Computational Visual Media

https://doi.org/10.1007/s41095-022-0288-2 Vol. 9, No. x, month 2022, xx–xx

Research Article

Autocompletion of repetitive stroking with image guidance

Yilan Chen[1], **Kin Chung Kwan**[2], **and Hongbo Fu**[1](✉)

Abstract Image-guided drawing can compensate for a lack of skill 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 an assisted drawing system to autocomplete repetitive short strokes during a users' normal drawing process. Users 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 the users' input style when certain repetition is detected. Users can accept, modify, or ignore the system's predictions and continue drawing, thus maintaining fluid control over drawing. Our key idea is to jointly analyze image regions and user input history to detect and predict repetition. The proposed system can effectively reduce the user's workload when drawing repetitive short strokes, helping users to create results with rich patterns.

Keywords Interaction; Autocompletion; Digital Drawing; Prediction; Texture Synthesis

1 Introduction

Drawing is a common form of artistic expression. By varying the strokes, texture, and shading, artists can create drawings in various styles [1]. However, it remains a largely manual process that may require significant artistic expertise and repetitive manual labor.

To reduce repetitive workload, various methods have been proposed to automatically synthesize strokes from user-provided examples [4–6] or through procedural steps [7]. However, these methods usually perform in batches, reducing user participation in the creative artistic process. Furthermore, since many methods have predefined styles and only allow users to modify a few global parameters, the final results may look monotonous and lack originality (see Figure 3). Other interactive systems [8, 9] preserve the normal drawing flow while automating significant stroke synthesis. We share a similar goal to theirs. However, they typically target experts, requiring artistic expertise for high-level picture composition and fine-grained control.

One common way to overcome this skill barrier is to use a reference photo as a scaffold for drawing, by tracing a reference photo physically using transparent paper, or digitally via layers in digital drawing applications. Prior research [10] shows that even when a reference image is used as a scaffold, people still enjoy the freedom of individual expression. We thus propose to enhance drawing using an image scaffold 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 effort but provides limited user control and interactivity. As the first attempt towards this goal, we focus on autocompleting repetitive *short* strokes, which are very common in pen-and-ink drawing (see Figure 2), under the guidance of a reference image. As in 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 fluid control of drawing. See Figure 1 for an example.

The major contribution of this paper is the technical design of an image-guided autocompletion drawing tool that can preserve the natural drawing process and individual user styles. Our approach is inspired by image analogy [4] and operation history analysis and synthesis [9] while leveraging two key insights. Firstly, since the act of drawing repetitive

1 School of Creative Media, City University of Hong Kong. Tat Chee Avenue, Kowloon, Hong Kong SAR, China. E-mail: Yilan Chen, yil.ellen.chan@gmail.com; Hongbo Fu, hongbofu@cityu.edu.hk.

2 University of Konstanz. University Street 10, 78464 Konstanz, Germany. E-mail: Kin Chung Kwan, kckwan@ieee.org.

Manuscript received: 2022-01-01; accepted: 2022-01-01

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

Fig. 1 Example of our system workflow. (a) A user stipples over a leaf region of a reference image 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), (c) which is then accepted by the user (green strokes: user inputs or accepted suggestions). (d) shows the manually drawn content (black, 261 strokes) and autocompleted content (red, 3510 strokes). (e) shows the final result; note the different repetitive stroke patterns in different regions. Our autocompletion system can reduce tedious repetitive input, while providing full user control.

strokes usually indicates specific intentions (e.g. filling an object or hatching a shaded region), we use common image features shared by the coherent repetitive strokes to infer the intended region. Secondly, the drawing is usually related to the underlying reference image (e.g. the density of strokes depends on image brightness). Therefore, we analyze the properties of both the drawing and the reference image to infer possible relationships as contextual constraints for stroke prediction.

We have implemented a prototype and conducted a pilot study with participants from different backgrounds to evaluate its utility and usability. The quantitative analysis and qualitative feedback, as well as various drawing results created by the users, suggest that our system effectively reduces a user's workload when drawing repetitive short strokes, helping users to create 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 [10–13], or providing educational guidance to novice users [14–16]. We share a similar goal to [17–19] of reducing the user's workload. However, these works use predefined algorithms to generate strokes guided by cursor movement and only take the user's input as an indicator of where to render, thus greatly limiting the user's artistic freedom. In contrast, we aim to provide more flexibility between automatic synthesis and manual artistic control by autocompleting tedious repetitions during the user's normal drawing processes.

2.2 Image-based Artistic Rendering

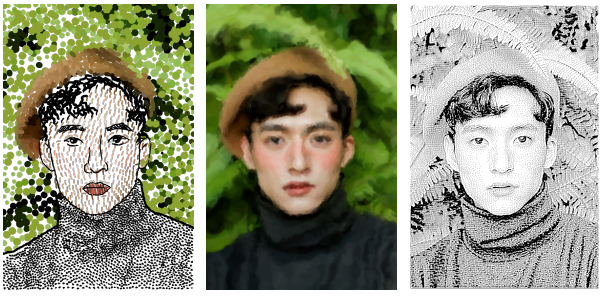
Our work is related to image-based artistic rendering (IB-AR) [20], especially stroke-based methods and example-based methods.

Stroke-based methods create artistic results from images by strategically generating brushstrokes whose properties (e.g. position, density, orientation, color, size) are related to image properties (e.g. gradient, edges, color, salience) [7]. Among those methods, the closest to ours are the early image-based pen-and-ink rendering methods [21, 22], 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 adjust parameters to produce the rendered output. In contrast, our system lets users directly provide exemplars on a reference image while silently inferring the distribution properties.

Example-based methods aim to model the visual features of example images for transferal. There are two major modeling approaches: the parametric approach [6, 23, 24] that is based on statistical analysis of stroke characteristics, thus preserving global textures better, and the non-parametric approach [4, 5, 25] based on patch-wise mapping, thus capturing local structures better. We combine both methods to generate strokes: the parametric approach is used to infer statistical relationships between stroke properties and image features, and the patch-wise matching method is used to preserve the local arrangement of strokes. Stylit [5] allows users to stylize a rendered ball and simultaneously propagates the

(a) By Alphonso Dunn **(b)** By Vincent van Gogh

Fig. 2 Inspiring manual drawings by artists.



(a) our result **(b)** produced with [2] **(c)** produced with [3]

Fig. 3 Style comparison. (a) our work is designed to reduce the workload of completing repetitive patterns during the manual drawing process. Full control over the drawing process leads to more dynamic results than (b) Photoshop's art history brush tool [2] and (c) StippleShop [3].

style to arbitrary 3D shapes. Our method shares a similar idea of interactive style propagation, but with two main differences. Firstly, instead of propagating a style globally, we propagate a style to perceptually similar local areas so that users can conveniently define different styles in different areas. Secondly, we represent drawings as discrete stroke operations instead of raster textures to better preserve their structure, e.g. changing the color or size of the drawn strokes.

2.3 Operation History-assisted Authoring

Operation histories [26] have been leveraged for various authoring tasks, such as sketching [9], animation [27, 28], modeling [29, 30], beautification of freehand drawings [31], and handwriting [32]. Our work is most closely related to that of Xing et al. [9], which autocompletes repetitive sketching by analyzing 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 case, an operation is an input stroke, so our work is also related to stroke pattern analysis and synthesis [8, 33–36]. These works disregard the temporal relationships between past strokes, and do not use image guidance, so are different from ours.

We summarise the major differences between our work and the discussed closely related work in Table 1.

Table 1 Differences between our tool and closely related work. *B* means generation is performed in batches, based on predefined attributes, while *Dy* means generation is based on dynamic operation history. *S* indicates operation on strokes, *P* on pixels. *Di* means users can specify a style by directly operating on the output, *I* indicates indirect interaction *Y* and *N* represent yes and no, respectively, for using image references.

Method	[22]	[4]	[6]	[8]	[9]	Ours
Reference	Y	Y	Y	N	N	Y
Process	B	B	B	B	Dy	Dy
Format	S	P	S	S	S	S
Operate	I	I	I	Di	Di	Di

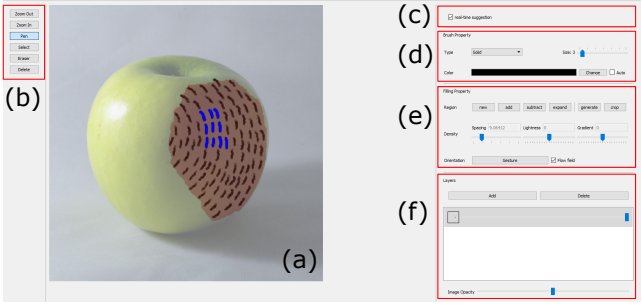


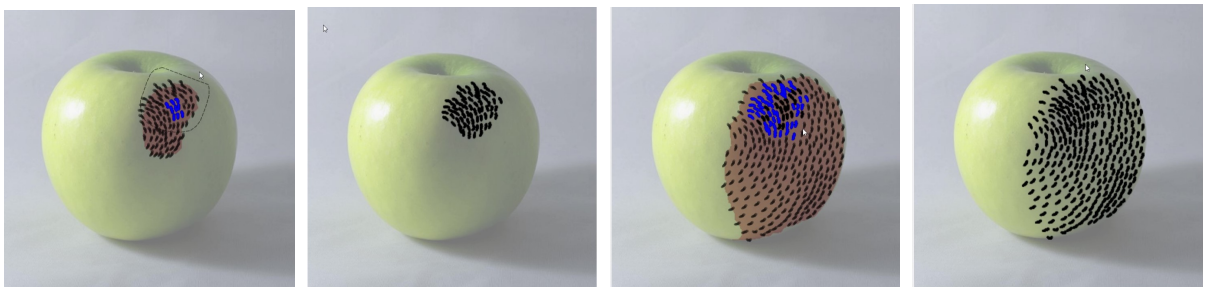
Fig. 4 User interface, comprising (a) a central drawing canvas, (b) a toolbar for drawing and selection, (c) a toggle-switch for autocompletion mode, (d) a brush property toolbar, (e) a filling property toolbar, and (f) a layer panel.

3 User Interface

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

3.1 Autocompletion

In autocompletion mode, our system automatically performs analysis whenever the user finishes a new stroke. When a potential repetition is detected, our system highlights the current repetitive strokes and an inferred propagation region, updates the inferred parameters in the filling property panel, and generates an autocompletion suggestion. Users can accept or reject the suggestion using hotkeys, accept part of it using lasso selection, or ignore it and continue to draw (Figure 5). The suggestion will continuously update according to user input.



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

Fig. 5 An example of autocompletion. The user selects part of the suggestion using the lasso tool (a) with the result shown in (b), then continues to draw leading to the updated suggestion (c), and accepts all the suggestions using a hotkey (d). The blue strokes in (a) and (c) indicate inferred exemplars from the user's input strokes.

3.2 Interactive Editing

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

Propagation region editing. Users can create, add, or subtract a region using the intelligent scissors tool [37], or expand an existing region by a fixed width (see Figure 4(e)) for stroke autocompletion. Figure 6 shows an example of creating a new region for stroke regeneration.

Density editing. Users can modify three parameters to adjust the density of the generated strokes: the average *spacing*, the *lightness* coefficient and the *gradient* coefficient. The latter two define the relationships between density and image lightness and gradient, respectively. Our system automatically updates these parameters upon prediction, and the updated parameters provide a starting point for user manipulation. Figure 7 shows an example.

Orientation editing. Our system automatically predicts whether the input exemplar is correlated with the image flow; orientation can also be adjusted by the user manually. The user can also modify the image flow field using the gesture brush, and the touched strokes will be rotated to be aligned with the gesture. 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-editing stroke properties. Users can select existing strokes and edit their properties, such as size and color.

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

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

4 Approach

Our system involves two key steps: (i) inferring the input exemplar, the output region, and the contextual constraints from the stroke history and the reference image, and (ii) synthesizing suggestive strokes accordingly. Section 4.1 first describes how to *synthesize* strokes, assuming all the information is available, and then Section 4.2 explains how to *infer* the necessary information for synthesis.

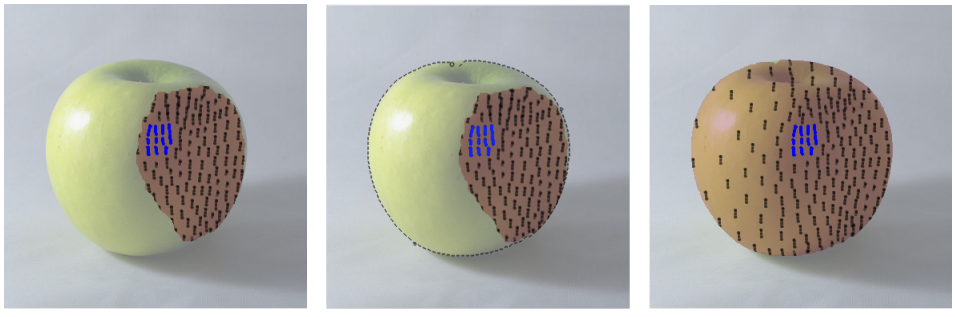
4.1 Stroke Synthesis

4.1.1 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 determine the 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 interaction with I in Section 4.2.

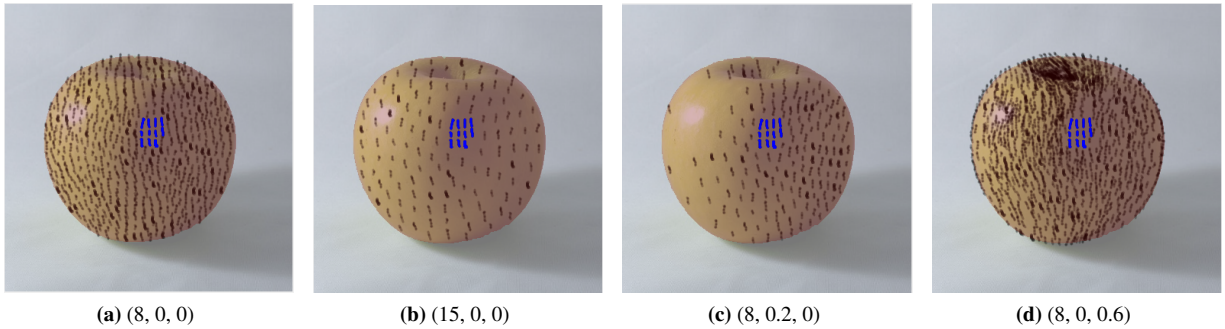
4.1.2 Idea

To support autocompletion using the reference image, we extend the discrete element texture synthesis method [9, 38], which represents strokes as point samples and iteratively improves the sample distribution by minimizing the neighborhood difference between the exemplar and the output, using an additional reference image. Firstly, we combine sample neighborhoods [38] with image features [4] to measure neighborhood differences. Secondly, the range and orientation of each sample neighborhood is determined by the radius and orientation maps inferred from the reference image. Figure 9 shows our key idea.



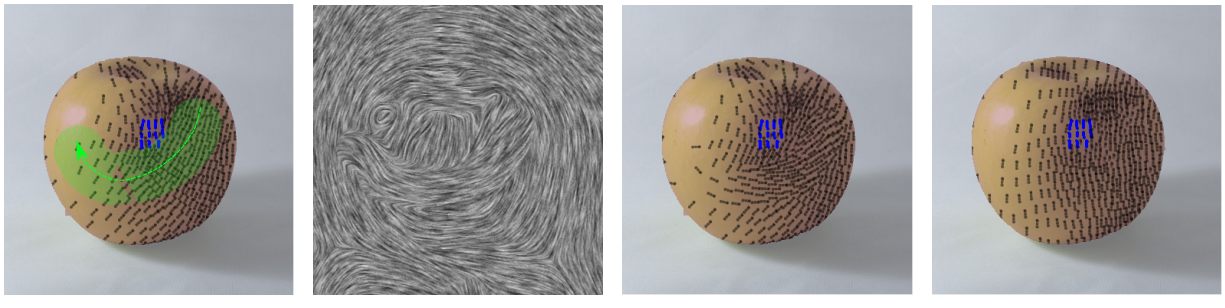
(a) initial **(b)** new region **(c)** result

Fig. 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).



(a) (8, 0, 0) **(b)** (15, 0, 0) **(c)** (8, 0.2, 0) **(d)** (8, 0, 0.6)

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



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

Fig. 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) Result without any orientation field.

4.1.3 Stroke representation

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 (see Figure 10) for efficiency of synthesis, without considering any other information about the original stroke. To take 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 sample points according to the updated centroid and direction.

4.1.4 Initialization

We pre-process the target region mask M by removing the area occupied by existing strokes in the same layer to avoid clutter, 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 several objectives, as detailed below.

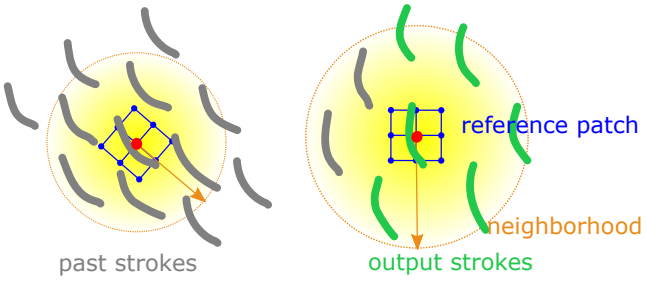


Fig. 9 Synthesis algorithm. We synthesize the predicted strokes (green) from previously drawn strokes (gray) by matching their neighborhoods as well as image features.

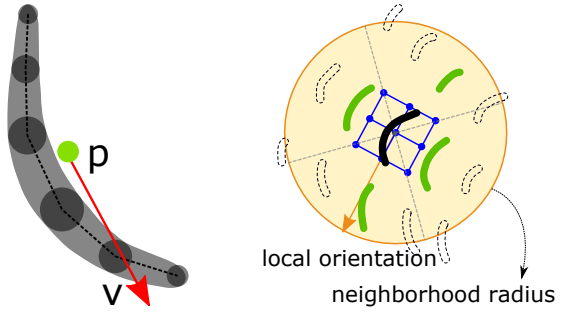


Fig. 10 Left: A stroke, with centroid p and dominant direction v. Right: The neighborhood of the black stroke includes the n ($n = 1$ in this example) closest strokes (green) from each quadrant and the middle image patch (blue pixel grid).

4.1.5 *Neighborhood term*

We define the neighborhood of a stroke s by 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. [38]) determine the neighboring strokes by spatial distance. Thus, the neighborhood radius should be large enough to capture any underlying pattern. However, this might include redundant strokes and thus decrease performance. Therefore, we adopt Zhao et al.'s method [39] to automatically find a minimal representative neighborhood, considering not only the distances between strokes but also their locations. As Figure 10b shows, we set the neighborhood radius of the center stroke s to $2R(s)$. We then divide all 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 $\mathbf{N}(s)$. 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 offsets in position and direction to be:

$$\hat{u}(s', s) = \begin{array}{l} \left(O(s)^{-1}\left(p(s') - p(s)\right)/R(s)\right., \\ \left. O(s)^{-1}\left(v(s') - v(s)\right)\right), \end{array} \quad (1)$$

where $O(s)^{-1}$ indicates rotating the vector inversely to $O(s)$. Note that the position and direction difference is computed in the local frame defined by the density map and orientation map. For an output stroke s_o and an input stroke s_i, we first find their best matching pairs $\{(s'_o, s'_i)\}$ in the neighborhoods $\mathbf{N}(s_o)$ and $\mathbf{N}(s_i)$ using the Hungarian algorithm [38, 40]. We use the norm-2 distance of the offsets from s_o or s_i in Equation (1) as the matching cost. The neighborhood distance is then defined as:

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

The second term measures the image feature distance d_I; μ ($= 0.1$ in our implementation) controls its relative weight. 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:

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

4.1.6 *Correction term*

Since the neighborhood term is a one-way matching from output neighborhoods to input neighborhoods, sometimes optimization may tend to leave out some void regions. Furthermore, the neighborhood term does not preserve the alignment of strokes to the image (e.g. see 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 $1/R$ as weight; we denote the computed region centroids as $\{\bar{p}\}$. Then we can minimize the total distance between each output stroke centroid and the region centroid, defined as follows:

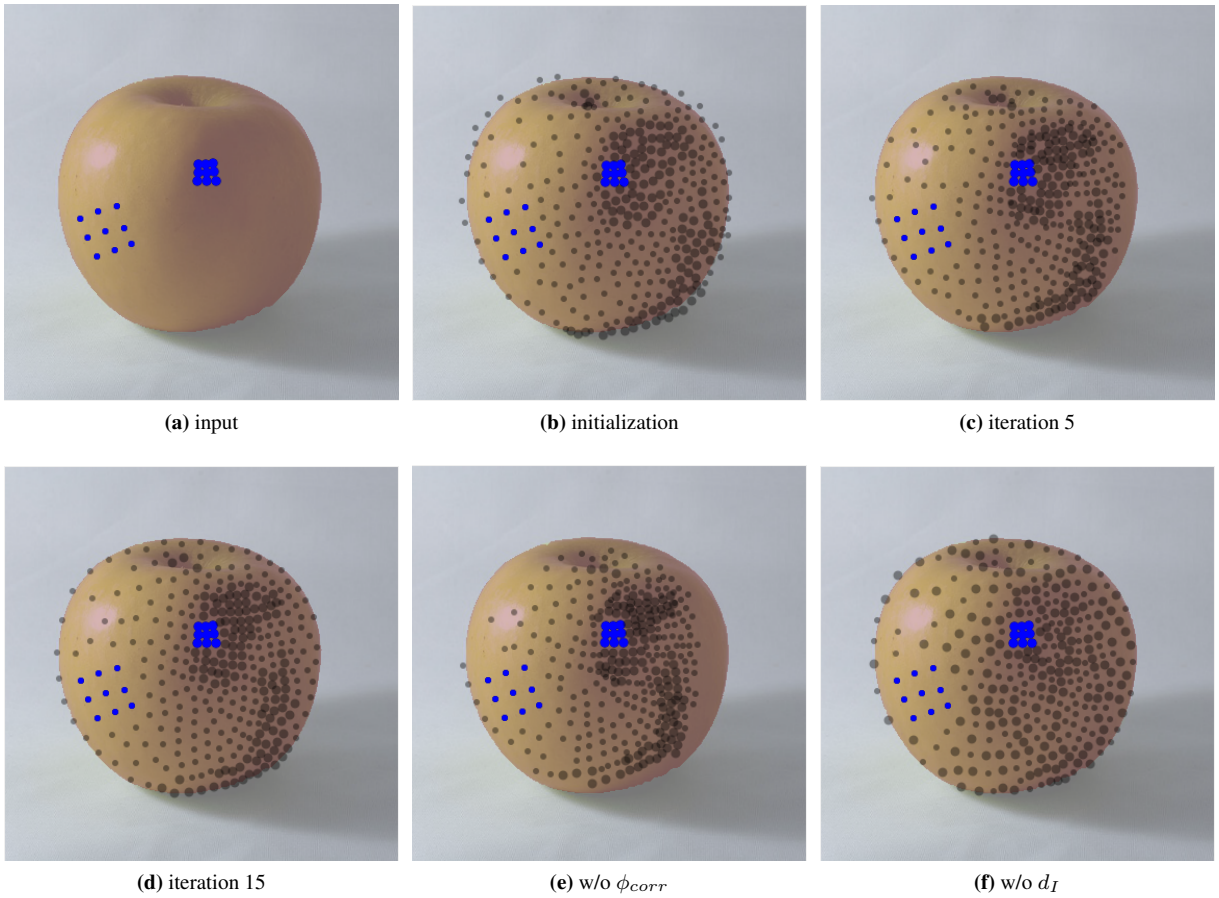
$$\phi_{\text{corr}}(X) = \sum_{s_o \in X} \left|p(s_o) - \bar{p}(s_o)\right|^2. \quad (4)$$

4.1.7 *Solver*

The energy function we aim to minimize is defined as:

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

We iteratively minimize the energy function following the EM methodology in [38]. In each iteration, for each output stroke s_o, we search for the closest matching 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 $w = (i/m)^2$, so that more weight is given to ϕ_{neigh} in earlier iterations, to optimize the



(a) input **(b)** initialization **(c)** iteration 5

(d) iteration 15 **(e)** w/o ϕ_{corr} **(f)** w/o d_I

Fig. 11 (a) Input. (b–d) Iteration process. (e,f) Ablation studies. Without the correction term ϕ_{corr} the predicted strokes tend to cluster together (e). Without the image term d_I the predicted strokes may not follow the reference sufficiently(f).

neighborhood distribution first before making corrections: this leads to better results.

Figures 11b–11d show iterative optimization of both objectives. For 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 we infer E, M, O, and R for synthesis from user interactions with I.

4.2.1 Input exemplar E

In this step we aim 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 [9] and they usually have specific intentions when drawing repetitive strokes, we assume the example strokes to be temporally consecutive and to 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 with similar shape and image features to the last stroke. Stroke shape similarity is measured using the Fréchet distance, while 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 [41] at a stroke's center. Alternatively, one could use different image features to capture different drawing intentions. 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. Back-traversal stops when the next stroke does not contain a 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 search process.

4.2.2 Output region M

The shared features of the obtained stroke exemplar also indicate the intended region. For instance, if all exemplar

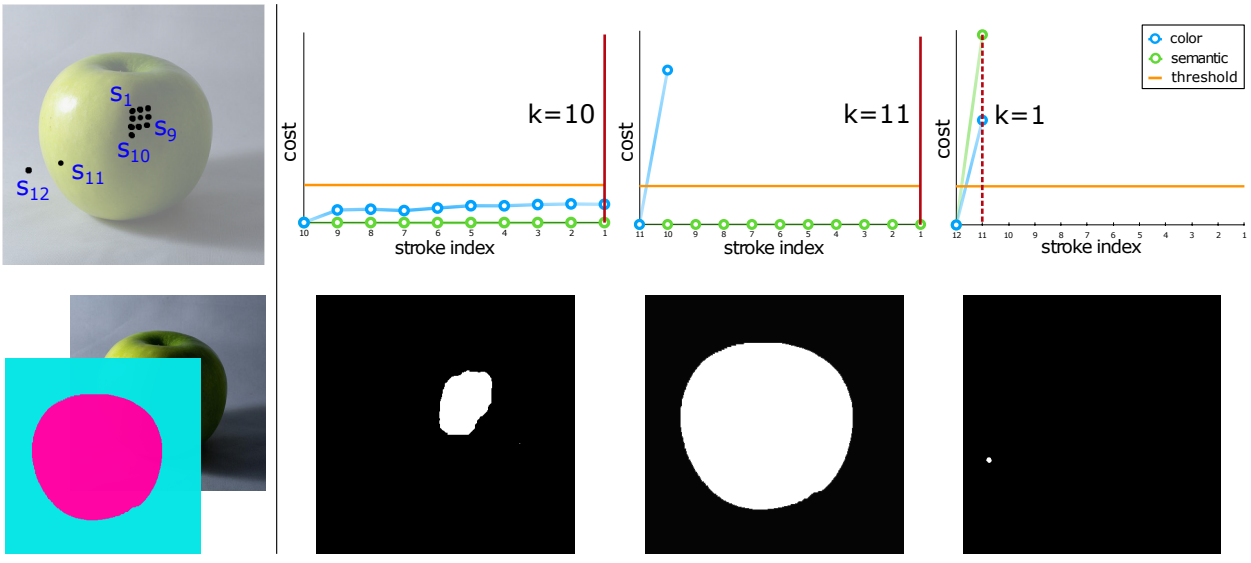


Fig. 12 Predicting the input exemplar and output region. Left, above: the input stroke sequence (black dots, only a few indices are shown for clarity) on the reference image. Left, below: image features. Right, above: threshold lines and the image feature cost curves for s_{10}, s_{11}, s_{12} respectively. Right, below: corresponding predicted output regions. 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 would not be used for synthesis.

strokes lie inside the same object segmentation region, it is very likely that the user intends to fill that region. Therefore, we use the shared features obtained in the exemplar grouping process to find a similar region for output.

Since we have only two features in our implementation, we simply obtain the region by GrabCut [42] if the Lab* color feature is shared by the exemplar strokes, we directly take the corresponding segmentation if the semantic feature is shared, and we 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 remainder, as it is less natural to propagate to distant regions.

4.2.3 Contextual constraints

Since the drawing is usually related to the underlying reference image, we analyze the properties of both the drawn strokes and the reference image to infer possible relationships to control the global distribution of strokes. The constraints we consider are orientation O and radius R.

Artists usually adjust stroke directions to convey curvature, but may sometimes randomize or fix 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) [43] 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 $15°$), 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 at each point of the orientation field.

Since density is inversely proportional to the spacing between strokes, we reframe the spacing problem as predicting a radius map that controls the extents 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:

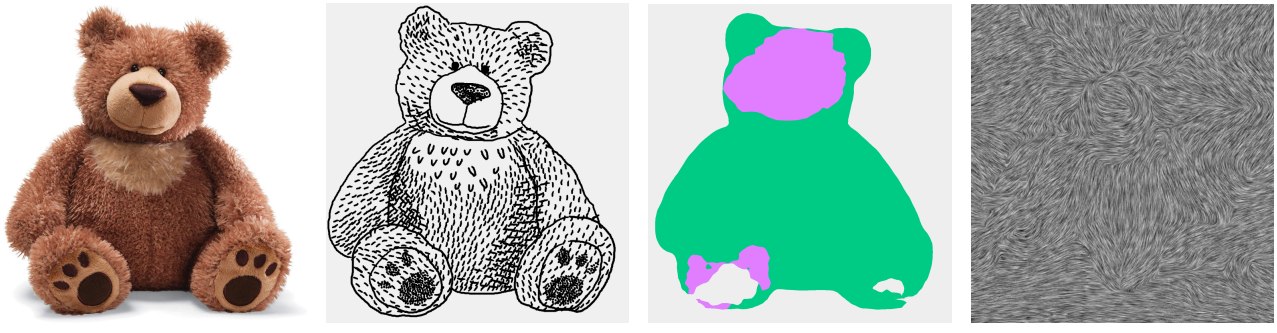
$$r = \begin{pmatrix} l & g & 1 \end{pmatrix} \cdot \mathbf{t}, \tag{6}$$

where $\mathbf{t}$ denotes the coefficients to be found. Using 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 to be uniform and create a constant radius map using the average spatial distance of the exemplar. We then update the user interface with the computed coefficients.

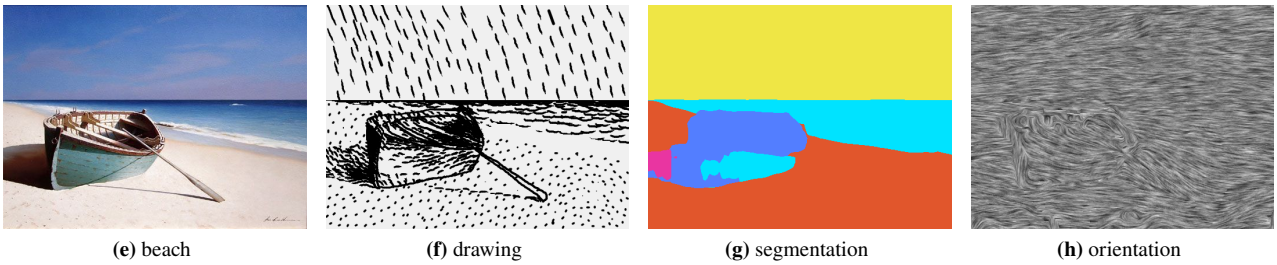
5 Evaluation

5.1 Approach

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



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



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

Fig. 13 Target session tasks. (a, e): Reference photos. (b, f): corresponding sample outputs.

In *autocompletion* mode, users had full access to our prototype, including autocompletion and interactive editing.

In *interactive batch filling mode* (batch mode), users were required to create a texture example first and then manually specify properties for batch filling. It simulates the sequential procedure in many IB-AR methods (e.g. [21]), although they rarely allow users to directly define examples on target images. This mode was performed using our system with autocompletion turned off.

In *fully manual drawing mode* (manual mode), users had 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.

5.2 Target Session

The goal of this session was to compare the three interaction modes in terms of utility and usability. Since we aim to facilitate drawing using an image scaffold, we included general users with different backgrounds but focusing on less skillful users, who are more likely to want to use reference images. We thus recruited 12 participants, including nine novices with little drawing experience, two amateurs with some experience (P3, P4), and a student majoring in illustration (P5). Most of the studies were conducted on a Lenovo Miix 520 tablet with a stylus, in a lab environment, except for two studies conducted remotely using a mouse (due to the covid pandemic).

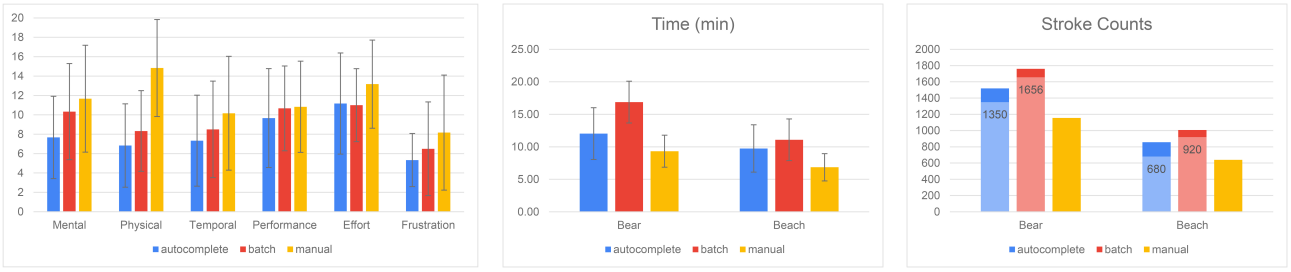
The study procedure consisted of a tutorial followed by target tasks, and took each participant about two hours in total.

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 training task. They were encouraged to vary the density and orientation of input strokes and to become familiar with the features of our system.

We used a within-subject design, in which each participant was asked to reproduce two target drawings (see Figure 13) in all three modes: autocompletion, batch mode, and manual mode. The target drawings contained an object and a landscape, common illustration topics (e.g. see Figure 2). The assigned order of modes was randomised over all 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 within about a dozen minutes, but without any hard time limit. After completing the two drawings in each mode, each participant filled in a NASA-TLX questionnaire [44]. Finally, we asked the participants about their preferred mode and usage experience, and for other comments.

5.3 Open session

The goal of this session was to observe how users interact with our system and to learn about users' subjective experiences.



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

Fig. 14 Target session results. (a) Average NASA-TLX scores from 12 participants. Lower scores are better. (b) Average completion time. (c) Average stroke counts. The number of system-generated strokes is labeled in each column.

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) using our system. The reference image was a portrait photo, also common in illustrations. The only requirement was that the drawings should contain some repetitive content. We again commenced with a tutorial and conducted the task on a Lenovo Miix 520 tablet with stylus. The participants were encouraged to think aloud and describe their thought processes and interaction during this session. After this task, participants could optionally create further drawings from any images they wanted. Since our prototype does not contain all common functionality of commercial drawing tools, we allowed the participants to retouch the resulting drawings, without adding more strokes, in Photoshop.

5.4 Results and Observations

5.4.1 Workload

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

5.4.2 Efficiency

We calculated the average completion time (Figure 14b) and stroke count (Figure 14c) for each mode and task. Generally, the system synthesized about 82% of the strokes in the autocompletion mode and about 92% of the strokes in batch mode. Although manual mode took the shortest time for participants to complete, it also resulted in the fewest total strokes. We thus calculated the strokes per minute for each mode: autocompletion (111.03, SD=38.76), batch (101.98, SD=45.13), manual (115.95, SD=46.73). It turns out that automatic generation did not improve 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 taken to explore alternative patterns, which, however, might be high in a fully manual case.

5.4.3 Quality

We asked 30 external volunteers to evaluate the quality of participants' drawings.We randomized all 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 average scores for each mode: autocompletion (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 was slightly better than for the fully manual drawings, but without a significant difference. From the participants' perspective, three novices commented that the automated strokes were better than their manual strokes, because they tend to become impatient when manually drawing all strokes, lowering quality.

5.5 Preferred mode

Seven participants preferred autocompletion mode while the other five participants preferred batch mode. Generally, autocomplete mode was considered more convenient, but less

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

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

Fig. 15 Example drawing results from the open session. Each case indicates the number of manual / autocompleted strokes.

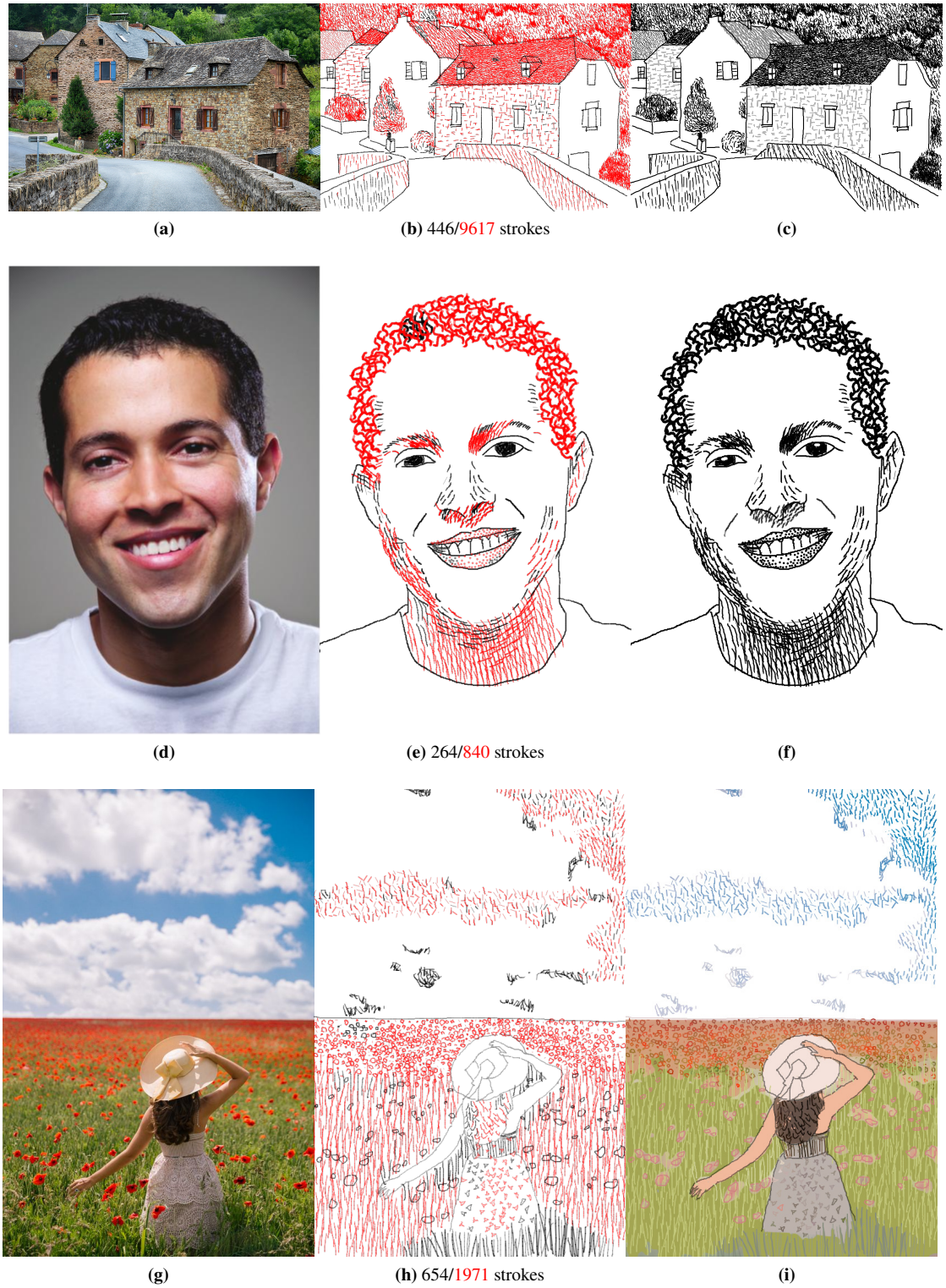
precise; batch mode was considered more precise, but requires too much interaction. P12 commented, *"autocompletion mode is more straightforward, because you can see the filling effects instantly without doing a lot of manipulation beforehand; while in batch mode, you have to remember the meanings of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

5.6 Creative results and experience

Figure 15 shows 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. Figure 16 demonstrates further 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, and one user commented that she *"felt creative when drawing with this system"*', because she could try 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 could be helpful in adding interesting textures to color paintings (e.g. see Figure 16i). Two users commented that the reduction in workload is useful, but they also complained about some inaccurate inferences of the autocompletion. We further discuss this problem in Section 6.

6 Limitations and Future Work

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



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

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

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

Fig. 16 Sample results. For each example: left: reference image, center: manual (black) and autocompleted (red) strokes, right: final drawings. In the last example, the strokes were created with our system first and then imported into Photoshop for background coloring.

Fig. 17 Example of visual blocking. Left: reference image. Right: canvas view.

6.1 Accuracy of autocompletion

We rely on simple Lab* color and semantic segmentation for region inference. While color features suffice for most cases, regions with similar colors but different semantics require sufficient segmentation accuracy for region inference (e.g. see Figure 13c). More advanced semantic selection methods (e.g. [45]) might help to infer more accurate regions. However, granularity of selection requires further study. For example, when users draw on a bear's limb, is the intended region the whole bear, or all limbs? We leave this as future work.

6.2 Visual blocking

Since the drawing and the system's suggestions are overlaid on the reference image, it can be difficult for users to see the image when selecting parts of the suggestions (see Figure 17) or adding a new layer of strokes. Although users can switch views using a hotkey, it might be helpful to provide some reference information, like image darkness or boundaries, through additional visual hints [10, 16].

6.3 Higher-level image features

We only consider 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 may be possible to extend the scope of autocompletion, such as autocompleting the sparse flowers in the foreground of Figure 16i through the correspondences between strokes and elements.

6.4 Stroke types

Our method only supports short strokes, while artists frequently also use long repetitive strokes [1]. It is worth investigating the possibility of incorporating continuous strokes [46] in our analysis and synthesis framework and extending support to different input strokes.

7 Conclusions

We have presented a new drawing concept and designed an assisted drawing system 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 system is able to generate results adapted 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.

Acknowledgements

We are grateful to Li-Yi Wei for his insightful comments and suggestions. We also thank the anonymous reviewers for feedback, and funding from Adobe Research and the Deutsche Forschungsgemeinschaft, Project-ID 251654672-TRR 161.

Declaration of competing interest

The authors have no competing interests to declare that are relevant to the content of this article.

References

[1] Dunn A. *Pen and Ink Drawing: A Simple Guide*. New Jersey: Three Minds Press, 2015. 1, 13

[2] Adobe. Paint stylized strokes with the Art History Brush. https://helpx.adobe.com/photoshop/using/painting-stylized-strokes-art-history.html, 2017. 3

[3] Martín D, Arroyo G, Rodríguez A, Isenberg T. A Survey of Digital Stippling. *Computers & Graphics*, 2017, 67: 24–44, doi:10.1016/j.cag.2017.05.001. 3

[4] Hertzmann A, Jacobs CE, Oliver N, Curless B, Salesin DH. Image Analogies. In *SIGGRAPH '01*, 2001, 327–340, doi:10.1145/383259.383295. 1, 2, 3, 4

[5] Fišer J, Jamriška O, Lukáč M, Shechtman E, Asente P, Lu J, Sýkora D. StyLit: Illumination-guided Example-based Stylization of 3D Renderings. *ACM Trans. Graph.*, 2016, 35(4): 92:1–92:11, doi:10.1145/2897824.2925948. 2

[6] Gerl M, Isenberg T. Interactive Example-based Hatching. *Comput. Graph.*, 2013, 37(1-2): 65–80, doi:10.1016/j.cag.2012.11.003. 1, 2, 3

[7] Hegde S, Gatzidis C, Tian F. Painterly rendering techniques: a state-of-the-art review of current approaches. *Computer Animation and Virtual Worlds*, 2013, 24(1): 43–64. 1, 2

[8] Kazi RH, Igarashi T, Zhao S, Davis R. 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*, CHI '12, 2012, 1727–1736, doi:10.1145/2207676.2208302. 1, 3

[9] Xing J, Chen HT, Wei LY. Autocomplete Painting Repetitions. *ACM Trans. Graph.*, 2014, 33(6): 172:1–172:11, doi:10.1145/2661229.2661247. 1, 3, 4, 7

[10] Xie J, Hertzmann A, Li W, Winnemöller H. PortraitSketch: Face Sketching Assistance for Novices. In *UIST '14*, 2014, 407–417, doi:10.1145/2642918.2647399. 1, 2, 13

[11] Kang HW, He W, Chui CK, Chakraborty UK. Interactive sketch generation. *The Visual Computer*, 2005, 21(8): 821–830, doi:10.1007/s00371-005-0328-9.

[12] Su Q, Li WHA, Wang J, Fu H. EZ-sketching: Three-level Optimization for Error-tolerant Image Tracing. *ACM Trans. Graph.*, 2014, 33(4): 54:1–54:9, doi:10.1145/2601097.2601202.

[13] Li G, Bi S, Wang J, Xu Y, Yu Y. ColorSketch: A Drawing Assistant for Generating Color Sketches from Photos. *IEEE Computer Graphics and Applications*, 2017, 37(3): 70–81, doi:10.1109/MCG.2016.37. 2

[14] Iarussi E, Bousseau A, Tsandilas T. The Drawing Assistant: Automated Drawing Guidance and Feedback from Photographs. In *UIST '13*, 2013, 183–192, doi:10.1145/2501988.2501997. 2

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

[16] Williford B, Doke A, Pahud M, Hinckley K, Hammond T. DrawMyPhoto: Assisting Novices in Drawing from Photographs. In *Proceedings of the 2019 on Creativity and Cognition*, C&C '19, 2019, 198–209, doi:10.1145/3325480.3325507. 2, 13

[17] Haeberli P. Paint by Numbers: Abstract Image Representations. In *SIGGRAPH '90*, 1990, 207–214, doi:10.1145/97879.97902. 2

[18] Benedetti L, Winnemöller H, Corsini M, Scopigno R. Painting with Bob: Assisted Creativity for Novices. In *UIST '14*, New York, NY, USA: ACM, 2014, 419–428.

[19] Tsai HC, Lee YH, Lee RR, Chu HK. User-guided line abstraction using coherence and structure analysis. *Computational Visual Media*, 2017, 3(2): 177–188. 2

[20] Kyprianidis JE, Collomosse J, Wang T, Isenberg T. State of the "Art": A Taxonomy of Artistic Stylization Techniques for Images and Video. *IEEE Transactions on Visualization and Computer Graphics*, 2013, 19(5): 866–885, doi:10.1109/TVCG.2012.160. 2

[21] Salisbury MP, Wong MT, Hughes JF, Salesin DH. Orientable Textures for Image-based Pen-and-ink Illustration. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, 401–406, doi:10.1145/258734.258890. 2, 9

[22] Hiller S, Hellwig H, Deussen O. Beyond stippling—Methods for distributing objects on the plane. In *Computer Graphics Forum*, volume 22, Wiley Online Library, 2003, 515–522. 2, 3

[23] Kalogerakis E, Nowrouzezahrai D, Breslav S, Hertzmann A. Learning Hatching for Pen-and-ink Illustration of Surfaces. *ACM Trans. Graph.*, 2012, 31(1): 1:1–1:17, doi:10.1145/2077341.2077342. 2

[24] Gatys LA, Ecker AS, Bethge M, Hertzmann A, Shechtman E. Controlling Perceptual Factors in Neural Style Transfer. *ArXiv e-prints*, 2016. 2

[25] Kaspar A, Neubert B, Lischinski D, Pauly M, Kopf J. Self Tuning Texture Optimization. *Computer Graphics Forum*, 2015, doi:10.1111/cgf.12565. 2

[26] Nancel M, Cockburn A. Causality: A Conceptual Model of Interaction History. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, New York, NY, USA: ACM, 2014, 1777–1786, doi:10.1145/2556288.2556990. 3

[27] Xing J, Wei LY, Shiratori T, Yatani K. Autocomplete Hand-drawn Animations. *ACM Trans. Graph.*, 2015, 34(6): 169:1–169:11, doi:10.1145/2816795.2818079. 3

[28] Peng M, Wei LY, Kazi RH, Kim VG. Autocomplete Animated Sculpting. In *UIST '20*, 2020, 760–777, doi:10.1145/3379337.3415884. 3

[29] Peng M, Xing J, Wei LY. Autocomplete 3D Sculpting. *ACM Trans. Graph.*, 2018, 37(4). 3

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

[31] Fišer J, Asente P, Sýkora D. ShipShape: A Drawing Beautification Assistant. In *Proceedings of the Workshop on Sketch-Based Interfaces and Modeling*, SBIM '15, Goslar Germany, Germany: Eurographics Association, 2015, 49–57. 3

[32] Zitnick CL. Handwriting Beautification Using Token Means. *ACM Trans. Graph.*, 2013, 32(4), doi:10.1145/2461912.2461985. 3

[33] Barla P, Breslav S, Markosian L, Thollot J. Interactive Hatching and Stippling by Example. Research Report RR-6461, INRIA, 2006. 3

[34] Ijiri T, Mêch R, Igarashi T, Miller G. An Example-based Procedural System for Element Arrangement. *Computer Graphics Forum*, 2008, 27(2): 429–436, doi:10.1111/j.1467-8659.2008.01140.x.

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

[36] Hsu CY, Wei LY, You L, Zhang JJ. Autocomplete Element Fields. In *CHI '20*, 2020, 1–13, doi:10.1145/3313831.3376248. 3

[37] Mortensen EN, Barrett WA. Intelligent Scissors for Image Composition. In *SIGGRAPH '95*, 1995, 191–198, doi:10.1145/218380.218442. 4

[38] Ma C, Wei LY, Tong X. Discrete Element Textures. *ACM Trans. Graph.*, 2011, 30(4): 62:1–62:10, doi:10.1145/2010324.1964957. 4, 6

[39] Zhao M, Zhu SC. Customizing Painterly Rendering Styles Using Stroke Processes. In *NPAR '11*, 2011, 137–146, doi:10.1145/2024676.2024698. 6

[40] Ma C, Wei LY, Lefebvre S, Tong X. Dynamic Element Textures. *ACM Trans. Graph.*, 2013, 32(4): 90:1–90:10, doi:10.1145/2461912.2461921. 6

[41] Zhao H, Shi J, Qi X, Wang X, Jia J. Pyramid Scene Parsing Network. In *CVPR*, 2017, 2881–2890. 7

[42] Rother C, Kolmogorov V, Blake A. "GrabCut": Interactive Foreground Extraction Using Iterated Graph Cuts. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, New York, NY, USA: Association for Computing Machinery, 2004, 309–314, doi:10.1145/1186562.1015720. 8

[43] Kyprianidis JE, Kang H. Image and Video Abstraction by Coherence-Enhancing Filtering. *Computer Graphics Forum*, 2011, 30(2): 593—602, doi:10.1111/j.1467-8659.2011.01882.x, proceedings Eurographics 2011. 8

[44] Hart SG, Staveland LE. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Advances in psychology*, volume 52, Amsterdam: Elsevier, 1988, 139–183. 9

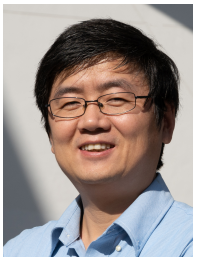
[45] Chen X, Zhao Z, Yu F, Zhang Y, Duan M. Conditional diffusion for interactive segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, 7345–7354. 13

[46] Tu P, Wei LY, Yatani K, Igarashi T, Zwicker M. Continuous Curve Textures. *ACM Trans. Graph.*, 2020, 39(6), doi:10.1145/3414685.3417780. 13

Author biographies

Yilan Chen received her Ph.D. degree from the School of Creative Media, City University of Hong Kong, Hong Kong, China. Her research interests include computer graphics and human-computer interaction.

Kin Chung Kwan received B.Sc. and Ph.D. degrees from the Chinese University of Hong Kong in 2009 and 2015. He is now a postdoctoral researcher at the University of Konstanz. His research interests include computer graphics and human-computer interaction.



Hongbo Fu is a Professor in the School of Creative Media, City University of Hong Kong. Previously, he had postdoctoral research training at the Imager Lab, University of British Columbia, Canada, and the Department of Computer Graphics, Max-Planck-Institut Informatik, Germany. He received a Ph.D. degree in computer science from Hong Kong University of Science and Technology in 2007 and a B.S. degree in information sciences from Peking University, China, in 2002. His primary research interests fall in the fields of computer graphics and human-computer interaction. He has served as an associate editor of The Visual Computer, Computers&Graphics, and Computer Graphics Forum.