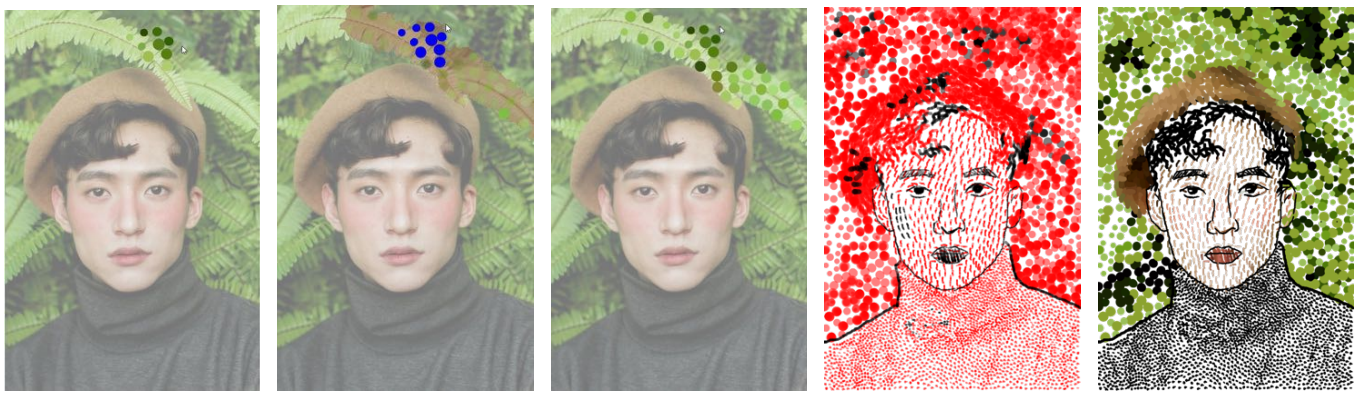
Autocomplete Repetitive Stroking with Image Guidance

Yilan Chen
City University of Hong Kong

Kin Chung Kwan
University of Konstanz

Li-Yi Wei
Adobe Research

Hongbo Fu
City University of Hong Kong



(a) user input (b) suggestion (c) accept (d) type visualization (e) result

Figure 1: *Example of our system workflow.* A user stipples over a leaf region of a reference image (a) while our system predicts what she might draw next (b) (blue strokes: inferred exemplars; pale red region: inferred target region; semi-transparent strokes: system suggestions), which is then accepted by the user (c) (green strokes: user inputs or accepted suggestions in this scene). (d) visualizes all the manually drawn content in black (261 strokes) and autocompleted content in red (3510 strokes). (e) shows the final result with different repetitive stroke patterns over different regions. Our autocomplete system can reduce tedious repetitive inputs, while being fully under user control.

ABSTRACT

Image-guided drawing can compensate for the lack of skills but often requires a significant number of repetitive strokes to create textures. Existing automatic stroke synthesis methods are usually limited to predefined styles or require indirect manipulation that may break the spontaneous flow of drawing. We present a method to autocomplete repetitive short strokes during users' normal drawing process. Users can draw over a reference image as usual. At the same time, our system silently analyzes the input strokes and the reference to infer strokes that follow users' input style when certain repetition is detected. Users can accept, modify, or ignore the system predictions and continue drawing, thus maintaining the fluid control of drawing. Our key idea is to jointly analyze image regions and operation history for detecting and predicting repetitions. The proposed system can effectively reduce users' workload in drawing repetitive short strokes and facilitates users in creating results with rich patterns.

CCS CONCEPTS

• **Computing methodologies** → **Computer graphics**; • **Human-centered computing** → **User interface design**.

KEYWORDS

Interactive System; Autocompletion; Digital Drawing; Prediction; Texture Synthesis

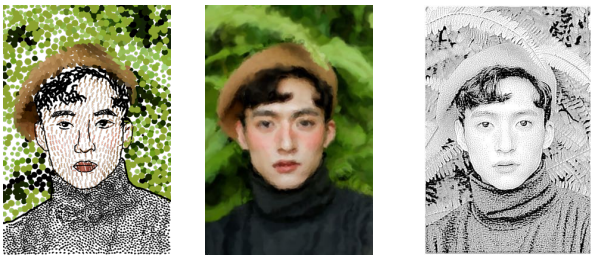
1 INTRODUCTION

Drawing is a common form of artistic expression. By varying the stroke, texture, and shading, artists can create drawings with various styles [5]. Yet, it remains a largely manual process that may require significant artistic expertise and repetitive manual labor.

Various methods have been proposed to synthesize user-initiated repetitive strokes [22, 48] to reduce the manual labor. However, such methods still require sufficient artistic expertise or experience for high-level picture composition. One common way to overcome this skill barrier is to use a reference photo as a scaffold for drawing, i.e., tracing a reference photo physically via transparent papers or digitally via layers in digital drawing applications. With a given reference, many methods exist to automate the synthesis of details, such as contours, textures, or strokes [3, 4, 7, 13, 25–27, 37, 43, 46], with the effects tunable via input parameters or exemplars. However, since these algorithms largely predefine the behaviors, their results may look canned (Figure 3) and cannot give users a sense of ownership. Furthermore, tweaking parameters or providing exemplars can break the spontaneous flow of direct drawing manipulation,

(a) ©Alphonso Dunn (b) ©Vincent van Gogh

Figure 2: *Inspiring manual drawings by artists.*



(a) our result (b) produced with [1] (c) produced with [30]

Figure 3: Our work is designed to reduce the workload of completing repetitive patterns during the manual drawing process. The full control of the drawing process leads to more dynamic results than (b) Photoshop's Art History Brush Tool [1] and (c) StippleShop [30].

which is important to creative decision making [18] and essential to a user's enjoyment and exploration [40].

Manual drawing provides sufficient freedom for individual expressing even when scaffolded with a reference image [47], and its typical interface (e.g., brush, eraser) is familiar to general users. Thus, we aim to enhance the manual drawing process and the typical UI design, by automating tedious repetitions. Our idea is to bridge the two extremes: *manual drawing*, which allows full control but can be tedious; and *image-based algorithmic synthesis*, which saves efforts but provides limited user control and interactivity. As the first attempt towards this goal, our approach focuses on autocompleting repetitive *short* strokes, which are very common in pen-and-ink drawing (Figure 2), under the guidance of a reference image. Like typical digital drawing applications, users can draw freely on a reference image with our system. Meanwhile, our system analyzes the relationships between user inputs and the reference image, detects potential repetitions, and suggests what users might want to draw next. Users can accept, reject, or ignore the suggestions and continue drawing, thus maintaining the fluid control of drawing. See Figure 1 for an example scenario.

The challenge of autocompletion is to predict suggestions that respect both users' inputs and the reference image. Our method is inspired by image analogy [13] and operation history analysis and synthesis [48] while leveraging two key insights. First, since the act of drawing repetitive strokes usually indicates specific intentions (e.g., filling an object or hatching a shading region), we use the common image features among the coherent repetitive strokes to infer the intended regions. Second, the drawing usually relates to the underlying reference image (e.g., the density of strokes with respect to the image lightness). Therefore, we analyze the properties of both the drawing and the reference image to infer possible relationships as contextual constraints for stroke prediction.

We implemented a prototype and conducted a pilot study with participants in different backgrounds to evaluate its utility and usability. The quantitative analysis and qualitative feedback, as well as various drawing results created by users, suggest that our system effectively reduces users' workload in drawing repetitive short strokes and facilitates users in creating results with rich patterns.

2 RELATED WORK

2.1 Image-assisted Drawing

Many drawing support tools adopt reference images and provide intelligent assistance to novices, e.g., beautifying users' sketches with extracted image features [20, 25, 41, 47], or providing educational guidance to novice users [16, 31, 45]. We share a similar goal to [3, 10, 43] so as to reduce the user workload. However, these works use predefined algorithms to generate strokes along cursor movement and only take users' input as an indicator of where to render, thus greatly limiting users' artistic freedom. In contrast, we aim to provide more flexibility between automatic synthesis and manual artistic control by autocompleting tedious repetitions during users' normal drawing processes.

2.2 Image-based Artistic Rendering

Our work is related to image-based artistic rendering (IB-AR) [23], especially stroke-based methods and example-based methods.

Stroke-based methods create artistic results from images by strategically generating brushstrokes whose properties (e.g., position, density, orientation, color, size) are related to the image properties (e.g., gradient, edge, color, salience) [12]. Among those methods, the closest to ours are the early image-based pen-and-ink rendering methods [14, 38], which allow users to input sample elements for distribution. However, users have to prepare the sample elements separately (usually as a standalone file) and then tweak parameters to view the rendered output. In contrast, our system lets users directly specify exemplars on a reference image while silently inferring the distribution properties.

Example-based methods aim to model the visual features of example images for transferring. There are two major modeling approaches: the parametric approach [8, 9, 19] that is based on the summary statistics of stroke characteristics and thus preserves the global textures better, and the non-parametric approach [7, 13, 21] that is based on patch-wise mapping and thus captures the local structures better. We combine both methods for generating strokes: the parametric approach to infer statistical relationships between stroke properties and image features, and the patch-wise matching method to preserve the local arrangements of strokes. Stylit [7] allows users to stylize a rendered ball and simultaneously propagates the style to arbitrary 3D shapes. Our method shares a similar idea in interactive style propagation but with two main differences. First, instead of propagating a style globally, we propagate a style to its perceptually similar local areas so that users can conveniently

define different styles in different areas. Second, we represent drawings as discrete stroke operations instead of raster textures for better preserving their structures and enabling procedural editing [39], such as changing the color or size of the drawn strokes.

2.3 Operation History-assisted Authoring

Operation histories [33] have been leveraged in different authoring tasks, such as sketching [48], animation [34, 49], modeling [35, 42], beautification of freehand drawings [6], and handwritings [53]. Our work is most closely related to that by Xing et al.'s [48], which autocompletes repetitive sketching by analyzing the dynamic operations recorded during authoring. Our method extends their work to consider additional information from a reference image and thus enables the propagation of strokes to regions with similar image attributes such as color or semantic meaning.

In our use scenario, an operation is an input stroke, so our work is also related to stroke pattern analysis and synthesis [2, 4, 15, 17, 22]. These works disregard the temporal relationship among past strokes and do not use image guidances and thus are different from ours.

To sum up, we list our major differences from the discussed closely related works in Table 1.

Table 1: *The differences between our tool and closely related works.* "batch" means the generation is performed in a batch, based on predefined attributes; "dynamic" means the generation is performed based on dynamic operation history. "direct" means users can specify a style by directly operating on the output. "Y" and "N" represent yes and no, respectively, for using image references.

Method	[14]	[13]	[9]	[22]	[48]	Ours
Reference	Y	Y	Y	N	N	Y
Process	batch	batch	batch	batch	dynamic	dynamic
Format	stroke	pixel	stroke	stroke	stroke	stroke
Operate	indirect	indirect	indirect	direct	direct	direct

3 USER INTERFACE

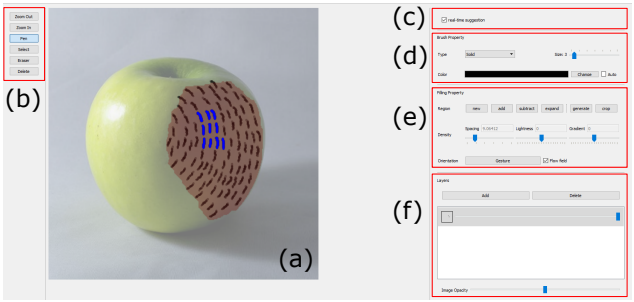
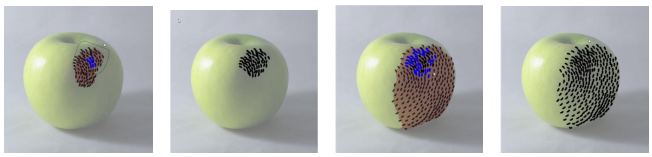


Figure 4: *User interface,* consisting of a central drawing canvas (a), a toolbar for drawing and selection (b), a toggle-switch of the autocomplete mode (c), a brush property toolbar (d), a filling property toolbar (e), and a layers panel (f).

Our prototype follows a standard digital drawing interface, with the added autocomplete feature, as shown in Figure 4. A user draws on top of a reference image displayed semi-transparently on the main canvas, while our system analyzes the input strokes and the reference image in the background.



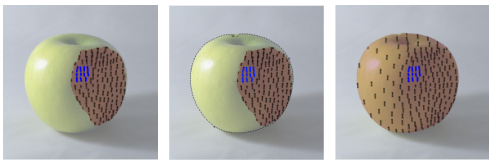
(a) selection (b) result (c) updated suggestions (d) final

Figure 5: *An example of autocompletion.* The user selects part of the suggestions via the lasso selection tool (a) with the result in (b), continues to draw leading to the updated suggestions (c), and accepts all the suggestions via a hotkey (d). The blue strokes in (a) and (c) indicate inferred exemplars from user-input strokes.

3.1 Autocomplete

In the autocomplete mode, our system automatically analyzes whenever the user finishes a new stroke. When a potential repetition is detected, our system highlights the currently repetitive strokes and an inferred propagation region, updates the inferred parameters in the filling property panel, and generates autocompletion suggestions. Users can accept or reject all the suggestions via hotkeys, accept part of them via lasso selection, or ignore them and continue to draw (Figure 5). The suggestions will keep updating according to user inputs.

3.2 Interactive Editing



(a) initial (b) new region (c) result

Figure 6: *Region editing example.* The initial prediction (a) contains only the brown region. The user-specified region (b) contains the entire apple, with the corresponding synthesis result in (c).

Our system provides a set of tools to refine the autocompleted results.

Propagation region editing. Users can create/add/subtract a new region using the intelligent scissors tool [32] or expand an existing region by a fixed width (Figure 4e) for stroke autocompletion. Figure 6 shows an example of creating a new region for stroke regeneration.

Density editing. Users can tweak three parameters to adjust the density of the generated strokes: the average *spacing*, the *lightness* coefficient and the *gradient* coefficient. The latter

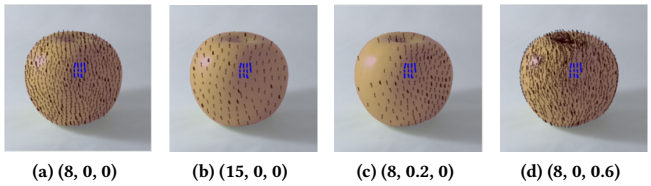


Figure 7: *Density editing example with different values of* spacing, lightness *and* gradient *parameters.* Larger spacing parameters lead to sparser strokes, while larger lightness and gradient parameters lead to larger stroke density variations.

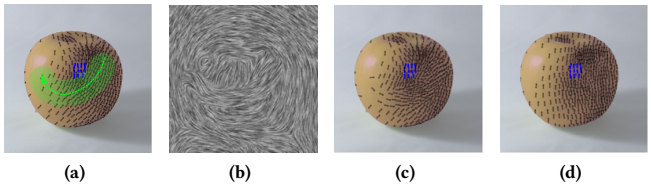


Figure 8: *Orientation editing example.* (a) User gesture. (b) Orientation field updated based on the user gesture and the original image flow field. (c) Updated result. (d) A result without any orientation field.

two define the relationships between density and image lightness/gradient, respectively. Our system automatically updates these parameters upon prediction, and the updated parameters provide a starting point for users to manipulate. Figure 7 shows an example.

Orientation editing. Our system automatically predicts whether the input exemplar correlates with the image flow, which can also be tweaked by users manually. Users can also modify the image flow field via the gesture brush, and the touched strokes will be rotated to align with the gesture direction. See Figure 8 for an example.

3.3 Auxiliary Functions

Our prototype also includes the auxiliary functions below. These are not unique to our system but can facilitate the usual drawing processes.

Post-edit stroke properties. Users can select the existing strokes and edit their properties, such as size and color.

Auto-color. This function, when toggled on, can automatically colorize strokes with color from the reference image.

Switch view. Users can press the space key to switch between the canvas view, reference view, and pure drawing view.

4 OUR APPROACH

To support the autocomplete functionalities described in Section 3, our system involves two key algorithm steps: (1) inferring the input exemplar, the output region, and the contextual constraints from the stroke history and the reference image; (2) synthesizing suggestive strokes accordingly. This section first describes how

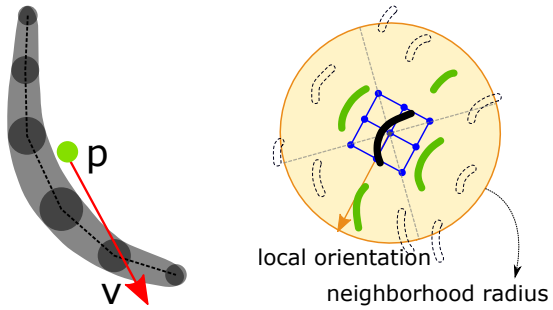


Figure 9: (a) A stroke, with centroid p and dominant direction v. (b) The neighborhood of the black stroke includes the n ($n = 1$ in this example) closest strokes (in green) from each quadrant and the middle image patch (blue pixel grid).

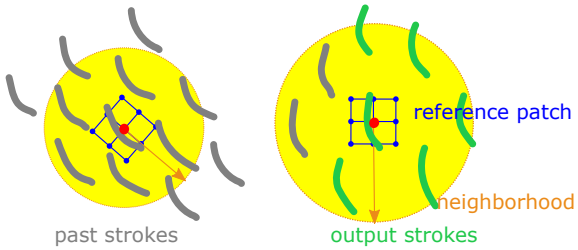


Figure 10: *Illustration of our synthesis algorithm.* We synthesize the predicted strokes (in green) from previously drawn strokes (in gray) by matching their neighborhoods.

to *synthesize* (Section 4.1) strokes, assuming all the information is available, and then explains how to *infer* (Section 4.2) the necessary information for synthesis.

4.1 Stroke Synthesis

Problem statement. The inputs to our stroke synthesis method include an exemplar E consisting of repetitive strokes, the reference image I, a target region mask M, an orientation map O, and a radius map R. Pixel values of R denote the extents of stroke spacing: a smaller value leads to a denser distribution. Our goal is to compute an output set of strokes X over the output region M, such that X is similar to E with respect to I. We describe how to infer E, M, O, and R from user interactions with I in Section 4.2.

Key idea. We extend the discrete element texture synthesis method [29, 48], which represents strokes as point samples and iteratively improves the sample distribution by minimizing the neighborhood difference between the exemplar and the output, with an additional reference image. First, we combine sample neighborhoods [29] with image features [13] for measuring neighborhood difference. Second, the range and orientation of sample neighborhoods are determined by the radius and orientation maps inferred from the reference image. Figure 10 shows our key idea.

Stroke representation. As shown in Figure 9a, a stroke s is an ordered list of sample points, each with a timestamp and appearance

attributes such as thickness and color. Here we focus on autocompleting short strokes, so we represent each stroke by its centroid p and the average direction v for efficiency during synthesis, without considering any other information of the original stroke. To take the drawing order into consideration, we obtain the dominant direction by averaging the vectors from the start point to each subsequent point. After synthesis, we reconstruct all the sample points according to the updated centroid and direction.

Initialization. We pre-process the target region mask M by removing the area occupied by existing strokes in the same layer to avoid cluttering, and then initialize the output X by generating sample positions with Poisson-disk sampling based on the radius map R. For each sampled position, we copy the input stroke with the smallest image feature distance d_I, which will be explained in Equation (2). We then optimize the output for a few objectives, as detailed below.

Neighborhood term. We define the neighborhood of a stroke s as both its neighboring strokes as well as an $R(s) \times R(s)$ image patch around its centroid, where $R(s)$ is the radius value at s. Prior methods (e.g. [29]) determine the neighboring strokes by spatial distances. Thus, the neighborhood radius should be large enough in order to capture an underlying pattern. However, this might include redundant strokes and thus decrease the performance. Therefore, we adopt Zhao et al.'s method [52] to automatically find a minimum representative neighborhood, considering not only the spatial distance between strokes but also their locations. As depicted in Figure 9b, we set the neighborhood radius of the center stroke s to $2R(s)$. We then divide all the strokes within the neighborhood radius into four quadrants with respect to the local frame defined by the orientation at $O(s)$, and collect the n nearest strokes from each quadrant as the representative neighborhood, denoted as $\mathbf{N}(s)$. In our implementation, we set $n = 4$ for the input exemplar and $n = 1$ for the output strokes to ensure that each output neighborhood can be maximally matched.

For a stroke s and a neighboring stroke $s' \in \mathbf{N}(s)$, we compute their difference in position and direction as:

$$\hat{u}(s', s) = \left(\frac{1}{R(s)} O(s)^{-1} \left(p(s') - p(s) \right), O(s)^{-1} \left(v(s') - v(s) \right) \right), \tag{1}$$

which is computed in the local frame defined by the radius map R and orientation map O. Therefore, the neighborhood distance between an output stroke s_o and an input stroke s_i is:

$$d_{neigh}(s_o, s_i) = \sum_{s'_o \in \mathbf{N}(s_o)} \left| \hat{u}(s'_o, s_o) - \hat{u}(s'_i, s_i) \right|^2 + \mu \underbrace{|I(s_o) - I(s_i)|^2}_{d_I}, \tag{2}$$

where s'_i is the matched input sample for s'_o via the Hungarian algorithm [28, 29], the second term measures the image feature distance d_I, and μ (= 0.1 in our implementation) controls the relative weighting. We use the mean $Lab*$ color of an $r \times r$ patch at the stroke centroid as the image feature vector. The overall neighborhood term to minimize is:

$$\phi_{neigh}(X, E) = \sum_{s_o \in X} \min_{s_i \in E} d_{neigh}(s_o, s_i). \tag{3}$$

Correction term. Since the neighborhood term is a one-way matching from the output neighborhoods to the input neighborhoods, sometimes the optimization would tend to leave out some void regions. Besides, the neighborhood term does not preserve strokes' alignment to the image (e.g., Figure 11e). To address these issues, we apply a correction term. We compute a weighted centroidal Voronoi diagram from all the strokes' center points, using $\frac{1}{R}$ as weight, and denote the computed region centroids as $\{\bar{p}\}$. Thus we can minimize the distance between each output stroke centroid and the region centroid, defined as follows:

$$\phi_{corr}(X) = \sum_{s_o \in X} |p(s_o) - \bar{p}(s_o)|^2 . \tag{4}$$

Solver. The energy function we aim to minimize is defined as:

$$\phi(X, E) = (1 - w)\phi_{neigh} + w\phi_{corr}. \tag{5}$$

We iteratively minimize the energy function following the EM methodology in [29]. In each iteration, for each output stroke s_o, we search for the most matched input stroke s_i to minimize ϕ_{neigh}, compute the Voronoi diagram centroid $\bar{p}$ to minimize ϕ_{corr}, and solve a least-squares system combining both terms. Let m be the total number of iterations. For the i−th iteration, we set $w = (i/m)^2$, which means that more weight is given to ϕ_{neigh} in the beginning of iterations, so that we can optimize the neighborhood distribution first before doing corrections, which leads to better results.

Figures 11b to 11d show the iterative optimization process of both the objectives. In comparison, Figure 11e shows the result without the correction term and Figure 11f shows the result without using the image neighborhood in both initialization and optimization.

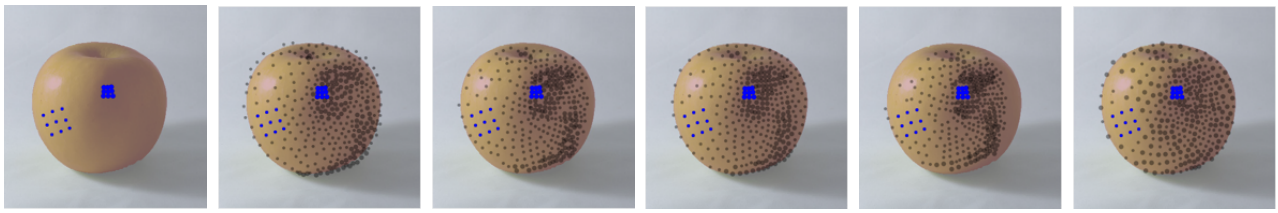
4.2 Inference

In this section, we describe how to infer E, M, O, and R used for our synthesis method in Section 4.1 from user interactions with I.

4.2.1 Input exemplar E. This step aims to detect whether stroke repetitions exist and obtain the repetitive group as an exemplar for the synthesis process. Since people usually draw strokes in a coherent manner [48] and they usually have specific intentions when drawing repetitive strokes, we assume the example strokes to be temporally consecutive and have certain similar properties.

We start from the last stroke input by the user and search backward in the stroke sequence to incrementally find strokes that have similar shape and image features to the last stroke. Specifically, the stroke shape similarity is measured with the Fréchet distance, and the image features include $Lab*$ color (weighted by 0.12, 0.44, and 0.44 to suppress the impact of lightness) and precomputed semantic segmentation [51] at a stroke's center. We compare the standard deviation of a feature in the traversed k strokes against a threshold (15/255 for the color feature, 1 for the segmentation feature) for similarity measurement. The back-traversal stops when the next stroke does not contain any similar feature or $k > 50$. These k strokes serve as the input exemplar for the synthesis process. See Figure 12 for an example of the incremental searching process.

4.2.2 Output region M. The shared features of the obtained stroke exemplar also indicate the intended region. For instance, if all of the exemplar strokes are inside the same object segmentation region, it is very likely that the user intends to fill that region. Therefore, we



(a) input (b) initialization (c) iteration 5 (d) iteration 15 (e) w/o ϕ_{corr} (f) w/o d_I

Figure 11: *Iteration process in (b) to (d) and ablation studies in (e) and (f).* Without the correction term ϕ_{corr} the predicted strokes tend to clutter together as in (e). Without the image term d_I the predicted strokes might not follow the reference sufficiently as in (f).

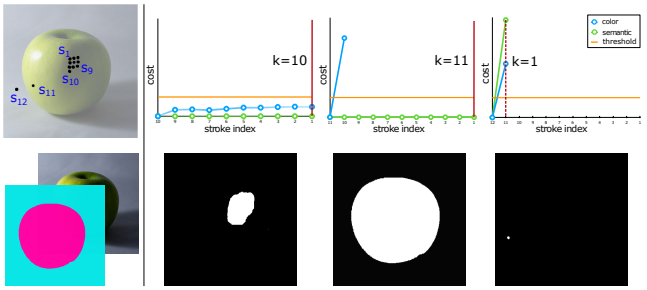


Figure 12: *An example of predicting the input exemplar and output region.* The left column shows the input stroke sequence visualized in black dots (only a few indices are shown for clarity) on the reference image (top) and the image features (bottom). The right columns show the threshold lines and the image feature cost curves for s_{10}, s_{11}, s_{12} respectively (top), and the corresponding predicted output regions (bottom). The cumulative number k is determined when both cost curves exceed the threshold. Note that the third region prediction result is only for demonstration: since the exemplar only contains one stroke (i.e., $k = 1$), it is not considered a valid exemplar and will not be used for synthesis.

use the shared features obtained in the exemplar grouping process to find a similar region for output.

Since there are only two features in our implementation, we simply obtain the region by GrabCut [36] if the *Lab∗* color feature is shared among the exemplar strokes, directly take the corresponding segmentation if the semantic feature is shared, and take the intersection if both features are shared. See Figure 12 for an example. When there are multiple disconnected regions, we retain the nearest region to the user's last stroke and discard the rest, because it is less natural to propagate to distant regions.

4.2.3 Contextual constraints. Since the drawing usually relates to the underlying reference image, we analyze the properties of both the drawn strokes and the reference image to infer possible relationships that control the global distribution of strokes.

Orientation O. Artists usually adjust the stroke directions to convey curvatures, but they may sometimes randomize or fix the stroke orientation regardless of the depicted objects to create different visual effects. Therefore, the problem is to decide which case the input exemplar implies. We first compute the edge tangent field (ETF) [24] for the reference image and then calculate the angles between the exemplar strokes and the ETF directions at their centroids. If the standard deviation of the angles is small (less than 15 degrees), we consider the stroke orientations to be related to the ETF and take the ETF as the orientation field; otherwise, we set a default global coordinate frame to each point of the orientation field.

Radius R. Since density is inversely proportional to the spacing between strokes, we reframe the problem as predicting a radius map that controls the extent of stroke neighborhoods. First, we compute the distance from each exemplar stroke to its nearest neighbor. We assume a linear relationship between these minimum distances r and the image features, including image lightness l and gradient strength g at a stroke's centroid, represented as:

$$r = \begin{pmatrix} l & g & 1 \end{pmatrix} \cdot \mathbf{t}, \tag{6}$$

where $\mathbf{t}$ denotes the coefficients to solve. With the fitted linear model, if the squared correlation value is lower than 0.5 (the closer to 1, the better explanation), we use the model to compute a radius map. Otherwise, we consider the density as uniform and create a constant radius map with the average spatial distance of the exemplar. We then update the UI with the computed coefficients.

5 EVALUATION

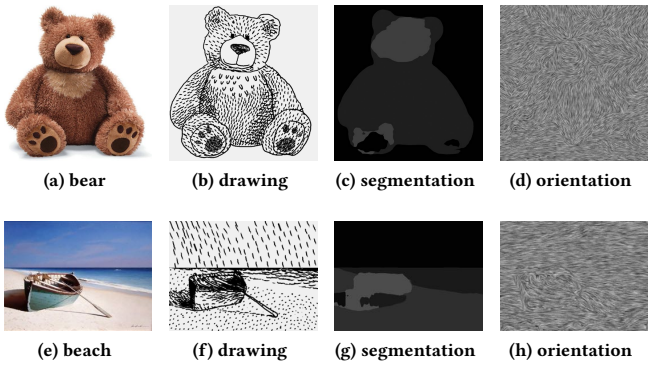
We conducted a pilot study to evaluate the utility and usability of our approach. We compared three modes through quantitative analysis and qualitative feedback.

Autocomplete Users have full access to our prototype, including autocomplete and interactive editing.

Interactive batch filling (aka *batch mode*) Users are required to create a texture example first and then manually specify the properties for batch filling. It simulates the sequential procedure in many IB-AR methods (e.g., [38]), although they rarely allow users to directly define examples on target images. This mode is performed on our system with the autocomplete function off.

Fully manual drawing (aka *manual mode*) Users have to manually draw each stroke without any automatic synthesis.

We also tested the expressiveness of our system through an open creation session and obtained comments for future improvements.



(a) bear (b) drawing (c) segmentation (d) orientation

(e) beach (f) drawing (g) segmentation (h) orientation

Figure 13: *Target session tasks*. Reference photos in (a) and (e), and the corresponding sample outputs in (b) and (f).

5.1 Target Session

The goal of this session is to compare the three interaction modes in utility and usability. Since we aim to facilitate image-scaffolded drawing, we hope to include general users from different background while focusing more on less skillful users, who are more likely to use reference images. We thus recruited 12 participants, including nine novices with little drawing experiences, two amateurs with some experiences (P3, P4), and a student majored in illustration (P5). Most of the studies were conducted on a Lenovo Miix 520 tablet with stylus in a lab environment, except two studies conducted remotely with mouse due to the pandemic.

The study procedure consisted of the following parts and took each participant about two hours in total.

Tutorial. Each participant was first given a brief introduction to our system and then asked to fill the apple in Figure 4 with short hatches as a warm-up task. They were encouraged to vary the density and orientation of input strokes and get familiar with the features of our system.

Target tasks. We used a within-subjects design, where each participant was asked to reproduce two target drawings (Figure 13) in all the three modes: autocomplete, interactive batch filling, and fully manual drawing. The target drawings include an object and a landscape, which are common illustration topics (e.g., Figure 2). The assigned order of modes was counter-balanced among all the participants. Since we focus on region filling, we asked the participants to draw the outlines of both images in advance, so that they could focus on drawing the textures during the study. We encouraged the participants to finish each drawing as soon as possible, preferably in a dozen of minutes, but without any hard time limit. After completing the two drawings in each mode, each participant filled in a NASA-TLX questionnaire [11]. At the end, we asked the participants about their preferred mode, usage experience and other comments.

5.2 Open session

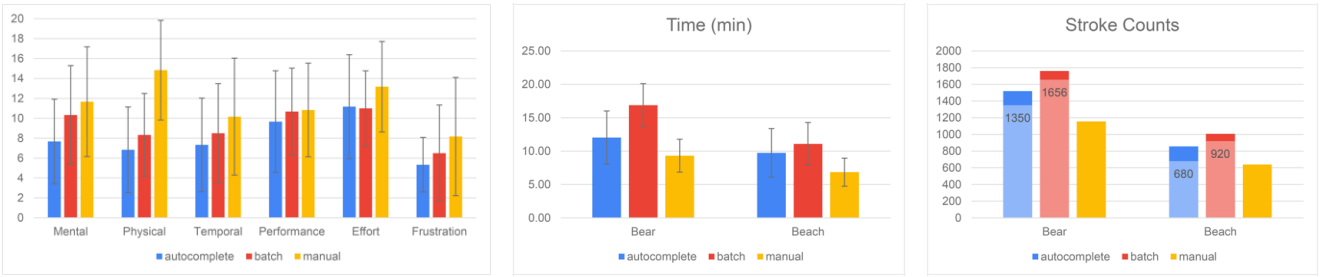
The goal of this session is to observe users' interaction with our system and learn about users' subjective experience. We invited seven participants (one professional artist, two amateurs and four novices) for this session. They were asked to create a drawing freely from the same reference image (Figure 15a) with our system. The reference image was a portrait photo, which is also common in illustrations. The only requirement was that the drawings should contain some repetitive content. We again gave a tutorial in the beginning and conducted the task on a Lenovo Miix 520 tablet with stylus. The participants were encouraged to think aloud and describe their thought process and interactions during this session. After this task, participants could optionally create more drawings with any images they want. Since our prototype does not contain all common functions in commercial drawing tools, we allow the participants to retouch the result drawings without adding more strokes in Photoshop.

5.3 Results and Observations

Workload. Figure 14a shows the perceived workload scores from the target session. Generally, the autocomplete mode received the lowest (i.e., best) scores for almost all the factors. One-way ANOVA showed the three modes have significant difference in physical demand ($F=10.69$, $p < 0.001$) while no significant difference in other factors. Regarding the physical demand, post-hoc pairwise tests showed that the autocomplete mode and batch mode were both rated significantly lower than manual mode, while had no significant difference from each other. This matches our expectation, since automatic synthesis should only reduce physical load and not cause extra pressure than manual work.

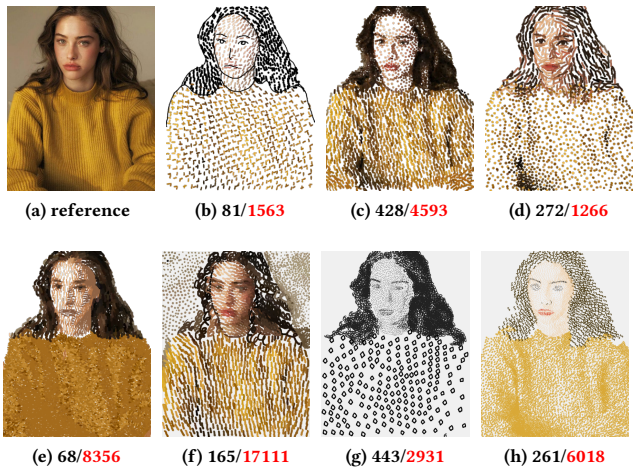
Efficiency. We calculate the average completion time (Figure 14b) and stroke count (Figure 14c) in each mode and each task. Generally, the system synthesized about 82% strokes in the autocomplete mode and about 92% strokes in the batch mode. Although the manual mode took the shortest time for the participants to complete, it also resulted in the fewest total number of strokes. We thus calculated the strokes per minute for each mode: autocomplete (111.03, SD=38.76), batch (101.98, SD=45.13), manual (115.95, SD=46.73). It turns out automatic generation did not improve the efficiency, probably because the users spent extra time adjusting and experimenting with the generated effects instead of just drawing strokes. It should be noted that such directed tasks omit the time for exploring alternative patterns, which, however, might be high in a fully manual case.

Quality. We asked 30 external volunteers to evaluate the quality of participants' drawings, as shown in Figure 19. We randomized all the drawings created by the participants, showed each output drawing alongside the target drawing, and asked volunteers to rate the resemblance of the output drawing to the target drawing, on a scale from 1 (very dissimilar) to 5 (very similar). The volunteers were instructed to focus more on the overall stroke distributions and flows instead of individual stroke thickness and detailed shapes. We calculated the average scores for each mode: autocomplete (3.10, SD=1.24), batch (3.09, SD=1.21), manual (2.98, SD=1.20). The quality of the drawings created with automatic synthesis is slightly better than the fully manual drawings, but without significant difference. From the participants' perspective, three novices commented the automated strokes were better than their manual strokes, because they tend to lose patience when manually drawing all strokes, which results in worse quality.



(a) NASA-TLX (b) time (c) # strokes

Figure 14: *Target sessions results.* (a) Average NASA-TLX scores from 12 participants. The lower the better. (b) Average completion time. (c) Average stroke counts. The number of system-generated strokes is labeled in each column.



(a) reference (b) 81/1563 (c) 428/4593 (d) 272/1266

(e) 68/8356 (f) 165/17111 (g) 443/2931 (h) 261/6018

Figure 15: *Example drawing results from the open session.* Each case is marked with the # of manual and autocompleted strokes.

Preferred Mode. Seven participants preferred the autocomplete mode while the rest five participants preferred the batch mode. Generally, the autocomplete mode is considered more convenient, yet less precise; the batch mode is considered more precise, but requires too many interactions. P12 commented, "*the autocomplete mode is more straightforward, because you can see the filled effects instantly without doing a lot of manipulation beforehand; while in the batch mode, you have to remember the meaning of parameters and tweak them in order to create strokes.*" P10 also said, "*Compared with batch filling, the autocomplete mode provides a quick guess of filled regions and allows me to get the results more quickly with less work.*" However, the autocomplete mode is "*less accurate at some vague and detailed regions, such as the shadows of the boat, where it tends to include some unwanted regions, so I have to manually subtract those regions, which is a bit tedious*", according to P3. The professional, P5, also preferred the batch mode for being able to precisely select the regions. Therefore, we consider the autocomplete function and the interactive editing function are complementary in usability.

Creation Results and Experience. Figure 15 shows the outcomes from the open session. Although from the same reference image and widely using repetitive short strokes, the study participants were able to create different results by varying the stroke shapes and arrangement. Figures 16 and 17 demonstrate some sample results. Regarding the creation experience, one user said "*it is playful, the final result is also good*"; two users described it as "*encouraging*", because the system allows beginners to quickly create stylistic drawings; one user commented that she "*felt creative when drawing with this system*", because she could test out patterns over image regions conveniently and she was more comfortable with drawing from a reference image than from scratch. The professional suggested that the tool itself was somewhat limited to pointillism and hatching styles, but can be helpful in adding interesting textures into color paintings (e.g., Figure 16i). Two users commented that the reduction of workload is useful, but they also complained about some inaccurate inference of autocompletion. We will discuss about this problem in Section 7.

6 CONCLUSION

We have presented a method to help users autocomplete repetitive short strokes with guidance from reference images while maintaining the flexible control of manual drawing. By extending operation history analysis and synthesis with image analysis, our method is able to generate results that adapt to reference images and users' prior inputs. We conducted a pilot study to validate the usefulness of our approach and show various drawing results from the users.

7 LIMITATIONS AND FUTURE WORK

From our observation and users' feedback, we identified several improvement opportunities.

Improve accuracy of autocompletion. We rely on simple *Lab∗* color and semantic segmentation for region inference. While color feature is sufficient for most cases, regions with similar colors but different semantics will require sufficient segmentation accuracy for region inference (Figures 13c and 13g). Since our segmentation map is precomputed, taking users' input as additional cues might help improve the segmentation accuracy (e.g., using interactive semantic segmentation methods like [50]).

Resolve visual blocking. Since the drawing and the system suggestions are overlaid on the reference image, it might be difficult for users to refer to the image when selecting parts of the suggestions (e.g., Figure 18) or adding a new layer of strokes. Although users can switch the views via a hotkey, it might be helpful to provide some reference information, like image darkness or boundaries, through additional visual hints [45, 47].

Figure 18: Example of visual blocking. Left: reference image. Right: canvas view.

Consider relationships with higher-level image features. We only consider the relationships between strokes and low-level image features, like colors and flows, over regions. By considering higher-level image features, such as elements and edges, it is possible to extend the scope of autocompletion, such as autocomplting the sparse flowers in the foreground of Figure 16i through the correspondences between strokes and elements.

Support more stroke types. Our method only supports short strokes, while artists also use long repetitive strokes frequently [5]. It is worth investigating the possibility of incorporating continuous strokes [44] in our analysis and synthesis framework and extending the support for different input strokes.

REFERENCES

[1] Adobe. 2017. Paint stylized strokes with the Art History Brush. **https://helpx.adobe.com/photoshop/using/painting-stylized-strokes-art-history.html**.

[2] Pascal Barla, Simon Breslav, Lee Markosian, and Joëlle Thollot. 2006. *Interactive Hatching and Stippling by Example.* Research Report RR-6461. INRIA. **https://hal.inria.fr/inria-00084569**

[3] Luca Benedetti, Holger Winnemöller, Massimiliano Corsini, and Roberto Scopigno. 2014. Painting with Bob: Assisted Creativity for Novices. In *UIST '14* (Honolulu, Hawaii, USA). ACM, New York, NY, USA, 419–428. **https://doi.org/10.1145/2642918.2647415**

[4] V. Alves dos Passos, M. Walter, and M. C. Sousa. 2010. Sample-Based Synthesis of Illustrative Patterns. In *2010 18th Pacific Conference on Computer Graphics and Applications.* 109–116. **https://doi.org/10.1109/PacificGraphics.2010.22**

[5] Alphonso Dunn. 2015. *Pen and Ink Drawing: A Simple Guide.* Three Minds Press.

[6] J. Fišer, P. Asente, and D. Sýkora. 2015. ShipShape: A Drawing Beautification Assistant. In *Proceedings of the Workshop on Sketch-Based Interfaces and Modeling* (Istanbul, Turkey) *(SBIM '15)*. Eurographics Association, Goslar Germany, Germany, 49–57. **http://dl.acm.org/citation.cfm?id=2810210.2810215**

[7] Jakub Fišer, Ondřej Jamriška, Michal Lukáč, Eli Shechtman, Paul Asente, Jingwan Lu, and Daniel Sýkora. 2016. StyLit: Illumination-guided Example-based Stylization of 3D Renderings. *ACM Trans. Graph.* 35, 4, Article 92 (July 2016), 11 pages. **https://doi.org/10.1145/2897824.2925948**

[8] L. A. Gatys, A. S. Ecker, M. Bethge, A. Hertzmann, and E. Shechtman. 2016. Controlling Perceptual Factors in Neural Style Transfer. *ArXiv e-prints* (Nov. 2016). arXiv:1611.07865 [cs.CV]

[9] Moritz Gerl and Tobias Isenberg. 2013. Interactive Example-based Hatching. *Comput. Graph.* 37, 1-2 (Feb. 2013), 65–80. **https://doi.org/10.1016/j.cag.2012.11.003**

[10] Paul Haeberli. 1990. Paint by Numbers: Abstract Image Representations. In *SIGGRAPH '90* (Dallas, TX, USA). 207–214. **https://doi.org/10.1145/97879.97902**

[11] Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Advances in psychology.* Vol. 52. Elsevier, 139–183.

[12] Siddharth Hegde, Christos Gatzidis, and Feng Tian. 2013. Painterly rendering techniques: a state-of-the-art review of current approaches. *Computer Animation and Virtual Worlds* 24, 1 (2013), 43–64.

[13] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. 2001. Image Analogies. In *SIGGRAPH '01.* 327–340. **https://doi.org/10.1145/383259.383295**

[14] Stefan Hiller, Heino Hellwig, and Oliver Deussen. 2003. Beyond stippling—Methods for distributing objects on the plane. In *Computer Graphics Forum*, Vol. 22. Wiley Online Library, 515–522.

[15] Chen-Yuan Hsu, Li-Yi Wei, Lihua You, and Jian Jun Zhang. 2020. Autocomplete Element Fields. In *CHI '20.* 1–13. **https://doi.org/10.1145/3313831.3376248**

[16] Emmanuel Iarussi, Adrien Bousseau, and Theophanis Tsandilas. 2013. The Drawing Assistant: Automated Drawing Guidance and Feedback from Photographs. In *UIST '13* (St. Andrews, Scotland, United Kingdom). 183–192. **https://doi.org/10.1145/2501988.2501997**

[17] Takashi Ijiri, Radomír Měch, Takeo Igarashi, and Gavin Miller. 2008. An Example-based Procedural System for Element Arrangement. *Computer Graphics Forum* 27, 2 (2008), 429–436. **https://doi.org/10.1111/j.1467-8659.2008.01140.x**

[18] Jennifer Jacobs, Sumit Gogia, Radomír Mundefinedch, and Joel R. Brandt. 2017. Supporting Expressive Procedural Art Creation through Direct Manipulation. In *CHI '17* (Denver, Colorado, USA). Association for Computing Machinery, New York, NY, USA, 6330–6341. **https://doi.org/10.1145/3025453.3025927**

[19] Evangelos Kalogerakis, Derek Nowrouzezahrai, Simon Breslav, and Aaron Hertzmann. 2012. Learning Hatching for Pen-and-ink Illustration of Surfaces. *ACM Trans. Graph.* 31, 1, Article 1 (Feb. 2012), 17 pages. **https://doi.org/10.1145/2077341.2077342**

[20] Hyung W. Kang, Wenjie He, Charles K. Chui, and Uday K. Chakraborty. 2005. Interactive sketch generation. *The Visual Computer* 21, 8 (01 Sep 2005), 821–830. **https://doi.org/10.1007/s00371-005-0328-9**

[21] Alexandre Kaspar, Boris Neubert, Dani Lischinski, Mark Pauly, and Johannes Kopf. 2015. Self Tuning Texture Optimization. *Computer Graphics Forum* (2015). **https://doi.org/10.1111/cgf.12565**

[22] Rubaiat Habib Kazi, Takeo Igarashi, Shengdong Zhao, and Richard Davis. 2012. Vignette: Interactive Texture Design and Manipulation with Freeform Gestures for Pen-and-ink Illustration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. 1727–1736. **https://doi.org/10.1145/2207676.2208302**

[23] Jan Eric Kyprianidis, John Collomosse, Tinghuai Wang, and Tobias Isenberg. 2013. State of the "Art": A Taxonomy of Artistic Stylization Techniques for Images and Video. *IEEE Transactions on Visualization and Computer Graphics* 19, 5 (2013), 866–885. **https://doi.org/10.1109/TVCG.2012.160**

[24] Jan Eric Kyprianidis and Henry Kang. 2011. Image and Video Abstraction by Coherence-Enhancing Filtering. *Computer Graphics Forum* 30, 2 (2011), 593––602. **https://doi.org/10.1111/j.1467-8659.2011.01882.x** Proceedings Eurographics 2011.

[25] G. Li, S. Bi, J. Wang, Y. Xu, and Y. Yu. 2017. ColorSketch: A Drawing Assistant for Generating Color Sketches from Photos. *IEEE Computer Graphics and Applications* 37, 3 (May 2017), 70–81. **https://doi.org/10.1109/MCG.2016.37**

[26] Yijun Li, Chen Fang, Aaron Hertzmann, Eli Shechtman, and Ming-Hsuan Yang. 2019. Im2pencil: Controllable pencil illustration from photographs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 1525–1534.

[27] Cewu Lu, Li Xu, and Jiaya Jia. 2012. Combining Sketch and Tone for Pencil Drawing Production. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering* (Annecy, France) *(NPAR '12)*. Eurographics Association, Goslar Germany, Germany, 65–73. **http://dl.acm.org/citation.cfm?id=2330147.2330161**

[28] Chongyang Ma, Li-Yi Wei, Sylvain Lefebvre, and Xin Tong. 2013. Dynamic Element Textures. *ACM Trans. Graph.* 32, 4, Article 90 (July 2013), 10 pages. **https://doi.org/10.1145/2461912.2461921**

[29] Chongyang Ma, Li-Yi Wei, and Xin Tong. 2011. Discrete Element Textures. *ACM Trans. Graph.* 30, 4, Article 62 (July 2011), 10 pages. **https://doi.org/10.1145/2010324.1964957**

[30] Domingo Martín, Germán Arroyo, Alejandro Rodríguez, and Tobias Isenberg. 2017. A Survey of Digital Stippling. *Computers & Graphics* 67 (Oct. 2017), 24–44. **https://doi.org/10.1016/j.cag.2017.05.001**

[31] Yusuke Matsui, Takaaki Shiratori, and Kiyoharu Aizawa. 2017. DrawFromDrawings: 2D Drawing Assistance via Stroke Interpolation with a Sketch Database. *IEEE transactions on visualization and computer graphics* 23, 7 (2017), 1852–1862.

[32] Eric N. Mortensen and William A. Barrett. 1995. Intelligent Scissors for Image Composition. In *SIGGRAPH '95.* 191–198. **https://doi.org/10.1145/218380.218442**

[33] Mathieu Nancel and Andy Cockburn. 2014. Causality: A Conceptual Model of Interaction History. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. ACM,

New York, NY, USA, 1777–1786. https://doi.org/10.1145/2556288.2556990

[34] Mengqi Peng, Li-Yi Wei, Rubaiat Habib Kazi, and Vladimir G. Kim. 2020. Autocomplete Animated Sculpting. In *UIST '20*. https://doi.org/10.1145/3379337.3415884

[35] Mengqi Peng, Jun Xing, and Li-Yi Wei. 2018. Autocomplete 3D Sculpting. *ACM Trans. Graph.* 37, 4, Article 132 (Aug. 2018).

[36] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. 2004. "GrabCut": Interactive Foreground Extraction Using Iterated Graph Cuts. In *ACM SIGGRAPH 2004 Papers* (Los Angeles, California) *(SIGGRAPH '04)*. Association for Computing Machinery, New York, NY, USA, 309–314. https://doi.org/10.1145/1186562.1015720

[37] Michael P. Salisbury, Sean E. Anderson, Ronen Barzel, and David H. Salesin. 1994. Interactive Pen-and-ink Illustration. In *SIGGRAPH '94*. 101–108. https://doi.org/10.1145/192161.192185

[38] Michael P. Salisbury, Michael T. Wong, John F. Hughes, and David H. Salesin. 1997. Orientable Textures for Image-based Pen-and-ink Illustration. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 401–406. https://doi.org/10.1145/258734.258890

[39] Martin Schwarz, Tobias Isenberg, Katherine Mason, and Sheelagh Carpendale. 2007. Modeling with Rendering Primitives: An Interactive Non-photorealistic Canvas. In *Proceedings of the 5th International Symposium on Non-photorealistic Animation and Rendering* (San Diego, California) *(NPAR '07)*. ACM, New York, NY, USA, 15–22. https://doi.org/10.1145/1274871.1274874

[40] B. Shneiderman. 1987. Human-computer Interaction. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, Chapter Direct Manipulation: A Step Beyond Programming Languages, 461–467. http://dl.acm.org/citation.cfm?id=58076.58115

[41] Qingkun Su, Wing Ho Andy Li, Jue Wang, and Hongbo Fu. 2014. EZ-sketching: Three-level Optimization for Error-tolerant Image Tracing. *ACM Trans. Graph.* 33, 4, Article 54 (July 2014), 9 pages. https://doi.org/10.1145/2601097.2601202

[42] R. Suzuki, K. Yatani, M. D. Gross, and T. Yeh. 2018. Tabby: Explorable Design for 3D Printing Textures. In *PG '18 Short Papers*. Eurographics Association, Goslar, DEU, 29–32. https://doi.org/10.2312/pg.20181273

[43] Hui-Chi Tsai, Ya-Hsuan Lee, Ruen-Rone Lee, and Hung-Kuo Chu. 2017. User-guided line abstraction using coherence and structure analysis. *Computational Visual Media* 3, 2 (2017), 177–188.

[44] Peihan Tu, Li-Yi Wei, Koji Yatani, Takeo Igarashi, and Matthias Zwicker. 2020. Continuous Curve Textures. *ACM Trans. Graph.* 39, 6, Article 168 (Nov. 2020), 16 pages. https://doi.org/10.1145/3414685.3417780

[45] Blake Williford, Abhay Doke, Michel Pahud, Ken Hinckley, and Tracy Hammond. 2019. DrawMyPhoto: Assisting Novices in Drawing from Photographs. In *Proceedings of the 2019 on Creativity and Cognition* (San Diego, CA, USA) *(C&C '19)*. 198–209. https://doi.org/10.1145/3325480.3325507

[46] Georges Winkenbach and David H. Salesin. 1994. Computer-generated Pen-and-ink Illustration. In *SIGGRAPH '94*. 91–100. https://doi.org/10.1145/192161.192184

[47] Jun Xie, Aaron Hertzmann, Wilmot Li, and Holger Winnemöller. 2014. PortraitSketch: Face Sketching Assistance for Novices. In *UIST '14* (Honolulu, Hawaii, USA). 407–417. https://doi.org/10.1145/2642918.2647399

[48] Jun Xing, Hsiang-Ting Chen, and Li-Yi Wei. 2014. Autocomplete Painting Repetitions. *ACM Trans. Graph.* 33, 6, Article 172 (Nov. 2014), 11 pages. https://doi.org/10.1145/2661229.2661247

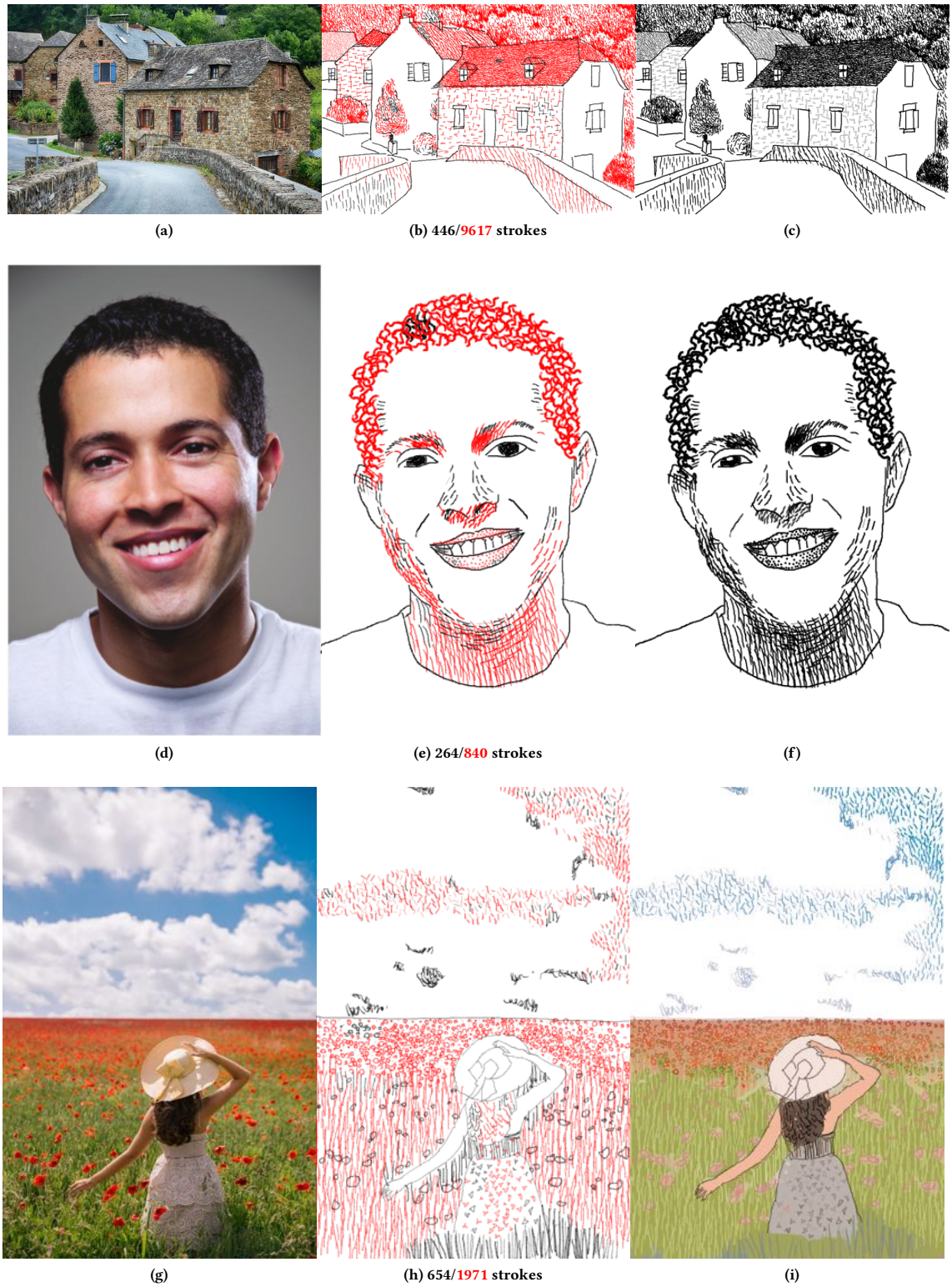
[49] Jun Xing, Li-Yi Wei, Takaaki Shiratori, and Koji Yatani. 2015. Autocomplete Hand-drawn Animations. *ACM Trans. Graph.* 34, 6, Article 169 (Oct. 2015), 11 pages. https://doi.org/10.1145/2816795.2818079

[50] Ning Xu, Brian Price, Scott Cohen, Jimei Yang, and Thomas Huang. 2017. Deep grabcut for object selection. *arXiv preprint arXiv:1707.00243* (2017).

[51] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. 2017. Pyramid Scene Parsing Network. In *CVPR*.

[52] Mingtian Zhao and Song-Chun Zhu. 2011. Customizing Painterly Rendering Styles Using Stroke Processes. In *NPAR '11* (Vancouver, British Columbia, Canada). 137–146. https://doi.org/10.1145/2024676.2024698

[53] C. Lawrence Zitnick. 2013. Handwriting Beautification Using Token Means. *ACM Trans. Graph.* 32, 4, Article 53 (July 2013), 8 pages. https://doi.org/10.1145/2461912.2461985



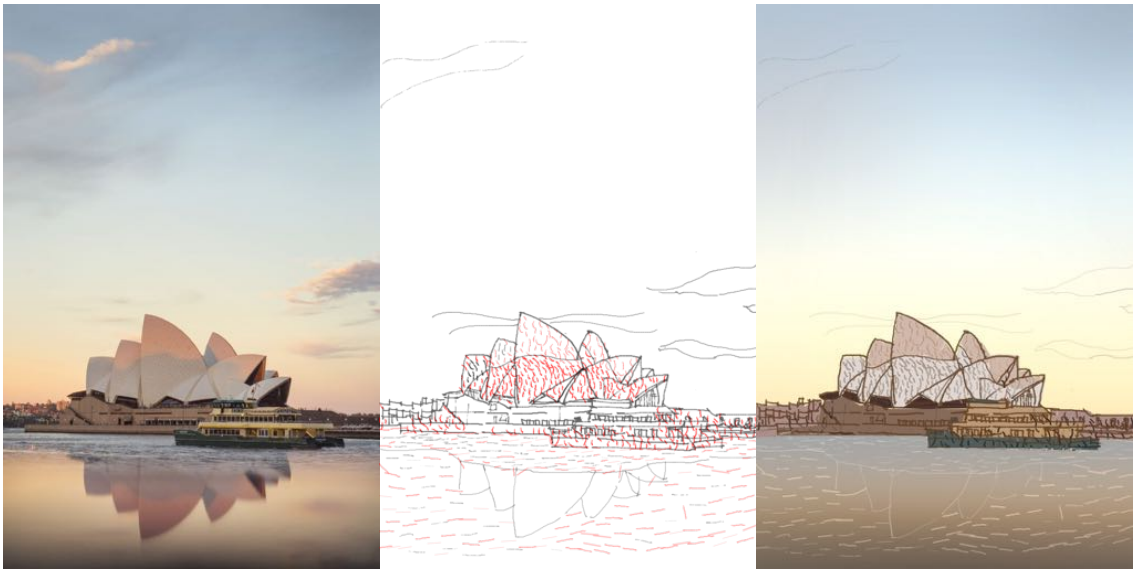
(a) (b) 446/9617 strokes (c)

(d) (e) 264/840 strokes (f)

(g) (h) 654/1971 strokes (i)

Figure 16: *Sample results.* In each example, the left column shows the reference images, the middle column visualizes the manual strokes (black) and autocompleted results (red) of the final drawings on the right column. In the last example, the strokes are created with our system first and then imported into Photoshop for background coloring.

(a) (b) 151/1590 strokes (c)



(d) (e) 470/551 strokes (f)

(g) (h) 39/1250 strokes (i)

(j) (k) 88/604 strokes (l)

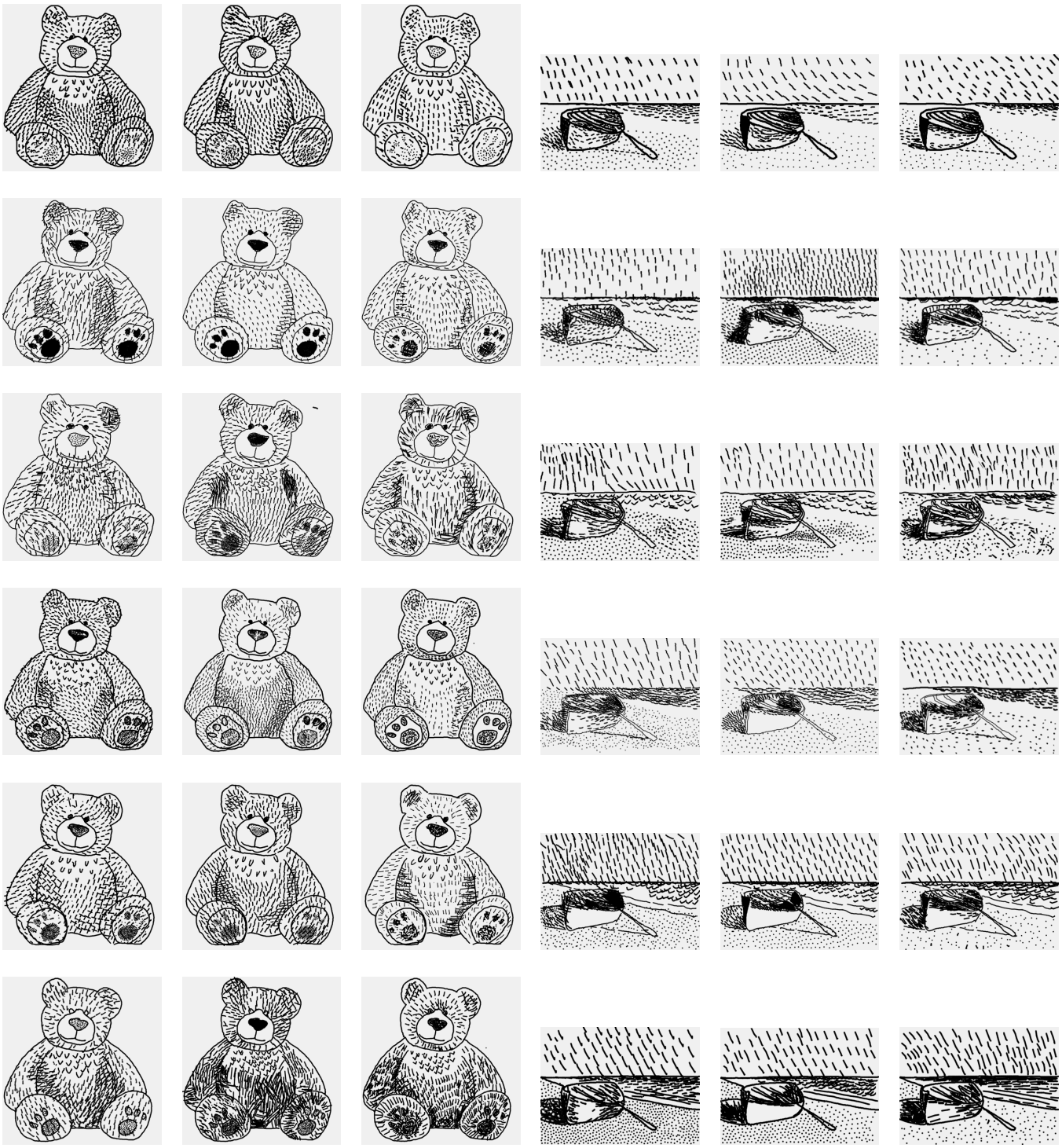
(m) (n) 163/939 strokes (o)

(p) (q) 322/2832 strokes (r)

(s) (t) 134/1039 strokes (u)

Figure 17: ***Additional results for Figure 16.***

Figure 19: ***Participants' drawings for the target session.***



Autocomplete Repetitive Stroking with Image Guidance