A Mixed-Initiative Interface for Animating Static Pictures

Nora S Willett*, Rubaiat Habib Kazi†, Michael Chen†, George Fitzmaurice†,
Adam Finkelstein*, Tovi Grossman†‡

*Princeton University †Autodesk Research ‡University of Toronto

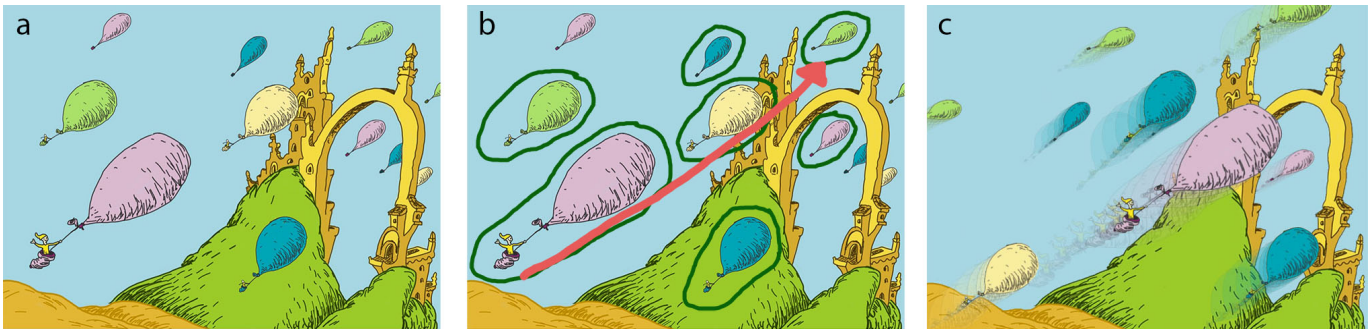

Figure 1: An example of animating a static picture using our system. (a) The original picture. (b) The user scribbles some sample objects (green) and their overall motion direction (red). (c) The resulting animation consists of kinetic textures [27] (including all the balloons) that are spatio-temporally consistent with the source picture, with proper depth ordering and an inpainted background.

ABSTRACT

We present an interactive tool to animate the visual elements of a static picture, based on simple sketch-based markup. While animated images enhance websites, infographics, logos, e-books, and social media, creating such animations from still pictures is difficult for novices and tedious for experts. Creating automatic tools is challenging due to ambiguities in object segmentation, relative depth ordering, and non-existent temporal information. With a few user drawn scribbles as input, our mixed initiative creative interface extracts repetitive texture elements in an image, and supports animating them. Our system also facilitates the creation of multiple layers to enhance depth cues in the animation. Finally, after analyzing the artwork during segmentation, several animation processes automatically generate kinetic textures [27] that are spatio-temporally coherent with the source image. Our results, as well as feedback from our user evaluation, suggest that our system effectively allows illustrators and animators to add life to still images in a broad range of visual styles.

Author Keywords

Kinetic textures; animation; dynamics; pictures.

ACM Classification Keywords

H.5.2 Information interfaces and presentation: User interfaces - Graphical interface.

UIST 2018, October 14–17, 2018, Berlin, Germany.

ACM ISBN 978-1-4503-5948-1/18/10 ...$15.00.
http://dx.doi.org/10.1145/3242587.3242612

INTRODUCTION

Dynamic drawings and cinemagraphs are becoming increasingly popular forms of media to portray scenes with animated, ambient textures and elements. Childrens books, logos, infographics, e-cards, academic papers [21], and gifs on social media make use of animated elements to enhance their appeal. In all of these examples, the artist carefully sets the temporal and spatial aspects of the animation to match their vision. In addition, the resulting animated video should loop seamlessly for a pleasing experience when embedded in the final applicaiton. Traditionally, such animations are generated from layered images [27, 43] or videos [19, 33, 38, 45, 48]. However, one way to direct an artist's vision is to make the animation resemble a static image. While the dynamics in a static picture are often imaginable, converting a static, raster image to an animated one is challenging [15]. This difficulty is primarily due to the lack of semantic object and depth information, which is necessary for animation. Furthermore, the image is not optimized for a particular animation data structure (i.e., rigging [25], vector graphics [27], meshes [23, 28, 51], hierarchy) that might be necessary for the corresponding animation framework. They also lack temporal information, and the decision about how objects should move often relies on artistic intuition or storytelling goals.

A common occurrence in pictures is the presence of repeating similar elements. They can take the form of falling snow, rain, and leaves or flying birds, insects and balloons. An artist could also create bubbles rising through the water, a school of fish swimming in a reef, or music notes drifting through the air. Such repeating elements are abundant in drawn pictures and photographs, where similar yet slightly varying objects are distributed across the image. Unlike previous animation systems [27] that enable artists to create animated textures from scratch, we start from a static image resembling the final

output. Animating manually from a static image is tedious and error-prone, since one must extract all such elements (using the GrabCut [39] algorithm or commercial tools, such as Photoshop [3]), and preserve their spatial arrangements over time. Additionally, the presence of repeating objects can give important cues about the dynamics of an associated animation.

We facilitate the creation of animated pictures from an input raster image (Figure 1a) using a few high-level guided strokes. First, our mixed initiative interface allows a user to decompose repeating elements in the scene. In this step, once the user circles a subset of the objects (Figure 1b), our system automatically finds similar objects in the image and extracts them into a separate layer. Second, we allow the extraction of multiple layers and user guided depth ordering of those layers. Third, from the user guided motion path and the extracted objects, our optimization step generates a kinetic texture [27] that is spatio-temporally coherent with the elements in the source image (Figure 1c). We analyze the semantics of the source scene to generate the necessary components and optimize the parameters of the kinetic textures to approximately preserve the spatial arrangements, density, and scale of the elements over time. Finally, we seamlessly loop the resulting animation. Our contribution is a new mixed initiative interface that synthesizes and extends existing image-processing algorithms and new animation optimization algorithms into an interactive sketch-based system for animating static images. As we will demonstrate, our algorithms and UI are applicable to a wide range of visual styles (i.e., paintings, cartoons, illustrations, photographs, infographics), which makes it suitable for a variety of application domains and scenarios.

To validate our system, we ran a series of observational feedback sessions with six participants. Our results show the potential of our system, point to areas of future work, and demonstrate that even first time users can animate static pictures in a short period of time - a task that is otherwise difficult for novices and tedious for experts using state of the art tools.

RELATED WORK

This section reviews prior work in creating animations from images and videos, and finding texture elements for high-level image editing.

Draco: The output of our system most closely resembles that of Kazi et al.'s Draco [27], because our system leverages the Sketchbook Motion iPad app [8] (Autodesk's implementation of Draco) as a renderer. However, we introduce several advances. Our method animates a single preexisting static image, whereas Draco requires the user to draw each part of the animation from scratch. Starting from an existing image allows users with limited drawing skills to design aesthetically pleasing animations. Draco also requires the user to manipulate all of the animation parameters to achieve their final result. In contrast, our system automatically computes the emitter line, emission frequency, and scale from user defined scribbles and the underlying picture. Finally, we introduce a method to automatically loop the resulting animation.

Animation from a single layer: Animating from a single image began with simple deformation [34] or adding varying patterns to simulate motion [20]. Horry et al. simulate depth by separating the foreground and background and adding simple 3D modeling to them [22]. The work most closely related to our goal is Chuang et al.'s addition of motion textures to a picture [15]. Their interface assists the user in separating areas for animation such as trees, water, boats and clouds. With time-varying displacement maps, they animate the separated regions creating effects like swaying, rippling, bobbing and translation. However, their method does not extract multiple instances of an object (i.e., finding all the boats). Our system can extract the same objects as Chuang et al. and assists in extracting similar objects in the image (i.e. multiple clouds, boats). For animation, Chuang et al.'s displacement maps result in a limited set of possible animations. Our animation process complements their work by focusing on kinetic textures [27], not stochastic motion textures, and extends their work by leveraging the properties of the original image elements when specifying animation parameters.

Sketch-based animation tools: Researchers have explored various methods for easy animation authoring, using motion sketching [17] and direct manipulation [23, 49, 50] techniques. However, in such cases, the visual elements to be animated must be drawn from scratch [7], or are already stored on separate layers as vector graphics [27], raster graphics [2], or meshes [10, 28]. Our input of a single static image lacks separate layer and data structure information, requiring a new technique to animate it.

Animation from videos: Using videos as input for animation creation supplies the missing information that a static image lacks. For instance, videos encode temporal information providing examples of movement and revealing background layers via disocclusion. Liu et al. describe how motion in video can be exaggerated for visualization and other purposes [35]. Video textures allow for continuous looping of the original frames so that a sequence is never repeated [41]. Cinemagraphs extend video textures but focus on animating a specific part of the image [9, 48]. Liao et al. explore creating looping videos with user control over the dynamism of the result [33]. Cliplets, while not looping, allow finer control over the spatial and temporal parts of the video [26]. Su et al. transfers video deformations to a separate single image [44]. While these techniques create compelling results, we focus on using a single image (with non-existent temporal information) as the input for our animation.

Finding texture elements: Image segmentation [42] methods are commonly used for extracting objects or texture elements from an image. When extracting a single object, GrabCut works well with user defined foreground and background marks [39]. The computer vision community has studied the problem of finding repeated elements in a picture [4, 14, 29, 31, 40]. In this line of work, the RepFinder system of Cheng et al. [14] is most suited to our problem. RepFinder [14] extends the GrabCut concept to the location of multiple similar objects with template matching [32]. However, our definition of similar encompasses deformation, and non-regular variations whereas RepFinder will only recognize objects which vary in color, position, orientation and scale. This distinction is

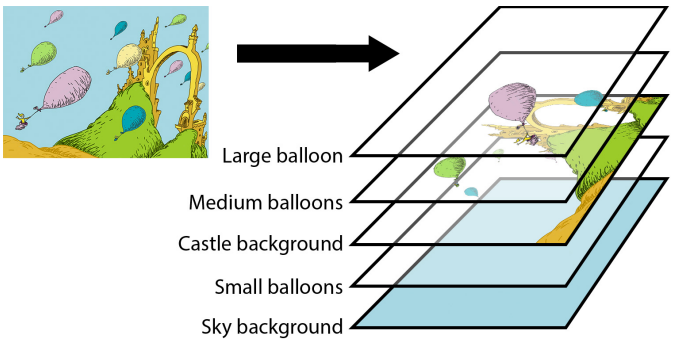

Figure 2: The layer stacks resulting from our framework.

particularly important when processing artwork and paintings, beyond photographs. Our method for finding texture elements leverages and enhances GrabCut to process simple user input marks into more informative masks for input. Once exemplar texture elements are found, researchers have proposed methods to characterize how new textures with similar statistics could be synthesized or used in practice [5, 6, 30, 36, 37, 46].

ANIMATING STATIC PICTURES: CHALLENGES

Given a static image as input, there is not necessarily an explicit 1-to-1 mapping to a corresponding animation. That is, there could be many reasonable interpretations as to what an "animated" version of a static image could look like. Our system's goal is to provide animators with the tools to create an animation which resembles, but does not necessarily exactly match, the original image. In some cases, the author may wish to generate an animation for which the original image is indistinguishable from a random subset of final animation frames. However, in other cases, the author may want more creative freedom resulting in divergence from the source image. In developing a system that facilitates both of these complementary goals, there are numerous challenges to overcome.

Depth ambiguities: One challenge is determining the depth ordering from a single image. How do we know if a small object is far away or just smaller than other objects in the image? For instance, the balloons in Figure 1 could have multiple possible explanations for their depth ordering. The different size balloons could be on separate layers with each smaller size behind/further away than the larger ones. In addition, the layer with the smallest balloons would fly behind the castle due to the occlusion clues from the image. In a different interpretation, the balloons could also be flying from close to the user in the lower left and then travel further away and through the castle, hence becoming smaller.

Temporal ambiguities: Once an image has been separated into layers, more challenges, such as a lack of temporal information, arise when trying to animate the image so that it resembles the original. For example, do the balloons in Figure 1 move in a straight line from the bottom left to the top right? Or do they curve, wiggle and rotate along the way? In addition, the image does not give clues to the speed of the objects. Are all the balloons moving slowly across the image? Or are the larger balloons moving faster?

Looping: Another challenge is the desire for the final animated video to loop seamlessly, with all animated elements in the scene matching at the start and end. Previous techniques for looping video (e.g., [9, 26, 33, 41]) rely on morphing and blending natural imagery texture elements and would not work as well for the kinds of discrete, visually distinctive, hand-drawn elements in many of our examples.

Overall, such ambiguities motivate a mixed-initiative approach for our problem. While certain inferences can be made, we cannot expect our algorithms to operate fully automatically since there are multiple ways an animation can be interpreted from a still image. For example, users should be able to specify how objects are separated into layers, or adjust their motion paths and speed. Providing users with an UI that allows them to specify all these constraints is in itself a challenge. In general, combining user input with automatic algorithms will allow artists the freedom to choose the interpretation of the original still image that best suits their artistic vision.

SYSTEM OVERVIEW

Our system allows a user to take a static image as input, and animate the objects with high-level gestures to create an intended animation that resembles the original image. We rely on a mixed-initiative approach, combining the power of automated image processing algorithms with simple and interactive tools to iterate on or guide the results. First, guided by user annotated exemplars, our system efficiently detects and extracts similar looking elements from the image. Once the elements, foreground, and background are separated into layers, the user rearranges them into the desired ordering for depth and occlusion effects (Figure 2). When animating the image, we use data from the original image to tune the animation parameters.

Extracting Repeating Objects

Our system targets images with multiple instances of objects that should be animated. For instance, there could be balloons of varying sizes and colors (Figure 1), lanterns of differing sizes (Figure 3), pieces of avocado (Figure 14), butterflies and raindrops (Figure 15a,e). Due to the large variety of repeating objects and the possible differences between them within a single image, we opt for an iterative, interactive approach. We allow the user to quickly select a subset of those objects which we use for animation. From these exemplar objects, we search for similar objects in the rest of the image to remove them from the background. At each of these steps, we provide interactive tools for users to refine our automatic results. Finally, we inpaint [12] the background and return two new layers: one transparent with only the exemplar objects and another completed background with all the selected objects removed.

Depth Ordering with Layers

While this new background will have all the animation objects removed, there might be other areas which the user wishes to separate into further layers. For instance, hills (Figure 1), people (Figure 3), chopsticks (Figure 14), engine parts or trees (Figure 15b,d) might be on a mid-ground layer with animated objects in front and behind. In this instance, we allow the user to supplement our automatic partitioning algorithm to select the exact part of the image to extract. We present the user with

Figure 3: The user interface, consisting of (a) a main canvas, (b) layer panel, the (c) global and (d) contextual toolbars.

a transparent layer containing the mid-ground object and an inpainted background (Figure 2). These layers allow the user to animate the mid-ground object without holes appearing in the animation. With these new layers and the ability to rearrange their ordering, the user clarifies any depth ordering ambiguities present in the source image.

Animation Optimization

We give the user the ability to add all the dynamic elements of a kinetic textures animation framework [27]. From a user drawn motion line, we leverage the elements' positions in the original image to automatically create an emitter - a path which generates particles at a defined frequency and velocity. We control the emission frequency to match the density of objects in the static source image. In some cases, emission objects may scale along their path (Figure 12) or might travel on different layers (Figure 2). We offer automatic methods to set these properties. We also introduce a method to create a seamless, looping animation while preserving temporal coherency for all objects in the scene.

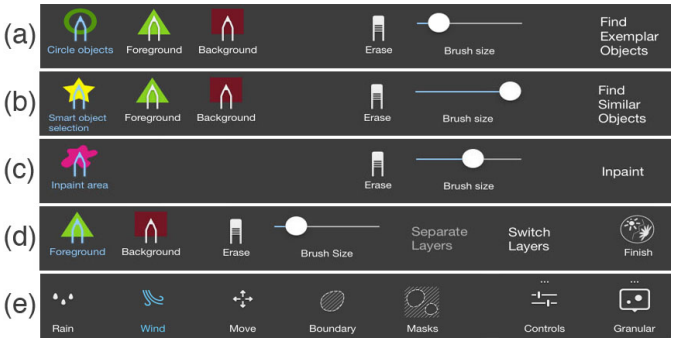

Figure 4: (a) Exemplar objects menu. (b) Similar objects menu. (c) Inpaint menu. (d) Separate layers menu. (e) Existing Sketchbook Motion animation menu.

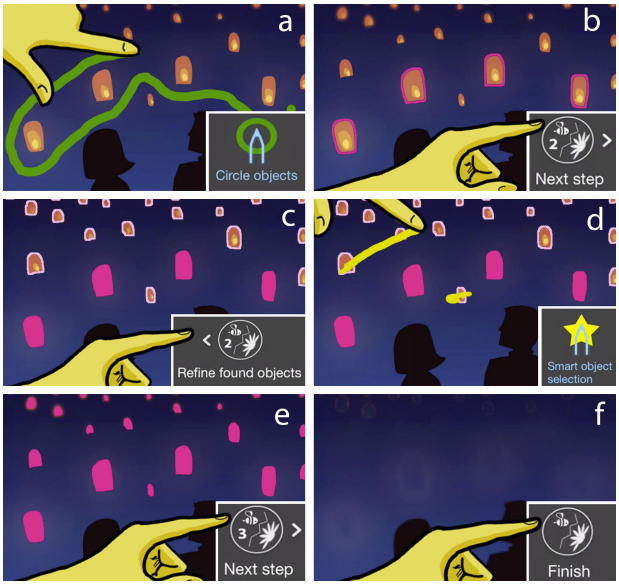

Figure 5: Interactive steps for separating animation objects. (a) Step 1: The user circles the objects. (b) The exemplar objects are shown for review. (c) Step 2: Similar objects are found in the image. The user decides to refine the found objects. (d) The Smart object selection brush is used to mark the light pink objects. (e) All the objects are found. (f) Step 3: The background is cleared of all objects.

USER INTERFACE

We designed and developed an interactive system allowing the user to animate an existing static picture. Our system is implemented as a plug-in within the Autodesk Sketchbook Motion iPad app [8], and borrows some of its existing UI elements. The interface has a main canvas, layer panel, global and contextual toolbars (Figure 3). A typical workflow starts with the user importing an image and decomposing the image into layers of animation and background objects. Finally, the user adds animation effects to the layers and exports the looping animated video. We elaborate on the workflow below.

Layer Panel

We use SketchBook Motion's unmodified layer panel, which is similar to those in other graphical editing tools [8]. Each of our decomposition and separation processes creates temporary layers storing brush markings and results. Each brush creates a new layer that is converted to black (background) and white (brush marks) masks as the input to our algorithms, described in the next section. After each process, we clear the temporary layers and replace them with the final layers. Users can also manually add, remove and duplicate layers.

Segmenting Objects to be Animated

As illustrated in the previous section, the first challenge is to extract elements to be animated from the background. Our UI guides the user through the three steps of (1) finding exemplar objects to be animated, (2) searching for similar objects in the scene, and (3) inpainting the background.

For Step 1, we extract the exemplar objects onto a separate layer. To begin, the "Circle objects" brush (Figure 4a) is used to draw circles around all the objects that the user will animate

later (Figure 5a). Objects can be circled separately (Figure 1b) or together (Figure 5a). Next, our algorithm separates the circled objects onto their own layer (Figure 5b). The user clarifies which pixels in each circle belong to the exemplar objects ("Foreground" brush) and which belong in the background ("Background" brush).

In Step 2, we search for similar, repeating objects in the rest of the image (Figure 5c). Objects, that we are confident are similar to the previously circled objects, are shown in filled dark pink. We are less confident for objects highlighted in light pink. The "Smart object selection" brush (Figure 4b) allows the user to replace light pink areas with dark pink to be inpainted (Figure 5d,e).

Finally, in Step 3, we inpaint all the previously marked dark pink areas (Figure 5f). The user can remove more areas with the "Inpaint area" brush (Figure 4c). When finished, a transparent layer with only the exemplar objects and a layer with the inpainted background are created.

Depth Ordering

For depth effects, the user might want to extract other parts of the image into separate layers. Before entering the separate layers menu, we automatically try to extract the foreground from the background. After refining the result with the "Foreground" and "Background" brushes, the "Separate layers" button re-runs our separation algorithm (Figure 4d). When done, we inpaint the area previously covered by the extracted foreground. The temporary layers are deleted and two new layers, a transparent one with only the foreground and another layer of the inpainted background, are created.

Animation

When animating, the user can add a kinetic or oscillating texture using Sketchbook Motion's existing interface [8]. For a kinetic texture, the user draws one or more motion path lines, indicated as the "Wind" button in Figure 4e. After drawing each line, we automatically create an emitter - a path which generates particles at a defined frequency and velocity (Figure 6a). We also allow the user to manually create an emitter. The user can choose to keep the emission at a fixed density, where we try to match the density of objects in the initial image (Figure 6b), or they can manually set the emission frequency and speed. In cases where there are objects of different scales in the original image, the user can scale the objects along the motion path (Figure 6c) or split the objects into two or three layers.

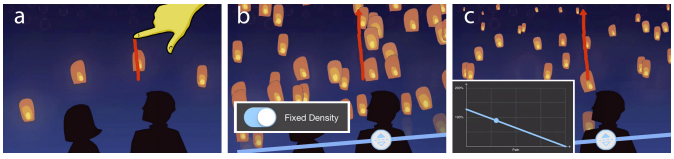

Figure 6: User interaction when animating. (a) The user draws a motion path. (b) A kinetic texture is automatically generated with an emitter (blue), where the density of the texture is set to approximate the original image. (c) The objects are automatically scaled over the path.

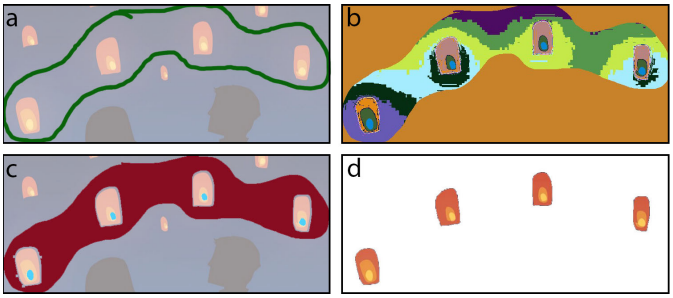

Figure 7: Finding exemplar objects. (a) User circles objects. (b) System separates pixels into 15 groups. (c) System tags the background (red) and foreground (light blue) groups. (d) GrabCut extracts the objects.

SYSTEM IMPLEMENTATION

Our system is implemented on an iPad, using OpenCV [13] and Image Stack [1] for image processing, GrabCut [39] for layer extraction, and PatchMatch [11] for inpainting. Supposing previous work on extracting repeating objects [14, 29] could run interactively on an iPad, either method could replace GrabCut [39] in our pipeline's layer extraction component.

Extracting Repeating Objects

To extract the objects from an image, our three step process allows the user the best combination of automatic help and detailed control for fine tuning.

Finding Exemplar Objects

When an image has multiple instances of objects, not every instance is desired in the final animation. Some instances might be too small, others might be occluded, and still others might lack details to look pleasing when animated. Hence, in the final animation, we only use exemplar objects that are circled by the user and extracted by our method (Figure 7a).

We start separating the circled areas by finding the contours of the mask from the "Circle objects" brush. A circled area is defined by having an inside and outside contour whose centroids are close together (less than five pixels apart). For each area, we use GrabCut [39] with five iterations to extract the objects. To run GrabCut, we need to determine sample areas of the background and foreground inside each circled area. We split the pixels inside the circle into 15 groups using k-means on the pixel color in the LAB color space (Figure 7b). Next, all groups that are underneath the circles are marked as background (Figure 7c). For the foreground, we select the group with the furthest average distance from all the background groups (Figure 7c). We initialize GrabCut with the foreground group and any foreground brush marks as the foreground, the circle and any background brush marks as the background and by default, all other pixels marked as probably background. Directly feeding the user's marks into GrabCut would be more tedious since many more marks would be required. The result is an image with only the circled objects visible (Figure 7d, Figure 8c).

With the brushes, the user can iterate on the result by carefully marking the interior of objects as foreground and anything else as background. In most cases, no refinements or only small refinements are necessary.

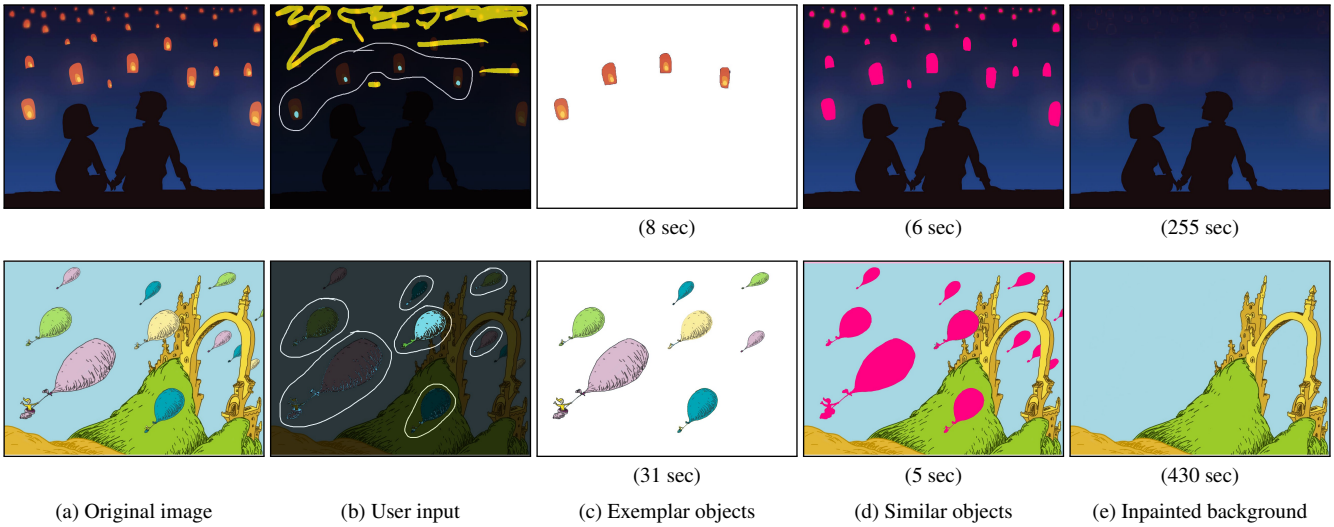

Figure 8: Summary of our segmentation pipeline for two example inputs. (a) Original image. (b) Input markings for all steps: white circles mark exemplar objects; light green shows foreground; yellow marks are for smart object selection. (c) Exemplar objects found by our system. (d) Dark pink indicates similar objects found for inpainting. (e) Final inpainted background layer ready for animated foreground. (Compute times in parentheses.)

Finding Similar Objects

Once the user specifies the exemplar objects, our goal is to find similar instances of those objects (raindrops, snowflakes, balloons, etc.) in the rest of the image. These instances can vary in shape, orientation, scale, color, and occlusion. From the previous step, we know the indications of foreground (objects) and background (all other pixels) within each circled area. We run GrabCut over the whole image with those indications and unknown pixels defaulting to probably background (Figure 9a). We declare an object to be a connected group of foreground pixels (Figure 9b). However, in our experience, this declaration results in some false positives. Hence, we filter the results based on normalized area and aspect ratio. The circled objects are used as ground truth to statistically determine if the others are outliers using quartiles. We iterate labeling outliers as truth until convergence. The objects marked as true are indicated in dark pink and the ones labeled as foreground from GrabCut but not filtered as true objects are highlighted in light pink (Figure 9c, Figure 8d).

Usually, this method finds many of the possible objects. Depending on which objects were selected, our filtering can be strict and result in numerous objects marked in light pink instead of dark pink. The user can refine this result by iterating with new markings for the foreground, background and smart object selection.

In addition, we experimented with a variety of machine learning algorithms (k-means, one-class SVM) to label the objects. These techniques did not return meaningful results because we usually only have three to eight ground truth examples to label around 10 to 100 other object instances.

Inpainting

Once all the objects are extracted, we use an implementation of PatchMatch to fill in the background holes [11] (Figure 8e). However, our system is modular so the inpainting method can be replaced by a more robust or specialized algorithm [16, 18, 24, 52]. In addition, the user can paint over additional areas to be filled. This interaction is helpful in the cases where we miss small parts of an object or a whole object due to occlusion.

Depth Ordering with Layers

Once again, we use GrabCut to split other parts of the image into more layers. To seed the process with foreground and background areas, we use a similar technique when finding the foreground inside a circled region. The pixels are separated into 30 groups using k-means on their LAB color. All groups that are either in the user defined background markings or in the absence of user markings, the top five non-transparent rows of the image, are set as the background. A single group that has the furthest average distance from all the background groups is selected as the foreground. All other pixels are marked as probably foreground. The result from GrabCut with this input is presented to the user. After refinements, the foreground area is inpainted using PatchMatch [11].

Animation Optimization

We show several methods to automatically enhance the creation and parameter tuning of an emitter to quickly create an animation that resembles the original still image.

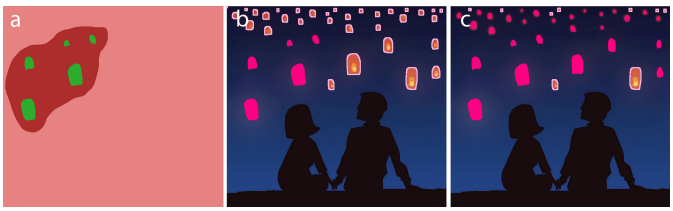

Figure 9: Finding similar objects. (a) The input to GrabCut. (b) Found objects from GrabCut. (c) Filtered found objects.

Emitter Creation

From the user drawn motion lines, we create a vector field [27] over the whole image and calculate an emitter line. To begin, we collect all exemplar and similar objects discovered in the object extraction step. From these objects, we calculate the minimum object area covered by the intersection of their expansion into a single merged area and their convex hull. Our initial guess of the emitter line is calculated by taking a line perpendicular to the motion line's beginning and finding its intersections with the convex hull's border. We push this line outside of the object area and then project the intersection points onto this new line (Figure 10a). These two endpoints serve as the initial guess to our emitter line.

In addition, we want the particles from our emitter to cover the whole region where the initial objects were located (Figure 10b). For each endpoint, we either contract it, until emitted particles travelling along the vector field switch from not intersecting the object area to intersecting it, or we expand it until particles that intersected the object area no longer do so. If the user draws multiple motion lines, we average the projected endpoints before checking for object area coverage (Figure 10c,d). To emit objects along the emitter line, we finely sample the line into emit points [27]. At each iteration, we determine, as described below, which emit point should emit an object. Each emit point chooses which exemplar object it will emit at that time.

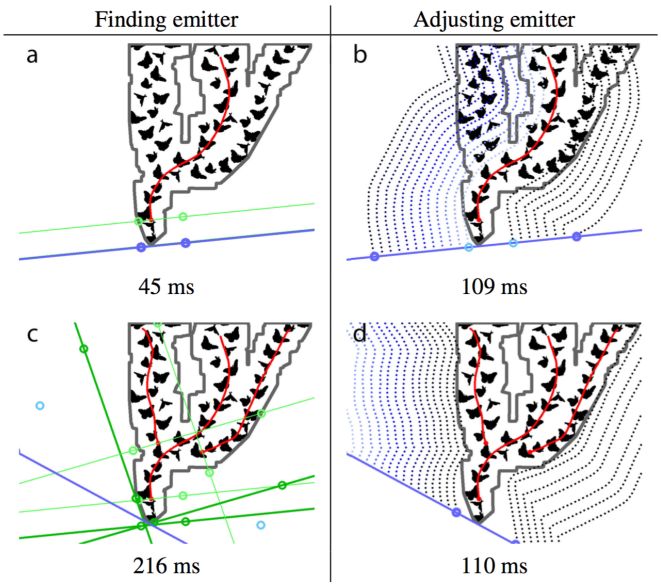

Figure 10: We automatically generate an emitter from user drawn motion paths (red). (a) First, we find the perpendicular line (light green) to the motion path and its intersection (light green) with the convex hull. Then, we push this line and its intersections (dark blue) outside the object area (dark gray). (b) From these initial points (light blue), we expand the emitter until particles completely cover the object area. These final dark blue points are the end points to the emitter. (c) When the user adds multiple motion paths, the endpoints (dark green) are averaged to create the light blue points. The line connecting these light blue points is pushed outside the object area. (d) The initial points contract to tightly cover the object area with emitted particles.

Emission Frequency

We control the emission frequency by setting the probability of emitting a new object at each time step. Our goal is to regulate the emission frequency such that the objects are distributed stochastically (but roughly uniformly) and the number of objects on the screen, *n*, is close to *N*, a target number based on the original image. We find *N* by determining the number of separate areas (connected components of pixels) found in the exemplar and similar object steps. All our results use *N* directly as found this way; but we also provide a control that allows the user to manually adjust this number up or down. Our supplemental materials show some examples of such adjustments. We emit a particle at time t_i based on the probability P_e as follows:

$$P_e(t_i) = \begin{cases} P_d(t_i - t_{i-1}), & \text{if } t_i < \frac{D}{v} \\ P_d(t_i - t_{i-1})C(n), & \text{otherwise} \end{cases}$$

This formulation splits the emission probability into two phases. An initial phase fills the object "pipeline" using a constant probability P_d of particle emission per time interval. Later, after the expected duration needed to fill the pipeline, we transition to a phase where a controller $C(n)$ modulates the emission probability upward or downward based on whether the current number of objects n is less than or greater than the target number N. The duration of the initial phase is the average distance D that objects travel along the vector field divided by their average velocity v. We set the "steady-state" emission probability P_d, to be the expected probability of an object disappearing from the scene, given by a similar calculation involving the time interval δ:

$$P_d(\delta) = \frac{Nv\delta}{D}$$

In principle, P_d and therefore P_e could exceed 1.0 when δ is large. However, we perform this test at a sufficiently high rate that this problem never occurs in practice. The controller $C(n)$ is a simple logistic curve, symmetric around the case where $n = N$ and scaled so $C(n) = 1$:

$$C(n) = \frac{2 * e^x}{e^x + 1}, \text{where } x = N - n$$

This controller ensures that the number of objects remains near the target number in the steady state, while allowing small fluctuations due to randomness. Figure 11a shows how the number of objects changes over the course of animation for the lantern result.

Finally, once we determine that a particle should be emitted, we need to choose an emission location from the emitter line. We choose randomly, weighting each potential emit point by its distance to the closest previously emitted object. This weight avoids emitting an object close to another existing object, giving the result a stochastic but more uniform distribution.

Auto-scaling

To automatically scale objects, we present two methods depending on whether we are scaling along a path or among layers (Figure 12). To scale objects along the motion path, the

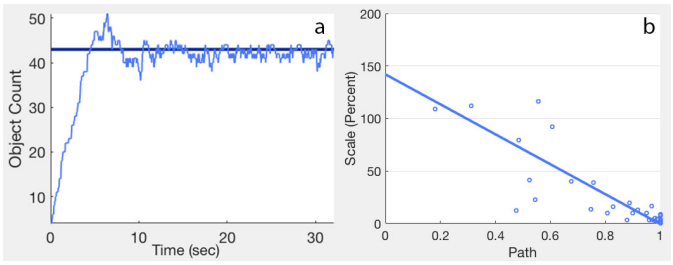

Figure 11: Our automatic animation calculations for the lantern example. (a) Changing the scene's number of objects over time. The dark blue line is N. (b) Fitting a line to objects' scale over the motion path.

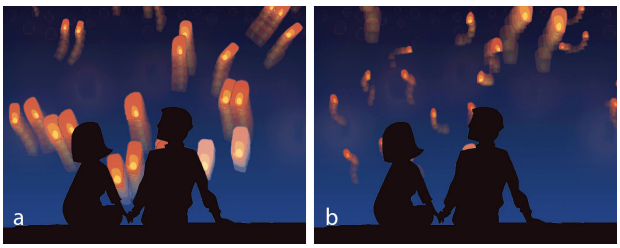

Figure 12: Ways to auto-scale objects. (a) Scaling over the motion path: objects scale up or down as they move along a path. (b) Separating into two layers: objects of different sizes move at different speeds.

y component is each object's pixel area divided by the average exemplar object pixel area. The x component is the percentage of motion path distance that object has traveled. We fit a line through these points and use it to scale the exemplar objects as they are emitted (Figure 11b, Figure 12a).

To scale objects among layers, we use k-means to cluster the exemplar objects based on their pixel areas. From each cluster, we create a new layer with only those objects, duplicating the emitter line and motion paths. To set the emission frequency of each layer, we cluster all the objects with k-means to determine N in each new layer. We set the speed of the largest object layer to be the same as the original layer. Each subsequent layer speed is proportional to that layer's average object area divided by the original layer's average object area (Figure 12b).

Looping

To create pleasing animations, we want the final exported video to loop seamlessly – the animated elements, including emitted particles, should match in the first and final frame. We address looping emitted particles separately from other animated objects. To begin, we record an animation sequence lasting from one to three minutes, making sure to encompass at least two full cycles of emitted particles.

To loop the emitter, we search for the time interval where the number and spacing of emitted objects is similar at the beginning and end (Figure 13a). In addition, we want the interval duration to be between two and four times the maximum lifetime of a particle. We define the distance D_{ij} between two particle emission windows starting at t_i and t_j as:

$$\sum_{k=0}^{K-1} \frac{K-k}{K} \Big(\mid (t_{i+k} - t_i) - (t_{j+k} - t_j) \mid + \mid e_{i+k} - e_{j+k} \mid \Big)$$

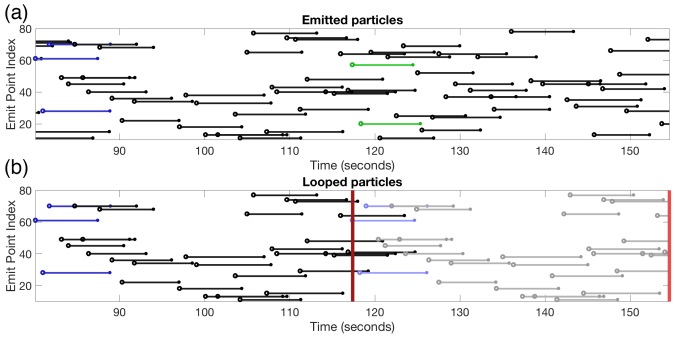

Figure 13: Finding an optimal interval over which to loop an emitter. (a) The windows with blue and green particles are the best matching groups. (b) Dark blue particles, and all black following up to the first red bar, are duplicated starting at the red bar (copies shown in light blue and gray). The red bars mark the start and end of the recording time interval, and the animations at these times match exactly.

where the ith particle is emitted from the emit point index e_i at time t_i. K is the number of emitted particles to compare between the two windows, and is proportional to the average emission frequency. We search over all possible animation intervals (with a minimum duration) seeking the one where the distance D_{ij} between the start and end is smallest. Search time is quadratic in the number of recorded particles. Once an optimal time interval is found, all of those particles are duplicated to the interval's end, forming the recording part of the looping animation (Figure 13b).

When looping other objects, we want their position, rotation, scale and alpha value to be C^1 across the loop seam. In addition, if there is an emitter in the scene, we keep the looping time interval the same as that previously found from the emitter. We search over all the saved animation data to determine the best interval which is close (within one second) to the time duration. Next, the attributes over the time interval are modified so that the position and tangent at the interval edges equals the average of the original values. We combine the original attribute values with a bezier curve formed by the difference of points and tangents at each edge point with the average. Finally, this time interval is remapped to exactly span the emitter time interval. Please see our supplemental materials for the looping animation results.

USER EVALUATION

To gain initial insights and feedback on our system, we ran an informal qualitative user evaluation.

Participants

We recruited six participants (four female and two male, aged 20-45) to test our system. Three subjects had extensive artistic experience and regularly created their own illustrations. Their animation skill level varied: one was a novice, four were experienced, and one was a professional. Four subjects had previous exposure to Sketchbook Motion [8]. Two subjects animated their own images (two drawings and two photographs) with our system (Figure 14). Participants were recruited from Autodesk and were compensated for their time.

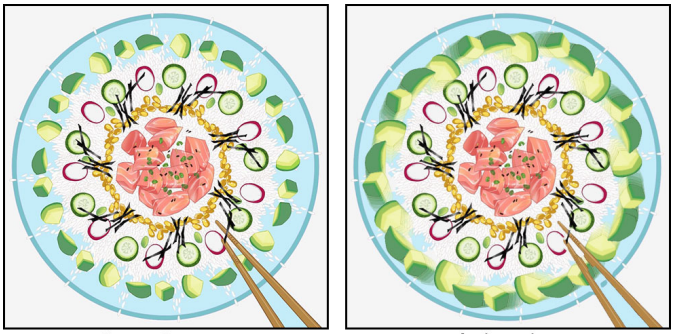

Figure 14: A participant animated one of her own previous illustrations.

Study Protocol

Each session lasted between one and two hours. To begin, we showed participants some results created with our system (Figure 15). This overview gave them a broad understanding of the animation types they would be creating. We demonstrated our system using the Snoopy example to extract the repeating snowflakes while interactively refining the results (Figure 15f). Once the snowflakes were extracted, we animated the example and auto-scaled the snowflakes into two layers. After answering any questions about the system, we had the participants animate the polka dots example (Figure 15c) as a simple warm-up task and the balloon example (Figure 1) as a more advanced testing task. When working on the polka dots example, we provided tips and suggestions if they were confused about the process. With the balloons, we did not provide suggestions while they were working with the system. After each example, we asked the participants questions about their experience.

Results and Observations

Participants found our system to be useful and mentioned that they would use it to produce animations for prototyping, social media, e-cards, background motions, and education. In their comments to us, they all reported that our system was faster than other tools, namely Photoshop [3] and AfterEffects [2]. They estimated that a similar task using those tools would take between 15 minutes to two hours compared to the six to ten minutes required for our system. One participant mentioned that it would have taken 30-40 minutes to redraw the image in Sketchbook Motion and reproduce the result. All agreed that other systems would be considerably more tedious to produce similar results. We also asked the users two questions ("How easy did you find the system to learn?" and "How easy did you find the system to use?") and recorded their responses as follows: *learn*: mean=3.8 (σ=0.42) and *use*: mean=3.9 (σ=0.49) – on a scale from 1="difficult" to 5="extremely easy."

When interacting with our system, subjects expressed a varied range of unique visions for their animations. (See the supplemental materials for examples.) When segmenting the polka dots image, some participants circled a range of colors and sizes for their exemplar objects. Other participants separated the polka dots by color into different layers. To achieve this result, they segmented the original image three times, with each time only extracting exemplar objects of a single color. For the balloon example, most participants circled a range of large, medium and small balloons for exemplar objects.

Some users had trouble remembering the steps of the segmentation process. Four were confused that only the circled exemplar objects appeared for animation. They thought the objects disappeared after step one, since we did not provide any visual indication of where they were. These issues could be resolved with an improved layout and icon indications. Also, four participants were unsure of what each brush, specifically the foreground and background brushes, could be used for. Hence, they had trouble refining the automatic results without help. Two users thought that the smart object selection brush circled the light pink objects instead of filling them. The amount of time in active use (without computation time) to extract the balloons ranged from 3 to 4.5 minutes.

During the animation process, all participants were pleased with the animations created. After drawing a motion line, most users adjusted the object speed. Some decided to replace our automatic emitter line with a manually drawn one. They were very enthusiastic about our method to auto split objects into different layers. When auto-scaling the balloons into multiple layers, some participants wanted to separate based on color or set a sliding scale for splitting by area. Users took between two and six minutes to animate their extracted objects. Despite some of these observed challenges, all users successfully animated the balloon image.

In addition to animating the balloon image, two users animated two images of their own. One participant extracted avocados from a food bowl image and animated them circling the other food (Figure 14). She also separated the chopsticks to place them on top of the animated avocados. In her other animation, rain fell behind a girl with an umbrella. Another user animated two of his photographs. In one, he extracted a hanging light and made it slowly swing. In the other, he extracted sparks and made them fly over a skateboard. Both users were impressed and pleased with the animations they added to their images. Please see the supplemental materials and video for the animations.

RESULTS

In addition to the four examples created by users, we also animated eleven other images. Six of these are displayed in Figure 15. For all the looping animated results, as well as the user input to create them, see our supplemental materials and video. The source pictures were in a variety of styles: illustrations, diagrams, infographics, photographs, and cartoons. Below we describe how the results were generated utilizing our mixed-initiative approach.

When extracting repeating objects, our method for finding objects inside the circles was completely automatic in four of the cases. Six examples required some additional marks to better define small parts of the foreground. For instance, some of the balloon connection strings (Figure 1 and Figure 15d) and black butterfly wing parts (Figure 15a) needed to be specified. When finding similar objects, seven examples needed some smart object markings to refine the filtering. In most cases, only a couple of objects needed to be marked. Two results,

the engine and Snoopy (Figure 15b,f), needed background markings for areas which were marked as objects but in reality, were not. When inpainting, three results used the inpaint brush to mark more areas to fill in.

To add more depth effects, nine results separated out large parts of the image into additional layers. For instance, the engine separated the piston and gear; the balloons separated the trees; and the frog separated the leaf (Figure 15b,d,e). With Snoopy, the snow, house and tree were extracted from the sky (Figure 15f).

For animation, most results only needed a single motion path. Three examples (the butterflies, engine and frog) needed multiple motion paths (Figure 15a,b,e). Five of our examples were complete after adjusting the speed. In two cases, we scaled the objects along the motion path. With four examples, we split the objects into two or three layers (Figure 15d,e,f). The layer with the smallest objects was positioned behind mid-ground layers (the frog and Snoopy in Figure 15e,f). To further enhance our animations, we added granular translation or rotation to the emitting objects [27]. We also adjusted the opacity on the lanterns (Figure 12) and butterflies (Figure 15a) so that the objects fade in as they appear.

Overall, the results show that it is possible to achieve pleasing animations across a spectrum of visual styles with our interactive and iterative interface. When our automated approaches did not produce the exact desired results, simple gestures could be used to quickly refine the results as necessary.

LIMITATIONS AND FUTURE WORK

While our system works well for a range of visual styles and effects, our participants pointed out several improvement opportunities. Since we rely on GrabCut to extract objects, we are constrained by its limitations. For instance, if objects vary in color and the user only circles objects of one color, we will miss labeling differently colored objects as foreground. However, the resulting animation would not resemble the source image because all emitted objects would be a single color. Another limitation of our algorithm is transparent objects. Since GrabCut only indicates each pixel as foreground or background, it cannot extract objects with transparency. Additional processing of GrabCut's output, or a new segmentation algorithm, is necessary to extract transparent objects [47]. In addition, our approach cannot separate overlapping objects, so they cannot appear in the final animation. However, they are easily removed from the background as shown in the balloons, raindrops, and snowflakes in Figure 15d,e,f.

Currently, our technique is suitable for animating repetitive elements using kinetic textures. However, to animate a variety of other phenomena such as secondary motion, fire, water, smoke, collisions, and clothes, physical simulation needs to be applied to the segmented objects [49]. In addition, mesh deformations or 3D models would also enhance the range of animation effects. These variations remain as future work.

During our user study, some participants had trouble remembering the function of each brush. For future UI improvements, we can improve the brush names and add tool tips for better explanation. Participants were also confused by whether a brush should be used for circling or filling in areas. Possibly changing the brush icons would help eliminate this confusion. Another suggestion was that splitting objects into multiple layers should occur before animation, not after. In the future, we hope to refine these workflow subtleties and conduct a more thorough evaluation of our system.

One drawback of our system is that the results are not created in real time. Extracting exemplar and finding similar objects took on average 10 seconds and at most 30 seconds. Our inpainting method took between four to eight minutes. The slow runtime of our inpainting process is mainly due to the particular iOS library implementation we incorporated. Inpainting can be fast, as demonstrated by Adobe Photoshop (desktop) and Adobe Photoshop Fix (iOS) [3], and is thus not an inherent limitation of our proposed process.

CONCLUSION

Creating animated pictures from scratch is challenging for amateurs and non-professionals. In this paper, we presented an approach that enables users to create animated pictures by taking a single static, raster image as input. By doing so, amateurs and professionals alike can leverage and animate existing pictures (irrespective of visual styles) in an interactive, iterative way, which is a very tedious process otherwise.

REFERENCES

1. A. Adams. 2017. Image Stack. (2017). **https://github.com/abadams/imagestack**
2. Adobe. 2017a. AfterEffects. (2017).
3. Adobe. 2017b. Photoshop. (2017).
4. Narendra Ahuja and Sinisa Todorovic. 2007. Extracting texels in 2.1 D natural textures. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE, 1–8.
5. Zainab AlMeraj, Craig S. Kaplan, and Paul Asente. 2013. Patch-based Geometric Texture Synthesis. In *Proceedings of the Symposium on Computational Aesthetics (CAE '13)*. ACM, New York, NY, USA, 15–19. DOI: **http://dx.doi.org/10.1145/2487276.2487278**
6. Zainab AlMeraj, Craig S. Kaplan, Paul Asente, and Edward Lank. 2011. Towards Ground Truth in Geometric Textures. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering (NPAR '11)*. ACM, New York, NY, USA, 17–26. DOI: **http://dx.doi.org/10.1145/2024676.2024679**
7. Christine Alvarado and Randall Davis. 2001. Resolving ambiguities to create a natural sketch based interface. In *Proceedings of IJCAI-2001*, Vol. 18. 1365–1371.
8. Autodesk. 2016. Sketchbook Motion: An app for adding movement to your art. **https://www.sketchbook.com/blog/motion-new-app-sketchbook-pro-members/**. (2016). Accessed: 2017-09-4.
9. Jiamin Bai, Aseem Agarwala, Maneesh Agrawala, and Ravi Ramamoorthi. 2013. Automatic cinemagraph

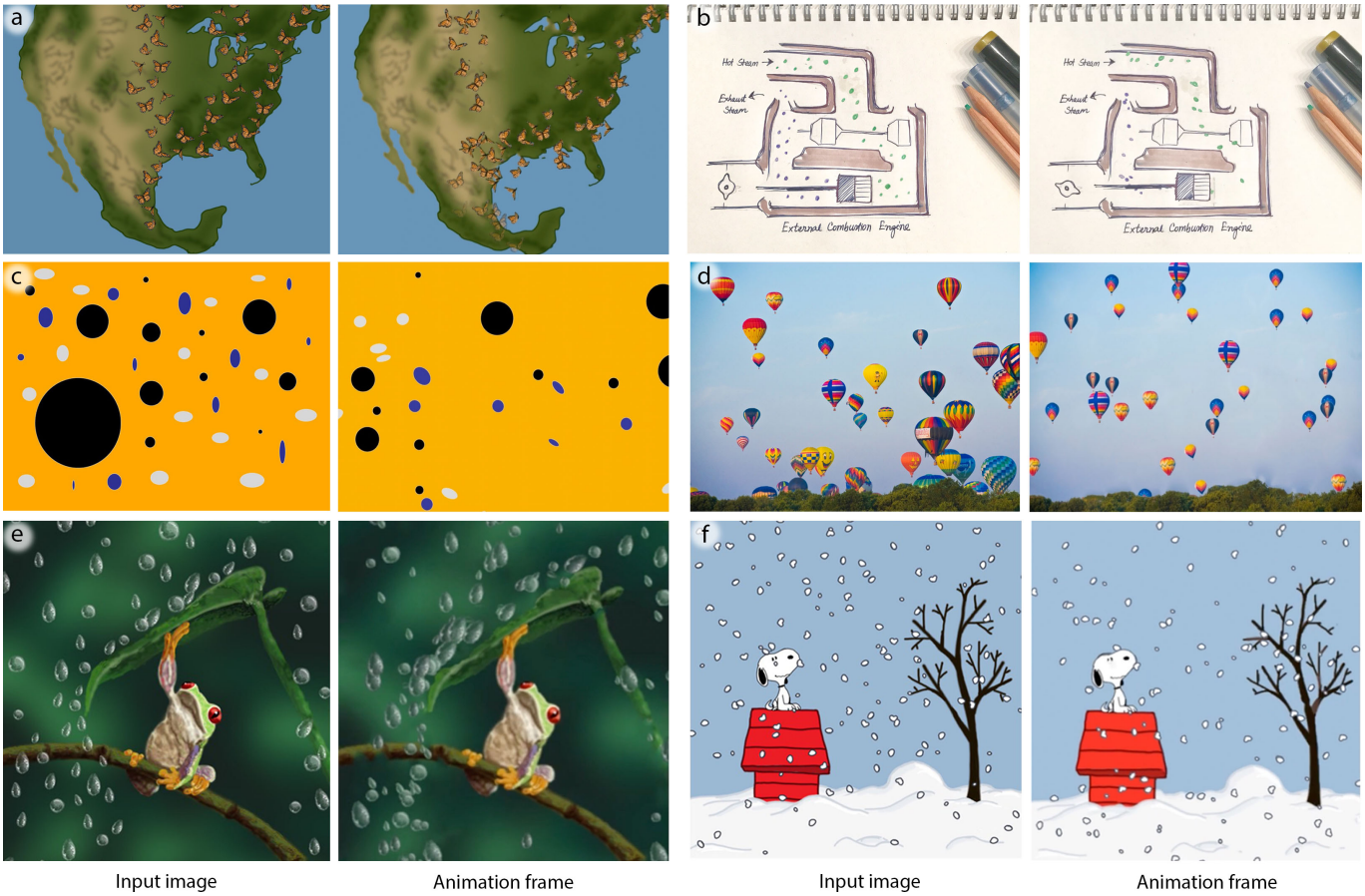

Figure 15: Some results created with our system. For animations and other details, see our supplemental materials and video.

portraits. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 17–25.

10. Yunfei Bai, Danny M. Kaufman, C. Karen Liu, and Jovan Popović. 2016. Artist-directed Dynamics for 2D Animation. *ACM Trans. Graph.* 35, 4, Article 145 (July 2016), 10 pages. DOI: **http://dx.doi.org/10.1145/2897824.2925884**

11. Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. 2009. PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing. *ACM Trans. Graph.* 28, 3, Article 24 (July 2009), 11 pages. DOI: **http://dx.doi.org/10.1145/1531326.1531330**

12. Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. 2000. Image Inpainting. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 417–424. DOI: **http://dx.doi.org/10.1145/344779.344972**

13. G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

14. Ming-Ming Cheng, Fang-Lue Zhang, Niloy J. Mitra, Xiaolei Huang, and Shi-Min Hu. 2010. RepFinder: Finding Approximately Repeated Scene Elements for Image Editing. *ACM Trans. Graph.* 29, 4, Article 83 (July 2010), 8 pages. DOI: **http://dx.doi.org/10.1145/1778765.1778820**

15. Yung-Yu Chuang, Dan B Goldman, Ke Colin Zheng, Brian Curless, David H. Salesin, and Richard Szeliski. 2005. Animating Pictures with Stochastic Motion Textures. *ACM Trans. Graph.* 24, 3 (July 2005), 853–860. DOI: **http://dx.doi.org/10.1145/1073204.1073273**

16. Antonio Criminisi, Patrick Perez, and Kentaro Toyama. 2003. Object removal by exemplar-based inpainting. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, Vol. 2. IEEE, II–II.

17. Richard C. Davis, Brien Colwell, and James A. Landay. 2008. K-sketch: A 'Kinetic' Sketch Pad for Novice Animators. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 413–422. DOI: **http://dx.doi.org/10.1145/1357054.1357122**

18. Iddo Drori, Daniel Cohen-Or, and Hezy Yeshurun. 2003. Fragment-based Image Completion. *ACM Trans. Graph.* 22, 3 (July 2003), 303–312. DOI: **http://dx.doi.org/10.1145/882262.882267**

19. FlixelStudios. 2017. Flixel: Magical Tools for Visual Storytelling. **https://flixel.com/**. (2017). Accessed: 2017-08-10.

20. William T Freeman, Edward H Adelson, and David J Heeger. 1991. *Motion without movement*. Vol. 25. ACM.

21. Tovi Grossman, Fanny Chevalier, and Rubaiat Habib Kazi. 2015. Your Paper is Dead!: Bringing Life to Research Articles with Animated Figures. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '15)*. ACM, New York, NY, USA, 461–475. DOI: **http://dx.doi.org/10.1145/2702613.2732501**

22. Youichi Horry, Ken-Ichi Anjyo, and Kiyoshi Arai. 1997. Tour into the Picture: Using a Spidery Mesh Interface to Make Animation from a Single Image. 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, 225–232. DOI: **http://dx.doi.org/10.1145/258734.258854**

23. Takeo Igarashi, Tomer Moscovich, and John F. Hughes. 2005. As-rigid-as-possible Shape Manipulation. In *ACM SIGGRAPH 2005 Papers (SIGGRAPH '05)*. ACM, New York, NY, USA, 1134–1141. DOI: **http://dx.doi.org/10.1145/1186822.1073323**

24. Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. 2017. Globally and Locally Consistent Image Completion. *ACM Trans. Graph.* 36, 4, Article 107 (July 2017), 14 pages. DOI: **http://dx.doi.org/10.1145/3072959.3073659**

25. Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine-Hornung. 2014. Bounded Biharmonic Weights for Real-time Deformation. *Commun. ACM* 57, 4 (April 2014), 99–106. DOI: **http://dx.doi.org/10.1145/2578850**

26. Neel Joshi, Sisil Mehta, Steven Drucker, Eric Stollnitz, Hugues Hoppe, Matt Uyttendaele, and Michael Cohen. 2012. Cliplets: Juxtaposing Still and Dynamic Imagery. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. ACM, New York, NY, USA, 251–260. DOI: **http://dx.doi.org/10.1145/2380116.2380149**

27. Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, Shengdong Zhao, and George Fitzmaurice. 2014. Draco: Bringing Life to Illustrations with Kinetic Textures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 351–360. DOI: **http://dx.doi.org/10.1145/2556288.2556987**

28. Rubaiat Habib Kazi, Tovi Grossman, Nobuyuki Umetani, and George Fitzmaurice. 2016. Motion Amplifiers: Sketching Dynamic Illustrations Using the Principles of 2D Animation. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 4599–4609. DOI: **http://dx.doi.org/10.1145/2858036.2858386**

29. Yan Kong, Weiming Dong, Xing Mei, Xiaopeng Zhang, and Jean-Claude Paul. 2013. SimLocator: robust locator of similar objects in images. *The Visual Computer* 29, 9 (2013), 861–870.

30. David H. Laidlaw. 2001. Loose, artistic" textures" for visualization. *IEEE Computer Graphics and Applications* 21, 2 (2001), 6–9.

31. Thomas Leung and Jitendra Malik. 1996. Detecting, localizing and grouping repeated scene elements from an image. In *European Conference on Computer Vision*. Springer, 546–555.

32. John P Lewis. 1995. Fast template matching. In *Vision interface*, Vol. 95. 15–19.

33. Zicheng Liao, Neel Joshi, and Hugues Hoppe. 2013. Automated Video Looping with Progressive Dynamism. *ACM Trans. Graph.* 32, 4, Article 77 (July 2013), 10 pages. DOI: **http://dx.doi.org/10.1145/2461912.2461950**

34. Peter Litwinowicz and Lance Williams. 1994. Animating images with drawings. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM, 409–412.

35. Ce Liu, Antonio Torralba, William T. Freeman, Frédo Durand, and Edward H. Adelson. 2005. Motion Magnification. *ACM Trans. Graph.* 24, 3 (July 2005), 519–526. DOI: **http://dx.doi.org/10.1145/1073204.1073223**

36. Hugo Loi, Thomas Hurtut, Romain Vergne, and Joelle Thollot. 2017. Programmable 2D Arrangements for Element Texture Design. *ACM Trans. Graph.* 36, 3, Article 27 (May 2017), 17 pages. DOI: **http://dx.doi.org/10.1145/2983617**

37. Gioacchino Noris, Daniel Sỳkora, A Shamir, Stelian Coros, Brian Whited, Maryann Simmons, Alexander Hornung, M Gross, and R Sumner. 2012. Smart scribbles for sketch segmentation. In *Computer Graphics Forum*, Vol. 31. Wiley Online Library, 2516–2527.

38. S O'Kane. 2015. Apple's new Live Photos feature turns your pictures into videos. **https://www.theverge.com/2015/9/9/9296829/apple-live-photos-feature-iphone-6s**. (2015). Accessed: 2017-09-4.

39. Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. 2004. "GrabCut": Interactive Foreground Extraction Using Iterated Graph Cuts. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 309–314. DOI: **http://dx.doi.org/10.1145/1015706.1015720**

40. Frederik Schaffalitzky and Andrew Zisserman. 1999. Geometric grouping of repeated elements within images. In *Shape, Contour and Grouping in Computer Vision*. Springer, 165–181.

41. Arno Schödl, Richard Szeliski, David H Salesin, and Irfan Essa. 2000. Video textures. In *Proceedings of the*

27th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co., 489–498.

42. Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence* 22, 8 (2000), 888–905.

43. Mikio Shinya, Masakatsu Aoki, Ken Tsutsuguchi, and Naoya Kotani. 1999. Dynamic Texture: Physically Based 2D Animation. In *ACM SIGGRAPH 99 Conference Abstracts and Applications (SIGGRAPH '99)*. ACM, New York, NY, USA, 239–. DOI: **http://dx.doi.org/10.1145/311625.312130**

44. Qingkun Su, Xue Bai, Hongbo Fu, Chiew-Lan Tai, and Jue Wang. 2018. Live Sketch: Video-driven Dynamic Deformation of Static Drawings. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA.

45. Meng Sun, Allan D Jepson, and Eugene Fiume. 2003. *Video input driven animation (VIDA)*. IEEE.

46. Daniel Sỳkora, John Dingliana, and Steven Collins. 2009. Lazybrush: Flexible painting tool for hand-drawn cartoons. In *Computer Graphics Forum*, Vol. 28. Wiley Online Library, 599–608.

47. Jianchao Tan, Marek Dvorožňák, Daniel Sýkora, and Yotam Gingold. 2015. Decomposing Time-lapse Paintings into Layers. *ACM Trans. Graph.* 34, 4, Article 61 (July 2015), 10 pages. DOI: **http://dx.doi.org/10.1145/2766960**

48. James Tompkin, Fabrizio Pece, Kartic Subr, and Jan Kautz. 2011. Towards moment imagery: Automatic cinemagraphs. In *Visual Media Production (CVMP), 2011 Conference for*. IEEE, 87–93.

49. Nora S. Willett, Wilmot Li, Jovan Popovic, Floraine Berthouzoz, and Adam Finkelstein. 2017b. Secondary Motion for Performed 2D Animation. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 97–108. DOI: **http://dx.doi.org/10.1145/3126594.3126641**

50. Nora S. Willett, Wilmot Li, Jovan Popovic, and Adam Finkelstein. 2017a. Triggering Artwork Swaps for Live Animation. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 85–95. DOI: **http://dx.doi.org/10.1145/3126594.3126596**

51. Jun Xing, Rubaiat Habib Kazi, Tovi Grossman, Li-Yi Wei, Jos Stam, and George Fitzmaurice. 2016. Energy-Brushes: Interactive Tools for Illustrating Stylized Elemental Dynamics. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 755–766. DOI: **http://dx.doi.org/10.1145/2984511.2984585**

52. Chao Yang, Xin Lu, Zhe Lin, Eli Shechtman, Oliver Wang, and Hao Li. 2017. High-resolution image inpainting using multi-scale neural patch synthesis. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Vol. 1. 3.