 7 Implementation

**Neighborhood** We use a temporal-spatial neighborhood similar to [Ma et al. 2013]. As exemplified in Figure 8a for the red stroke, we extract its 2 previous temporal operations (yellow strokes) and nearby spatial operations within a distance of 0.05 canvas size. For acceleration, instead of using every operation within the aforementioned spatial vicinity, we extract at most 2 longest context operations (the long green stroke) and at most 4 matching operations that are temporally the nearest (the blue strokes), both upper bounds subject to availability. The notion of matching operation is defined below.

**Search window** For the search step in Section 6.1, we search within a local temporal-spatial window for similar operations. Specifically, we use a temporal (workflow) window size of 30 previous operations, and a spatial window size of 0.05 (same as the spatial neighborhood size above).

**Matching operation** Presetting a similarity threshold for selection of matching operations is nearly impossible as such threshold is usually content-dependent (e.g. the dis-similarity between stipples is much smaller than coliseum windows), thus we adopt an online method to measure such threshold dynamically. Specifically, we find the most similar operations within the aforementioned search window as a reference, and define as matching operations those with dissimilarity smaller than twice the smallest dissimilarity.

For acceleration, instead of matching with the whole temporal-spatial neighborhood, we separate the process into two steps: temporal matching followed by spatial matching. Specifically, we first use the temporal neighborhood to search the candidate matching operations (e.g. the blue strokes in Figure 8a), from which we use spatial neighborhood for further sifting (e.g. the blue stroke on the right side of branch in Figure 8a). The similarity thresholds in both steps are described above.

**Sample parameters** We set the appearance weighting $\alpha$ in Equation (3) to be 0.1 for neighborhood matching, and 0.9 for analysis in Section 6.2. So in Equation (14), the painting parameters are mainly influenced by the second constraint term. For the temporal weighting $\beta$ in Equation (3), it includes weightings for both global time stamp $\beta_1$ and sample-id $\beta_2$. For $\beta_1$, we set it to be 0 in spatial neighborhood matching and infinity in temporal neighborhood matching to enforce same drawing order. For $\beta_2$, we set it to be infinity for neighborhood matching to emphasize global difference and 0 for context analysis in Section 6.2 to emphasize local difference.

For parameter $\sigma_1$ in Equation (18), the value is dependent on the specific attributes. For position and direction, we set $\sigma_1$ to be 5 and 0.1, respectively. For color, we set $\sigma_1$ as 0.1 times the maximal color value, e.g. 25 for $[0,\ 255]$.

## 8 Evaluation

We have conducted a pilot user study to evaluate the usability and quality of our system. The study considers the following modes: fully manual drawing as in traditional tools, our auto-complete function, and our workflow clone function.

### 8.1 Participants and apparatus

Our participants include 1 professional artist and 9 computer science graduate students with background in graphics. All participants are experienced in digital sketching and software tools. Throughout the study, participants performed the sketching tasks on a 13 inch laptop with a Wacom tablet.

### 8.2 Procedure

The study consists of four sessions: warm-up, target sketch, open sketch, and final interview. On average, the entire study takes about two hour per participant.

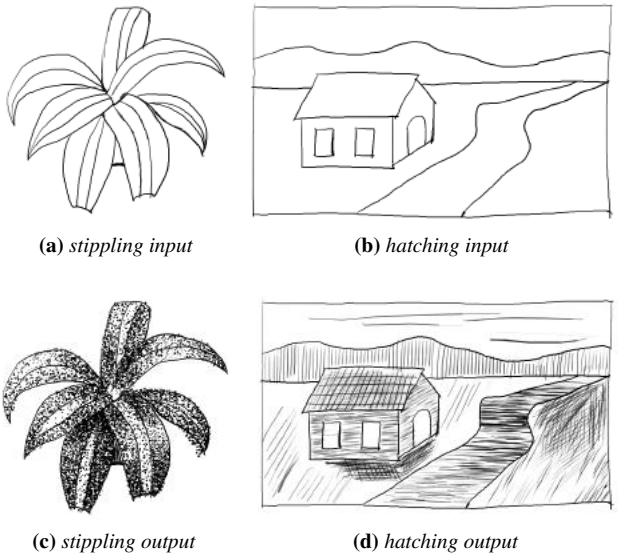
**Warm-up session** The warm-up session is designed to familiarize the participants with the UI and various modes of our system. The tasks consist of filling in the interiors of simple object contours with stipples and hatches. One of the authors guided participants throughout the entire process.

**Target sketch session** The goal is to measure the efficiency among three different sketching modes. Each participants was asked to imitate the shading style of two reference images created by our collaborating artist, and shade the given outline shapes with stipples and hatching (Figure 11). The three sketching modes are arranged among participants in counter-balanced orders.

**Open sketch session** The goal is to observe participants behavior and identify potential usability issues of our system. Participants were free to pursue open-ended drawing using our system with the only requirement of creating drawings with at least a certain amount of repetitive content, e.g. for shading or structures. One of the authors accompanied the participants through the session and encouraged them to try out different features provided in the system.

### 8.3 Results and discussion

Figure 12 provides quantitative measures of completion time and painting operations counts for the target sketch tasks in Figure 11. The result shows that using *clone* & *auto* modes yields 224% & 73% higher average operation count per minutes for stippling (Figure 11c) and 73% & 127% higher for hatching (Figure 11d), against the fully manual mode. Both modes enable participants to finish



**(a)** *stippling input* **(b)** *hatching input*

**(c)** *stippling output* **(d)** *hatching output*

**Figure 11:** *Target sketch tasks. We specify the object shapes and boundary contours (top row), and ask the participants to hatch/stipple the interiors to achieve the desired shading and artistic effects (bottom row).*

the tasks within less time while accomplishing more strokes on the canvas.

The result also suggests that *clone* is most effective for stippling while *auto* performs better for hatching. We learned from the post-hoc interview that participants used the workflow clone function more frequently for stippling because stipples are isotropic and thus the outcomes are more predictable, while the hatching patterns are anisotropic and dependent to the local context and thus the participants found *auto* more predictable and easier to use.

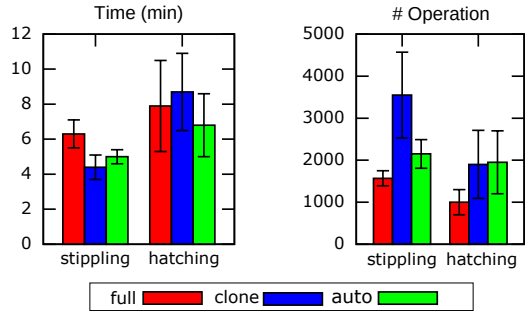



**Figure 12:** *Performance measure for different targets. We measure the completion time and operation number for stippling (Figure 11c) and hatching (Figure 11d) among three modes:* full *— fully manual drawing,* clone *— workflow clone,* auto *— auto-complete.*

Figure 13 shows the ratio of system-generated operations (labor reduction) and undo operations (undo usage) to the total number of operations. The labor reduction measure indicates that both *auto* and *clone* condition help users avoid significant amounts of repetition. The undo usage chart shows that *auto* mode requires 77% and 73% less undo operations than *clone* mode for stippling and hatching. When asking about the difference, some participants commented that sometimes *clone* takes certain amount of trial-and-error to achieve desired outcome. In particular, small source area may lead to visible repetition while larger source area may lead to unexpected or undesired variations.

Finally, Figure 14 provides subjective feedbacks from participants about the functions in our system. To further clarify the usability of

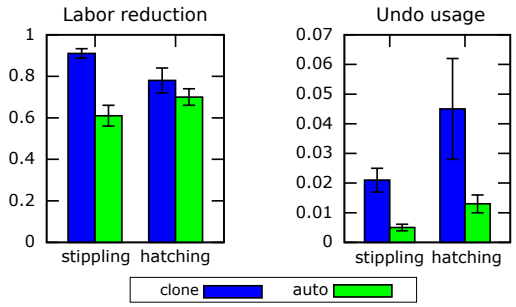



**Figure 13:** *Operation usage. Labor reduction (left) measures the ratio of system-generated operations to all operations. Undo usage (right) is measured by the ratio of undo to all operations.*

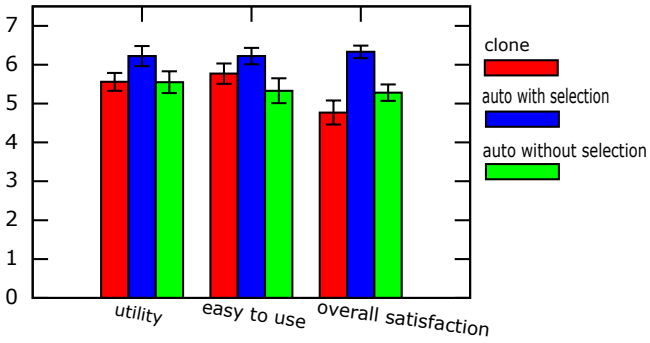



**Figure 14:** *User feedback. We evaluate our workflow clone function and auto-complete function with and without the selection brush. All quantities are expressed as mean±std in a 7-point Likert scale.*

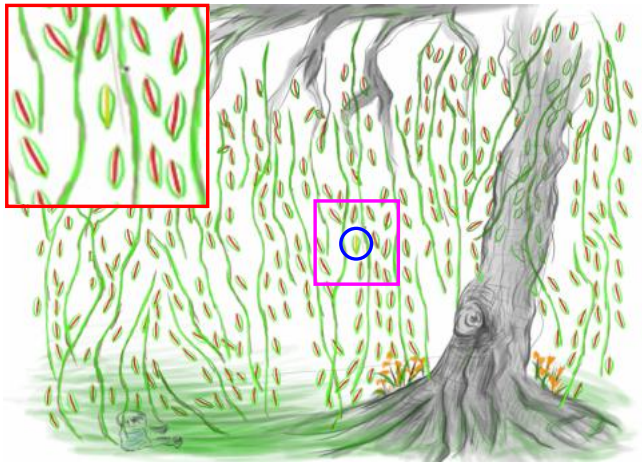
*auto* mode, we collect the rating for *auto* with and without selection brush (Figure 3d) respectively. Result shows that overall participants are more satisfied with the *auto* mode and considered it fits more naturally to the ordinary painting flows than the *clone* mode. Participants also seem to prefer the additional flexibility provided by the selection brush despite the expense of additional gestures.

## 9 Additional Drawing Results

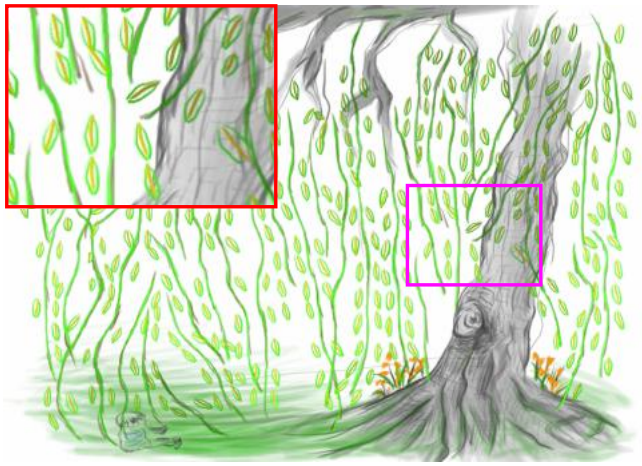
Figure 15 demonstrates a color drawing example. The user first draws the branches and leaf outlines (with the help of our auto-complete), and then decides to add interior veins for all leaves. Manually doing so would require a lot of repetitions especially given the varying colors and shapes of the leaves. Using our system, the user just needs to edit one leaf and having similar changes automatically propagated to other leaves.

Figure 15 also demonstrates the ability of color in addition to structure adaption of our system. Specifically, if the user draws leaves with colors close to the surrounding branches, our method will choose the local consistency to have the newly synthesized leaves with colors similar to the branches. In contrast, if the leaves have similar colors that are sufficiently different from the branch colors, our method will choose the global consistency to have leaves maintain similar colors among themselves.

As described in Section 3.2, our system provides both path and region gestures for workflow clone, which works better than traditional clone for adapting the cloned drawings with the existing context. Figure 6 demonstrates the path gesture, while Figure 16 demonstrates the region gesture. With the assistance of our system, it is possible for novices to draw a variety of sketches with high amount of repetitions as demonstrated in Figure 18.



**(a)** *before + prediction*



**(b)** *after*

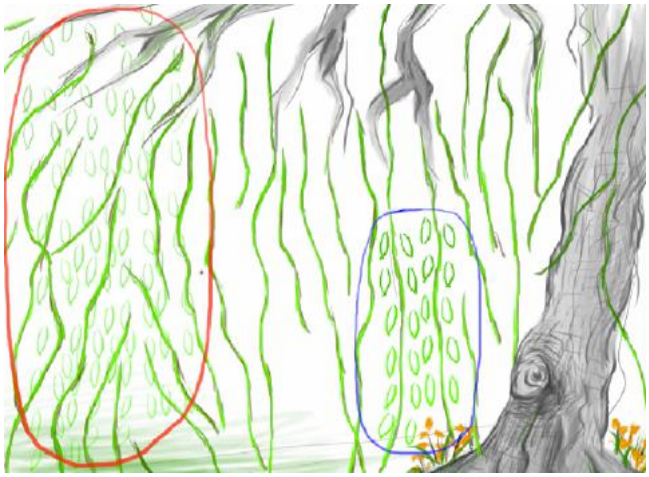
**Figure 15:** *Color drawing example. The user first draws a bunch of willow branches and leaf outlines via our system, then decides to add interior veins to all leaves. Instead of manually repeating the process, the user just edits one leaf (indicated by blue circle) and our system predicts the potential strokes for other leaves (shown in red), as in (a). Similar to Figure 3, the user can accept, reject, or brush-select the prediction. Our auto-complete adjusts to the color-shape context; notice that each interior structure fits the leave outlines with slightly different colors and shapes as in (b). For visualization, each magenta area is enlarged into the top-left red area.*

## 10 Limitations and Future Work

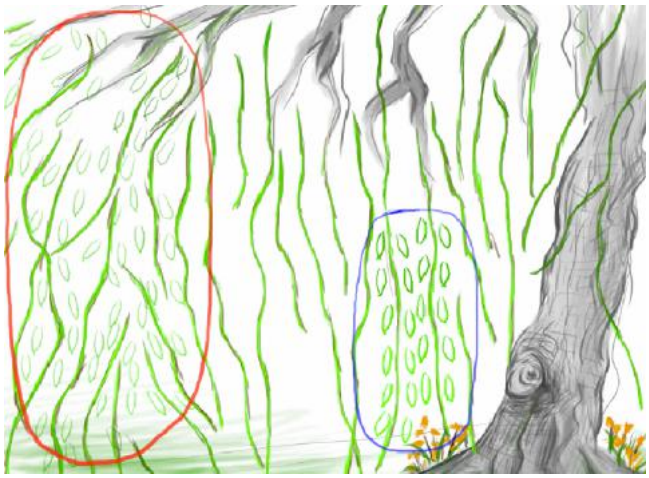
The scope of our auto-complete function depends on the kinds and amounts of repetition that can be detected. It might miss potential repetitions (false negatives) as well as provide inaccurate predictions (false positives) during sufficiently rapid or complex context changes, as exemplified in Figure 17. Thus, a potential future work is to incorporate better pattern detection methods such as [Guy et al. 2014; Huang et al. 2014].

While striving for a simple UI with minimal user gestures, our design might not accommodate more complicated user intentions, such as perspective foreshortening as in [Kazi et al. 2012]. Achieving the best balance between simplicity and functionality can be a good topic.

Our current method is restricted to simple operations consisting of stipples, hatches, or strokes. A good next step is extension for richer set of digital painting and photo editing operations as well as other



**(a)** *traditional clone*



**(b)** *workflow clone*

**Figure 16:** *Context-aware workflow clone. Our system provides both path (shown in Figure 6) and region (shown here) control for workflow clone.*

**Figure 17:** *Limitation case. Our prediction can fail with rapid changing context, such as the rightmost half window.*

interactive tasks [Cypher 1991].

Our current prototype focuses on painting workflow from a single user in a single session. Extending the workflows to multiple users and/or sessions can provide more variations for the prediction as well as further applications such as analyzing drawing styles, generating tutorials, controlling the variations of repetitions, and transferring or combining styles from multiple artists or paintings.

## Acknowledgements

We would like to thank Andy Siska for video dubbing, Carrie de Souza for clarifying that this paper was not withdrawn, our user study participants for their time and feedbacks, and the anonymous reviewers for their constructive suggestions. This work was partially supported by a HKU postgraduate fellowship (UPF) and general research fund *Dynamic Element Textures* (HKU 717112E).

## References

BONANNI, L., XIAO, X., HOCKENBERRY, M., SUBRAMANI, P., ISHII, H., SERACINI, M., AND SCHULZE, J. 2009. Wetpaint: scraping through multi-layered images. In *CHI '09*, 571–574.

CALLAHAN, S. P., FREIRE, J., SANTOS, E., SCHEIDEGGER, C. E., SILVA, C. T., AND VO, H. T. 2006. Vistrails: visualization meets data management. In *SIGMOD '06*, 745–747.

CHEEMA, S., BUCHANAN, S., GULWANI, S., AND LAVIOLA, JR., J. J. 2014. A practical framework for constructing structured drawings. In *IUI '14*, 311–316.

CHEN, H.-T., WEI, L.-Y., AND CHANG, C.-F. 2011. Nonlinear revision control for images. In *SIGGRAPH '11*, 105:1–10.

CHEN, H.-T., GROSSMAN, T., WEI, L.-Y., SCHMIDT, R. M., HARTMANN, B., FITZMAURICE, G., AND AGRAWALA, M. 2014. History assisted view authoring for 3d models. In *CHI '14*, 2027–2036.

CYPHER, A. 1991. Eager: Programming repetitive tasks by example. In *CHI '91*, 33–39.

DENNING, J. D., AND PELLACINI, F. 2013. Meshgit: diffing and merging meshes for polygonal modeling. In *SIGGRAPH '13*, vol. 32, 35:1–35:10.

DENNING, J. D., KERR, W. B., AND PELLACINI, F. 2011. Meshflow: Interactive visualization of mesh construction sequences. In *SIGGRAPH '11*, 66:1–66:8.

DIXON, D., PRASAD, M., AND HAMMOND, T. 2010. icandraw: Using sketch recognition and corrective feedback to assist a user in drawing human faces. In *CHI '10*, 897–906.

DOBOŠ, J., MITRA, N. J., AND STEED, A. 2014. 3d timeline: Reverse engineering of a part-based provenance from consecutive 3d models. *Computer Graphics Forum 33*, 2, 135–144.

FERNQUIST, J., GROSSMAN, T., AND FITZMAURICE, G. 2011. Sketch-sketch revolution: An engaging tutorial system for guided sketching and application learning. In *UIST '11*, 373–382.

FU, H., ZHOU, S., LIU, L., AND MITRA, N. J. 2011. Animated construction of line drawings. In *SIGGRAPH Asia '11*, 133:1–133:10.

GLEICHER, M., AND WITKIN, A. 1994. Drawing with constraints. *Vis. Comput. 11*, 1 (Jan.), 39–51.

GROSSMAN, T., MATEJKA, J., AND FITZMAURICE, G. 2010. Chronicle: capture, exploration, and playback of document workflow histories. In *UIST '10*, 143–152.

GUERRERO, P., JESCHKE, S., WIMMER, M., AND WONKA, P. 2014. Edit propagation using geometric relationship functions. *ACM Trans. Graph. 33*, 2 (Apr.), 15:1–15:15.

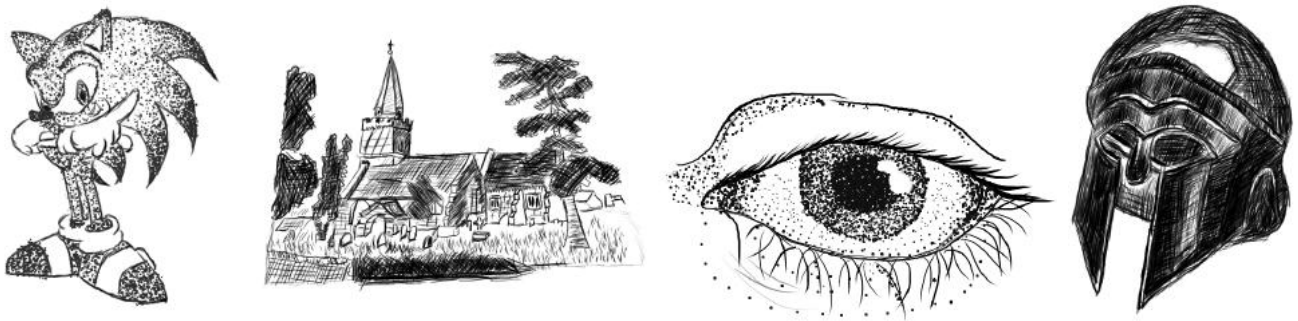
GUY, E., THIERY, J.-M., AND BOUBEKEUR, T. 2014. Simselect: Similarity-based selection for 3d surfaces. *Computer Graphics Forum 33*, 2, 165–173.

HU, S.-M., XU, K., MA, L.-Q., LIU, B., JIANG, B.-Y., AND WANG, J. 2013. Inverse image editing: Recovering a semantic editing history from a before-and-after image pair. In *SIGGRAPH Asia '13*, 194:1–194:11.

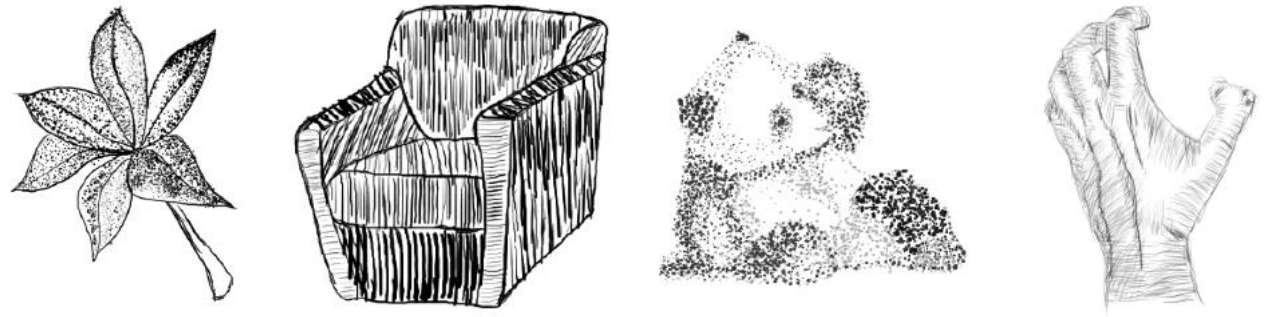
HUANG, Q., GUIBAS, L. J., AND MITRA, N. J. 2014. Near-regular structure discovery using linear programming. *ACM Trans. Graph. 33*, 3 (June), 23:1–23:17.

IARUSSI, E., BOUSSEAU, A., AND TSANDILAS, T. 2013. The drawing assistant: Automated drawing guidance and feedback from photographs. In *UIST '13*, 183–192.

KAZI, R. H., IGARASHI, T., ZHAO, S., AND DAVIS, R. 2012. Vignette: interactive texture design and manipulation with freeform gestures for pen-and-ink illustration. In *CHI '12*, 1727–1736.



**(a)** *sonic: 7318 op, 12 min* **(b)** *scene: 5662 op, 15 min* **(c)** *eye: 5457 op, 13.5 min* **(d)** *helmet: 4651 op, 11 min*



**(e)** *flower: 6657 ops, 15 min* **(f)** *chair: 872 ops, 6.9 min* **(g)** *panda: 6668 ops, 19 min* **(h)** *hand: 2228 ops, 15 min*

**Figure 18:** *Example sketches produced by participants via our system with the corresponding statistics in number of operations and drawing time in minutes.*


KAZI, R. H., CHEVALIER, F., GROSSMAN, T., ZHAO, S., AND FITZMAURICE, G. 2014. Draco: Bringing life to illustrations with kinetic textures. In *CHI '14*, 351–360.

KONG, N., GROSSMAN, T., HARTMANN, B., AGRAWALA, M., AND FITZMAURICE, G. 2012. Delta: a tool for representing and comparing workflows. In *CHI '12*, 1027–1036.

KUHN, H. 1955. The hungarian method for the assignment problem. *Naval research logistics quarterly 2*, 1-2, 83–97.

KURLANDER, D. 1993. Chimera: example-based graphical editing. In *Watch what I do: programming by demonstration*, 271–290.

KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. In *SIGGRAPH '05*, 795–802.

LAFRENIERE, B., GROSSMAN, T., AND FITZMAURICE, G. 2013. Community enhanced tutorials: improving tutorials with multiple demonstrations. In *CHI '13*, 1779–1788.

LANDES, P.-E., GALERNE, B., AND HURTUT, T. 2013. A shape-aware model for discrete texture synthesis. *Computer Graphics Forum 32*, 4, 67–76.

LEE, Y. J., ZITNICK, L., AND COHEN, M. F. 2011. Shadowdraw: Real-time user guidance for freehand drawing. In *SIGGRAPH '11*, 27:1–27:10.

LIMPAECHER, A., FELTMAN, N., TREUILLE, A., AND COHEN, M. 2013. Real-time drawing assistance through crowdsourcing. In *SIGGRAPH '13*, vol. 32, 54:1–54:8.

LU, J., GEORGHIADES, A. S., GLASER, A., WU, H., WEI, L.-Y., GUO, B., DORSEY, J., AND RUSHMEIER, H. 2007. Context-aware textures. *ACM Trans. Graph. 26*, 1 (Jan.).

LU, J., BARNES, C., DIVERDI, S., AND FINKELSTEIN, A. 2013. Realbrush: painting with examples of physical media. In *SIGGRAPH '13*, vol. 32, 117:1–117:12.

LU, J., BARNES, C., WAN, C., ASENTE, P., MECH, R., AND FINKELSTEIN, A. 2014. Decobrush: Drawing structured decorative patterns by example. In *SIGGRAPH '14*.

LUKÁČ, M., FIŠER, J., BAZIN, J.-C., JAMRIŠKA, O., SORKINE-HORNUNG, A., AND SÝKORA, D. 2013. Painting by feature: texture boundaries for example-based image creation. In *SIGGRAPH '13*, 116:1–116:8.

MA, C., WEI, L.-Y., AND TONG, X. 2011. Discrete element textures. In *SIGGRAPH '11*, 62:1–10.

MA, C., WEI, L.-Y., LEFEBVRE, S., AND TONG, X. 2013. Dynamic element textures. In *SIGGRAPH '13*, vol. 32, 90:1–90:10.

MAULSBY, D. L., WITTEN, I. H., AND KITTLITZ, K. A. 1989. Metamouse: Specifying graphical procedures by example. In *SIGGRAPH '89*, 127–136.

NANCEL, M., AND COCKBURN, A. 2014. Causality: A conceptual model of interaction history. In *CHI '14*, 1777–1786.

TONG, X., ZHANG, J., LIU, L., WANG, X., GUO, B., AND SHUM, H.-Y. 2002. Synthesis of bidirectional texture functions on arbitrary surfaces. In *SIGGRAPH '02*, 665–672.

WEI, L.-Y., HAN, J., ZHOU, K., BAO, H., GUO, B., AND SHUM, H.-Y. 2008. Inverse texture synthesis. In *SIGGRAPH '08*, 52:1–9.

WEI, L.-Y., LEFEBVRE, S., KWATRA, V., AND TURK, G. 2009. State of the art in example-based texture synthesis. In *Eurographics '09 STAR*, 93–117.

WINKENBACH, G., AND SALESIN, D. H. 1994. Computer-generated pen-and-ink illustration. In *SIGGRAPH '94*, 91–100.

ZITNICK, C. L. 2013. Handwriting beautification using token means. In *SIGGRAPH '13*, 53:1–53:8.