

Kitty: Sketching Dynamic and Interactive Illustrations

Rubaiat Habib Kazi¹, Fanny Chevalier², Tovi Grossman¹, George Fitzmaurice¹

¹Autodesk Research ²INRIA

rubaiat.habib@gmail.com, fanny.chevalier@inria.com, {tovi.grossman, george.fitzmaurice}@autodesk.com

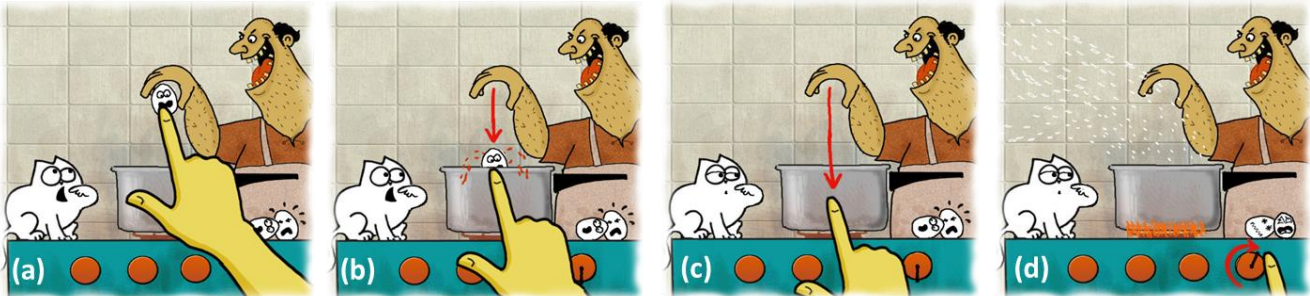


Figure 1: Example of a dynamic interactive illustration authored with Kitty. (a) Objects in the scene are interactive: the egg held by the cook can be dragged down, as if falling into the pot, triggering subsequent animations, such as soup splashes (b) and closing of the cat’s eyelids (c). Turning the knob increases the fire and steam (d). The resulting dynamic illustration captures the living nature of the scene, where the gas stove flames burn and steam emits from the pot.

ABSTRACT

We present Kitty, a sketch-based tool for authoring dynamic and interactive illustrations. Artists can sketch animated drawings and textures to convey the living phenomena, and specify the functional relationship between its entities to characterize the dynamic behavior of systems and environments. An underlying graph model, customizable through sketching, captures the functional relationships between the visual, spatial, temporal or quantitative parameters of its entities. As the viewer interacts with the resulting dynamic interactive illustration, the parameters of the drawing change accordingly, depicting the dynamics and chain of causal effects within a scene. The generality of this framework makes our tool applicable for a variety of purposes, including technical illustrations, scientific explanation, infographics, medical illustrations, children’s e-books, cartoon strips and beyond. A user study demonstrates the ease of usage, variety of applications, artistic expressiveness and creative possibilities of our tool.

Author Keywords

Sketching; interactive illustrations; causal animation.

ACM Classification Keywords

H.5.2. Information interfaces and presentation: User Interfaces – Graphical Interface.

“Dynamic pictures will someday be the primary medium for visual art and visual explanations” – Bret Victor [27]

INTRODUCTION

Due to the recent advances in mobile devices and web technologies, there is a growing demand for interactive illustrations to enrich the content found on mediums such as websites, e-books, and teaching materials. In contrast to static pictures and videos that are consumed by a passive observer, interactive illustrations respond to a user’s input, providing a more playful or informative experience. For example, an interactive illustration of a mechanical system invites exploration through interaction as users investigate hypotheses, which may help develop insights about significant relationships and causal patterns [12, 13].

Traditionally, crafting such interactive illustrations is achieved through programming or scripting [1, 2, 6]. This requires the artist to work with abstract textual representations of visual entities, their interactive behavior, and their relationships to other entities. To alleviate these prerequisite programming skills, researchers and practitioners have developed notable tools for authoring interactive illustrations, relying on techniques such as drag-and-drop interaction [20], sketching [17] and visual programming [27]. While powerful, these tools are usually tailored for special purpose applications (for example, simulating blood flow [28]).

In this paper, we present Kitty, a sketch-based tool for authoring a wide range of interactive dynamic illustrations (Figure 1). Kitty builds upon Draco [8], a recent system that allows users to create dynamic illustrations – timeless, continuous animated textures that capture the living quality of a scene. Dynamic illustrations are driven by parameters, encoding *visual, spatial, temporal* and *quantitative*

information of the entities. Kitty extends Draco to incorporate interactivity and functional relationships between these parameters into the illustrations.

Kitty supports both an authoring and presentation mode. An artist (*author*) can produce illustrations to be interacted with, while a viewer (*user*) can directly manipulate an entity's parameters (*e.g.* turn the knob in Figure 1) which causes subsequent changes of the parameters of the related entities and animations (*e.g.* flames burn more intensely).

To specify the interactive behavior of entities, Kitty relies on an underlying *relational graph* structure, where nodes correspond to the entities and edges represent the functional correspondence between the entities' parameters. Kitty displays this graph in the context of the illustration, and authors can specify edges and their associated functions through freeform sketching. This enables an intuitive interaction, without requiring any programming or scripting.

Before presenting the Kitty user interface, we describe a general framework that can support the authoring of interactive dynamic illustrations. The generality of this framework makes our tool applicable for a variety of purposes, including technical and medical illustrations, scientific explanation, infographics, children's e-books, cartoon strips and beyond. A user study demonstrates the ease of usage, variety of applications, artistic expressiveness and creative possibilities of our tool. We conclude by discussing limitations and future opportunities.

RELATED WORK

The breadth of related works in crafting dynamic illustrations is large. Here, we focus our review on prior representative tools for authoring animated illustrations, explanatory illustrations, and sketch-based user interfaces.

Crafting Dynamic Illustrations

Many tools for crafting various types of dynamic graphics have been proposed, including tools for creating animations [1, 2, 8], presentations [14], and visualizations [4, 6, 7]. Of particular relevance are tools that provide support to specify relationships between entities and interactivity.

Support for Relational Dynamics

In Flash [1] and visualization toolkits [4, 6, 7], entities' parameters can be accessed as variables in scripts that enable coordination between them. Other professional animation tools [2, 3] allow users to specify procedural relationships between the attributes of visual entities by scripting 'Expressions' with equations. While powerful, these tools' lack of interactive graphical support to ease the manipulation of the relationships between entities, and require users to deal with abstract textual representations. Scratch [20] is another system relying on visual programming for children to create their own animations with programmable constructs. Rather than programming, Kitty provides an interface where relationships can be defined by directly manipulating the underlying relational graph, displayed in the context of the illustration.

Support for Interactivity

Flash [1] enables the creation of interactive media, but again, this is achieved through programming, like in visualization toolkits [4, 6, 7]. Presentation tools [14] also support interactivity, though they are typically limited to triggers which progress an animation through a linear global timeline. Kitty capitalizes on freeform sketching, and allows artists to directly sketch the correspondence function between visual parameters. Such an approach not only facilitates the authoring of expressive relational patterns, but the different entities in the resulting interactive dynamic illustrations can also be directly manipulated.

Systems for Visual Explanatory Illustrations

Static visual explanatory illustrations (*i.e.* diagrams) are effective for communicating design ideas, technical assembly, medical systems and scientific concepts [24]. Tools have been proposed to manually [28] or automatically [16] generate such diagrams. To depict dynamics, they typically rely on arrows and text, which do not convey timing and manner of movements and require users to *infer* relations rather than *perceive* them.

When carefully designed, animated diagrams can yield benefits beyond their static counterparts [25]. Most of the tools to create dynamic diagrams rely on physical modeling [23] or are designed for special-purpose applications, such as explaining medical phenomena [26] and illustrating dynamic fluid systems [28], which limits variety of usage.

In both cases, these tools often employ a graph model to represent the dynamics of a system [16, 28], an effective conceptual representation to encode relationships. We employ a similar model in Kitty, which the artist can directly manipulate through sketching. The generality of such an approach makes our tool expressive, and applicable to a wide array of application domains.

Sketch-Based User Interfaces

Sketching is a powerful tool for rapid design and visual thinking, and many sketch-based applications have been explored for domains such as interface prototyping [9, 18], mathematical expressions [10] and data storytelling [11].

Most relevant to our work, Draco [8] is a tool designed to ease animation authoring through freeform sketching. Draco allows users to create rich animation effects using *kinetic textures* – sets of similar objects with coordinated motion. Unlike previous sketch-based animation tools, Draco facilitates the animation of object collections, while providing global and granular motion controls. In spite of their expressiveness, the resulting artifacts are not interactive and do not capture the functional relationship between the visuals. Kitty builds on and extends Draco to depict the dynamic relationships and make them interactive. Kitty relies on the sketch-based paradigm, and provides a general framework for the creation of a variety of types of interactive dynamic illustrations.

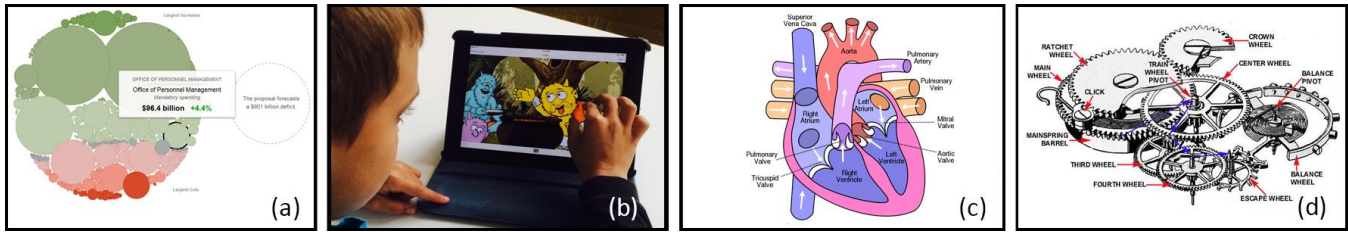


Figure 2: Example domains that could benefit from dynamic and interactive illustrations. (a) An infographics with a bubble set depicting the US budget allocation, in New York Times, (b) a children's e-book illustration of a monster scene, (c) a medical illustration explaining blood circulation of the heart, and (d) a technical illustration showing how a mechanical watch works.

MOTIVATION: APPLICATIONS AND PRINCIPLES

With the current growth and advancement of multimedia devices and web technologies, there are a number of domains where dynamic and interactive illustrations could be valuable or are already being deployed, such as interactive infographics and children's e-books (Figure 2). Dynamic illustrations have numerous advantages over static representations. They can be more compelling and engaging, and they can also help depict situational dynamics of phenomena [25]. The associated authoring tools, however, often come with a complex interface and typically require programming skills, which greatly limits the targeted audience and require a tedious process that may discourage even the most experienced users. We aim to alleviate this challenge, by providing an easy-to-use, yet expressive tool for authoring interactive dynamic illustrations through freeform sketching and direct manipulation.

A dynamic illustration consists of visual entities, driven by parameters representing *information* that changes over time. This information is consumed and assimilated by observers to build *knowledge*. In order to provide an effective tool for authoring dynamic illustrations, we need to understand how our brain processes a visual scene and what types of information is absorbed. This understanding is imperative to provide effective support to construct the visual entities, parameterize the visuals and capture the dynamics.

The Visual Thinking Process

In his influential work *The Back of the Napkin* [21], Roam illustrates a simple perceptual framework depicting how our brain processes information, based upon prior neurological findings [19]. Roam breaks down the visual thinking process into the “six W’s” as follows:

- *Object* Who and What
- *Quantity* How many and how much
- *Position in Space* Where
- *Position in Time* When
- *Relations* How
- *Interpretation* Why

When processing a visual scene, we first identify the *objects* – visually distinguishable entities, *e.g.* a dog, a doorknob – involved, and in which *quantity* they appear. Each of these objects and collections of objects occupies a

specific location in the scene, *e.g.* the dog on the right, the trees in the background. Given a spatial referential, objects configuration can be characterized by transformations with regard to the origin (*i.e.* translation, rotation and scale). In a dynamic representation, the objects themselves and their position can change over time. Our vision system also tracks these *temporal* variations, *e.g.* the dog walks to the left. Such changes are usually depicted using simple animation. All three above components (what, where and when) can effectively define a dynamic visual scene. But there is much more that the human brain processes than the individual behavior of objects: we also extract functional relations between objects, including coordination and cause-and-effect (*e.g.* throwing a pebble in a pond results in water splashes and ripples).

The last two points of Roam’s framework refer to such *relations* (how) and subsequent knowledge we construct through *interpretation* (why). In other words, when a particular event occurs and triggers a chain effect, we infer functional relationships and the impact of one thing upon another, resulting in new knowledge about causal patterns.

Interactivity

Prior literature on education and learning suggests that correlations and cause-and-effect relationships are best apprehended when the observer is given control over the pace of the animation, [12, 13, 25]. Thus, not only is it important for dynamic illustrations to accurately convey information, this information will also be best assimilated if the observer can manipulate the entities directly.

To summarize, a dynamic visual scene consists of *visual entities* with *spatial* and *temporal* properties. Most research efforts have been devoted to facilitating the authoring of such parameters at the individual level of objects and object collections, while editing functional relations between them – though critical to the understanding of a dynamic scene – has not been directly addressed. In traditional animation tools, the artist is required to manually coordinate temporal events on a linear global timeline, a tedious process that leaves little opportunities for interactive exploration of individual relations. Kitty rather focuses on the importance of depicting and presenting functional relationships between visuals entities and depicting knowledge in an interactive fashion. In the following, we describe the framework that we designed to meet these goals.

BACKGROUND: DRACO KINETIC TEXTURES

Before describing our new framework, we first provide necessary background information on the Draco system [8] which Kitty is built upon. Draco introduced *kinetic textures*, a novel animation component that encodes collections of objects and their associated motion.

A kinetic texture consists of a *patch* and *motion properties*. The patch is a set of representative objects for the collection. The motion properties – *global motion* and *granular motion* – define an object’s movement. Draco introduced two types of kinetic textures: *emitting textures* and *oscillating textures* (Figure 3).

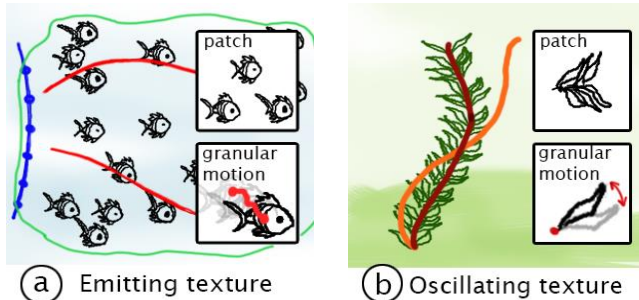


Figure 3: Kinetic textures: (a) Emitting texture, defined by a source patch, emitter (blue), global motion paths (red) and granular motion, and (b) oscillating texture, defined by a source patch, brush skeleton (brown), oscillating skeleton (orange), and granular motion. Figure taken from [8].

Emitting Textures

Emitting textures continuously emit a stream of objects, following a global motion trajectory guided by motion path(s) (e.g. fish swimming) (Figure 3a). The objects emanate from an *emitter*, and follow a global motion trajectory, which is guided by an underlying motion vectors field. This field is computed from a set of *motion path(s)*. Additional emitting texture attributes include a *texture outline* and *mask(s)*. Objects are removed when they cross the outline, and hide as they pass through a mask. Additional attributes control the dynamics of the animation: the *emission velocity* controls velocity of the objects and the *emission frequency* controls the frequency of emissions.

Oscillating Textures

Oscillating textures consist of a duplicated collection of elements from the patch along a *brush skeleton* (e.g. leaves along the stem of a plant) (Figure 3b). The global motion of an oscillating texture is defined by the oscillatory movement between the initial brush skeleton and a target *oscillating skeleton*. Oscillating textures can also have a *texture outline* and *mask(s)*. Additionally, an oscillating texture has a *velocity* attribute which defines the speed of the global oscillations.

Granular Motion

Granular motions can also be applied to individual objects of the collection: *translation* defines a two-dimensional path that the individual objects move within (Figure 3a) and *rotation* defines a pivot point and path that the objects rotate

about (Figure 3b). A *velocity* attribute controls the speed of any granular motion. The final trajectory and orientation of individual objects in the collection results from the combination of the global and granular motions.

DYNAMIC INTERACTIVE ILLUSTRATIONS: FRAMEWORK

With respect to our motivation, Draco’s kinetic textures satisfy a subset of the visual thinking process, but provide no interactivity. Based on these remaining open challenges, we designed a framework to further support the visual thinking process and also provide interactivity. Our framework consists of 1) The *objects and collections* which serve as the visual entities in a scene, 2) the *parameters* defining the visual properties of these entities, 3) a *relational graph* structure which defines the functional relationships between the entities, and 4) *interactivity* of the elements in the dynamic illustration.

Objects and Collections (What and How Much)

A visual scene comprises one or multiple identifiable visual entities characterized by *attributes* that define their visual (e.g. color, shape) and spatial (e.g. location, scale) properties. In dynamic illustrations, visual entities can be animated, which adds a set of *temporal attributes* (e.g. motion). In our framework, we refer to individual animated entities as *objects*, defined by a set of strokes and their attributes. An individual animated object can have a continuous ambient motion, e.g. a boat rocking on the sea, and also global motion, e.g. the same boat moving forward.

Our framework supports collections of objects using *kinetic textures* [8]. We use both emitting textures and oscillating textures as described above, which can possess both global and granular motion.

Objects and kinetic textures form the base components in dynamic illustrations. They are characterized by attributes that define their visual and temporal properties. We also use a subset of these attributes as *parameters* in functional relations and for input control.

Parameters (Where, When, and How Much)

We refer to an attribute as a *parameter* if its value can be changed in response to user interaction, or based on other visual entities’ behaviors. Parameters are used as the basis of our system to create a dependency mapping between the visual entities. While any attribute could possibly be considered as a parameter, we discuss here only the subset of attributes that we will use for our implementation of Kitty. Our choices were guided by informal pilot studies which helped us identify attributes that would be the most useful to have control upon in our framework.

For individual objects, the parameters are the *translation* (along a trajectory), *rotation* (about a pivot), *scale*, *opacity* and *ambient speed* of the object’s continuous ambient motion.

The global parameters for the kinetic textures include *opacity*, *emission frequency*, and *emission speed* for

emitting textures and *opacity* and *skeleton length* for oscillating textures. The skeleton length defines how long the brush skeleton extends. The *granular velocity* serves as a single local parameter.

Relational Graph (How and Why)

In a dynamic scene, visual entities can interact with one another, depicting the coordination and cause-and-effect relations between them. Similar to prior work on explanatory illustrations [16, 28], we use an underlying *relational graph* to capture such information, where nodes correspond to visual entities and edges encode the functional relations between entities' parameters (Figure 4).

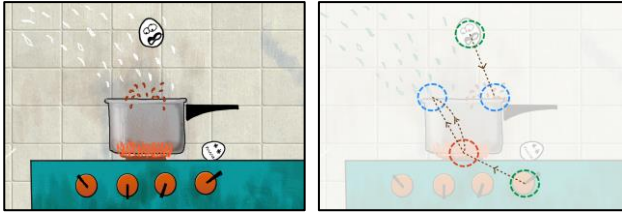


Figure 4: A simplified version of Figure 1 (left) and its associated relational graph (right).

Nodes

Nodes in the relational graph represent the visual entities – objects and kinetic textures. The graph of Figure 4 has five nodes, corresponding to two objects (green nodes), two emitting textures (blue) and one oscillating texture (orange). Each node has an associated set of parameters that can be used to create a functional dependency with other nodes.

Edges

Edges in the relational graph capture the functional relationship between the parameters of a source node and a target node. Each edge is directed, and has an associated *functional mapping* – a correspondence function that encodes the dependent behavior from a specific parameter of the target node to a specific parameter of the source node. For example, in Figure 4 the steam intensity (emission frequency parameter) is a function of the rotation of the knob (global rotation parameter): the more the knob is turned, the higher the emission frequency will be.

Two types of events could cause a source parameter to change. If the source entity is *interactive*, then the associated parameter could be directly manipulated. Alternatively, the source parameter itself could be the target of another relation, making it an intermediary parameter in a *chain effect*. In both cases, the target parameter will also change, as defined by the edge's functional mapping.

Functional Mapping

To specify the dependent behavior of a target parameter based on the status of a source parameter, we rely on a two-dimensional function as follows:

$$t_p = f(s_p)$$

where s_p and t_p denote the value of the source and the target parameters of the linked visual entities.

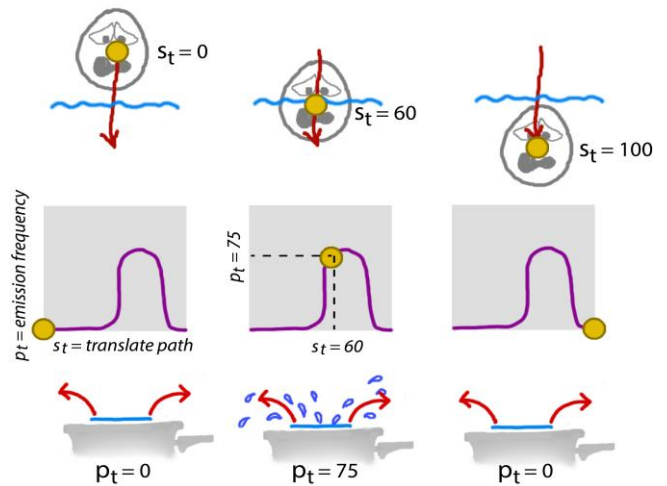


Figure 5: Functional relationship between the translation path of the egg and the emitting frequency of soup splashes.

For example, in Figure 4 the egg is interactive. Moving the egg into the pot causes the soup to splash. Here, the relation involves the translation of the egg as the source parameter s_p , and the emission frequency of the soup splashes as the target parameter t_p . The associated mapping function f is such that emission frequency is highest when the egg hits the surface (Figure 5).

Chain Effects

Multiple edges can be combined in sequence, resulting in a chain of relations across the parameters. In Figure 4, the global rotation parameter of the knob (object) impacts the scale parameter of the fire (oscillating texture), which in turn impacts the emission frequency of the steam (emitting texture). The resulting chain effect is such that when turning the knob, the fire increases in intensity, which causes more intense steam from the pot.

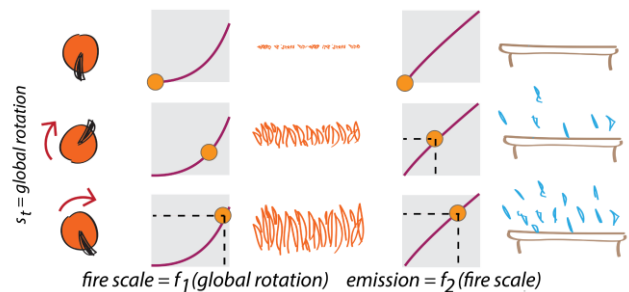


Figure 6: Chain effect with functional relationships.

Interactivity (How and Why)

The interactivity of a visual entity is defined at the node level. Each node can have one of its parameters assigned for input control, to enable manipulation. For example, in Figure 4, the egg's global translation parameter, and the knob's global rotation parameter are both interactive. Note that a visual entity does not necessarily have to be involved in a relation to be interactive.

KITTY: IMPLEMENTATION

We designed and implemented Kitty, a sketching tool to author interactive dynamic illustrations. Artists can create animated objects and textures and specify how the parameters of an illustration’s visual entities are related and respond to each other. Artists can also define how these objects respond to user interaction. Figure 7 shows the Kitty user interface, consisting of a main canvas, as well as global and contextual toolbars. In addition to supporting authoring, Kitty also provides a presentation mode for both authors and subsequent viewers to interact with the illustrations.

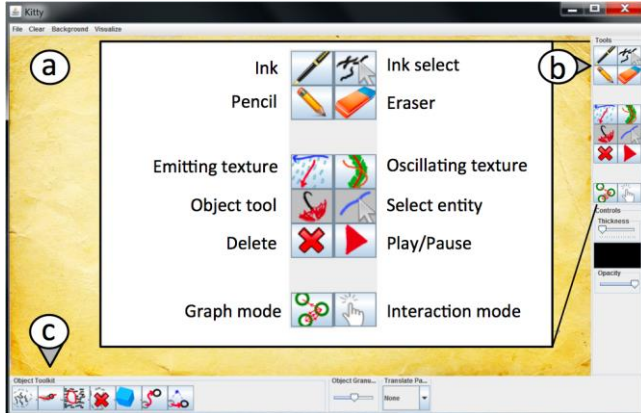


Figure 7. Kitty user interface comprising: (a) the main canvas, (b) the global toolbar and (c) contextual toolbar.

Global Toolbar

The global toolbar is comprised of the main drawing tools and tools for creating animated objects (Figure 7b). It also provides access to a graph mode, where the artist can define the underlying relational graph, and to an interaction mode.

The *ink* tool is the primary drawing tool, and has an associated color widget, brush thickness slider, and opacity slider. Artists can select a group of ink strokes with the lasso *stroke selection* tool to move or delete them. A *pencil* tool is also available to draw a rough outline of an illustration, which can be deleted with the *eraser* tool.

Dynamic and interactive entities are created using the *object* tool, *emitting texture* tool, and *oscillating texture* tool. The artist can also select an existing visual entity with the *entity select* tool and subsequently edit its properties using the tools in the contextual toolbar (Figure 7c).

Creating Visual Entities

Kitty supports three types of animated visual entities: objects, emitting textures and oscillating textures.

Creating and Editing Objects

To create an object, the artist starts by sketching the set of strokes defining the object using the *ink* tool (Figure 8a). She then defines these strokes as components of a new object, by clicking on the *object* tool in the global toolbar and selecting the strokes with a lasso (Figure 8b).

The object contextual toolbar allows the artist to further configure the created object: the *translation path* tool is

used to sketch a global translation path; the *mask* tool is used to draw a mask region where the object is hidden; and the *ambient rotation* and *ambient translation* tools allow the artist to define continuous motion. For ambient rotation, a pivot point can first be dragged to set the origin of rotation (Figure 8c). Specifying ambient rotation and translation is done through direct manipulation of the associated object (Figure 8d).

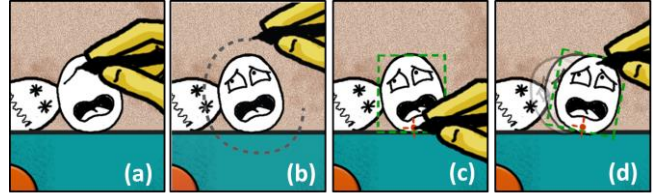


Figure 8: Creating and editing objects. (a) The artist sketches an egg with the *ink* tool and (b) defines the strokes as a new object. (c) Upon creation, the bounding box (green outline) and default pivot point (in red) are displayed. The artist drags the pivot point to a new location, and (d) specifies a continuous motion through direct manipulation using the *ambient rotation* tool.

Creating and Editing Emitting and Oscillating Textures

We used similar interactions to Draco’s for authoring kinetic textures. We refer the reader to the Draco system for a complete description of the interactions [8]. Sketching the example strokes and an emitter creates an *emitting texture*, while sketching example strokes and a brush skeleton creates an *oscillating texture*. The example strokes are also displayed in a small patch widget which can be used to add granular motions.

Similar to objects, the artist can edit the components of the currently selected texture through the contextual toolbar. For an emitting texture, she can edit the *motion paths* and *emitter*. For an oscillating texture, she can define or edit the *oscillating skeletons*. In addition, for both types of kinetic textures, the artist can define a *mask* and *outline*, and induce granular motion to the texture by directly manipulating the strokes in the example patch widget.

Defining Functional Relationships

To create functional relationships between the visual entities, the artist is required to enter the *graph mode* from the global toolbar. The system animates to an abstract graph view of the visual entities and their relations, displayed in the context of the illustration canvas (e.g. Figure 4). Each entity in the illustration is automatically associated to a node in the relational graph.

We use a force-directed layout algorithm [5] to guarantee minimal distance between nodes, facilitating their selection. The nodes are also color-coded to distinguish between objects (green nodes), emitting textures (in blue), and oscillating textures (in orange). When exiting the graph mode, visual entities snap back to their original location in the illustration. Creating a functional relationship between two visual entities follows a three-step process.

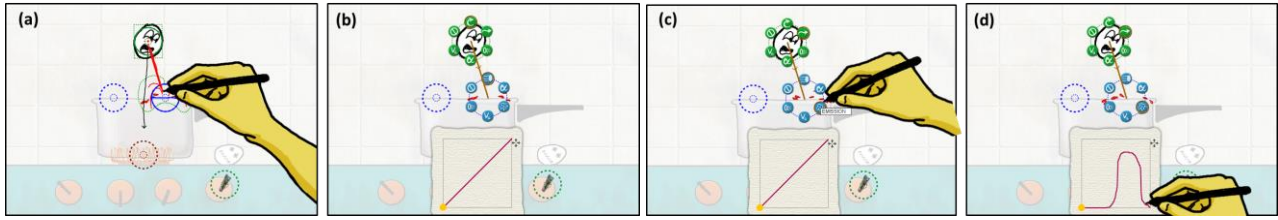


Figure 9: (a) Sketching a path between the source object node (the egg) and the target emitting texture node (soup splash), creates a relation between the two visual entities. (b) Upon creation, the relation control widget (radial menus and mapping function widget) are made visible. (c) The artist selects the parameters for the relation, then (d) directly sketches the mapping function.

Step 1: Specify the source and target nodes.

To connect two nodes in the graph, the artist simply sketches a stroke from the source node to the destination node (Figure 9a). This creates an edge between the two nodes. All edges are directed, but if desired, bi-directional edges can be created by specifying two separate edges in opposite directions.

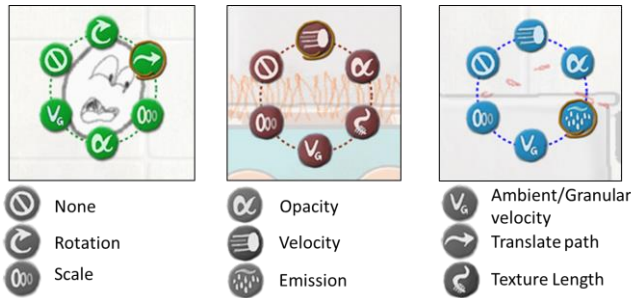


Figure 10: The parameter radial menus for an object (left), an oscillating texture (middle) and an emitting texture (right).

Step 2: Select parameters.

Upon creation of an edge, Kitty displays all candidate parameters of the source and target nodes in radial menus (Figure 9b). Figure 10 illustrates the radial menus and parameters associated with each of the three node types. The artist can select the pair of particular parameters for the functional mapping by clicking on the corresponding icons on the menus (Figure 9c). Each edge is associated to one source parameter s_p and one target parameter t_p . Creating multiple edges between the same nodes allows an author to define mappings for additional pairs of parameters.

Step 3: Specify the functional mapping.

When an edge is created or selected, a function mapping widget visualizes the functional correspondence between the associated source parameter s_p (x axis) and the target parameter t_p (y axis). By default, this function is set to identity ($t_p = s_p$), which the artist can overwrite by directly sketching a new curve within the widget (Figure 9d). The widget is equipped with an *exploration cursor* that the artist can drag within the mapping space to help identify relevant mapping points: when moving this cursor, the visual entities are automatically updated to preview the corresponding source and target parameter values.

Kitty also allows an artist to load a mapping function from an external quantitative data file (csv format), enabling

precise functional relationships between visual entities. This can be particularly useful in the domain of infographics.

Editing Edges and Nodes

Existing edges and nodes can be selected to adjust their associated parameters and mapping functions. A selected edge can also be deleted using the *delete* tool in the global toolbar. Deleting an edge removes the functional relation between entities. Deleting a node also deletes all of the associated edges in the graph.

Setting Interactivity

The interactivity of objects can also be specified in the graph mode. This is accomplished by tapping on a node and selecting the desired interaction parameter in the radial menu (or the *none* icon for no interactivity). In the resulting interactive illustration, all visual entities with a defined interaction parameter are considered to be interactive and can be manipulated through direct manipulation.

Presentation Mode

The artist can switch to the *presentation mode* to test the resulting interactive dynamic illustration. The presentation mode is also used by subsequent viewers of a saved illustration.

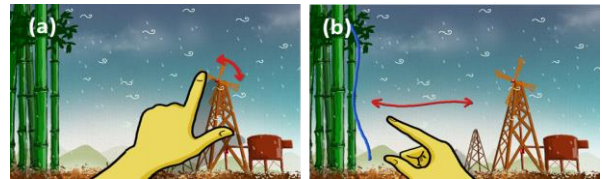


Figure 11: Interaction guides as the user hovers over the canvas. a) Direct manipulation is used for spatial parameters, such as rotating the windmill. b) Swiping gestures are used for non-spatial parameters, such as increasing the wind speed.

Kitty displays red visual cues when the user hovers over an interactive visual entity with a mouse or stylus, or when the user taps the screen when using direct touch. These guides indicate the possible interactive gestures (Figure 11). For spatial parameters (e.g. rotation and translation), the expected direct manipulations are used, such as directly rotating the windmill (Figure 11a). For other parameters, such as emission frequencies, a swiping gesture towards and away from the entity will decrease and increase the parameter value respectively. Figure 11b shows an example of increasing the wind emission speed.

As the user performs a gesture, the value of the corresponding parameter changes, and all subsequent dependent parameters in the illustration are updated in real-time. A global setting can be used to toggle the visual cues, to accommodate the desired level of guidance.

Chain effects are carried out by traversing the relational graph, starting at the node of interaction. Any edge which would induce a cycle is not traversed. This prevents infinite loops while still allowing for bi-directional relationships.

Resulting Interactive Dynamic Illustrations

Inspired from the examples that motivated us to develop Kitty (Figure 2), we authored several interactive dynamic illustrations using our tool (featured in Figure 13-Figure 15 and in the accompanying video): a weather infographics, a technical illustration showing how an electrical bell works, a comic-style animation illustrating the passing of time, and a children's illustration. These results help demonstrate the creative capabilities and expressive power of Kitty.

USER EVALUATION

We conducted a user evaluation with both professional and amateur animators and illustrators, to gain insights about the capabilities and potential applications, and identify any limitations or opportunities for future advancements. This study was also used to gather feedback on how our system compares to existing approaches, although we do not perform any sort of formal comparison to existing commercial tools.

Participants

We recruited eight participants (2 female) aged 24-45 for our study. Participants completed a questionnaire about their background before the study began: all participants have moderate to good experience with sketching and illustration and are proficient with sketch-based digital painting tools. Two were professional animators (*P5*, *P8*), and the six other participants had little to moderate experience with animations.

Study Protocol

The study was conducted using the Wacom CINTIQ 21ux tablet display at our laboratory or at the professional animators' studios. The evaluation lasted for approximately 60 minutes, consisting of the three following steps.

Overview and training (20-25 minutes). Participants completed five training tasks designed to familiarize them with the system's drawing tools, kinetic textures, visual entities functional relationships, and interaction modes. The facilitator provided light guidance during these tasks.

Exercise task (15-25 minutes). At the end of the training, participants were given an exercise task (Figure 4) to reproduce with Kitty. The exercise task required participants to author five visual entities, including two objects (egg, knob), two emitting textures (soup splash, emitting smoke) and one oscillating or emitting texture (fire) on top of a provided background image. Participants were also required to coordinate and create the four

functional relationships between the entities. A video of the target scene was prompted in a separate display to facilitate the task. The facilitator recorded observations but did not intervene during the exercise task.

Freeform usage (20-25 minutes). After completing the exercise task, participants were free to explore the tool on their own to craft illustrations. Participants then completed a questionnaire about the system.

Results

In general, participants responded positively to the user interface and functionality of Kitty. Participants commented that crafting visual entities and specifying their functionalities was easy to learn and use. On a 5-point Likert scale (1=very difficult to 5=very easy), participants rated 4.4 on average for the ease of crafting visual entities (min: 4) and also 4.4 for the ease of specifying functional relationships (min: 4).

All participants commented that specifying and sketching the functional relationship was simple to comprehend and easy to use. We were encouraged to see that all of them were very quick at picking up the mechanisms behind the graph structure and the mapping function. The visualization in the graph mode (color coded nodes, radial menus, edges) facilitated the association between the graph elements and their corresponding visual entities (objects and textures).

Participants commented that crafting interactive illustrations using the conceptual relational graph in Kitty would be tedious and time consuming with other tools. Our participants were proficient with Flash (all but *P1* and *P2*), ToonBoom Harmony (*P8*) Maya (*P2*, *P5*), PowerPoint (*P1*, *P6*, *P7*), After Effects (*P4*) and Lightroom (*P5*) to create animated and interactive contents.

P1: "Very nice to use, fluid interaction; can create results that I cannot imagine doing any other way."

P6: "Just four tools to do anything, I cannot get myself lost! For the use I have of animation [Note: presentations], it's more than enough and give me finer control than timeline."

P8 [pro]: "I only bothered adding interactivity to animations for my website. I know how to make them with Flash, but programming the most basic interactions takes me forever. I'm really impressed I made this [exercise task] so quickly."

P1 pointed out that functional relations in Kitty provide an elegant alternative way to coordinate animations, conceptually easier to create and manipulate in contrast to timeline based 2D animation systems that require planning. *P5* and *P7* also found that the timeless nature of Kitty alleviates planning requisites, enabling a more spontaneous and playful authoring.

P1: "I use PowerPoint to create animated technical diagrams, having a timeline with relative events starting one after another. So, if I have to change anything, I have to re-order all the transitions [...] this [system] is much more flexible to co-ordinate between the animations."

P3: "I like the ability to create functional animated graphics in a matter of minutes... It's almost like being able to do Flash stuff without learning ActionScript and being bothered by timeline... I have to prepare a day to figure out how to do this (exercise task) with Flash."

P6: "If I had this tool in PowerPoint, I would put interactive illustrations everywhere in my presentations. For now, I merely create very basic animations, because it takes me too much time to craft something satisfying."

Exercise Task Performance

All the participants were able to complete the exercise task without assistance. On average, the participants took 22 minutes (min 9:30, max 34:00) to complete the exercise task. This is encouraging to us, indicating that after only a short training period, participants could accomplish realistic tasks in reasonable time duration. During the studies, our eight participants encountered 10 errors in total. Errors include drawing an edge in wrong direction (twice by one participant), tool mix-up when participants created an emitting texture when the intended effect was an oscillating texture (twice), and choosing the wrong parameter (three times, the task required to select emission, but the participant chose velocity). A common mistake was that participants did not initially specify the interactivity of the nodes (8 times). This was easily resolved in all cases, but participants had assumed that interactivity was automatically specified. Several participants accidentally added ambient rotation to the knob, causing the knob to rotate continuously.

Observed strategies

Participants used different strategies to author the entire illustration. Three participants broke down the process into three phases, first drawing all of the visual entities involved in the illustration, then setting interactivity for all of them, and finally specifying the relationship with repeated switches between the graph mode and the presentation mode for fine tuning. The other five participants broke down the process by relations instead. For each of the three relations, they started with the creation of the source and the target nodes, then directly entered the graph mode to specify the relation before checking the result in the presentation mode. Almost all the participants used control points to coordinate the animations and the functional relationships, which they commented was a useful addition for quick exploration of the mapping space.

Overall, the strategies used demonstrate that Kitty requires little planning; we observed the participants complete the subtasks as they came to mind. This suggests that the underlying graph abstraction was a suitable design.

Follow-Up Usage

Two of the participants volunteered to continue to use the tool after the study and sent us sample illustrations which they created. Figure 16 demonstrates that even a simple relational graph can create a compelling illustration, while Figure 17 demonstrates a visually rich illustration which

consists of numerous entities and functional relationships (see also the accompanying video figure).

DISCUSSION

Our evaluation indicates that users are able to quickly grasp the concepts of the Kitty system and author their own illustrations after only a short training period. Our observations of user strategies demonstrated that the system is flexible to a variety of effective workflows. The sample content created by two of our participants, in addition to the samples which we have created, demonstrates that Kitty successfully provides a sketch-based interface for authoring a variety of rich interactive and animated effects.

While these are all encouraging results, there are also a number of limitations and opportunities for future improvements that warrant discussion. Below we report and discuss the most interesting insights gathered from our own experiences and observations made during the evaluation.

Limitations

Grouping of Visual Entities

Currently, Kitty does not support the definition of groups of visual entities. Objects and kinetic textures are the atomic components of our system and cannot be grouped into collections. Such hierarchical groupings would allow for richer animation expression, and could also speed up repetitive operations. Some participants mentioned that the lack of this capability was limiting in their creative process. In particular, grouping visual entities into hierarchical-objects would allow the coordinated movement of multiple animated elements that together form some higher level object (P5, P6).

Grouping objects or relations would also allow authors to apply common parameters to a set of visual entities in a single interaction. Participants said it would be helpful if they could replicate the same mapping function across relations (P7, P8). One participant wanted a smoke emitting texture to follow the global motion of a car (P5); another participant wanted several oscillation textures' phases to be synchronized, which is not currently supported.

We are confident that these types of grouping and coordination operations could be added to the Kitty system without introducing significant complexity to the interface.

Time-varying parameters

Kitty occupies an interesting space between the timeless dynamic illustrations composed of looping animated textures, and traditional animations based on a global timeline. The timeless nature of Kitty's visual entities is at the root of the underlying conceptual framework. However, some participants mentioned that they would like to have controls for more sophisticated storytelling, such as deformations over time, or the redefinition of the motion paths and masks of visual entities according to another entity's parameter (P2, P7, P8).

Other suggestions made by participants included support of physics such as momentum, which could make resulting artifacts look more natural. In the future, it would be interesting to explore the combination of sketch-based and physics-based animation effects. We also hope to further explore the scalability of Kitty, to see if additional considerations would be required to accommodate significantly larger or more complex illustrations.

Applications

Participants pointed out a number of potential application scenarios apart from artistic use cases, including technical and scientific diagrams, presentations, education material, interactive comics, e-books, prototyping, data journalism, point-and-click games, and more broadly, visual media for the web. Currently, we use our own file format to store interactive illustration authored with Kitty. As suggested by one of the participants (P8), the ability to export dynamic illustrations in an HTML5 format, would certainly contribute to the adoption of Kitty.

CONCLUSION

We have presented Kitty, a sketching tool to author animated and interactive illustrations driven by parameters. Kitty enables artists to create functional correspondence between visual entities with freeform sketching, offering a new way to author dynamic systems. Kitty is particularly beneficial in that dynamic and interactive illustrations can be authored in the matter of minutes, without any scripting or programming prerequisites. The timeless and interactive nature of the resulting illustrations allows subsequent viewers to play and replay specific animations or effects within an illustration. Overall, participants found the associated framework to be simple, yet expressive. We believe tools like Kitty requiring no programming language might place crafting high quality interactive and dynamic contents in the hands of a broader audience.

REFERENCES

1. Adobe. Flash. www.adobe.com/products/flash.html
2. Autodesk. 3D Studio Max. tinyurl.com/Adsk-3Dsmax
3. Autodesk. Maya. tinyurl.com/Adsk-maya
4. Bostock, M., Ogievetsky, V., & Heer, J. (2011). D³ data-driven documents. *Visualization and Computer Graphics*, IEEE Transactions on, 17(12), 2301-2309.
5. Fruchterman, T. M., & Reingold, E. M. (1991). Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11), 1129-1164.
6. Fry, B. Processing. www.processing.org.
7. Heer, J. Prefuse toolkit. prefuse.org/
8. Kazi, R. H., Chevalier, F., Grossman, T., Zhao, S. & Fitzmaurice, G. (2014) Draco: Bringing Life to Illustrations with Kinetic Textures. *ACM CHI*. In press.
9. Landay, J. A. (1996). SILK: sketching interfaces like crazy. *ACM CHI Conference companion*, 398-399.
10. LaViola, J. J., & Zeleznik, R. C. (2007). MathPad 2: a system for the creation and exploration of mathematical sketches. *SIGGRAPH courses*.
11. Lee, B., Kazi, R. H., & Smith, G. (2013). SketchStory: Telling more engaging stories with data through freeform sketching. *IEEE Trans. on Vis. and Comp. Graphics*, 19(12), 2416-2425.
12. Lowe, R. (2001). Beyond 'Eye-candy': Improving learning with animations. *Apple University Consortium*.
13. Lowe, R. (2003). Animation and learning: selective processing of information in dynamic graphics. *Learning and instruction*, 13(2), 157-176.
14. Microsoft. PowerPoint. microsoft.com/powerpoint
15. Miller, T., & Stasko, J. (2001). The InfoCanvas: information conveyance through personalized, expressive art. *ACM CHI Extended Abstracts*. 305-306.
16. Mitra, N. J., Yang, Y. L., Yan, D. M., Li, W., & Agrawala, M. (2010). Illustrating how mechanical assemblies work. *ACM Trans, on Graphics*, 29(4), 58.
17. Moscovich, T., & Hughes, J. F. (2001). Animation sketching: An approach to accessible animation. *Master's Thesis*, CS Department, Brown University.
18. Newman, M. W., Lin, J., Hong, J. I., & Landay, J. A. (2003). DENIM: An informal web site design tool inspired by observations of practice. *Human-Computer Interaction*, 18(3), 259-324.
19. Ramachandran, V.S., & Blakeslee, S. (1998). Phantoms of the brain: Probing the mysteries of the human mind.
20. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., & Kafai, Y. (2009). Scratch: programming for all. *Communications of the ACM*, 52(11), 60-67.
21. Roam, D. (2008). The back of the napkin.
22. Schwan, S., & Riempp, R. (2004). The cognitive benefits of interactive videos: Learning to tie nautical knots. *Learning and Instruction*, 14(3), 293-305.
23. Scott, J., & Davis, R. (2013, October). Physink: sketching physical behavior. *ACM UIST adjunct*. 9-10.
24. Suwa, M., & Tversky, B. (2002). External representations contribute to the dynamic construction of ideas. *Diagrammatic representation and inference*. 341-343.
25. Tversky, B., Heiser, J., Mackenzie, R., Lozano, S., & Morrison, J. (2008). Enriching animations. *Learning with animation*, 304-356.
26. Vainio, T., Hakkarainen, K., & Levonen, J. (2005). Visualizing complex medical phenomena for medical students. *ACM CHI*. 1857-1860.
27. Victor, B. (2009). Drawing Dynamic Visualizations. *CUSE*.
28. Zhu, B., Iwata, M., Haraguchi, R., Ashihara, T., Umetani, N., Igarashi, T., & Nakazawa, K. (2011). Sketch-based dynamic illustration of fluid systems. *ACM Trans. on Graphics*, 30(6), 134.

APPENDIX: AUTHOR AND PARTICIPANT GENERATED EXAMPLES

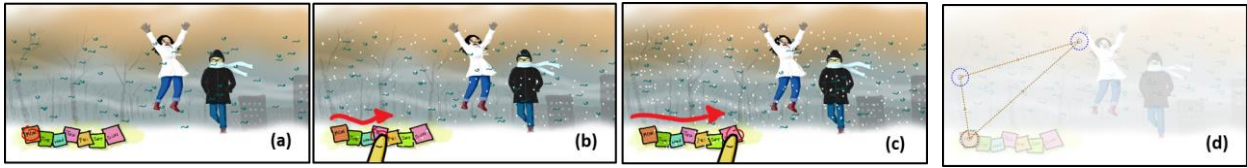


Figure 12 [author generated]: Animated weather infographics which updates dynamically as the user slides the calendar marker.

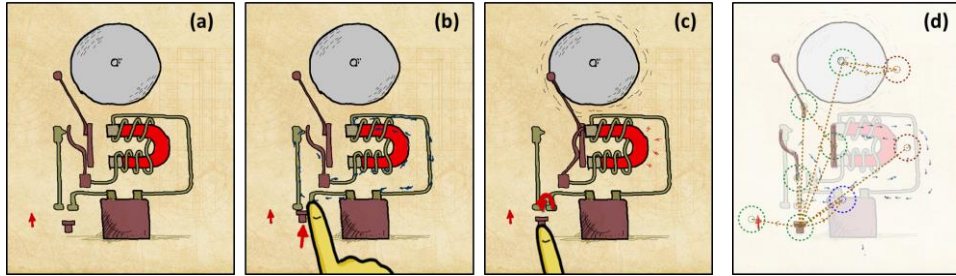


Figure 13 [author generated]: An interactive and animated mechanical diagram illustrating how an electric bell works.

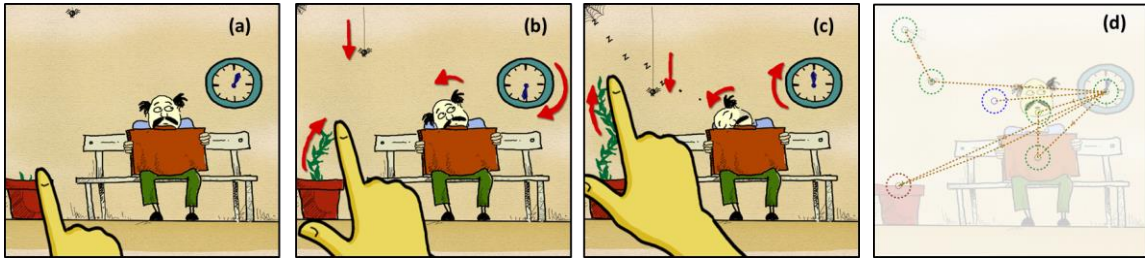


Figure 14 [author generated]: An interactive comic-style animation. Time passes as the user drags the plant or the clock hand.



Figure 15 [author generated]: An interactive children's illustration. Animations respond to user interactions.

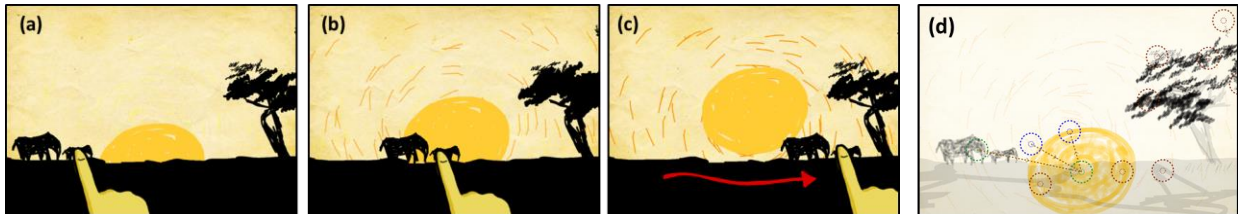


Figure 16 [participant generated]: A user drags the elephant across the horizon, and the sun rises.

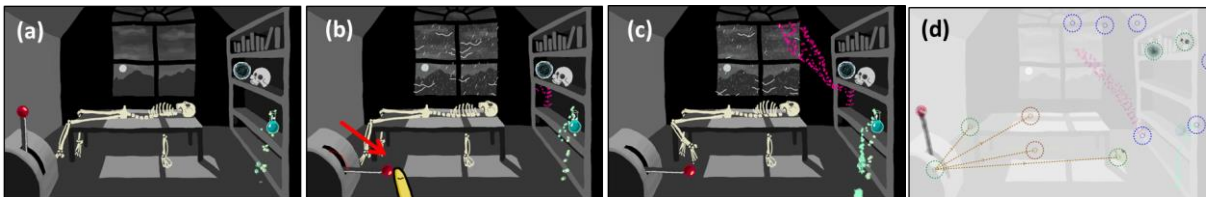


Figure 17 [participant generated]: Dr. Frankenstein's Laboratory comes to life when the user pulls the lever.