Esquisse: Using 3D Models Staging to Facilitate the Creation of Vector-based Trace Figures

Axel Antoine[1], Sylvain Malacria[2], Nicolai Marquardt[3], and Géry Casiez[1]

[1] Univ. Lille, CNRS, Centrale Lille, Inria, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France
{axel.antoine, gery.casiez}@univ-lille.fr
[2] Inria Lille - Nord Europe, France
sylvain.malacria@inria.fr
[3] University College London, UK
n.marquardt@ucl.ac.uk

Abstract. Trace figures are contour drawings of people and objects that capture the essence of scenes without the visual noise of photos or other visual representations. Their focus and clarity make them ideal representations to illustrate designs or interaction techniques. In practice, creating those figures is a tedious task requiring advanced skills, even when creating the figures by tracing outlines based on photos. To mediate the process of creating trace figures, we introduce the open-source tool *Esquisse*. Informed by our taxonomy of 124 trace figures, *Esquisse* provides an innovative 3D model staging workflow, with specific interaction techniques that facilitate 3D staging through kinematic manipulation, anchor points and posture tracking. Our rendering algorithm (including stroboscopic rendering effects) creates vector-based trace figures of 3D scenes. We validated *Esquisse* with an experiment where participants created trace figures illustrating interaction techniques, and results show that participants quickly managed to use and appropriate the tool.

Keywords: Trace figures · 3D models staging · Vector graphics · Blender

1 Introduction

Trace figures are static illustrative figures created to capture the essence of a situation, removing unnecessary details by limiting the graphical representation to the most important contours/outlines of the shown objects and people (for example, the visual abstract of our Figure 1 includes several trace figures). Trace figures, even when augmented with graphical annotation overlays (such as text, arrows or circles), minimize visual clutter and are effective at describing the essence of an interactive scenario.

Likely because of their clarity and focus, trace figures are gaining popularity in the HCI community, especially for illustrating a novel interaction technique

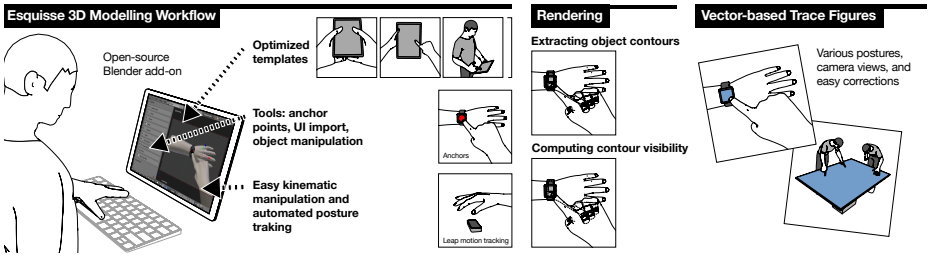

Fig. 1: Workflow of *Esquisse*: (left) facilitating staging of 3D scenes with templates, anchor points, direct UI import, kinematic manipulation and automated posture tracking, (center) *Esquisse*'s algorithm rendering the tracing images, and (right) final vector-based trace figures.

or interactive systems. As a result, they can be found in many papers, presentations or posters: over three years of UIST proceedings [1,2,3] for instance, more than 25% of the published papers used trace figures, often used to demonstrate gestures, provide an overview of a system setup or illustrate a design space.

Producing trace figures from scratch without a model requires very good drawing skills. Therefore, many users have to rely on manual photo tracing in order to produce the outlines of devices and people shown in the illustrations [9]. This manual tracing is a time-consuming and tedious task and eventually results in a limited workflow where even small changes of the drawn people, postures or devices might require starting again the manual tracing process from scratch. This is even more of a problem in cases where several hand postures must be produced or when users are required to change the point of view of the illustration. So far, research on the production of illustrations has been mostly focused on facilitating the production of new types of illustrations such as animated or interactive illustrations [19,18], whether they illustrate interactive scenarios or not. However, despite the frequent use of trace figure to illustrate interactive scenarios, and the challenges of the process to produce them, we are lacking software tools that are targeted at facilitating their production.

In this paper, we support the hypothesis that 3D staging is a good alternative to mediate the production of static trace figures to illustrate interactive scenarios. More precisely, we present *Esquisse*, an open-source tool implementing an innovative 3D staging workflow for producing trace figures (Figure 1). In particular, the underlying idea behind *Esquisse* is to capitalize on the flexibility of a 3D modeling software for staging 3D scenes that are subsequently rendered as a trace-figure vector file that can still be post-edited in a vector-graphics software afterwards. We provide 3D models, tools and techniques to make this workflow accessible to every people even if they do not have previous experience with 3D modeling software.

The workflow implemented by *Esquisse* works as follows. Users open a 3D scene using one of the provided templates. If needed, they can directly modify the position and pose of the objects using interaction techniques provided by

Esquisse that simplify the manipulation of these 3D objects. For instance, users can easily specify a hand posture using a Leapmotion hand tracking camera[4], or by adding a *contact point anchor* which will snap a desired finger to this contact point and modify the hand pose according if the point is moved on the display. They can also add other avatars and objects to the scene if required. Users then position the camera in the 3D space according to their needs. They can also integrate interface screenshots in various displays. Optionally they can add a stroboscopic effect to illustrate the motion of a finger or object. A simple click to a *Render* button then calls *Esquisse*'s rendering algorithm which generates a trace figure of the scene, integrating all added interface images, and outputs a vector-based file (examples of trace figures produced with *Esquisse* can be seen in Figure 1 and other figures throughout the paper; further renderings can be found in the Supplementary Material).

In summary, *Esquisse*'s workflow facilitates the rapid production of trace figures, as well as the adaptation of existing figures. Our work makes the following contributions: (i) introducing a novel workflow for quickly creating vector based trace figures; (ii) presenting the design of interaction techniques to facilitate this workflow; (iii) developing an innovative rendering pipeline, adapted from existing techniques; (iv) implementing this workflow into a Blender add-on. To allow members of the HCI community to create trace figures with *Esquisse*, it is released as a free open-source[5] add-on for the 3D modelling software Blender [14].

The remainder of the paper is structured as follows. After presenting a taxonomy of illustrative figures for HCI scientific contribution, we discuss the existing workflows for creating trace figures. We then describe details of *Esquisse*'s interaction techniques and its implementation, before discussing the evaluation and figures created by participants.

2 Taxonomy of Trace Figures

We define trace figures as graphical representations of the most essential features of a scene by using contours/outlines of objects, people and the environment. In this section, we will introduce a classification of different kinds of trace figures and discuss typical characteristics. To better understand the use of trace figures within the HCI community, we analysed the use of trace figures in the papers published over three years (2015-2017) of UIST proceedings [1,2,3]. Many publications at UIST include details on gestural interaction techniques, use of interaction techniques and technical details on device use, and are a subset of the spectrum of publications published in the HCI community. Our initial sampling of proceedings indicated a common use of photo tracing in UIST papers, but our analysis below similarly translates to other HCI publications (e.g., Interact or ACM CHI), though the overall counts and percentages would probably differ.

Overall, we found 124 trace figures (often multiple per paper) in the 222 accepted UIST papers of these three years (trace figures in 29% of the papers).

[4] LeapMotion – https://www.leapmotion.com/
[5] Github Esquisse – http://ns.inria.fr/loki/esquisse/

We constrained our search so that all trace figures included in our sample set need to include at least one part of a person's body (e.g., a person's fingers, hand or full body), and therefore we did not include pure technical line drawing of components or devices. By following a systematic coding (by two researchers) and iterative thematic analysis of the figure sample set we identified the following five categories:

1. **Demonstration of gestures (64):** this most common type of trace figure illustrates a hand or body gesture (e.g., swiping, pressing, turning). Trace figures work well to clearly illustrate the accurate postures of hands or a person's body ([13, fig 13], [29, fig 1]).
2. **Overview of system setup or assembly (26):** these illustrations show, for example, the setup of tracking cameras, the setup of hardware (e.g., [9, fig 1d]) or the technical assembly of a wearable device.
3. **Interaction sequences (14):** these sequences show multiple frames of an interaction with a system (e.g., key steps of the interaction with a screen interface).
4. **Design space illustrations (13):** this can be a technical design space (e.g., variations of a tangible form factor) or a design space of gestures (e.g., a collection of 8 different hand postures).
5. **Other (7):** this includes any other uses of trace figures, for examples as part of tables, flow diagrams, etc.

By continuing our analysis and coding, we further identified the following characteristics of trace figure illustrations:

- **C1 — Person's body, hands or fingers:** the most common drawn element across all figures was a person's hand (total of 218 drawings of a person's hand, often multiple per figure, in 43 papers). This corresponds to the high number of figures in HCI papers demonstrating gestures ([9, fig 1d]). A person's body (or most of the body, like the upper torso) was drawn as part of figures 158 times across 22 papers. Individual fingers were drawn 51 times in 7 papers ([13, fig 13]) and a person's head (without rest of the body) 16 times (in 5 papers).
- **C2 — Devices and objects:** phones/tablets (32) and wearable devices such as smartwatches (20) were the most commonly drawn devices in our sample. 37 of the trace figures included some kind of tangible devices or other physical objects (e.g. active tokens, door handles). Other less frequently drawn objects and devices included: white boards and interactive tabletops (12), VR/AR headsets (7), tracking cameras (4), and car interiors (2) for automotive UI interaction.
- **C3 — Screen user interfaces:** 31 of the trace figures included screenshots of graphical user interfaces. For example, screenshots show the interface elements on mobile phones or smartwatch screens (see screenshots embedded in [13, fig 13] and [16, fig 3]).
- **C4 — Environment:** most photo traces (116) do not include any drawn details of the physical environment (e.g., background, furniture). Only 8

figures (e.g., [22, fig 1]) include traces lines delimiting the size of a room, or showing furniture and other objects in the environment. Since an important goal of photo traces is to reduce visual clutter, it seems that most figures strictly limit the traced details.

- **C5 — Use of colour:** photo traces can be pure black-and-white line drawings (34) but many include colour: 45 drawings used coloured areas within the drawing (e.g., solid fill of a person's hand in a skin-coloured tone), and 58 drawings used colour for making annotations and labels that stand out against the black and white trace figure (e.g., coloured arrows on top of the drawing, or coloured text labels).
- **C6 — Annotations:** most of the trace figures (121) include meta annotations such as labels, text descriptions, arrows or motion path lines. Most common annotations (besides text labels) are arrows (63) and touch contact point visualisations (37), for example by showing a coloured circle at the position where a person's fingers are touching the screen ([13, fig 13] and [16, fig 3]).
- **C7 — Static vs dynamic:** most figures depict a static scene (e.g., a single moment in time, a static setup), but a small number of figures (6) were designed for illustrating motion or movement (e.g., moving arms for a body gesture). Those usually used stroboscopic effect, where previous states/positions are drawn in different colours or line styles. McAweeney *et al.* made the same observation in their taxonomy focused on how gestures can be represented [26].
- **C8 — Use of perspective:** 55 trace figures in 43 papers use some sort of perspective (e.g., [25, fig 2] and [22, fig 1]) , which can be required to illustrate specific interactive scenarios (*e.g.* users around a table top, device tilting).

Our categorization and identified characteristics are of course depending on our sample and the kind of publications at UIST, and it is likely that further categories and properties can be identified when widening the sample. However, our categories and characterizations are a first-order approximation of the kind of interaction technique trace figures created for publications within the HCI community.

We used this characterisation of trace figures to directly inform the design of *Esquisse*. We decided to support various hand gestures and full-body models in the application to capture much of the use cases for trace figures we found in the literature. Commonly used device types (such as phones, tablets and smartwatches) became part of our template library (explained shortly). We also decided to directly integrate different colouring and tracing mechanisms (e.g., see through lines, coloured areas), and to simplify the use of human body representations in the visualisations (through templates and automated hand tracking).

3 Practices for Creating Trace Figures

3.1 Model-based Methods

While producing trace figure is a common practice, especially for HCI researchers, surprisingly little work has been devoted to ease the production of such materials. The most commonly used method remains manual photo-tracing [9,33] which can be described as follows. First, users stage the situation they want to illustrate using physical devices and other people standing in as actors, and take a photo of it. Then, they transfer this photo to a computer or a tablet, and open it as a background layer in an image editing software. Third, they sketch traces on top of the photo using their favourite input device and graphics editing software (usually Inkscape or Adobe Illustrator). Once the sketch is complete, they may add visuals such as overlaying an interface over a display or adding arrows, texts or contact points. Finally, they export the trace figure, most of the time as a vector graphic file so it can be resized at will.

This manual photo-tracing workflow still suffers from several limitations. First, it is time consuming and relatively tedious. It also requires users to have devices as well as friends or colleagues available that can be used as models. It may also require users to build physical props, for instance to illustrate interaction with devices they do not actually own or that exist only as 3D models. Another limitation is that some "minor" modifications (for instance a change in hand posture) might require users to start the whole process from the beginning. Similarly, changing the angle of the illustration requires users to repeat the process with a new photo.

Some of the vector graphics editing software provide tools to convert a photo into a drawing-like vector figure. However, these methods rely on juxtapositions of colour fills and result in a different visual quality than trace figures. Typically for a hand, the skin will result in several colour fills to recreate the gradient resulting of the lightning condition. Moreover, such an approach will vectorize the entire image, including background or elements that will ultimately be removed from the trace figure. A method letting the user perform local adjustment has thus been proposed [34].

Edge detection algorithms, based on the seminal work by John Canny [6], seem an alternative solution for tracing the contours of a model photography, but these algorithms usually fill pixels rather than producing a vector path, and may result in low-quality results depending on lighting conditions and quality of the source image. Mixed initiative [28,30] lets the human *lead* the edge detection, but still suffers from lighting conditions and blurry objects.

Specifying the contours of an object is also typical from rotoscoping technique[6] used in animation, which consists in drawing on top of consecutive images (typically a video). The traditional workflow for rotoscoping however differs from the manual photo-tracing as it still consists in creating several closed shapes for each object in order to ease subsequent manipulation [24]. As a result, tools

[6] Rotoscoping – https://en.wikipedia.org/wiki/Rotoscoping

specifically designed for easing rotoscoping, such as Roto++ [24], are less suitable for producing trace figures.

The comic book drawing tool ClipStudio Paint[7] could facilitate the production of trace figure when a photo model cannot be provided, since it provides 3D avatars that can be used as support to manually photo trace on top of them. However, this tool is mostly targeted at drawing characters and is not adapted to the illustration of HCI scenarios . Finally, DemoDraw [9] captures whole body movements from a user through a Microsoft Kinect and produces a minimalist illustration that can be overlaid with arrow annotations or using stroboscopic effects. *Esquisse* extends beyond this earlier related work by supporting the easy creation of vector-based photo tracings of both body movements, gestures and input devices.

3.2 Staging-based Methods

2D staging can be used to produce illustrations similar to trace figures, typically by using cliparts found online or suggested by Google AutoDraw[8] (a tool based on machine learning that retrieves clipart images based on approximate sketches drawn by the user) to rapidly produce trace figures without relying on a photo model. Those approaches require the appropriate cliparts (which are not necessarily available in vector format), and the 2D nature of clipart does not allow changes to correct the perspective of the figure which can be essential for illustrating interactive scenarios involving multiple users in a smart living room, or interaction techniques on a smartphone based on device tilting (for example, see Figures 1 in [22] and [31]).

SketchStudio is a 2.5D staging tool for prototyping animated design scenarios, based on body avatars and graph nodes that depict the journey of that avatar in space and time [21], scenes that can be played back through a dedicated viewing interface. There are, however, limitations when using this tool for the creation of photo traces. First, it is focused on *large scale* interactive scenarios involving a user that moves around devices, making it more difficult to use for fine interactions such as touch-based interaction on a smartphone. Furthermore, all interactive devices and interfaces must still be sketched, thus requiring drawing skills from the user, and the tool does not produce a vector-graphics rendering of the scene. Finally, several participants in an experiment conducted with SketchStudio mentioned that it becomes limited for ideas that need to be illustrated in 3D form, which is why Kim et al. suggest to improve their system to support the use of 3D models.

Finally, ComiPo! [9] is a design tool that relies on 3D models to make it possible to design manga-like comic books. 3D staging to produce 2D renderings here solves many problems associated with perspective. However besides the type of rendering, this tools could not be used for the production of trace figures for HCI

[7] ClipStudio Paint – https://www.clipstudio.net/en/
[8] AutoDraw – https://www.autodraw.com/
[9] ComiPo! – https://www.comipo.com/

because of the lack of support for aspects like the inclusion of interfaces or the production of vector format. Moreover, it does not use a camera system, which means that changing the perspective of a scene requires users to independently rotate all the objects of that scene.

4 Esquisse Workflow and Interaction

To use *Esquisse* the user starts Blender with the *Esquisse* add-on[10] previously installed. We chose Blender as it is a free and cross-platform software widely used to model, animate and render 3D scenes. However, as a complete 3D modelling environment, Blender provides controls with many degrees of freedom that are not needed for illustration production, which is why we overrode most default controls with simpler interaction techniques specifically implemented for *Esquisse*. These techniques are located in a dedicated panel located on the left side of the Blender interface (see Figure 2, left). To create a vector graphics illustration with *Esquisse*, users stage a 3D scene using one of the pre-existing templates as starting point or manually adding 3D objects to the scene. For objects featuring a screen (e.g. a smartphone) they can optionally load an external file with a screenshot image of the interface, which gets automatically rendered as part of the screen of the selected device. They then position and align the objects using the specific interaction techniques of *Esquisse* built to ease the manipulation of articulated objects (and in particular simplifying the modification of hand postures), but they can still use default Blender controls if they prefer. Next they orient the viewport to the desired point of view. If wanted, they can add a stroboscopic effect to illustrate the motion of a finger, for example. Last they render the scene to create the trace figure as a vector-based SVG file. Optionally, they can tweak or add graphical elements using any vector graphics editing software like Inkscape or Illustrator. Note that these steps do not necessarily have to be performed in that specific order. In the following we detail each step of this workflow.

4.1 Choosing a Template and Objects

We identified in our taxonomy (C1 and C2) that many trace figures only vary in details, which is why *Esquisse* provides Templates with pre-arranged sets of people and objects used in these typical scenarios. Each Template defines a 3D scene in Blender whose objects are loaded in the scene currently opened, for example two hands holding a phone, a person interacting with a smartwatch, or two people around an interactive tabletop. Templates can be easily selected from the gallery of thumbnail images of all currently available templates (Figure 2, (1) Esquisse *Library* tab). We created Templates based on the most commonly used postures and devices from our taxonomy, but new templates can be easily added by saving a scene and adding it to the *Esquisse* library.

[10] Esquisse on GitHub – http://ns.inria.fr/loki/esquisse/

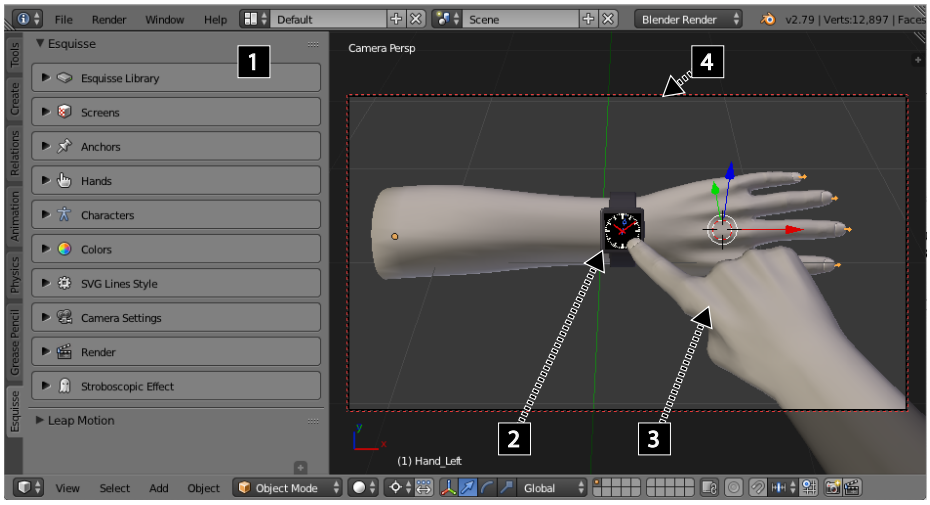

Fig. 2: Blender with the *Esquisse* add-on. All controls specific to *Esquisse* are located in a dedicated panel located on the left side of the interface (1). The right side shows the 3D view of the scene with several objects from the *Esquisse* library such as a SmartWatch with its GUI (2) and hand models (3)), and the frame of the current camera view (4).

In a similar way, users can complete a 3D scene by adding new objects (e.g., phones, avatars) from the object library of *Esquisse*. Custom 3D objects can obviously also be added to the scene using the import feature from Blender.

4.2 Adding GUIs in the Scene

Esquisse allows users to insert UIs (which was the case of 31 figures from our taxonomy, see C3) on 3D objects featuring a screen. A *screen* is a 3D plane object with a specific tag on which a user interface (defined in an external file) can be imported and displayed on.

Common objects with displays (smartphones, tablets, TVs, smartwatches and tabletops) are already provided in *Esquisse* with a defined *screen* area. It is also possible to add a *screen* to any other 3D object directly in *Esquisse*, by adding a *screen* object to the scene and setting the corresponding 3D object as its parent.

4.3 Facilitating Object Positioning

Simple objects like smartphones or tabletops can be positioned in the 3D scene easily using the 3-axis manipulator already available in Blender for translating, rotating and scaling the objects. However, when it comes to complex objects like a person's hand or body, manipulating the global position, rotation and scale of the combined object is not enough as these models are more complex and

the position and orientation of each of their sub-objects (e.g., bones for a hand model) need to be modified. *Esquisse* provides *rigged* models that take into account child-parent relationships between bones. That being said, modifying a body or hand posture while keeping a coherent visual appearance can be a difficult and a time-consuming process for novice users [4,32]. In the following, we introduce techniques to simplify this task.

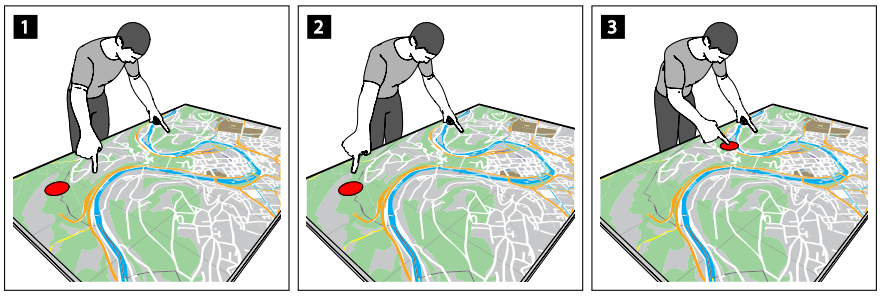

Fig. 3: 1) A red anchor is added to the screen of the tabletop. 2) The index of the right hand is linked to the anchor. 3) The anchor is moved and the arm follows.

Anchors. We define an ANCHOR as a point on an object that can be used to constrain the posture of another 3D object (*e.g.* a hand model). For example, ANCHORS allow users to specify a contact point on a display, attach a specific finger of a hand model to it, and then to modify the location of this contact while preserving the kinematic constraints of the hand (i.e., *Esquisse* makes sure that the fingers are touching the ANCHOR points with a realistic hand posture when reachable).

Once in ANCHOR mode, anchors can be added on any object (*e.g.* a screen) by clicking on the desired locations on the object. The user then links anchors to parts of another another object (*e.g.* fingertips of a hand). The anchors define constrains between the two objects that are then used by the Blender inverse kinematic solver to compute the position and orientation of the armature when the anchors are moved in 3D. *Esquisse* also provides a mode to move a linked object with the anchors, to avoid changing the current pose of the armature.

Fingers Flexion and Extension. The manipulation of non-anchored fingers may remain difficult [4,32], therefore *Esquisse* provides two specific controls for modifying hand poses. Inspired by Achibet *et al.* who control the pose of a virtual hand using sliders on a tablet [4], we added slider controls to manipulate the flexion of every finger of our hand model. Changing the value of a slider updates the flexion of the corresponding finger in the 3D scene in real time by adjusting the orientation of each bone of the finger. This method allows users to rapidly modify hand poses, for instance to describe a mid-air gestural vocabulary.

We also implemented a LeapMotion integration so the hand posture can be demonstrated. Similarly, body postures could be demonstrated using a Kinect like in DemoDraw [9].

4.4 Snap to Camera Point of View

In Blender cameras can be moved and rotated like any other object. As rotations can be difficult to perform, we implemented a feature to snap the camera object to the current viewport position. In this way the user can simply move the viewport to the desired point of view and then 'teleport' the camera so it matches its location and orientation.

4.5 Stroboscopic Effect

Esquisse allows users to generate stroboscopic figures, used to convey the movement of objects over time [27], by rendering each step with an increasing amount of transparency (C7 in our taxonomy). Once the scene is set up (Figure 4, top left), the user can enter in stroboscopic mode by clicking on the appropriate button in the interface. This allows users to define different *keyframes*, each saving the positions of all the current objects in the scene. *Esquisse* uses keyframes to detect the objects that moved and create a version of the corresponding objects (i.e. a copy of the objects displayed in wireframe) as illustrated Figure 4, top right (a keyframe version of the anchor and the linked hand are created when the anchor is moved). Keyframe objects can also be manipulated in the scene if necessary. Additional frames can be defined between keyframes in *Esquisse* using linear interpolation (see Figure 4, bottom left, where two steps are defined using interpolation).

All of the presented techniques in this section are designed to allow users inexperienced with 3D modelling softwares to rapidly stage a scene with 3D models. Most importantly, the techniques allow rapid iteration and experimentation (e.g., quickly changing the posture of a hand or camera view of the scene), something that is not possible to do when manually creating trace figures.

5 Esquisse Rendering Process

From the 3D scene, *Esquisse* is rendering vector images focusing on the contours of the objects. One important requirement of our rendering approach is to create closed contours only, so users can modify if they want the rendered objects in a vector graphics editing software afterwards (*e.g.* moving objects or modifying the colour or opacity of an object).

A first possible approach consists in propagating virtual points along the edges of an object mesh, by considering their visibility, to form the contours of the 3D object [17]. Unfortunately, this technique requires a high mesh density to accurately compute meshes' visibility, which yields to time consuming renderings. A second approach [12] consists in building a view map assigning properties

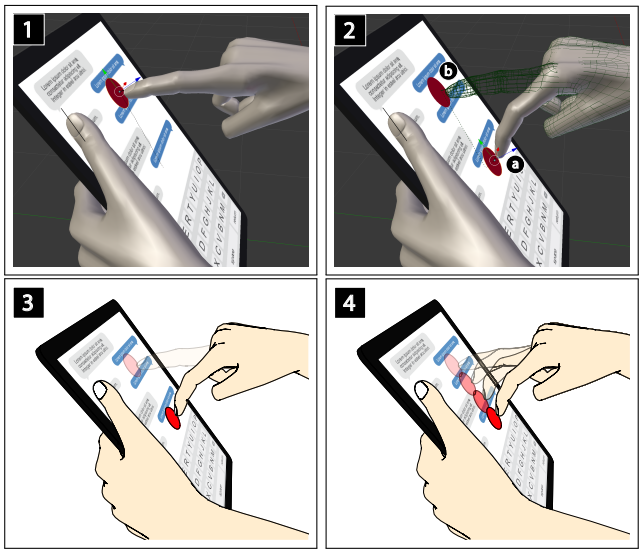

Fig. 4: 1) The index of the right hand is linked to a red anchor. 2) As soon as the hand moves in stroboscopic mode (a), a wireframe version of the object is created (b). 3) *Esquisse* rendering of the scene. 4) *Esquisse* rendering of the scene when 2 additional frames are defined by interpolation between the two keyframes.

to edges and then extracting contours by selecting specific edges based on their nature and visibility in order to chain them to form strokes. Unfortunately, extracted contours are split in several strokes which can be problematic to produce accurate polygons filling in a vector-based format. A third approach consists in creating layers containing full objects with the most effective cuts [10]. This facilitates the edition of SVGs in a graphics editing software but at a cost of longer rendering times to compute the depth order of the different layers.

As a result we chose to adapt the approach proposed by Eisemann *et al.* [11] which provides fill regions support by using the 2D arrangement package of the CGAL[11] library to extract 2D regions formed by 2D strokes. In that way, it is possible to extract at the same time line contours and fill polygons.

5.1 Esquisse 3D to SVG Rendering

This section explains how paths are calculated and how the final SVG rendering is performed.

From 3D to 2D paths We adapted the two approaches introduced in [11,12] to our needs and implemented them in Blender as follows.

[11] https://www.cgal.org

Step 1: *Computing a view map for the scene*
Similarly to [11,12], the first step iterates over all the edges of the objects to build a view map of the scene, by characterizing the *nature* of each edge (crease, silhouette, etc.) as well as its *visiblity* (visible or hidden). We use the Freestyle implementation already available as an extension in Blender.

Step 2: *Selecting and grouping edges of interest*
Second step consists in organizing the edges by selecting all the edges that correspond to a SILHOUETTE, CONTOUR or a CREASE, and to group them depending on their visibility. Note that keeping hidden edges is crucial as it can be required to illustrate some interactive scenarios (typically instances of back-of-device interaction).

Step 3: *Computing and associating filling regions*
Once all the edges of interest of the different objects have been grouped, we compute the regions formed by the 2D projection of the visible ones. As [11], we use the *2D arrangement* package provided by CGAL, which provides the regions formed by the 2D projection of the edges.

Step 4: *Associating 2D regions to their corresponding 3D objects*
We associate a region to its corresponding object in the scene by picking for a random point inside the region (using the GPC library[12]) and raycasting this point to the 3D scene to identify the corresponding 3D object. This step is used to group regions of a same object, as well as select a default filling colour for each region.

From Paths to SVG rendering The SVG file produced by *Esquisse* comprises two main layers, one for fillings and visible strokes, and a second for hidden strokes (which must be drawn on top of fills to be visible). The first layer contains one sub-layer per object grouping all its fillings and visible strokes, which eases the post-editing using other softwares. The line style used for each type of contour, as well as the colour of each object can also be specified by the user in the *Esquisse* interface. Filled contour layers are produced to avoid stroke layers from overlapping.

Generating filled contours containing holes. To appear as a hole, the points of the inside contour have to be oriented counter-clockwise while the points of the outer contour should be clockwise[13] for a *non-zero* fill rule. As CGAL 2D arrangement package returns regions under a hierarchy, regions inside other regions can be considered as holes and thus, be drawn in a counter-clockwise way.

Projecting UIs on screens. Our algorithm also needs to project possible images of graphical user interfaces on their corresponding screens in the 3D scene. For that, we use numpy linear algebra solver to compute the planar homography from the four corners of the user interface to the four corners of the screen object in the scene. However, considering SVG does not currently support perspective matrix

[12] http://www.cs.man.ac.uk/~toby/gpc/
[13] https://www.w3.org/TR/SVG/painting.html

transformations, we apply the homography to all the geometric objects of the user interface by computing the new coordinates. Some elements like ellipses, arcs or text first need to be discretized into segments or Bezier curves in order to be projected. All the projected elements are then grouped and a clipping path is applied to this group using the visible contours of the screen object, in order to avoid overlapping artefacts (*e.g.* a finger in front of the smartphone's display).

Rendering a stroboscopic figure. To render a stroboscopic figure, we adapt the rendering pipeline as follows. First, it computes the four rendering steps for each keyframe. Then it iterates over these keyframes and computes the SVG rendering for an object only if it has moved since the previous keyframe.

Doing multiple renderings implies having multiple contours for some of the objects (keyframes and final), and thus, requires to carefully choose the z-index of each object in the SVG file.

First, all objects that did not move in the scene are positioned with a low z-index. Then, we iterate over each keyframe and draw its objects with increased transparency and z-index .

Note that these modifications marginally deter rendering time, which remains below 5 seconds for the Figures 3 and 4, when running on the i7 4GHz computer described in the study section.

6 Study: Illustrating Interactions

We evaluated *Esquisse* by asking HCI students and researchers to produce visual illustrations of interaction techniques from the literature using our tool.

6.1 Method

Participants and apparatus. We recruited 8 participants ($\bar{x}$=33, σ=10), all researchers in HCI. Three were academic researchers, three PhD students and two Post-graduate students. None of these participants had previous experience with the Blender software. The experiment was conducted with Blender v2.79 with the *Esquisse* add-on installed, running on an iMac 27' 5K display with i7 4Ghz. Three input devices were available: a Logitech G9 Laser computer mouse, an Apple Magic Trackpad 2 and a LeapMotion. Adobe Illustrator and Inkscape were also installed and available to use at anytime if the participants wanted to complete the vector graphics file produced with *Esquisse*.

Procedure. Participants were invited to sit in front of the computer and were instructed that the experiment consisted in producing trace figures. We first showed them a set of trace figures extracted from the taxomony to clearly explain what trace figures are. After this introduction, we introduced the main task which consisted in illustrating one interaction technique from a set of four published in the HCI community (AuraSense [37], Put-that-there [5], TiltReduction [7] and Stitching [15]) that were described using the original accompanying

videos. We chose these interaction techniques because they rely on a variety of devices (*e.g.* smartphone, smartwatch, large display) and input modalities (device motion, touch, skin and speech input).

Participants were first presented with a 10 minutes long video explaining Blender basic controls and Esquisse features, and were then given 30 minutes to illustrate one interaction using Esquisse. They were also told that they could edit the produced SVG file afterwards with a graphics editing software.

Since the experiment's main focus was not the production of 2D interfaces, we provided a set of 2D interfaces associated to the 4 interaction techniques, to be used as screen interfaces. We measured the time it took participants to create the tracing figure (up to 30 minutes) and made observer notes about the strategy used by each participant. This experiment employed a *think aloud* protocol [23] encouraging participants to comment on *Esquisse* while producing the illustration. Figures produced by the participants can be found as additional material to this paper.

6.2 Interaction and Strategy in *Esquisse*

Templates. All participants but two started their illustration using one of the predefined templates. Interestingly, the two participants who did not [P3, P8] were producing illustrations for *Stitching* which consists of a pen gesture spanning over two interactive tablets, a situation for which *Esquisse* does not provide a template for. Both P3 and P8 manually added two mobile devices and one right hand with a stylus and placed them in the scene. In the other cases, *Esquisse* provides predefined templates close enough to the interactions to illustrate.

Anchors. As for templates, all participants used anchors except the two participants who had to illustrate *Stitching* [P3, P8]. In this case, all participants adopted a similar workflow: they simply moved the right hand with the stylus over the two devices on an axis without using anchors. Interestingly, one of these participants wanted to anchor the stylus [P8], which is not available in *Esquisse* but could be quickly implemented. His idea was to put an anchor on the tablet screen, then link the anchor to the stylus tip, and so, moving the stylus and the hand at the same time when moving the anchor. Overall, all participants understood and adopted the use of anchors very quickly.

Stroboscopic effect. All participants but one used the stroboscopic feature of *Esquisse* in order to illustrate motion. Two of them [P1,P2] who had to illustrate *AuraSense* used the interpolation feature in order to generate additional frames between the two defined keyframes. Once again, these participants exhibited a similar workflow, positioning the right hand with an anchor, adding a keyframe, moving the right hand to a different position and then set the interpolations. The participant who did not use the stroboscopic effect [P5], considered himself as figure expert, and did it intentionally to build instead a storyboard-like figure for *TiltReduction* using 3 distinct illustrations.

3D manipulations. All participants but two used the anchoring system and the 3-axis translation manipulator to make all the displacements they wanted, without expressing difficulties. The only two participants [P5,P6] who rotated objects during staging had to illustrate *TiltReduction*, which involves a rotation from the wrist to tilt a smartphone. One of them [P6] was the only participant who had difficulties with Esquisse and failed to create what he wanted, commenting that *"3D manipulations are too difficult for me"*. That being said, he ignored easier alternative strategies that could be used to rotate objects, such as anchoring all the fingers to force the hand pose, only move the camera to a different point of view, or to use the Leap Motion. The latter was adopted by [P5] to more easily change the hand orientation. Overall, only two participants used the Leap Motion for changing the hand posture, the other one [P2] was illustrating *AuraSense* and used it to define an index pointing posture for the right hand. Finally, all the participants used the camera snapping functionality and no one moved using the 3-axis manipulator.

SVG post-editing. While all participants were invited to do so, only three participants [P4,P5,P7] post-edited the figure produced with *Esquisse* in a vector-grahics editing software. While two performed only minor adjustments, [P5] built a 3 images storyboard and overlaid an arrow depicting motion over one figure.

6.3 Participants Subjective Feedback.

Learning and use. Overall, participants appreciated *Esquisse*, reporting that it was *"easy to use and fun"* [P4, P5, P8] and *"fast"* [P5, P7, P8], even if *Esquisse* requires a learning phase for objects manipulation and Blender interface [P1, P3, P8]. Participants quickly understood all the functionalities by just watching the 10 minute tutorial video [P1, P3, P4, P5]. Even better, [P1] was *"impressed"* and said it was *"fast to learn and master the different functionalities"*, making *Esquisse "interesting considering [his] poor artistic skills"*.

Post editing. Globally, participants were satisfied with the *Esquisse* rendering results, but some still prefer to refine the figure afterwards in a graphics editing software [P4, P5, P7]. More importantly, participants saw *Esquisse* as a rapid prototyping tool *"for making quick illustrations using complex objects to be modified afterwards"* [P5, P7].

Improvements. Participants were enthusiastic with *Esquisse* and commented on several additional features that could enrich its functionalities. [P2] commented that the capacity to control any object of the scene using motion sensing (as Yoon *et al.* did with a smartphone [35]) could be useful to make 3D staging even easier. [P4] would have liked to identify the keyframes objects in the scene, for example by using a different colour or a number associated to the keyframe index. [P8] commented that the ability to modify a User Interface file within *Esquisse* directly in the scene would optimize the workflow. Finally, two participants [P5, P8] commented that the capacity to overlay arrows *"between here and*

there" directly within *Esquisse*, either using the anchors or optionally displaying arrows between the keyframes, would remove the need to open the produced SVG file in a graphics-editing software afterwards. These suggested improvements are not challenging and will be implemented in the future versions of *Esquisse*.

7 Discussion and Conclusion

Trace figures are powerful and frequently used materials for illustrating HCI research papers. Based on a characterization of trace figures used in the HCI community, we designed *Esquisse*, a tool that implements a novel workflow based on 3D objects manipulation to stage a 3D scene and export it as a vector trace figure. We contributed interaction techniques to facilitate this workflow (including the use of anchor points to help the manipulation of complex articulated objects such as hands and the integration of stroboscopic effects) and a rendering pipeline to extract the different contours, fill them and finally generate a vector-based SVG file. We implemented *Esquisse* as an add-on for the open source 3D modelling software Blender and evaluated its usability in a qualitative experiment with 8 participants. The results of this experiment suggest that users, even when not familiar with Blender nor used to produce trace figures, managed to quickly produce trace figures illustrating interaction techniques from the literature with little to no experience in the production of trace figures or use of a 3D modelling software.

Esquisse thus provides an alternative workflow to produce trace figures, that can be used by users who believe they do not have the skills required to perform manual photo-tracing. *Esquisse* can also benefit users who usually rely on manual photo-tracing, for instance, as a rapid-prototyping tool as reported by participants in our experiment. A 3D scene staged with *Esquisse* can also be exported *as is* as a model for manual photo-tracing, which could be useful in situations where one has to illustrate a "complex and heavy" interactive scenario (for instance involving tabletops, large displays, several users, etc.) without having to prepare the physical setup required to take a photo of the scene.

There are limitations when using *Esquisse*. Typically, while adding models to *Esquisse* is as easy as adding a scene to its template sub-folder, designing these templates still need basic 3D modeling and staging skills. We anticipate that in the future, progress in computer vision and computer graphics research will allow to extract 3D objects from a photography to automatically create templates [36,8,20].

An aspect of *Esquisse* that can be seen as both an advantage or a drawback is that it has been implemented as a Blender add-on. We made this decision because Blender provides a solid environment for 3D modelling and object manipulation that *Esquisse* benefits from, as well as because it simplifies its distribution and maintenance. Because of this, however, the interaction with *Esquisse* is constrained by what the Blender environment allows add-ons to do (e.g., the design of the user interface for add-ons is limited). In that respect, we designed *Esquisse* by trying to provide the best user experience and interaction

that Blender allowed us to design. For example, we added dedicated slider controls to ease the definition of hand and body poses and implemented a direct LeapMotion integration to help users to manipulate hand poses. Similarly, inspired by DemoDraw [9], we plan to implement a Microsoft Kinect integration to ease the manipulation of body poses.

Most trace figures overlay meta annotations (such as texts or arrows). Besides contact points, *Esquisse* does not currently support a way to add these annotations and the user still has to open the produced SVG file in a graphic editing software to add them afterwards. However, meta annotation are usually simple geometric shapes or text layers that should be straightforward to integrate in the add-on. Adding meta-annotation support to *Esquisse* could be added by mimicking how anchors are currently added: the meta-annotation would be a shape or text that would be positioned over an invisible plan that would not be rendered when the SVG file is produced.

We hope our open-source release of *Esquisse* will help the production of trace figures that are gaining popularity in the HCI community as they are effective in capturing the essence of interactive scenarios.

References

1. UIST '15: Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology. ACM, New York, NY, USA (2015)
2. UIST '16: Proceedings of the 29th Annual Symposium on User Interface Software and Technology. ACM, New York, NY, USA (2016)
3. UIST '17: Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology. ACM, New York, NY, USA (2017)
4. Achibet, M., Casiez, G., Lécuyer, A., Marchal, M.: Thing: Introducing a tablet-based interaction technique for controlling 3d hand models. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems. pp. 317–326. CHI '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2702123.2702158
5. Bolt, R.A.: "put-that-there": Voice and gesture at the graphics interface. In: Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques. pp. 262–270. SIGGRAPH '80, ACM, New York, NY, USA (1980). https://doi.org/10.1145/800250.807503
6. Canny, J.: A computational approach to edge detection. In: Readings in Computer Vision, pp. 184–203. Elsevier (1987)
7. Chang, Y., L'Yi, S., Koh, K., Seo, J.: Understanding users' touch behavior on large mobile touch-screens and assisted targeting by tilting gesture. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems. pp. 1499–1508. CHI '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2702123.2702425
8. Chen, T., Zhu, Z., Shamir, A., Hu, S.M., Cohen-Or, D.: 3-sweep: Extracting editable objects from a single photo. ACM Transactions on Graphics (TOG) **32**(6), 195 (2013)
9. Chi, P.Y.P., Vogel, D., Dontcheva, M., Li, W., Hartmann, B.: Authoring illustrations of human movements by iterative physical demonstration. In: Proceedings of the 29th Annual Symposium on User Interface Software and

Technology. pp. 809–820. UIST '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2984511.2984559
10. Eisemann, E., Paris, S., Durand, F.: A visibility algorithm for converting 3d meshes into editable 2d vector graphics. In: ACM SIGGRAPH 2009 Papers. pp. 83:1–83:8. SIGGRAPH '09, ACM, New York, NY, USA (2009). https://doi.org/10.1145/1576246.1531389
11. Eisemann, E., Winnemöller, H., Hart, J.C., Salesin, D.: Stylized vector art from 3d models with region support. In: Computer Graphics Forum. vol. 27, pp. 1199–1207. Wiley Online Library (2008)
12. Grabli, S., Turquin, E., Durand, F., Sillion, F.X.: Programmable style for npr line drawing. In: Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques. pp. 33–44. EGSR'04, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2004). https://doi.org/10.2312/EGWR/EGSR04/033-044
13. Han, J., Lee, G.: Push-push: A drag-like operation overlapped with a page transition operation on touch interfaces. In: Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology. pp. 313–322. UIST '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2807442.2807457
14. Hess, R.: The Essential Blender: Guide to 3D Creation with the Open Source Suite Blender. No Starch Press, San Francisco, CA, USA (2007)
15. Hinckley, K., Ramos, G., Guimbretiere, F., Baudisch, P., Smith, M.: Stitching: Pen gestures that span multiple displays. In: Proceedings of the Working Conference on Advanced Visual Interfaces. pp. 23–31. AVI '04, ACM, New York, NY, USA (2004). https://doi.org/10.1145/989863.989866
16. Huang, D., Zhang, X., Saponas, T.S., Fogarty, J., Gollakota, S.: Leveraging dual-observable input for fine-grained thumb interaction using forearm emg. In: Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology. pp. 523–528. UIST '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2807442.2807506
17. Karsch, K., Hart, J.C.: Snaxels on a plane. In: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering. pp. 35–42. NPAR '11, ACM, New York, NY, USA (2011). https://doi.org/10.1145/2024676.2024683
18. Kazi, R.H., Chevalier, F., Grossman, T., Fitzmaurice, G.: Kitty: Sketching dynamic and interactive illustrations. In: Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology. pp. 395–405. UIST '14, ACM, New York, NY, USA (2014). https://doi.org/10.1145/2642918.2647375
19. Kazi, R.H., Chevalier, F., Grossman, T., Zhao, S., Fitzmaurice, G.: Draco: Sketching animated drawings with kinetic textures. In: ACM SIGGRAPH 2014 Posters. pp. 5:1–5:1. SIGGRAPH '14, ACM, New York, NY, USA (2014). https://doi.org/10.1145/2614217.2614221
20. Kholgade, N., Simon, T., Efros, A., Sheikh, Y.: 3d object manipulation in a single photograph using stock 3d models. ACM Transactions on Graphics (TOG) **33**(4), 127 (2014)
21. Kim, H.J., Kim, C.M., Nam, T.J.: Sketchstudio: Experience prototyping with 2.5-dimensional animated design scenarios. In: Proceedings of the 2018 Designing Interactive Systems Conference. pp. 831–843. DIS '18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3196709.3196736
22. Lander, C., Gehring, S., Krüger, A., Boring, S., Bulling, A.: Gazeprojector: Accurate gaze estimation and seamless gaze interaction across multiple displays.

In: Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology. pp. 395–404. UIST '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2807442.2807479
23. Lewis, C., Rieman, J.: Task-centered user interface design. A Practical Introduction (1993)
24. Li, W., Viola, F., Starck, J., Brostow, G.J., Campbell, N.D.F.: Roto++: Accelerating professional rotoscoping using shape manifolds. ACM Trans. Graph. **35**(4), 62:1–62:15 (Jul 2016). https://doi.org/10.1145/2897824.2925973, http://doi.acm.org/10.1145/2897824.2925973
25. Lo, J., Torres, C., Yang, I., O'Leary, J., Kaufman, D., Li, W., Dontcheva, M., Paulos, E.: Aesthetic electronics: Designing, sketching, and fabricating circuits through digital exploration. In: Proceedings of the 29th Annual Symposium on User Interface Software and Technology. pp. 665–676. UIST '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2984511.2984579
26. McAweeney, E., Zhang, H., Nebeling, M.: User-driven design principles for gesture representations. In: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. pp. 547:1–547:13. CHI '18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3173574.3174121
27. McCloud, S.: Understanding Comics: The Invisible Art (1993)
28. Mortensen, E.N., Barrett, W.A.: Intelligent scissors for image composition. In: Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques. pp. 191–198. SIGGRAPH '95, ACM, New York, NY, USA (1995). https://doi.org/10.1145/218380.218442
29. Nancel, M., Vogel, D., De Araujo, B., Jota, R., Casiez, G.: Next-point prediction metrics for perceived spatial errors. In: Proceedings of the 29th Annual Symposium on User Interface Software and Technology. pp. 271–285. UIST '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2984511.2984590
30. Neufeld, E., Popoola, H., Callele, D., Mould, D.: Mixed initiative interactive edge detection. In: Graphics Interface. pp. 177–184 (2003)
31. Ruiz, J., Li, Y.: Doubleflip: A motion gesture delimiter for mobile interaction. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 2717–2720. CHI '11, ACM, New York, NY, USA (2011). https://doi.org/10.1145/1978942.1979341
32. Wu, Y., Huang, T.S.: Hand modeling, analysis and recognition. IEEE Signal Processing Magazine **18**(3), 51–60 (2001)
33. Xie, J., Hertzmann, A., Li, W., Winnemöller, H.: Portraitsketch: Face sketching assistance for novices. In: Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology. pp. 407–417. UIST '14, ACM, New York, NY, USA (2014). https://doi.org/10.1145/2642918.2647399
34. Xie, J., Winnemöller, H., Li, W., Schiller, S.: Interactive vectorization. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. pp. 6695–6705. CHI '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3025453.3025872
35. Yoon, D., Lee, J.H., Yeom, K., Park, J.: Mobiature: 3d model manipulation technique for large displays using mobile devices. In: 2011 IEEE International Conference on Consumer Electronics (ICCE). pp. 495–496 (Jan 2011). https://doi.org/10.1109/ICCE.2011.5722702
36. Zheng, Y., Chen, X., Cheng, M.M., Zhou, K., Hu, S.M., Mitra, N.J.: Interactive images: cuboid proxies for smart image manipulation. ACM Trans. Graph. **31**(4), 99–1 (2012)

37. Zhou, J., Zhang, Y., Laput, G., Harrison, C.: Aurasense: Enabling expressive around-smartwatch interactions with electric field sensing. In: Proceedings of the 29th Annual Symposium on User Interface Software and Technology. pp. 81–86. UIST '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2984511.2984568