# Catify Drawing System

Marissa Sorkin
Georgia Institute of Technology
Atlanta, Georgia, USA
marissasorkin@gatech.edu

Rohit Vemuri
Georgia Institute of Technology
Atlanta, Georgia, USA
rohitvemuri@gatech.edu

## Abstract

Catify is a web-based drawing application that offers an interactive, cat-themed interface for users to draw strokes on a canvas. The application provides a common set of authoring features including draw, erase, clear, color selection, laser pointer, and more, to offer flexible and mutable properties. The system introduces novel interaction techniques, including implementing keyboard press input for changing or correcting shape after selecting a stroke, dynamically changing the stroke path width via scroll wheel input, and allowing the user to draw cubic Bezier curves. The web application is built using TypeScript, React, and CSS.

## 1 Motivation

While there are numerous drawing tools available, many lack intuitive interaction techniques, especially for users seeking a more engaging and dynamic drawing experience. We recognized the need for an application that not only offers standard drawing functionalities but also introduces innovative features that enhance the user's creative process. Specifically, in modern drawing applications, multi-modal inputs are under-explored to provide greater affordances for the user, and through Catify, we wanted to provide this.

One of the primary motivations behind Catify was to challenge traditional drawing methods by incorporating unique interaction techniques. These techniques include the ability to transform strokes into shapes through keyboard inputs, dynamically adjust stroke widths using the mouse scroll wheel, and intuitively draw cubic Bezier curves. Such features not only add a layer of versatility to the drawing experience but also make the process more engaging and accessible to users with varying levels of skill and creativity.

In addition to enhancing functionality, we aimed to design an interface that is both aesthetically pleasing and user-friendly. The choice of a cat-themed interface was driven by the desire to create a visually engaging and cohesive experience that stands out from the standard interfaces of most drawing applications. This approach also aligns with our goal of making the drawing process more enjoyable and less intimidating for users, particularly for those who are new to digital drawing.

Another motivating factor was the opportunity to leverage modern web technologies such as ReactJS, TypeScript, and CSS. These technologies offer advanced capabilities in terms of state and hook management, type checking, and styling. By utilizing these technologies, we aimed to showcase the potential of web-based applications in delivering rich and interactive user experiences.

## 2 Discussion of Related Works

In the realm of stroke correction or shape correction within drawing applications, there is a notable body of research that focuses on dynamic correction techniques during the drawing process. This contrasts with our implementation in Catify, where drawing correction is initiated by the user through keyboard inputs post-drawing.

"ShipShape: A Drawing Beautification Assistant" by Fiser et al. [2015] presents an innovative approach to the visual fidelity of freehand sketches. ShipShape is a beautification assistant that allows users to retain the simplicity and speed of sketching while automatically rectifying output images by recognizing implicit geometric relations. This tool, functioning with general Bezier curves and integrated into Adobe Illustrator, offers undo/redo operations and scale independence [8]. The concept of real-time automated correction in ShipShape contrasts with our system implementation.

Another significant contribution is "Fluid Sketches: Continuous Recognition and Morphing of Simple Hand-Drawn Shapes" by Arvo and Novins [2000]. This paper describes a sketching interface where raw input strokes are continuously morphed into ideal geometric shapes, providing users with a dynamic and interactive drawing experience. The system implements time-dependent transformations to morph sketches into pre-defined shapes, offering immediate feedback and a sense of control over the drawing process [3].

Additionally, "Self-correctional 3D Shape Reconstruction from a Single Freehand Line Drawing" by Oh and Kim [2003] explores the field of 3D shape reconstruction from 2D sketches. Their self-correctional algorithm aims to minimize the distortion often encountered in traditional reconstruction methods. By correcting the shape and drawing simultaneously using geometric error metrics, this method refines sketch reconstructions progressively [13]. This approach to dynamic correction during the drawing process is distinct from Catify's user-initiated shape correction, highlighting a different aspect of user interaction and control in drawing applications.

These works collectively showcase the trend towards real-time, dynamic shape correction in the drawing process. In contrast, Catify's approach, allowing users to apply shape

corrections at their discretion post-drawing, offers an alternative perspective, emphasizing user control and flexibility in the creative process.

Two papers stand out for their innovative approaches to user interaction techniques that use scroll or scroll-like input. The first paper, "Elastic scroll for multi-focus interactions" by Kazuki Takashima et al. [2012], introduces a novel multi-focus scroll interface utilizing a content distortion technique. This interface treats displayed content like an elastic material, allowing it to be shrunk and stretched using finger gestures. The first step in this two-step operation involves dragging to elastically distort the content, temporarily showing the results of the viewport transition. This allows users to maintain both the original and the new focus within the viewport. The second step includes three simple gestures for scrolling, restoring, and zooming out, enabling users to adjust their focus as needed [15]. This concept of elastic scrolling and multi-focus interaction aligns with Catify's approach of intuitive and responsive user interaction, particularly in how we handle canvas navigation and manipulation.

The second paper, "An exploration of pen rolling for pen-based interaction" by Xiaojun Bi et al. [2018], delves into an underutilized aspect of pen input: pen rolling. While Catify uses mouse scroll inputs, the principles explored in this paper are parallel to our approach. The authors examine the use of pen rolling as an additional input modality, distinguishing between intentional pen rolling for interaction and incidental pen rolling during regular writing or drawing. Through their studies, they establish parameters for effective pen rolling interactions and explore a design space that includes rotational tasks, color selection, and simplified mode selection [4]. Although the implementation differs (pen rolling vs. mouse scrolling), the underlying concept of enhancing user interaction through innovative input techniques is a common thread that resonates with our objectives in Catify.

Both papers contribute valuable insights into the realm of user interaction, offering perspectives that have influenced the design and implementation of novel interaction techniques in Catify. The focus on user-friendly, intuitive, and efficient interaction methods in these studies aligns with our goal of creating a seamless and engaging user experience in our drawing web application.

In the domain of geometric modeling and digital graphic design, Bezier curves are a critical component. Defined by control points, these curves are integral to rendering smooth and precise lines. While the control points might not always be on the curve itself, their influence on the curve's shape is substantial, akin to a magnetic pull, making Bezier curves a vital tool for creating intricate and well-defined designs. While Bezier curves can be defined as having several control points, for the purpose of this paper, when referring to Bezier curves, we will strictly be referring to cubic Bezier curves defined by two control points.

The complexity of a Bezier curve is indicated by its degree, which is the number of its control points minus one. To illustrate, consider the quadratic Bezier curve, which has three control points. To visualize this curve, it is segmented into smaller parts. Points are selected at a proportional distance $'t'$along each segment formed by the control points. The point at this proportion $'t'$ on the new segment is a part of the Bezier curve. This principle can be extended to Bezier curves of higher orders.

A related work in this field is Jean Gallier's paper, "A simple method for drawing a rational curve as two Bezier segments." Gallier [1999] presents a method for representing a closed rational curve, defined by control points, as two Bezier curve segments. The key to this method is the derivation of control points for a rational curve G(t) from another curve F(t), through a projectivity mapping. This enables the complete tracing of curve F across an infinite range by plotting the segments F[(r,s]) and G([r,s]). Gallier's research, employing geometric reasoning about dividing the real projective line, introduces a new perspective on using Bezier segments for rational curve depiction [9].

The work of Gordon and Riesenfeld [1974], "Bernstein-Bezier Methods for the Computer-Aided Design of Free-Form Curves and Surfaces," is also crucial. They explore Bernstein polynomial approximation and its use in parametric Bernstein polynomials, as pioneered by Bezier at Renault. Their study examines Bezier techniques and their applications in geometric design, highlighting the preference for smoothness over exact fit in design challenges [10].

A contemporary development in this area is the 2020 paper "SketchADoodle: Touch-surface Multi-stroke Gesture Handling by Bezier Curves" by Grolaux et al. [2020]. This study introduces a new mathematical framework for representing multi-stroke gestures on touch surfaces using Bezier curves. This method enables direct manipulation of gestures in an Android application, demonstrating the practical, real-time use of Bezier curve techniques in gesture-based user interfaces [11].

## 3 Development Environment

In the development of Catify, TypeScript was the language of choice due to its organization capabilities and its proficiency in handling complex features. These features, like types, interfaces, and the handling of points, curves, and strokes, are vital for the application's complexity and scale. TypeScript's compatibility with React's hook management system is a significant advantage, providing rigorous type checking and error prevention, which are crucial for debugging and managing specific objects and interfaces. This added layer of application safety, ease of program compilation, and organization is beneficial compared to standard JavaScript.

React's state and hook management, specifically `useEffect` and `useState`, efficiently manage user interactions such as

mouse wheel scrolls and canvas clicks, which are fundamental to a drawing application. React's powerful framework facilitates a responsive and dynamic user experience, improving the interactivity of Catify. CSS was employed for its effective styling capabilities and simplicity. With over a decade of use and extensive documentation, CSS aids in creating an appealing and user-friendly interface.

Visual Studio Code served as the primary tool for programming, compilation, executing the application, debugging, and file organization. Its versatile and user-friendly interface made it an ideal choice for development needs. GitHub was used for version control, managing the codebase, tracking changes, and facilitating collaboration. Yarn was used for package management, which is specifically tailored for ReactJS applications. Yarn's provided functionality to reliably handle package installations and updates. It ensured that all dependencies for the web application had a smooth development workflow.

## 4 Implementation

As mentioned previously, we are using React's powerful hook and state management tools to implement Catify. Catify was build from scratch using no pre-existing open-source data. The single third-party library we utilized was Tippy for tooltips.

### 4.1 ReactJS Hooks Utilization

The use of ReactJS hooks, `useEffect` and `useState`, is a crucial aspect of the development of the application. `useEffect` maintains the application's state across multiple renders. This functionality is essential for preserving user-created strokes, shapes, and curves on the canvas. Given the dynamic nature of drawing applications, `useEffect` is invoked frequently, ensuring that changes such as the current width setting, color selection, and the array of strokes and shapes are consistently updated and rendered correctly. `useState`, on the other hand, manages the interactive state changes within the application. This hook is critical for operations such as adding new strokes or changing tool settings, allowing for a responsive and fluid user experience.

### 4.2 External Libaries

In addition to React hooks, there were integrated select external libraries to improve functionality and user experience. `Styled-components` helped to create custom-styled elements within React, providing a cohesive integration of design and functionality. This was particularly useful for toggling between different tool states, ensuring an intuitive user experience. Another key integration was Tippy, a third-party library used for creating tooltips.

### 4.3 Novel Techniques

Catify introduces three novel techniques that enhances the user's drawing experience.

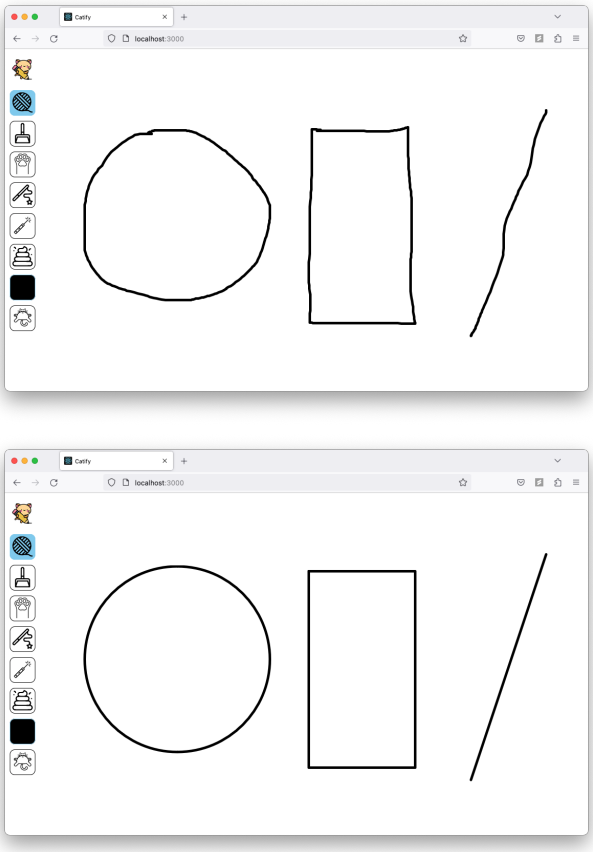

**Figure 1.** Correcting shapes based on key stroke input. The top image demonstrates the free-hand drawing of the shapes, and the bottom image highlights the shapes after they've been selected and corrected; the user would press "C" for circle, "R" for rectangle, and "L" for line.

**Shape Transformation via Keyboard Input**. In a departure from conventional drawing applications, users can transform freehand strokes into predefined shapes using keyboard inputs. This feature adds an extra layer of flexibility and creativity to the drawing process. Users can select a stroke and press specific keys ("C" for circle, "R" for rectangle, "L" for line) to instantly convert their strokes into these shapes. The transformation algorithms calculate the necessary parameters, such as the center and radius for circles or corner coordinates for rectangles, to accurately render the desired shapes.

**Dynamic Stroke Width Adjustment**. We also introduced a novel feature allowing users to dynamically adjust the

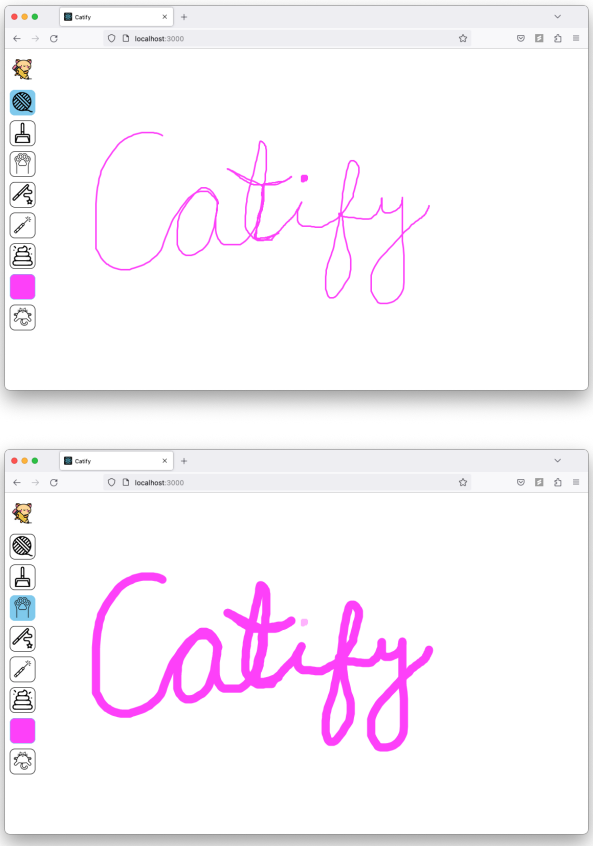

**Figure 2.** Dynamic stroke width adjustment seen in the images above. The top image has a thin stroke, created by scrolling down, while the bottom image has a wide stroke, adjusted by scrolling up.

width of strokes using the scroll wheel. This feature provides an intuitive and immediate way to modify the thickness of strokes, enhancing the creative control users have over their drawings. The implementation ensures real-time responsiveness, with the stroke width visually updating as the user scrolls.

**Cubic Bezier Curve Tool**. Theline cubic Bezier curve tool is another standout feature of Catify. Users can easily create complex Bezier curves by simply clicking to place four points on the canvas: two endpoints and two control points. This intuitive method lowers the barrier to creating advanced graphical elements, making it accessible even to users with limited drawing experience.

### 4.4 Tools and Features

**Draw**. This tool is synonymous with a pencil tool on standard authoring system applications. This allows users to draw strokes on the canvas, and users can select their path

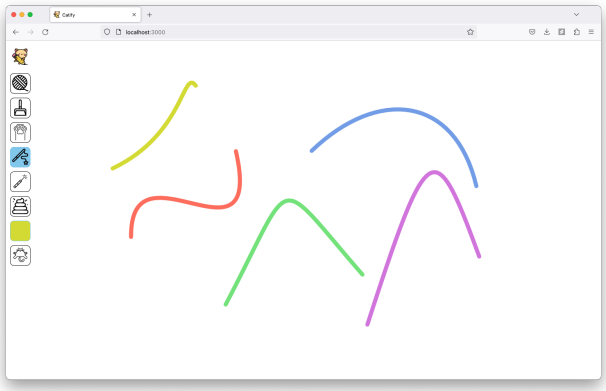

**Figure 3.** Drawing Bezier curves via the curve tool.

and alter them with other tools, such as changing the color and the thickness.

**Select**. The select tool is used to interact with strokes. The user can click on a stroke to modify it with the given tools.

**Eraser**. The user first selects a stroke, and the eraser tool removes the selected stroke.

**Curve**. The curve tool produces a curved stroke by clicking four points on the canvas. The first two clicks are the endpoints of the Bezier curve. The next click will define the first control point that corresponds with the first endpoint, and the final click will define the second control point and render the curve.

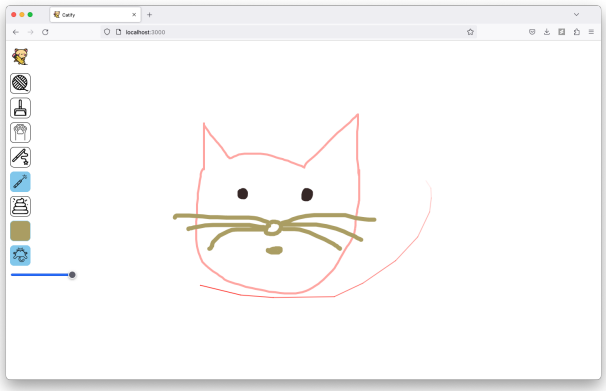

**Figure 4.** Laser pointer tool demonstrated by the trailing red path which follows the cursor.

**Laser Pointer**. The laser tool provides a digitally implemented laser pointer, allowing the user to call attention to specific areas on the canvas. There is a trailing effect to mimic

laser feedback.

**Clear Canvas**. The canvas page can be completely cleared with the Clear Canvas tool.

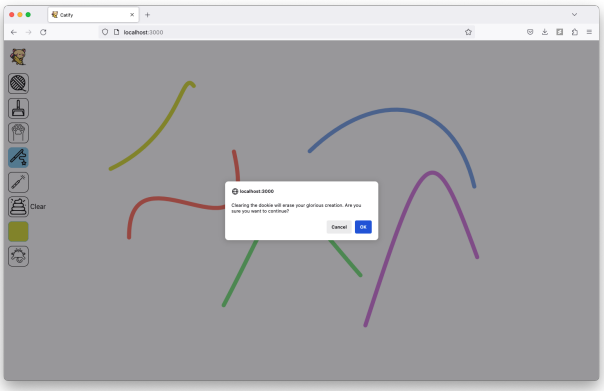

**Figure 5.** Warning modal that displays to confirm if the user wants to clear their selection.

**Warning Modal**. Along with the clearing canvas functionality comes a warning modal, since implementing this change will be permanent. We decided to go with a popup confirmation approach owing to the research findings from "Confirmation Responses: In-context, Visible, & Predictable Design versus Popup Windows" from Evgeniy Abdulin [2016], where the paper concludes that popup modals tend to result in fewer user errors [1].

**Color Selection**. Users can select a specific color for their stroke via the color tool.

**Thickness**. Users can change the thickness of a selected stroke via a button slider by mouse clicking and dragging to increase and decrease stroke size.

**Tooltips**. We have integrated tooltips to contextualize our cat-themed icons through phrases seen typically in drawing applications. We based this off of research done at the University of Washington demonstrated in the research paper "Tipper: Contextual Tooltips that Provide Seniors with Clear, Reliable Help for Web Tasks" by Yibo Dai et al. [2015] [6]. This research paper illustrates that users who have contextual tooltips enabled have a significantly easier time understanding functionalities of a web application, and we wanted to carry these research findings forward in Catify.

## 5 Discussion

In our implementation of Catify, a web-based drawing application with an interactive cat-themed interface, we have

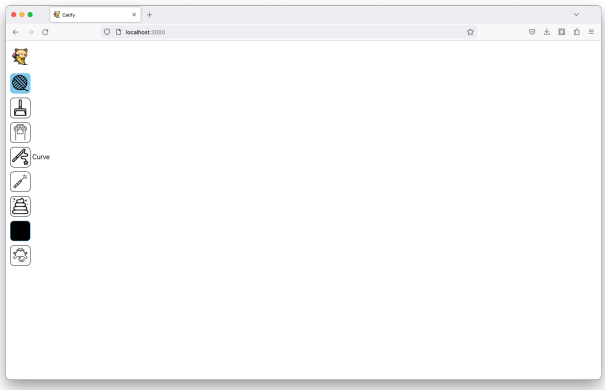

**Figure 6.** Tooltip labeled text "Clear" when hovering to the right of the clear icon.

focused on improving the usability and user experience of three key novel features: the integration of Bezier curves, shape correction with keystroke input, and changing the width of a stroke via scroll wheel input.

The basis of our drawing application falls on shapes and strokes on a canvas, always re-rendered when the user swaps functionality. React's hook `useEffect` prevents erasing data from the canvas. Each stroke is defined as a series of points connected by lines, and there are two shapes that are stored as well: circles and curves. We relied on web APIs to render circles (`CanvasRenderingContext2D.arc()`) and curves (`CanvasRenderingContext2D.bezierCurveTo()`).

**Bezier Curves:** The application's ability to allow users to draw cubic Bezier curves is one of our standout features. These curves, defined by two endpoints and two control points, provide high flexibility and precision in drawing. The user-friendly design, where users click on the canvas to establish four points (the first two being endpoints and the next two control points), has been developed with Fitt's Law in mind. However, there are limitations in terms of editing these curves post-creation, as users cannot select the stroke for further modification.

**Shape Correction with Keystroke Input:** The implementation of keystroke inputs for transforming free-hand strokes into predefined shapes (circles, rectangles, and lines) after selection is another novel aspect of Catify. This feature, which relies on mathematical calculations to estimate shape parameters based on stroke data, is a novel tool for users seeking precision and speed in creating geometric shapes, as using keyboard input to correct shapes is a unique feature in drawing applications.

**Analyzing the current eraser's functionality:** When developing the project, we decided that eraser's function should

be limited to removing an entire stroke on the canvas, similar to modern drawing applications. However, the research paper "An Algorithm For Sketching Design with Pencil and Eraser" by Guo and Zhang [2006] explores removing portions of a stroke when erased, as this feature aligns with experiments conducted by users with a pencil and paper. As such, one of the weaknesses of Catify is the inflexible eraser tool. To remedy this, we need to fix two things. Currently, a stroke is defined by a series of points connected by a line, and we can modify the eraser to remove points of a stroke aligning, with the findings of Guo and Zhang's research [12]. Next, we will make the eraser draggable so users can erase multiple strokes at the same time, as the user currently can only click to remove a single stroke rather than dragging the cursor to remove multiple strokes.

**Changing Stroke Width via Scroll Wheel Input:** The dynamic alteration of stroke path width through scroll wheel input allows users to quickly change the thickness of their stroke real-time. This feature allows for a seamless and immediate response to user input, adding to the overall flexibility and responsiveness of the drawing experience.

## 6 Future Work

While Catify's tools offer novel interaction techniques in web-based drawing applications, there are more features that could be added to offer users enhancements to their drawing experience.

Firstly, we plan to introduce advanced editing tools to allow users the functionality to edit strokes and curves. Specifically, we want to increase flexibility when editing a user's Bezier Curve, since these curves once on a canvas cannot be modified. As such, defining the anchor points and the control points so a user can stretch, move, and modify the path of a Bezier curve would provide this flexibility.

Furthermore, we also wish to enhance the precision of shape correction through keyboard input. At present, the application identifies stroke endpoints and edge points to create shapes such as circles, rectangles, and lines. For example, in creating a rectangle, the application determines its corners by identifying the maximum and minimum $x$ and $y$ coordinates. These coordinates are then utilized as the rectangle's corners. A similar approach is used for circles, where a center point and radius are inferred from these edge points. Moving forward, our objective is to improve the system's capability to recognize whether a stroke closely resembles a rectangle or circle before the shape is actually formed. While this will require a detection algorithm, we believe it will significantly benefit our users by delivering more accurate and intuitive shape corrections.

This is based on "SKETRACK: Stroke-Based Recognition of Online Hand-Drawn Sketches of Arrow-Connected Diagrams and Digital Logic Circuit Diagrams" by Oguz Altun and Orhan Nooruldeen [2019]. SKETRACK uses classical and specialized machine learning algorithms including a modified support vector machine (MSVM) classifier and spectral clustering algorithms like KNN and Euclidean distance. This research paper explores extracting and recognizing visual objects from continuous stroke streams, which is in line with Catify's purposes [2].

Additionally, there are other research papers that outline the use of neural networks to detect hand-drawn figures. The paper "DrawnNet: Offline Hand-Drawn Diagram Recognition Based on Keypoint Prediction of Aggregating Geometric Characteristics" by Jiaqi Fang et al. [2022] explores developing a convolutional neural network DrawnNet to extract geometric shapes from hand-drawn figures, including rectangles, squares, or diamonds. Currently, users need to use keyboard input to define a shape [7]. Instead of this, we can develop a smart tool, either through keyboard input or a button on screen, to detect the drawn shape and render a corresponding figure.

Another goal is adapting Catify to be usable across all output and input devices. "Dynamically Adapting GUIs to Diverse Input Devices" by Carter et al. [2006] explores this possibility. This paper explores using different switch inputs to draw shapes in a paint application [5]. While Catify is already responsive and can be used across output devices, some tools need to be adapted to be used without a pointing device, such as the laser pointer. For example, the mouse scroll wheel input for dynamically changing stroke width would need to be adapted for tablet input or touch-screen input devices.

In addition to these enhancements, a significant area of future development is integrating pressure sensitivity into the drawing system. This inclusion is inspired by the foundational work on pressure widgets by Ramos et al. [2004]. Their research explores using continuous pressure data from styluses to operate multi-state widgets, a concept that aligns perfectly with the interactive nature of Catify. Implementing pressure sensitivity will allow users to have more control over their strokes, such as varying the thickness based on the pressure applied, thus adding another layer of realism and flexibility to the drawing experience [14].

## 7 Conclusion

Catify stands out for its innovative approach to digital drawing, particularly in its implementation of Bezier curves, shape correction, and dynamic stroke width adjustment. As such, Catify's purpose is to explore and integrate new interaction techniques through web-based environments. We aim to further enhance its capabilities and introduce new features in future iterations of the application and focus on enhancing the precision and range of these features to further improve the user experience.

## References


[1] Evgeniy Abdulin and Dorrit Billman. 2016. Confirmation Responses: In-Context, Visible, & Predictable Design versus Popup Windows. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI EA '16)*. Association for Computing Machinery, New York, NY, USA, 2969–2975. https://doi.org/10.1145/2851581.2892403

[2] Oğuz Altun and Orhan Nooruldeen. 2019. SKETRACK: Stroke-Based Recognition of Online Hand-Drawn Sketches of Arrow-Connected Diagrams and Digital Logic Circuit Diagrams. *Scientific Programming* (Nov 2019). https://doi.org/10.1155/2019/6501264

[3] James Arvo and Kevin Novins. 2000. Fluid Sketches: Continuous Recognition and Morphing of Simple Hand-Drawn Shapes. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology* (San Diego, California, USA) *(UIST '00)*. Association for Computing Machinery, New York, NY, USA, 73–80. https://doi.org/10.1145/354401.354413

[4] Xiaojun Bi, Tomer Moscovich, Gonzalo Ramos, Ravin Balakrishnan, and Ken Hinckley. 2008. An Exploration of Pen Rolling for Pen-Based Interaction. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology* (Monterey, CA, USA) *(UIST '08)*. Association for Computing Machinery, New York, NY, USA, 191–200. https://doi.org/10.1145/1449715.1449745

[5] Scott Carter, Amy Hurst, Jennifer Mankoff, and Jack Li. 2006. Dynamically Adapting GUIs to Diverse Input Devices. In *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility* (Portland, Oregon, USA) *(Assets '06)*. Association for Computing Machinery, New York, NY, USA, 63–70. https://doi.org/10.1145/1168987.1169000

[6] Yibo Dai, George Karalis, Saba Kawas, and Chris Olsen. 2015. Tipper: Contextual Tooltips That Provide Seniors with Clear, Reliable Help for Web Tasks. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI EA '15)*. Association for Computing Machinery, New York, NY, USA, 1773–1778. https://doi.org/10.1145/2702613.2732796

[7] Jiaqi Fang, Zhen Feng, and Bo Cai. 2022. DrawnNet: Offline Hand-Drawn Diagram Recognition Based on Keypoint Prediction of Aggregating Geometric Characteristics. *Entropy* 24, 3 (2022). https://doi.org/10.3390/e24030425

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

[9] Jean Gallier. 1999. A Simple Method for Drawing a Rational Curve as Two Bézier Segments. *ACM Trans. Graph.* 18, 4 (oct 1999), 316–328. https://doi.org/10.1145/337680.337696

[10] William J. Gordon and Richard F. Riesenfeld. 1974. Bernstein-Bézier Methods for the Computer-Aided Design of Free-Form Curves and Surfaces. *J. ACM* 21, 2 (apr 1974), 293–310. https://doi.org/10.1145/321812.321824

[11] Donatien Grolaux, Jean Vanderdonckt, Thanh-Diane Nguyen, and Iyad Khaddam. 2020. SketchADoodle: Touch-Surface Multi-Stroke Gesture Handling by Bézier Curves. *Proc. ACM Hum.-Comput. Interact.* 4, EICS, Article 87 (jun 2020), 30 pages. https://doi.org/10.1145/3397875

[12] Fenghua Guo and Caiming Zhang. 2006. An Algorithm For Sketching Design with Pencil and Eraser. In *2006 First International Symposium on Pervasive Computing and Applications*. 403–407. https://doi.org/10.1109/SPCA.2006.297607

[13] BeomSoo Oh and ChangHun Kim. 2003. Self-Correctional 3D Shape Reconstruction from a Single Freehand Line Drawing. In *Proceedings of the 2003 International Conference on Computational Science and Its Applications: PartIII* (Montreal, Canada) *(ICCSA'03)*. Springer-Verlag, Berlin, Heidelberg, 528–538.

[14] Gonzalo Ramos, Matthew Boulos, and Ravin Balakrishnan. 2004. Pressure Widgets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vienna, Austria) *(CHI '04)*. Association for Computing Machinery, New York, NY, USA, 487–494. https://doi.org/10.1145/985692.985754

[15] Kazuki Takashima, Kazuyuki Fujita, Yuichi Itoh, and Yoshifumi Kitamura. 2012. Elastic Scroll for Multi-Focus Interactions. In *Adjunct Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology* (Cambridge, Massachusetts, USA) *(UIST Adjunct Proceedings '12)*. Association for Computing Machinery, New York, NY, USA, 19–20. https://doi.org/10.1145/2380296.2380307