FreeStyle: A Sketch-based Wireframing Tool

Savinay Narendra, Sheelabhadra Dey, Josiah Coad, Seth Polsley, and Tracy Hammond

1 Abstract

User interfaces have classically involved WIMP (windows, icons, mouse, and pointing) paradigm. Current wireframing tools in the early stages of web interface design focus on using drag and drop features, that is, selecting elements from a menu and inserting them in the interface. Most designers prefer to sketch early interface ideas on paper. A system that allows them to draw sketches would make the process more intuitive, thus going beyond traditional methods of website interface designing. We have developed an interactive tool called FreeStyle that allows designers the ability to quickly sketch a web-based interface electronically. The designer can draw web-based HTML elements on the canvas and in response, FreeStyle transforms the sketch into an interface. Additionally, we also provide the functionality of downloading the code developed from the sketch.

Savinay Narendra
Department of Computer Science & Engineering, Texas A&M University, e-mail: savinay@tamu.edu

Sheelabhadra Dey
Department of Computer Science & Engineering, Texas A&M University, e-mail: sheelabhadra@tamu.edu

Josiah Coad
Department of Computer Science & Engineering, Texas A&M University, e-mail: josiahcoad@tamu.edu

Seth Polsley
Sketch Recognition Lab, Department of Computer Science & Engineering, Texas A&M University, e-mail: spolsley@tamu.edu

Tracy Hammond
Sketch Recognition Lab, Department of Computer Science & Engineering, Texas A&M University, e-mail: hammond@cse.tamu.edu

2 Introduction

As computers grow more powerful, less expensive, and more widely available, people are expecting them not only to perform obvious computational tasks, but also to assist in people-oriented tasks, such as writing, drawing, and designing. This shift is causing some user-interface (UI) researchers to rethink the traditional reliance on methods that are more machine-oriented and to look at ways to support properties like ambiguity, creativity, and informal communication. The idea is to bend computers to people's way of interacting, not the other way around.

When professional designers first start thinking about a visual interface, they often sketch rough pictures of the screen layouts. Their initial goal is to work on the overall layout and structure of the components, rather than to refine the detailed look-and-feel. Designers use these sketches and other "low-fidelity techniques" to quickly consider design ideas, later shifting to interface construction tools or handing off the design to a programmer. Unfortunately, this transition forces the designer to specify too many details. Much of the design literature recommends drawing rough sketches of design ideas, yet most interface construction tools, and even prototyping tools, require the designer to specify much more of the design than a rough sketch allows. These tools force designers to bridge the gap between how they think about a design and the detailed specification they must create.

Another key lesson from the design literature is the value of iterative design. It is important to iterate quickly in the early stages of design because that is when radically different ideas can and should be examined. The need to turn out new designs quickly is hampered by tools that require detailed designs. This over-specification can be tedious and may also lead to a loss of spontaneity. Thus, the designer may be forced to abandon computerized tools until later in the design process or forced to change design techniques in a way that is not conducive to early creative design.

Additionally, research indicates that the use of current interactive tools in the early stages of development places too much focus on design details like color and alignment rather than on the major interface design issues, such as structure and behavior. Research done in [11] found that colleagues give more useful feedback when evaluating interfaces with a sketchy look. The designers reported that current user interface construction tools are a hindrance during the early stages of interface design. What designers need are computerized tools that allow them to quickly sketch rough design ideas.

We have developed an interactive web-based wireframing tool that allows website designers to quickly sketch a website having certain HTML elements and in response, our tool develops a website containing all the sketches that the user drew converted into the corresponding HTML element having custom values. Designers can test the interface at any point, not just when they finish the design. When they are satisfied with their early prototypes, they can have FreeStyle transform the sketch into an operational interface using real widgets, according to a particular look and feel.

3 Prior Work

Our work draws inspiration from [6] SILK (Sketching Interfaces Like Krazy), an informal sketching tool that combines many of the benefits of paper-based sketching with the merits of current electronic tools. With SILK, designers can quickly sketch an interface using an electronic pad and stylus, and SILK recognizes widgets and other interface elements as the designer draws them. Unlike paper-based sketching, however, designers can exercise these elements in their sketchy state. SILK also supports the creation of storyboards, that is, the arrangement of sketches to show how design elements behave such as how a dialog box appears when the user activates a button. Storyboards are important because they give designers a way to show colleagues, customers, or end users early on how an interface will behave. After the designer tests the interface and iterates the design as needed, SILK transforms the rough design to a more finished looking implementation. But, our system resembles a more real world scenario which automatically generates the website in real-time while sketching the HTML elements on the canvas. This provides real-time feedback to the website designers, thereby improving their efficiency and quality of work.

Our work is based on an approach developed in [2] which presents a fast, simple and compact approach to recognize scribbles (multi-stroke geometric shapes) drawn with a stylus on a digitizing tablet. Regardless of size, rotation and number of strokes, the method identifies the most common shapes used in drawings, allowing for dashed, continuous or overlapping strokes. The method combines temporal adjacency, fuzzy logic and geometric features to classify scribbles with measured recognition rates over 97%. In 1963, Sutherland [9] presented Sketchpad the first interactive system that used one light pen to draw diagrams directly over the screen surface. Nevertheless, due in part to ergonomic problems with the light pen and the invention of the mouse in 1964, current graphical interfaces relegated pen-based interactions to specific CAD applications. Things changed with the appearance of the first pen computers in 1991, even though pens were used mainly to input text through handwriting recognition.

The Newton system [7], one of the first hand-held pen-based computers, incorporates handwriting, shape and gesture recognizers. While the shape recognizer only handles uni-stroke, axis-aligned shapes, the gesture recognizer is more suitable for text editing than for drawing schematics. An approach to recognize schematic drawings was developed at the University of Washington [1], which recognized a small number of non-rotated shapes and did not distinguish open/closed, dashed or bold shapes. [12] described an interactive editor based on the StateCharts [4] formalism. Although the editor used a mouse and direct manipulation, many of the ideas described in [12] express an approach based on diagram recognition guided by syntax. [3] described a system fundamentally based on sketches that are partially interpreted for use in architecture drawings, using a recognizer substantially simpler and less robust than ours. Although this recognizer is trainable, the number of identified shapes is apparently smaller than ours. [6] used a recognizer developed by Rubine in [8] to sketch graphical arrangements of bi-dimensional documents. Rubine's [8] recognizer is a trainable single-stroke gesture recognizer that uses a classic linear discriminator training algorithm. Other authors have proposed more complex methods, involving neural networks [10], to identify a small number of geometric shapes (rectangles, squares, circles, ellipses and triangles). These shapes are recognized independently of size and rotation, but they cannot be drawn using dashed or bold lines. Even though the authors claim good performance for their method, the paper does not present clear measurements to back their claims.

4 System Overview

The domain for which we developed the sketch recognition system required us to identify simple shapes such as rectangles, triangles, circles, and line-segments. A geometric recognizer that recognizes multi-stroke shapes perfectly suits our application. We decided to use the technique described by Fonseca et. al. [2] in their paper to identify shapes.

4.1 Finding the Convex Hull

The first step of our algorithm involves finding the convex hull of a set of points using the Jarvis march algorithm [5]. It helps in removing ambiguity of shapes that are not drawn perfectly since most of times when drawing with free hand it is difficult to generate perfect figures. Taking the convex hull of the set of data points obtained in a stroke approximates it to the actual shape that the user intended to draw. In the next step, our algorithm calculates the bounding box of the figure by finding the extreme points of the figure. So, in the drawn sketch if the maximum x coordinate is given by x_max, the maximum y coordinate is given by y_max, the minimum x coordinate is given by x_min, and the minimum y coordinate is given by y_min, the coordinates of the corner points of the bounding-box will be (x_min, y_min), (x_max, y_min), (x_max, y_max), and (x_min, y_max).

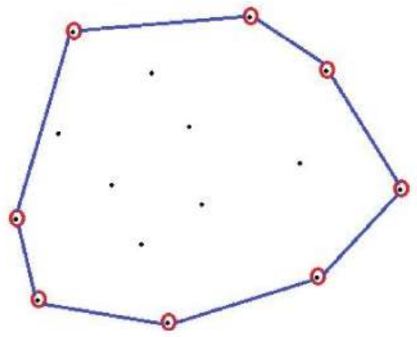

Fig. 1: Convex Hull - indicated by the blue lines

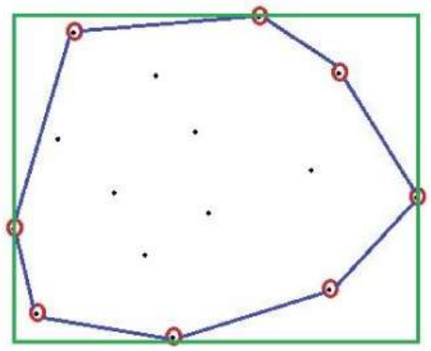

Fig. 2: Bounding Box - indicated by the green lines

4.2 Recognition of Elementary Shapes

For classification, we compute a feature that is unique for each shape that we are concerned with. Once we are done finding the convex hull and the bounding box of the sketch, we compute their areas. The ratio (ρ) of the area of the convex hull to the area of the bounding box is one of the features that can be helpful for classification.

$$\rho = \frac{Area\, of\, the\, Convex\, Hull}{Area\, of\, the\, Bounding\, Box}$$

In the ideal case, ρ would be 1, $\pi/4$, 0.5, and 0 for a rectangle, a circle, a triangle, and a line segment respectively. This method of classification is pretty intuitive and foolproof for identifying the elementary shapes that we are concerned with since it is based on the geometrical properties.

Humans usually do not draw shapes perfectly and hand-drawn sketches often contain a lot of noise. Due to this reason most of the times the ideal values for ρ are not obtained. To resolve the issue, we consider a range in which ρ would lie. Experimentally we found that the bounds for ρ lie between 0.8 and 1 for a rectangle, between 0.7 and 0.8 for a circle, between 0.4 and 0.6 for a triangle and between 0 and 0.1 for a line segment.

Based on the value of these ratios we built a set of decision rules that were used to classify the drawn figures as a rectangle, a circle, a triangle, and a line segment. These decision rules provided the correct classification when the shapes were drawn using a single continuous stroke.

4.3 Recognition of Complex Shapes

Wireframing tools consist of complex shapes other than the elementary shapes that we have listed above. A shape drawn inside another shape can be considered a complex shape. Once we are done drawing a shape in a single stroke, our recognition algorithm recognizes the shape and waits for the user to draw another figure for a time threshold of 2 seconds. If the user draws a figure after the 2 seconds then the algorithm identifies the figure drawn previously and the current figure as separate wireframing elements. Otherwise, the previously drawn shape and the current shape are considered to be a part of a single wireframing element. The shape that is drawn after drawing another figure in the case of complex shapes is usually drawn inside the first figure. So, the size of the shape drawn second is in general smaller than the first figure. This helps us decide if the shape drawn second is actually a part of a complex figure by comparing the size of its bounding box with the previously drawn figure. If the 2 shapes are found to combine together to form a single wireframing element then we run the recognition algorithms on the 2 shapes separately. This identifies the 2 shapes. Thereafter, we extract some other features such as the location of the 2^{nd} shape with respect to the 1^{st} shape, the slope, and the length of the stroke. This helps us form the decision rules to identify the wire-framing elements. For example, if the recognizer identifies that the first shape drawn is a rectangle and the second shape drawn is a triangle, there are two possible wireframe elements that they could refer to; a video element or a drop down menu. But, if we observe Fig. 5, the video element has the triangle at the center of the figure drawn first while the drop down element shown in Fig. 11 has the triangle drawn towards the right end side of the figure drawn first. This allowed us to introduce an attribute, the relative location of the second shape with respect to the first shape and differentiate a video element from a drop down menu element.

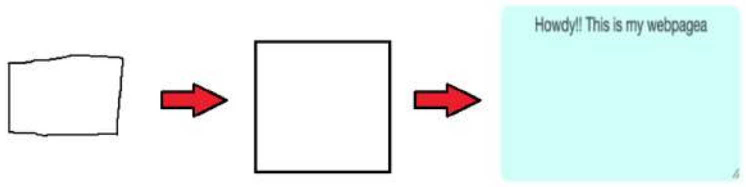

Fig. 3: A rectangle drawn is beautified to generate a text area element

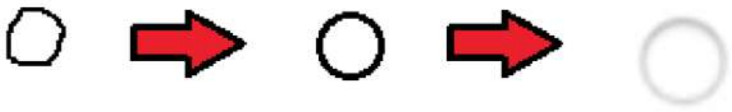

Fig. 4: A circle drawn is beautified to generate a radio button element

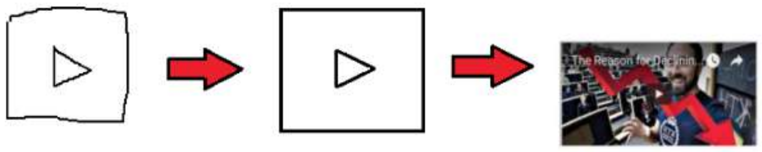

Fig. 5: A shape with a triangle drawn inside a rectangle is beautified to generate a video element

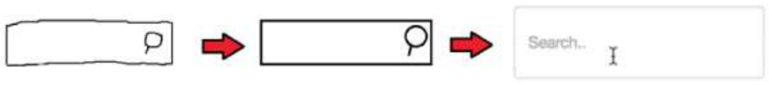

Fig. 6: A shape with a lens shape drawn inside a rectangle is beautified to generate a search box element

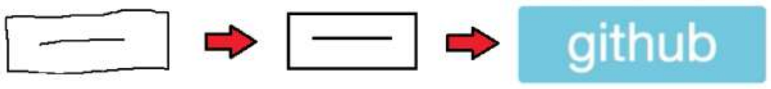

Fig. 7: A shape with horizontal line drawn inside a rectangle is beautified to generate a button element

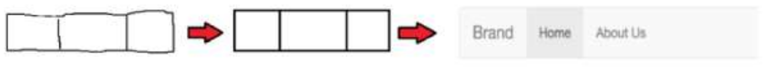

Fig. 8: A shape with vertical lines drawn inside a rectangle is beautified to generate a navigation bar element

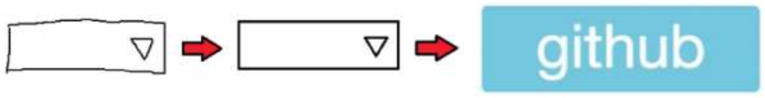

Fig. 9: A shape with a triangle drawn inside a rectangle towards the right end is beautified to generate a drop-down menu element

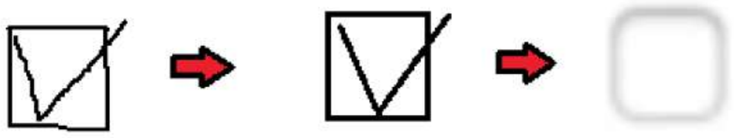

Fig. 10: A shape with a tick mark drawn inside a rectangle is beautified to generate a check box element

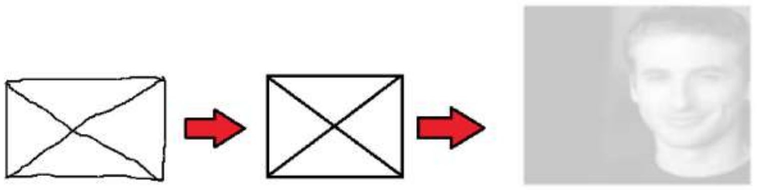

Fig. 11: A shape with a cross mark drawn inside a rectangle is beautified to generate an image element

4.4 Beautification of Elements

Once the shapes are correctly recognized and are classified as the correct wire-framing elements, we beautify the elements. This lends clarity to the wireframing elements and also provides us the freedom to add customizable features. The beautified element is introduced in place of the hand-sketched element. The size of the beautified element is equal to the size of the bounding box of the shape that the user drew. Thereafter, we allow the user to

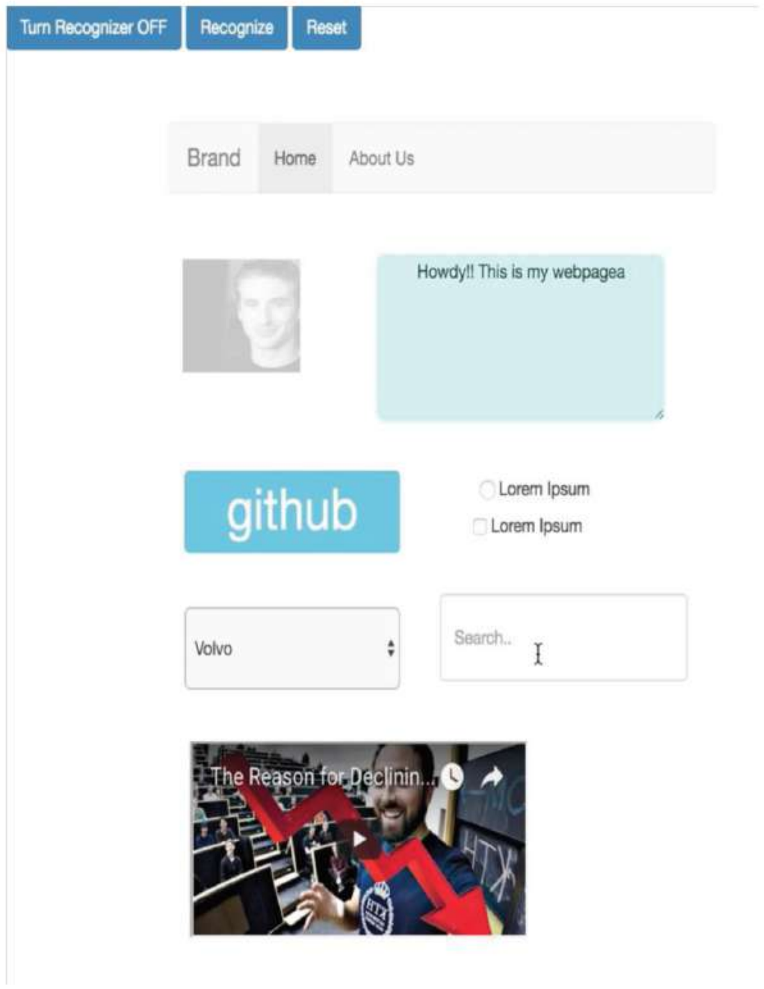

Fig. 12: A basic website created in our interface using wireframing elements

customize elements according to her preference. For example, on identifying that a particular shape refers to a video, the user can insert the link to the video that she intends to insert. The video will be embedded in the area created for the video element. Similarly, once a cross drawn inside a rectangle is identified as an image element, the user has the freedom to insert the link to an image and the image is displayed in the area created for the image element. This makes the layout of the website to be visually appealing. Since, our project also targets amateurs who are new to web-designing or people who just wish to design their own website, this feature provides an interesting visual feedback to the user. It helps them anticipate what their website might actually look like.

Professional designers will look for features where they can alter or tinker with the HTML code of the elements that they sketched. We have introduced a feature where once the element drawn by the user is identified, the HTML code for the element is displayed on the right hand of the canvas. Users can use the HTML code to alter the features of the wire-framing elements. FreeStyle works by giving the user a blank white canvas which they can sketch freely on. The recognition algorithm can recognize upto nine distinct elements at this time. These elements are standardly identified as a developer would draw them. For example, a rectangle drawn with crossed diagonal lines inside is an image element. Once identified, the recognizer removes the drawn sketch and replaces it with the code element that the sketch represents (which in the case of an image drawn would be the $< img >$ tag). Inside the HTML tag is information about the size and the position to match the element which was drawn. The code associated is shown temporarily on the side of the screen at the time of insertion. The code generated as a result of the design is easily downloadable at any time through the click of a button.

4.5 Dynamic Functionality

While FreeStyle is designed to be as quick and easy as possible to create a website design, we have also added additional functionality to the beautified elements that allows the website to go beyond a concept assistant to a functional website builder. Each element has a unique functionality which is accessible through selecting the element after beautification. For example when a user first draws the symbol for a button (a rectangle with a horizontal line drawn inside), the sketch goes away and an button icon image is added as a placeholder for the actual button. Based on the user's intentions, this may be enough or they could choose to select the beautified icon which it then prompts for a button link and text to display. At this point, the button becomes a live link, that is a clickable button. Each element has similar functionality that would allow someone with minimal technical knowledge the ability to make a basic yet functional website.

5 Evaluation

Although a good User Interface is essential for our software to provide a delightful experience to the users, it is not possible to quantify satisfaction. Hence, the criterion that we used for testing our application is to measure the accuracy of the recognition of the shapes. We asked 5 of our classmates

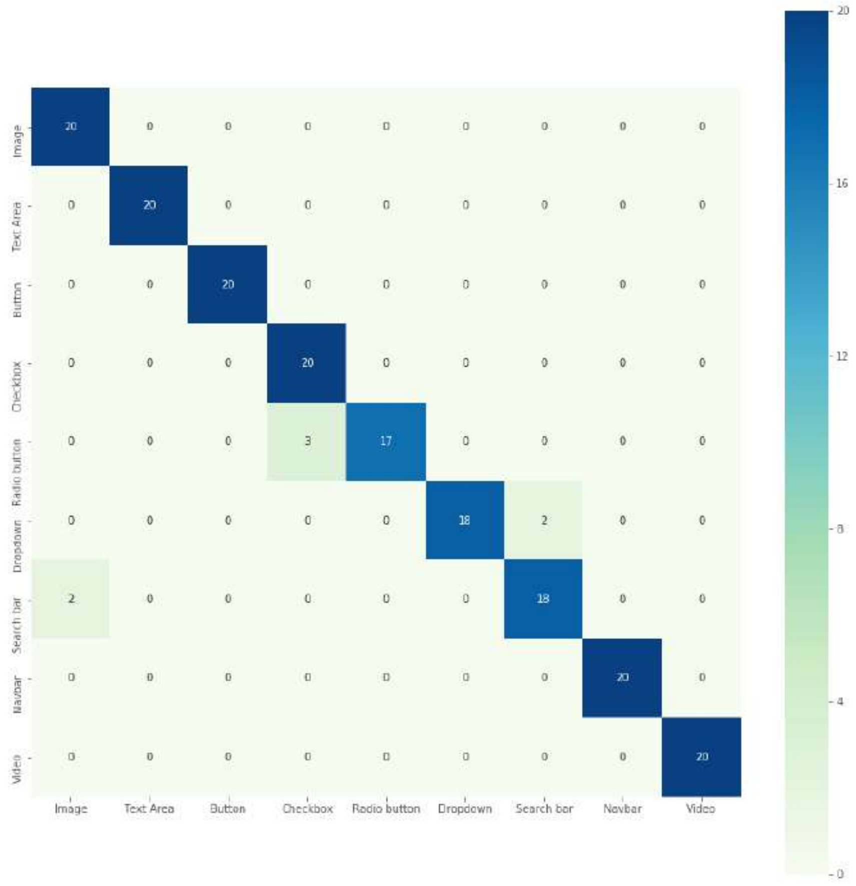

Fig. 13: Confusion matrix

from the Sketch Recognition course at the Texas A&M University to draw shapes that were defined by us on the canvas. We took a small sample since we wanted to only test a working prototype of our application. All the participants were briefed regarding the working of our system. The experiments were performed in a controlled manner in which the participants were asked to draw shapes that were close to ideal shapes with low variance. Each of them drew each shape in the canvas 4 times. On running the recognizer for the nine shapes corresponding to navbar, video, image, textarea, dropdown, search bar, checkbox, radio button, and button for twenty times each, we obtained the confusion matrix as depicted in Fig. 13. We find that our recognizer has a pretty high accuracy for recognizing shapes. The recognizer was able to identify not only the simple shapes but some complex shapes as well including radio button, dropdown and search bar with an accuracy of 100%.

Sometimes, it misclassified a radio button as a checkbox, a dropdown as a search bar and a search bar as an image. However, the error percentage was low and we were able to get accurate results most of the time. The conditions in which the experiments were done can be considered as ideal and in real world we do not expect our recognizer to perform extremely well.

6 Conclusion

The current drag and drop wireframing tools accessible to website developers lack the feel of pen and paper which often proves restrictive in the early brainstorming stage of website interface design. We believe that the FreeStyle system is an effective use of sketch recognition research. Our recognition techniques are effective in shape identification which can then further recognize more complex shapes as a combination of simple shapes. Our current methods identify simple shapes by comparing the ratio of the area of the bounding box to the area of the convex hull. Additionally, our method of identifying complex shapes through their relative position to each other, their relative size, and their orientation has been shown to be effective for identifying wireframing sketches. FreeStyle has applications much larger than the tool for the website designer. It can be scaled to be the tool for anyone who wants to make a simple personal website with little or no technical experience.

7 Future Work

We wish to make FreeStyle accessible to all as an effective tool. In this vein, we plan to make additions which would aid in ease of use and extend functionality. We would like to add move, edit, and delete functionality through drawing circles around elements and scribbling elements out. We also wish to apply a similar system to what is shown in SILK of multiple page navigation through drawing arrow from page to page. Additionally, we plan to build into our algorithm a normalized confidence feedback. We have experimented with this by using the shapes features to create a type of shape profile which can then be statistically compared against the attributes of the each ideal shape profile. By producing such feedback, we would be able to tell if the user is trying to draw an element or leave an annotation in which case we would leave whatever they sketched on the canvas without beautification. To improve FreeStyle's accuracy we plan to make a data driven recognizer that employs a deep neural network. FreeStyle currently lives up to its name by really allowing its users to sketch anywhere on the canvas but our research indicated that users and professional developers would like the option to switch on a grid system which their drawn elements would snap to. We plan to implement a customizable grid with various containers, much like the open-source CSS project "Bootstrap". Finally, we plan to continue to further our research in FreeStyle through case studies with both website building newcomers and professional developers.

References

1. Apte, A., Vo, V., Kimura, T.D.: Recognizing multistroke geometric shapes: an experimental evaluation. In: Proceedings of the 6th annual ACM symposium on User interface software and technology, pp. 121–128. ACM (1993)
2. Fonseca, M.J., Pimentel, C., Jorge, J.A.: Cali: An online scribble recognizer for calligraphic interfaces. In: AAAI spring symposium on sketch understanding, pp. 51–58 (2002)
3. Gross, M.D.: The electronic cocktail napkin - a computational environment for working with design diagrams (1996)
4. Harel, D.: Statecharts: A visual formalism for complex systems. Science of computer programming **8**(3), 231–274 (1987)
5. Jarvis, R.A.: On the identification of the convex hull of a finite set of points in the plane. Information processing letters **2**(1), 18–21 (1973)
6. Landay, J.A.: Silk: sketching interfaces like krazy. In: Conference companion on Human factors in computing systems, pp. 398–399. ACM (1996)
7. Menuez, D., Kounalakis, M.: Defying Gravity: The Making of Newton. Beyond Words Pub. (1993)
8. Rubine, D.: The automatic recognition of gestures. Ph.D. thesis, Carnegie Mellon University (1991)
9. Sutherland, I.E.: Sketchpad a man-machine graphical communication system. Transactions of the Society for Computer Simulation **2**(5), R–3 (1964)
10. Ulgen, F., Flavell, A.C., Akamatsu, N.: Geometric shape recognition with fuzzy filtered input to a backpropagation neural network. IEICE transactions on information and systems **78**(2), 174–183 (1995)
11. Wong, Y.Y.: Rough and ready prototypes: Lessons from graphic design. In: Posters and short talks of the 1992 SIGCHI conference on human factors in computing systems, pp. 83–84. ACM (1992)
12. Zhao, R.: Incremental recognition in gesture-based and syntax-directed diagram editors. In: Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems, pp. 95–100. ACM (1993)