FEASY: A Sketch-based Tool for Finite Element Analysis

Sundar Murugappan
C-Design Lab
School of Mechanical Engineering
Purdue University
West Lafayette, IN 47906
Email: sundar.murugappan@gmail.com

Cecil Piya
C-Design Lab
School of Mechanical Engineering
Purdue University
West Lafayette, IN 47906
Email: cpiya@purdue.edu

Maria C. Yang[*]
Dept. of Mechanical Engineering and the Engineering Systems Division
Massachusetts Institute of Technology
Cambridge, MA 02139
Email: mcyang@mit.edu

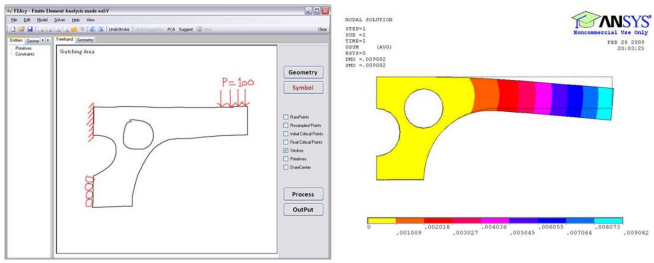
Karthik Ramani [†]
Donald W. Feddersen Professor
School of Mechanical Engineering
Purdue University
West Lafayette, IN 47906
Email: ramani@purdue.edu

ABSTRACT

*Freehand sketching is an integral part of early design process. Recent years have seen an increased interest in supporting sketching in computer-based design systems. In this paper, we present FEAsy (**F**inite **E**lement **A**nalysis made ea**sy**), a naturalistic environment for static finite element analysis. This tool allows users to transform, simulate and analyze their finite element models quickly and easily through freehand sketching. A major challenge here is to beautify freehand sketches and to this extent, we present a domain- independent, multi-stroke, multi-primitive method which automatically detects and uses the spatial relationships implied in the sketches for beautification. Further, we have also developed a domain-specific rules-based algorithm for recognizing commonly used symbols in FEA and a method for identifying different contexts in finite element modeling through combined interpretation of text and geometry. The results of the user study suggest that our proposed algorithms are efficient and robust. Pilot users found the interface to be effective and easy to use.*

---

[*]Associate Professor of Mechanical Engineering
[†]School of Electrical Engineering (by courtesy)



(a) (b)

Fig. 1. The FEASY interface showing (a) a hand-drawn sketch of an example 2d bracket, and (b) the deformation results in ANSYS.

1 Introduction

Freehand sketching is an activity that can take place throughout the engineering design process and is a natural, efficient and convenient way to capture, represent and communicate design ideas [1, 2]. Sketches are particularly useful in early stages of design where their fluidity and ease of construction enable creativity and the rapid exploration of ideas [3, 4]. Over the past few years, there has been increased interest in supporting freehand sketching in user interfaces and tools for various applications in diverse domains such as Computer aided Design (CAD), simulation and computer animation. As freehand sketch-based interfaces mimic the pen-paper paradigm of interaction, they may provide a host of advantages over the traditional windows, icons, menus and pointers (WIMP) style Graphical User Interfaces (GUI). Designers can seamlessly and directly interact with the computer with only limited training, whereas in menu based interfaces, users are forced to learn the system rather than the system having to learn users' intentions.

In this paper, we describe FEAsy (***F****inite* ***E****lement****A****nalysis made ea****sy***), a sketch-based interface for static finite element analysis. FEAsy allows users to transform, simulate and analyze their finite element models quickly and easily through freehand sketching (Fig. 1). A major challenge here is the need for techniques to transform ambiguous freehand strokes of a sketch into usable parametric geometric entities making up a formal diagram. Such a transformation is referred to as 'Beautification' [5–7]. Most current beautification methods do not consider important information implied in sketches such as the spatial relationships between different primitives in a stroke and between strokes [8]. These spatial relationships are usually represented as geometric constraints (like parallelism and tangency). Cognitive studies show that users preferentially attend towards certain geometric features while drawing and recognizing shapes [9]. To this extent, we present a multi-stroke, multi-primitive beautification method that identifies these spatial relationships and uses them to drive beautification. We also posit that, it is more intuitive to specify loading and boundary conditions through symbols as shown in Fig. 1, than through traditional menu-based input. Hence, we have developed a domain-specific algorithm for recognizing commonly used symbols in FEA. In addition, we have also developed an algorithm for combined interpretation of geometry and symbols in the sketch to identify different contexts like loading and boundary conditions observed in finite element analysis.

We foresee this tool to be used in engineering education and early design. It will allow analyses to be conducted even earlier in the design process because it reduces the reliance on preparing formal computer models beforehand. Formal, computational models such as CAD models may require a great deal of preparation and a clear understanding of design details. However, at the preliminary stages of design, hand generated sketches are often more appropriate to explore a wide range of potential ideas. Our tool would permit formal analysis of ideas that exist as only a hand drawn sketch. It can be used as a learning tool for undergraduate students especially in mechanical and civil engineering. The students can use this tool to quickly verify answers to hand-worked problems and also in preliminary stages of design projects to evaluate their ideas.

1.1 Contributions

This paper extends our prior work [10], and makes a number of contributions to research in sketching, interfaces, and analysis in engineering design:

1. A multi-stroke, multi-primitive beautification method that incorporates automatic constraint detection and solving, to transform the ambiguous freehand input into more structured formal drawings.
2. A symbol and text recognition algorithm for the finite element domain.
3. An algorithm for combined text and geometry interpretation, based on different contexts.
4. A novel interface that integrates freehand sketching, geometry constraint solving and symbol recognition in a unified framework for structural and thermal finite element analysis.

2 Related Work

This section provides an overview of the past work in beautification and sketch-based interfaces for varied applications.

2.1 Beautification - Segmentation and Recognition

Two of the main challenges that have hindered the development of a robust beautification system are: *Segmentation* - identification of critical points on the strokes and, *Recognition* - classifying the segment between adjacent critical points as low level-geometric primitives (like lines, circles and arcs). Much earlier work [5, 11–13] assumed that each pen stroke represented a single primitive such as a line segment or a curve in sketches. In spite of their simplicity, the strategy based on single primitive or stroke usually results in a less natural interaction because of the constraints imposed on the users' drawing freedom. Other works [14, 15] have also utilized pre-defined templates of higher order splines to neaten sketch inputs and smoothly combine the segments. By taking advantage of the interactive nature of sketching, several works [16–18] have used the pen-speed and curvature properties of the stroke to determine the critical points. They found that it was natural to slow the pen when making intentional discontinuities in the shape. When a user is sketching at a constant speed, many segmentation points will be missed due to this biased assumption. Kim and Kim [19] proposed new metrics based on curvature - local convexity and local monoticity for segmentation. Hammond et al. [20] introduced an effective method to find corners in polylines. Their method is founded on a simple concept based on the length between two points. They showed higher accuracy over Sezgin et al. [16] and Kim et al. [19]. Other approaches to segmentation utilized artificial intelligence, such as the template-based approach [21], conic section fitting [3], and domain-specific knowledge [22]. Despite their relative success in sketch segmentation, these are dependent on various restrictive conditions. For example, a large number of sketch examples are required for the purpose of training the computer in the methods proposed in [3], otherwise, the segmentation performance will be affected. For recognizing the segments, Shpitalni et al. [3] and Zhang et al. [23] used a least-squares based method. Sezgin et al [16] and Wolin et al. [20] compared the Euclidean distance between adjacent critical points and accumulated arc length of the segment. The ratio of arc length to Euclidean distance is close to 1 for a linear region and significantly higher for curved region. Xiong et al. [24] improved the algorithm described in [20] to include curves in addition to just lines. However, the algorithm does not recognize corners at a place where a line smoothly (tangentially) transitions into an arc and also those between two arcs.

More recently, with the advent of commodity level depth sensors (e.g. MS Kinect^{TM}) there has been some research devoted towards segmentation and recognition of free-form 3D strokes drawn in mid-air using finger based gestures. For example, Taele et al. [25] investigates techniques for developing intelligent interfaces and optimal interaction techniques for surfaceless sketching. Similarly, Babu et al. [26] provide a system from recognizing free-hand 3D input strokes and matching them to specific pre-defined 3D symbols. However, 3D sketching methods are limited as they cannot provide tactile feedback required for controlled sketch creation and also require advanced display media for co-location of the interaction and modeling spaces. As a result, their use is restricted to simple symbolic inputs and are therefore unsuitable for creating shapes for structural analysis. Wang et al. [27] utilize stereoscopic display with bimanual interactions through digital motion trackers to enhance sketching in 3D. But, their system relies on dedicated hardware that are not commonly available.

2.2 Sketch-based Interfaces

The emergence of pen-input devices like Tablet PCs, large electronic Whiteboards and PDAs have led to demand for sketch based interfaces in diverse applications [6]. Here, we list a few examples of such existing experimental systems. In CAD based applications like QuickSketch [28] and SKETCH [29], the user has to draw objects in pieces i.e., only one primitive at a time, thereby reducing the sense of natural sketching. Arisoy et al. [30] utilize a predictive modeling approach to automatically complete preliminary rough sketches created by users. Our interface also facilitates automatic completion of sketches, but in addition provides a suggestive interface to allow users to explore different possibilities resulting from their input.

Sketch based interfaces have also been used in early design [31] and in user-interface design [32]. Shadowdraw [33] provides dynamically adaptive suggestions in the background to guide users create aesthetically pleasing sketches. Similarly, in Juxtapose [34], sketch inputs are used to drive search of 2D images with the intent of making serendipitous discoveries during clipart composition. In contrast, our system provides dynamically updating suggestions to guide parametric sketching for engineering applications. Gesture-based systems have been explored in 2D pen-based applications [35, 36] where input strokes are converted or replaced with predefined primitives. Other works have also explored creation of 3D wireframe models based on multi-view planar sketch inputs [37] or scaffold based perspective drawings [38]. In contrast, our work is mainly related to creation of 2D geometry for structural analysis. ParSketch [39] is a sketch-based interface for editing 2D parametric geometry. MathPad [40] is a tool for solving mathematical problems. Kara et al. [41] developed a sketch-based system for vibratory mechanical systems. Krichoff's pen [42] is a pen-based tutoring system that teaches students to apply Kirchhoff's voltage law and current law. Hutchinson et al. [43] developed a unified framework for structural analysis. They used an existing freehand sketch recognition interface which is not robust in handling freehand strokes representing multiple primitives combined together. In addition open circular arcs or curves are not handled, constraining the variety of input that

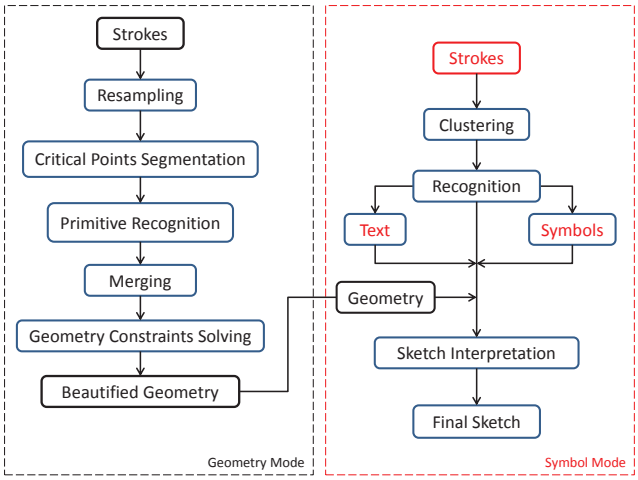


Fig. 2. The System Pipeline with two modes of input - 'Geometry' and 'Symbol'.

can be specified and also designer's drawing freedom. Moreover, the system does not address the problems related to the ambiguous nature of freehand input.

For symbol recognition, Fonseca et al.. [44] developed an online scribble recognizer called CALI. The recognition algorithm uses Fuzzy Logic and geometric features, combined with an extensible set of heuristics to classify scribbles. Since their classification relies on aggregated features of the pen strokes, it might be difficult to differentiate between similar shapes. Kara et al.. [45] described a hand-drawn symbol recognizer based on a multi-layer image recognition scheme. Similarly, Johnson et al. [46] enable users to apply standard drafting symbols to define constraints such as equality and perpendicularity, and edit 2D shapes by latching or erasing sketch segments. However, these methods require training, and in the case of [45] is also sensitive to non-uniform scaling. Veselova et al. [9] used results from perceptual studies to build a system capable of learning descriptions of hand-drawn symbols which are invariant to rotation and scaling.

3 Overview of the Approach

Freehand sketches are usually composed of a series of strokes. A stroke is a set of temporally ordered sampling points captured in a single sequence of pen-down, pen-move and pen-up events [47]. Sketches can be created in our system using any of a variety of devices that closely mimic the pen-paper paradigm. We use a Wacom Cintiq 21UX digitizer with stylus, tablet-PCs and a traditional mouse. Both Wacom and tablet PCs are particularly suited to natural interaction, enabling the user to sketch directly on a computer display. Figure 2 shows the pipeline through which the input strokes are processed in our system. In FEAsy, the strokes are input in either the 'geometry' mode or in the 'symbol' mode. Accordingly, the raw input strokes representing geometry are colored in black and those representing symbols are colored in red. Each stroke input in geometry mode is beautified, i.e. decomposed into low-level geometric primitives with minimal error. The system then identifies the spatial relationships between the primitives. These relationships are represented as geometric constraints which are then solved by a geometry constraint solver. The output from the solver is the beautified version of the input which is updated on the screen automatically. The strokes input in symbol mode are processed unlike in geometry mode. The red colored strokes are first clustered into stroke-groups. Then, the stroke-groups are classified as either text or symbol and recognized. Finally, the symbols, text and geometry are interpreted together for understanding the various contexts in the sketch. The sketch is now ready for finite element problem study. Sections 4-6 describe each of these steps in detail.

3.1 An Example

Figure 3 shows a step-by-step process of analyzing a bracket from its freehand sketch. The user starts the geometry creation process by sketching a freehand stroke in the 'geometry' mode as shown in Fig. 3 (a). The green circle is the starting point of the stroke and the red circle depicts the end point. The blue arrows show the direction of the stroke. At the completion of the stroke, the system automatically beautifies the input. The beautified output is shown in Fig. 3 (b). Next, the user adds a freehand stroke to the sketch. In this case, it is a hole in the bracket (see Fig. 3 (c)). The final result after beautification is shown in Fig. 3 (d). The geometry creation process for the bracket is now complete. The user then switches modes to insert symbols. Figure 3 (e) shows a beautified sketch with input symbols. In the next step, the symbols are recognized and the sketch is interpreted. The output is updated on the screen as shown in Fig. 3 (f)). Once the sketch is complete and processed, the user specifies the material properties, element description and meshing parameters for 'Finite

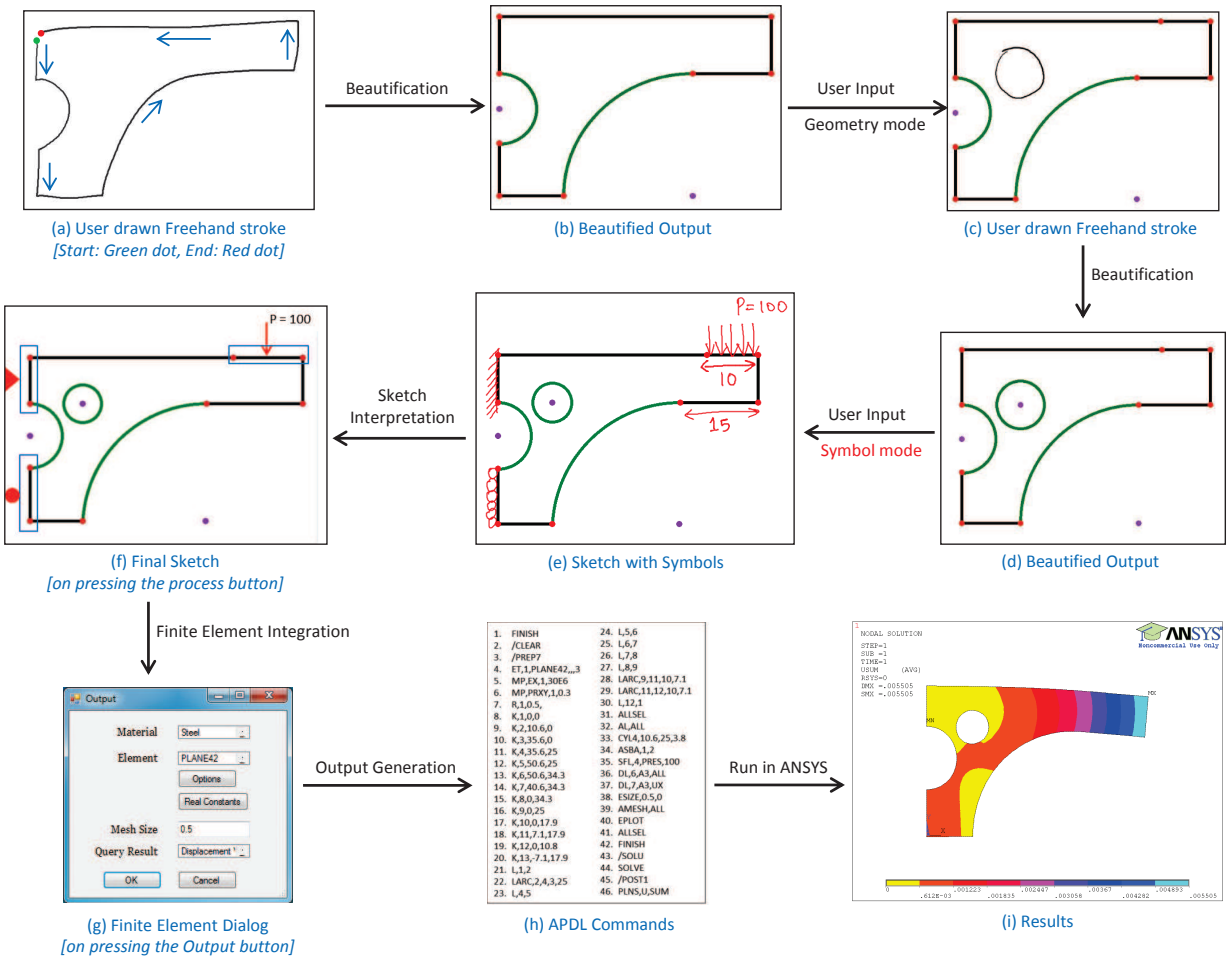


Fig. 3. Step-by-step process of analyzing a bracket. (a) User drawn freehand stroke in geometry mode, (b) beautified output, (c) a freehand stroke representing a circle is added to the sketch, (d) result after beautification, (e) symbols (red colored strokes) input in symbol mode, (f) final sketch after sketch interpretation, (g) finite element integration dialog, (h) generated ANSYS specific commands, (i) deformation results in ANSYS

Element integration' (Fig. 3 (g)) and all the information is exported as a set of commands suitable for import in ANSYS (Fig. 3 (h)). These commands are then run and solved in ANSYS. Figure 3 (i) shows the deformation results of the bracket.

4 Beautification

Beautification aims at simplifying the representation of the input where the various points of the strokes are interpreted and represented in a more meaningful manner. Our approach for transforming the input to formalized representations (i.e. beautification) is based on the architecture shown in Fig. 4. There are five steps in the pipeline, namely - resampling, segmentation, recognition, merging and geometry constraint solving. Figure 4 shows the various steps along with the actual outputs generated in the system. However, it is to be noted that only the final beautified sketch is visible to the users and other intermediate outputs are generated for illustration purposes. Figure 4(a) shows a user drawn freehand stroke. This is an example of a single stroke representing multiple primitives connected together. Figure 4(b) shows the raw data points (blue circles) as sampled by the hardware and Fig. 4(c) illustrates the uniformly spaced points after resampling (green circles). The segmentation step explained in section 4.2 identifies the critical points (red circles) shown in Fig. 4(d). Then, the segments between the adjacent critical points are recognized and fit with primitives (Fig. 4(e)). The status of the freehand sketch after merging is shown in Fig. 4(f). Finally the sketch is beautified considering the geometric constraints (Fig. 4(g)). The aforementioned steps are explained in detail in the following sections. For simplicity, we limit the discussions to a single stroke in a sketch. All the other strokes are processed similarly.

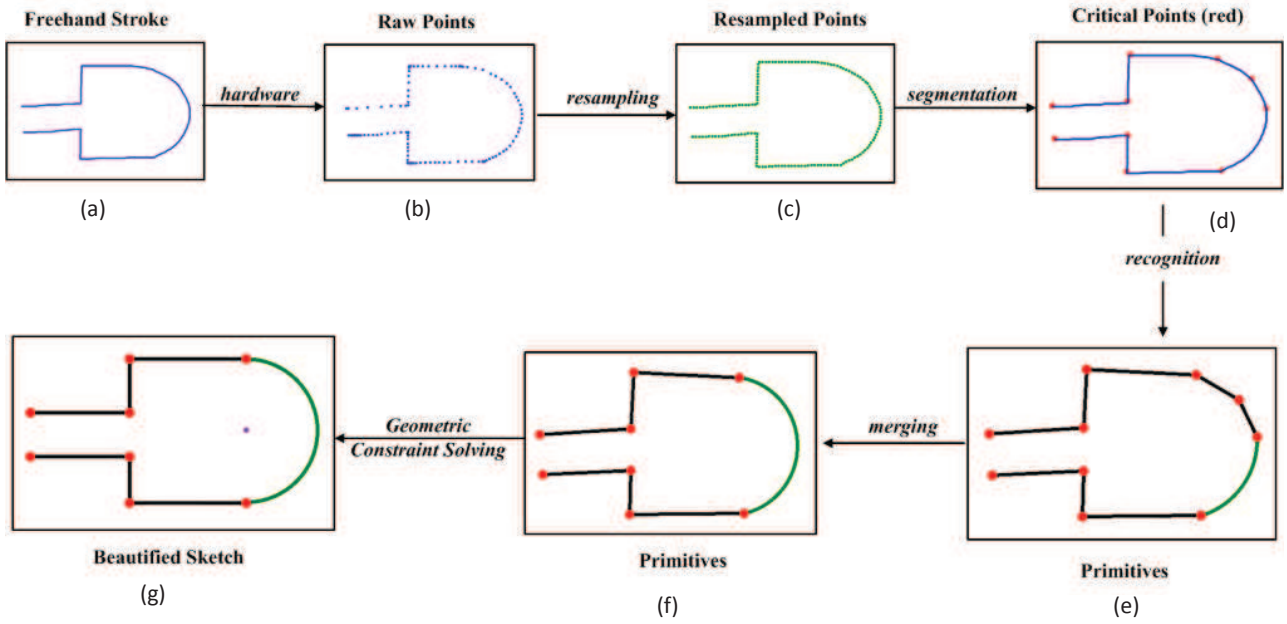


Fig. 4. Beautification of a freehand stroke

4.1 Stroke Resampling

The sampling frequency of the mechanical hardware coupled with the drawing speed of the user result in non-uniform samples of the raw freehand input. Evenly spaced points are important for the segmentation algorithm to work efficiently. To achieve uniform sampling, we resample the points of the input stroke such that they are evenly spaced. We used a fixed interspacing distance, Id of 200 HIMETRIC units (1 HIMETRIC = 0.01mm = 0.0378 pixels). The resampling algorithm discards any sample within the I_d of earlier samples and interpolates between samples that are separated by more than I_d. The start and end points of the stroke are by default added to the resampled set of points. Figure 4(c) shows the result of resampling for the stroke.

4.2 Segmentation

In our system, a single freehand stroke can represent any number of primitives connected together. The task of the segmentation routine is to find those critical points that divide the stroke into its constituent primitives. These critical points are 'corners' of the piecewise linear strokes and also the places where curve and line (curve) segments connect. Our segmentation algorithm builds upon the approach described in [20], which works well for strokes composed of only line segments. One of the drawbacks of this method is that the algorithm often misses identification of corners at heavily obtuse angles. We address this drawback and also improve their algorithm to accommodate curves in addition to line segments. We are interested in improving this algorithm especially for its simplicity (easy to program) in implementation, high efficiency and at the same time not being computationally intensive. They described a measure called straw (chord length) which in essence is a naive representation of the curvature of the stroke. The *straw* at each point p_i is computed as $straw_i = |p_{i-w}, p_{i+w}|$, where w is a constant window and $|p_{i-w}, p_{i+w}|$ is the euclidean distance between the points p_{i-w} and p_{i+w}. As the stroke turns around a corner, the *straw* length starts to decrease and a local minimum value corresponds to a likely critical point. However, when there is smooth continuity between a line and an arc or between two arcs, the *straw* length does not vary much and it fails to identify the transition in such regions. Hence, we use another such measure, *chord angle* which is effective in identifying these gradual changes in addition to finding the corners.

After resampling the stroke, we compute *chord angle* for the resampled points p_w to p_{n-w}. where 'n' is the total number of resampled points and w is a constant window. The *chord angle* at each point p_i is computed as follows

$$\measuredangle p_i = \arccos\left(\frac{\overrightarrow{p_i p_{i-w}} \cdot \overrightarrow{p_i p_{i+w}}}{|\overrightarrow{p_i p_{i-w}}||\overrightarrow{p_i p_{i+w}}|}\right) \tag{1}$$

The likely critical points of the stroke are those indices where the 'chord angle' is a local minimum, which is lesser than a threshold ('t'). Figure. 5 shows the computation of the chord angle. The blue circles represent the resampled points and 'θ'

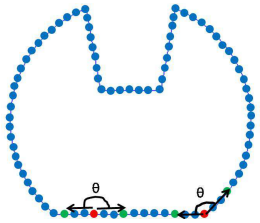


Fig. 5. 'Chord Angle' computation. The blue circles represent the resampled points. θ represents the 'chord angle computed for the resampled point (red) using a window size of 3 (green points)

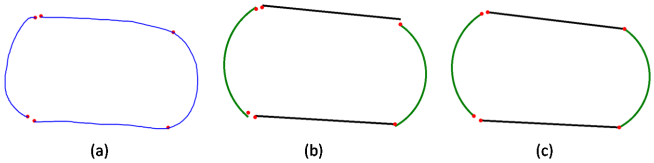


Fig. 6. Recognition. (a) shows the freehand stroke with critical points. (b) shows the results of least squares fitting. There are discontinuities between adjacent segments. (c) shows the results of our algorithm

represents the 'chord angle' computed using the equation 1. To avoid the problem posed by choosing a fixed threshold, we set the threshold to be equal to the median of all the chord angle values. For the stroke in Fig. 4(a), the initial set of critical points obtained is shown in Fig. 4(d). By default, the start and end points of a stroke are considered as critical points. A window of uniformly spaced points is used to compute the curvature (chord angle), which smoothens out the noise, if any in the input stroke. The larger the window, the larger the smoothing effect resulting in missed critical points. Like [17], we found that setting the window size, w = 3 to be effective irrespective of the user or the input device used.

4.3 Recognition

The next task after segmentation is to classify and fit the segments between adjacent critical points as low-level geometric primitives. The current implementation of our system recognizes lines, circular arcs and circles. Our recognition method is based on least squares analysis [48], but the computation of parameters of best fit line and circular arc differ from the traditional approach. Usually, the least square fit of lines and arcs result in the end points of the primitives to be moved to new locations as shown in Fig. 6. These new positions do not coincide with the original critical points of the stroke and hence cause discontinuities between adjacent primitives of the stroke. To prevent such discontinuities, we fix the endpoints of the primitives to coincide with original critical points and then perform the analysis. Figure 6 shows the actual result of our recognition algorithm which has no discontinuities.

4.3.1 Fitting a straight line

Let $S_N = \{p_i = (x_i, y_i) | i = 1, 2, ..., N\}$ be the given points of the segment and let $P_1(x_1, y_1)$ and $P_N(x_N, y_N)$ be the end points of S_N. These N points are fitted by a straight-line, $y = mx + c$, where m and c represent the slope and the intercept, respectively. As the end points of the line segments are fixed, the slope and the intercept can be estimated as follows,

$$m = \frac{y_N - y_1}{x_N - x_1} \tag{2}$$

$$c = \frac{1}{2}[(y_N - mx_N) + (y_1 - mx_1)] \tag{3}$$

The average distance from the points (x_i, y_i) to the fitted straight line, E_l, can be calculated using the following formula:

$$E_l = \frac{\sum_{i=1}^{N} |(mx_i + c) - y_i|}{N} \tag{4}$$

4.3.2 Fitting a circular arc

S_N can also be fitted as a circular arc, $(x-a)^2 + (y-b)^2 = R^2$, where $C(a, b)$ is the center of the arc and R is the radius.

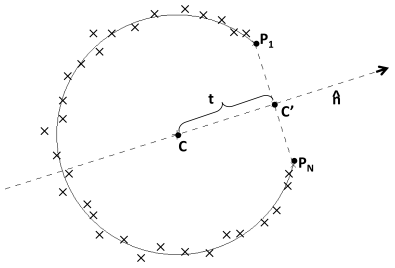


Fig. 7. Least Squares Arc Fitting

As the start and end points of the arc are fixed, the center of the arc should lie on the perpendicular line that passes through the mid-point of the line connecting the end points of the arc (Figure 7). Let $P_1(x_1, y_1)$ and $P_N(x_N, y_N)$ be the end points of the arc, $C'(a', b')$ be the mid point of line joining P_1 and P_N, and $\hat{n} = (n_x, n_y)$, the normal to the line joining P_1 and P_N. Therefore,

$$C' = \frac{1}{2}(P_1 + P_N) \tag{5}$$

$$C = C' + t\hat{n},\ \mathrm{t}\ \varepsilon\ \Re \tag{6}$$

The center, $C(a, b)$ is the solution to the problem

$$\textit{Find C minimizing} \sum_{i=2}^{N-1} \|P_i - C\|^2 \tag{7}$$

using Eqn. 6, the solution to Eqn. 7 is

$$t = \frac{1}{N-2} \sum_{i=2}^{N-1} (x_i - a')n_x + (y_i - b')n_y \tag{8}$$

and hence the radius, $R = \|P_N - C\|$. The average distance from the points (x_i, y_i), i = 1 to N, E_a, can be calculated using the following equation,

$$\mathrm{E}_a = \frac{\sum_{i=1}^{N} |\sqrt{(x_i - a)^2 + (y_i - b)^2} - R|}{N} \tag{9}$$

After finding the errors, the segment is typically classified by the primitive that matches with the least error. However, line segments can always be fit with high accuracy as an arc with a very large radius. In such cases, if the arc length is less than 15 degrees, we classify it as a line. Similarly, an arc is classified as a circle if its arc length is close to 2π.

4.4 Merging

The initial critical points set obtained through segmentation routine may contain some false positives. The merging procedure repeatedly merges adjacent segments, if the fit for the merged segment is lower than a certain threshold. For every i^{th} segment, we try merging it with $i - 1^{st}$ and $i + 1^{st}$ segment. Let these new segments be seg_1 and seg_2. The fit errors for seg_1 and seg_2 are calculated according to section 4.3. For the segment with least error among seg_1 and seg_2, merging occurs if and only if the error is less than the sum of the corresponding errors of the original segments. For example, in figure 4(e), the two lines and an arc on the right were merged into one single arc (see Fig. 4f).

4.5 Geometry Constraint Solving

Geometric constraints are usually classified as either (1) explicit constraints, which refer to the constraints that are explicitly specified by the user such as dimensions - distance between a point and a line or angle between two lines, (2)

	Point	Line	Arc (Circle)
Point	Coincidence, Horizontal Alignment, Vertical Alignment	Point on Line	Coincidence, Point on Arc (Circle)
Line	Point on Line	Parallel, Perpendicular, Collinear	Tangency
Arc (Circle)	Coincidence, Point on Arc (Circle)	Tangency	Tangency, Concentricity

Fig. 8. Implicit geometric constraints inferred in our system for beautification.

implicit constraints, which refer to the constraints that are inherently present in the sketch such as concentricity and tangency. It is natural for users to express geometric constraints implicitly when they are sketching. Our system infers and satisfies the constraints automatically without much intervention from user using the method described in [49]. Figure 8 lists the the different kind of constraints inferred in our system between points, lines, circular arcs and circles. We have integrated the LGS2D [50] geometry constraint solver with our system for constraint solving purposes. The set of primitives along with the constraints are input to the solver and after satisfying the constraints, the solver returns the modified primitives with their new locations. Figure 4 (f) and (g) show the primitives of the sketch before and after constraint solving respectively. The core technology of LGS2D is a combination of symbolic and numerical methods for solving systems of geometrical constraints.The main symbolic method used in LGS2D is a variation of constraint graph analysis, based on abstract degree-of-freedom approach [51].

4.6 Resolving ambiguities with Interaction

Any recognition system is not devoid of ambiguities. Our system provides the interface to correct the errors through simple interactions. Errors in segmentation include missed and unnecessary critical points. In our system, when the user taps on or near a critical point with the stylus, the system first removes that critical point and the corresponding two primitives that share this point. This results in an unrecognized segment which is then classified and refit. The user can also add a segmentation point in a similar manner. The nearest point on the stroke to the clicked location is used as the input point where the existing primitive is broken into two primitives. Errors in segment recognition correspond to primitive misclassification. An input stroke drawn by holding down a button on the stylus is recognized as a pulling gesture. The primitive that is closest to the starting point of this gesture is the one to be pulled and accordingly its classification is altered i.e. if the primitive was a line, it is refit as a circular arc and vice versa. Additionally, the user can erase a primitive, a stroke or a part of stroke using the eraser end of the stylus, just as using a pencil eraser.

5 Symbol Recognition and Sketch Interpretation

The symbols drawn in finite element domain, both in academia and research have well- defined and standardized forms. The list of symbols commonly used in finite element domain (i.e. for loading and boundary conditions) is shown along with other symbols recognized in our system in Fig. 9. Fig. 10 (a) shows an example of beautified 2D bracket drawn in ‘geometry’ mode. The sketch consists of 7 line segments (L1 - L7), two circular arcs (A1 and A2) and a circle (C1). For visual clarity, once the geometry is beautified, the recognized lines are drawn in black and the arcs (circles) in green. Fig. 10 (b) shows the various red colored strokes input in ‘symbol’ mode that represent dimensions, loading and boundary conditions. The following section describes the various steps in processing these strokes for symbol recognition and sketch interpretation (Fig. 10).

5.1 Clustering

The first step in processing this collection of symbol strokes is to cluster them into smaller groups. We use both a temporal and a spatial proximity strategy to group strokes. This stems from the observation that a group of strokes comprising a symbol are generally drawn close to each other and continuously. In addition, the system should not constrain the user to complete a symbol before moving on to the next one. For example, if the user specifies ‘P=100’ as a loading condition initially and later wishes to change it to ‘P=1000’, the operations required must be as simple as adding a zero to the input rather than have to erase and rewrite the whole text again. Hence, the criteria for clustering requires the strokes to be within a spatial threshold distance of 100 HIMETRIC units and (or) the time gap between continuous strokes is less than 500 milliseconds. Figure 10 (c) shows the results of the clustering, where a dashed bounding box is drawn around each group.

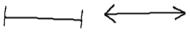
Symbols / Text	Description
	Fully Constrained
	Roller
	Load (force / pressure)
	Moment
	Dimension (Length or Diameter)
	Dimension (Radius)
F, P, T	Force, Pressure, Temperature
=	Equal to
0, 1, 2, 3….9	Numbers (temperature load, dimensions)

Fig. 9. The list of Finite Element symbols recognized in our system

5.2 Text and Symbol Recognition

The next step is recognition of each stroke-group, where each stroke- group is comprised of either text or symbols. We use the height ($<$1.2centimeter) and width of bounding box ($<$2.2centimeter) as the criteria to distinguish between text and symbols. The stroke-groups that are classified as text are next recognized using the built-in handwriting recognizer (Microsoft Tablet PC SDK). The texts in the sketch are primarily of two types: 1) loading conditions (force, temperature or pressure) with alphabets - F, T or P on the left hand side of an 'equal to' symbol and numbers on the right hand side, and 2) dimensions, which are made up of only numbers. This observation helps in robust recognition of text and also helps in correcting misclassification of texts and symbols. After the identification of texts, the next step is to recognize the remaining stroke-groups. On quick observation, one can see that almost all of the symbols are comprised of either lines and (or) circles and only the 'Moment' symbol consists of an arc. Also, some symbols like 'Roller' have different variations, where there is a difference in the number of circles drawn. Though these symbols seem different, there are certain distinct properties for each symbol or group of symbols that are different from other symbols (or groups). For example, the 'fully constrained' symbol is different from 'roller' symbol, as it can be distinguished with the presence or absence of circles. In this case, the number of circles does not matter for the differentiation. We have created similar heuristic based rules to recognize different symbols. The reason behind using such an approach is that the number of symbols in this set is finite and each symbol has some distinct properties that can be used to differentiate from the other symbols in spite of the possible variations. Also, there is no training required. For the recognition of various symbols, we have built custom recognizers by extending SIGER (Simple gesture recognition library) [52] using vector strings and regular expressions.

5.3 Sketch Interpretation

The sketch needs to be interpreted after beautification and symbol recognition. Generally, users draw related objects in such a way that they are closer to each other. We use this observation to associate and group objects to provide context. For example, in Fig. 10 (d), the 'load' symbols, 'P=100' and line L6, combine together to imply the meaning that a pressure load of 100 units is applied on the line in negative y-direction. The various contexts observed in finite element analysis can be classified into three categories, namely *loading conditions*, *boundary conditions* and *dimensions*. Accordingly, the various symbols (Fig. 9) fall into these categories. We use this classification information and spatial proximity reasoning of the bounding boxes to understand the different contexts in the sketch. Applied loads in the system are either point-loads or uniform loads which can be forces, pressure or temperature (depending on the problem). The magnitude and direction of the loads are determined from the text and direction of arrow. When there is only one load symbol detected, it refers to a point-load and the detected load is applied to the nearest point (node) in the geometry. If a pattern of load symbols is inferred next to hand written text, then the closest starting and end points of the arrows are found and the system searches for a nearest

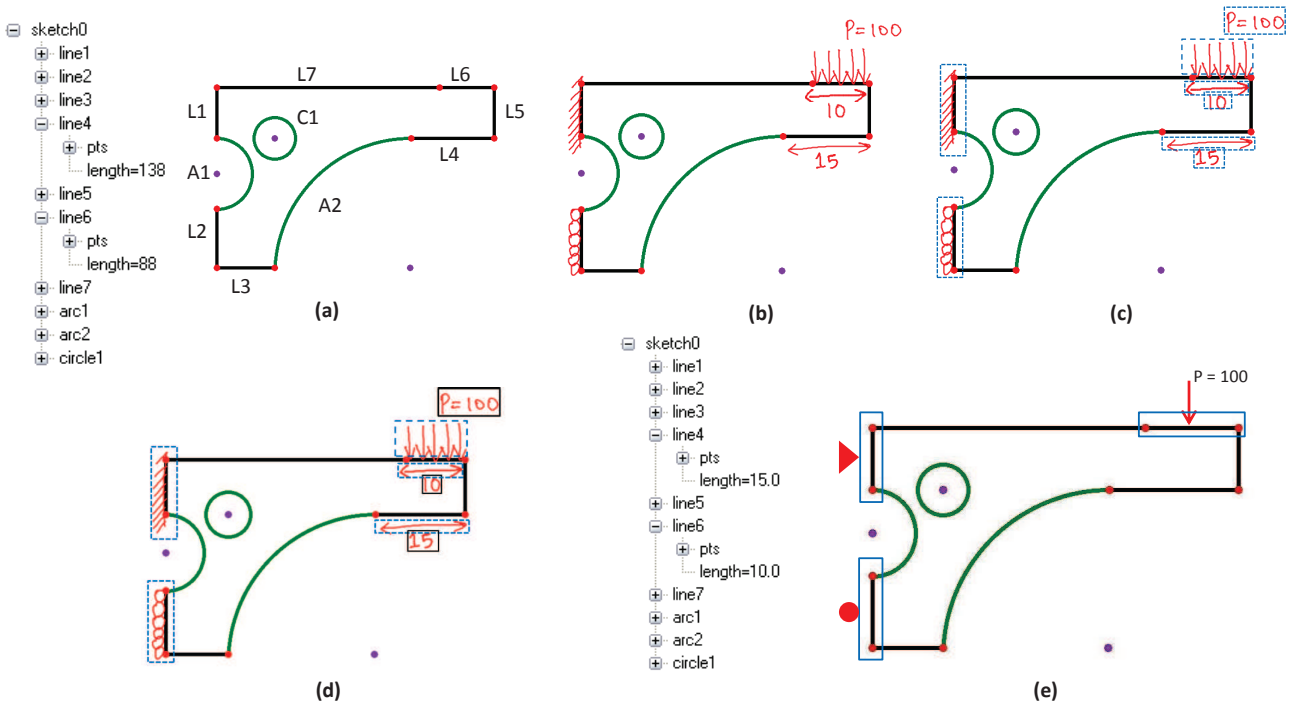


Fig. 10. Symbol Recognition and Sketch Interpretation. (a) A beautified sketch at the end of 'geometry' mode. (b) Red-colored strokes represent the dimensions, loading and boundary conditions drawn in 'symbol' mode. (c) Clustering of strokes into stroke-groups represented by dashed bounding boxes. (d) Classification of stroke-groups into text (black) and symbols (blue). (e) Final sketch after sketch-interpretation

primitive on the geometry and applies to it. The types of boundary conditions are either fully constrained or constrained in only one direction (specified with a roller symbol). The specific direction i.e., x- or y- direction is determined from the orientation of the symbol, for example, like the pattern of circles in the roller symbol. Like loads, boundary conditions can be applied either to a single point or a primitive. Finally, the interpreted dimensional constraints are satisfied by the solver and the sketch gets updated accordingly. Figure 10 (e) shows the final sketch after interpretation of different contexts in the sketch. The freehand input symbols are replaced with recognized text and symbols. Line L1 is 'fully constrained' which is indicated by a bounding box and a triangle, and line L2 is constrained along the x- direction, indicated by a bounding box and a circle. The dimensions of lines L4 and L6 have been updated, which is reflected in the tree.

The text, geometry and symbols in a sketch have an inherent structure and they all combine only in some specific ways. For example, a dimensional value can never be associated with a straight single headed arrow; a loading condition can never be associated with a 'fully constrained' symbol; any arrow cannot exist on its own without an associated text group. This kind of reasoning helps to correct errors automatically allowing for robust sketch interpretation.

6 Finite Element Integration

The final step is to setup the problem for finite element analysis. Our system provides the interface to (see Fig. 3(g)) input the material information, element type and description, and mesh size (if necessary). Our current implementation of the system supports three types of elements which are commonly used in structural, thermal and static finite element analysis. Similarly, the users can also specify what results they wish to view after the analysis. Currently, the system allows users to choose from von Misses stress, reaction forces, deflections and temperature. Figure 3(g)-(i) shows the finite element integration for the bracket in Fig. 10. Here, the three dimensional bracket is modeled as a two-dimensional problem with uniform thickness = 0.5inches. The finite element specific parameters (ANSYS) include material: steel, $\varepsilon = 30e6$, $\nu = 0.3$; element type: PLANE42; element size: 0.5. After specifying the necessary input, the system exports the model geometry, boundary conditions, loads, material, element and meshing information to a unified file specific for ANSYS (APDL commands). Figure 3(h) shows the generated ANSYS specific code and figure 3(i) shows the 'displacement vector sum results' plotted results in ANSYS.

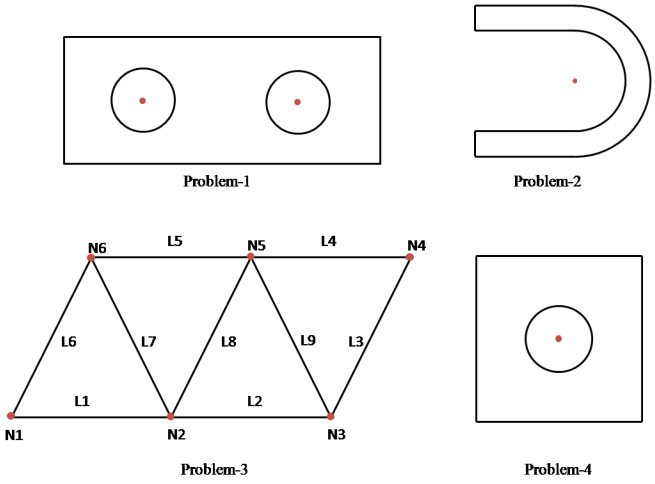


Fig. 11. Problems used in user study.

7 User Study

We conducted a preliminary user study to test the system. Through this study we aimed to find out if the users were able to finish the task given and whether they were able to accomplish it with fewer interactions and strokes than a system that supports only single-primitive strokes . Also, we wanted to receive feedback from participants about the tool for future improvements.

System: Our prototype was ported on to a PC with Wacom Cintiq 21UX LCD display. This display offers the users a way to work naturally and intuitively by using a digital pen, directly on the surface of an LCD display.

Subjects: Six graduate students in mechanical engineering participated in this study and all of them were familiar with sketching aspects of CAD programs (like AutoCAD and Pro/Engineer) and hence were well aware of use of geometric constraints in making diagrams. They were also familiar with using ANSYS for finite element analysis. In addition, they had used digitizing media like Tablet PCs and (or) PDAs before but not the Wacom line of products.

Measurement: The measures in this study included critical points segmentation accuracy, primitives recognition accuracy, symbol recognition accuracy and context interpretation accuracy.

Training Process: The participants were trained for ten minutes with the capabilities of the system, i.e., the two modes of input - 'geometry' and 'symbol'; beautification of the freehand strokes in geometry mode; symbol recognition and sketch interpretation; and finally, finite element integration. In addition to illustrating the work flow, we also demonstrated its limitations, i.e. the system recognizes only lines, arcs and circles and does not handle over-tracing (making several overlapping strokes, such that the strokes are perceived as a single object collectively); interaction techniques (like clicking on a critical point) for correcting errors during beautification and symbol recognition; and finally, symbol recognition might fail when two different symbols overlap each other. In addition, the participants were given 15 minutes to get acquainted with the system.

Task: On the completion of the training process, the participants were asked to sketch and solve the four problems shown in Fig 11. The total amount of time given was one hour. The problems were carefully chosen in such a way that they tested all the different capabilities of our system. In addition these examples illustrate a good range of problems that can be solved using our system. Each problem had a verbal description of the boundary conditions, loads and dimensions (collectively termed as non-geometric information) accompanied by a graphic that represented just the geometry, devoid of dimensions and symbols. The reason behind such a formulation was to analyze how the users input the non-geometric information in the symbol mode and also to remove any bias on how the information should be input. The four problems were chosen in such a way that they were diverse and at the same time be able to test all the capabilities of the system. The problem types were: (1) a static plane stress structural problem (Fig. 11a), (2) a static two-dimensional truss problem (Fig. 11b), (3) a three-dimensional structural problem modeled as a static, two-dimensional problem with constant thickness (Fig. 11c), and (4) a steady-state heat conduction problem (Fig. 11d). The problem descriptions are as follows

Problem 1: Plot the von Mises Stress for the shape in Fig. 11(a) and the following loading conditions. A flat rectangular plate is made of steel (ε = 210000 MPa and ν = 0.3) with two holes and a constant thickness of 0.75cm. The width of rectangular plate = 20cm and height = 10cm. The two holes must be completely inside the plate and on the same imaginary horizontal line, but should not touch the edges of the plate or each other. The left end of the rectangular plate is welded (fully constrained) and a uniform pressure of 0.1MPa acts along the right end of the plate. Use PLANE 42 element in plane stress with thickness and a mesh size of 0.25.

Problem 2: Plot the displacement vector sum for the shape in Fig. 11(b) and loading conditions. The material properties

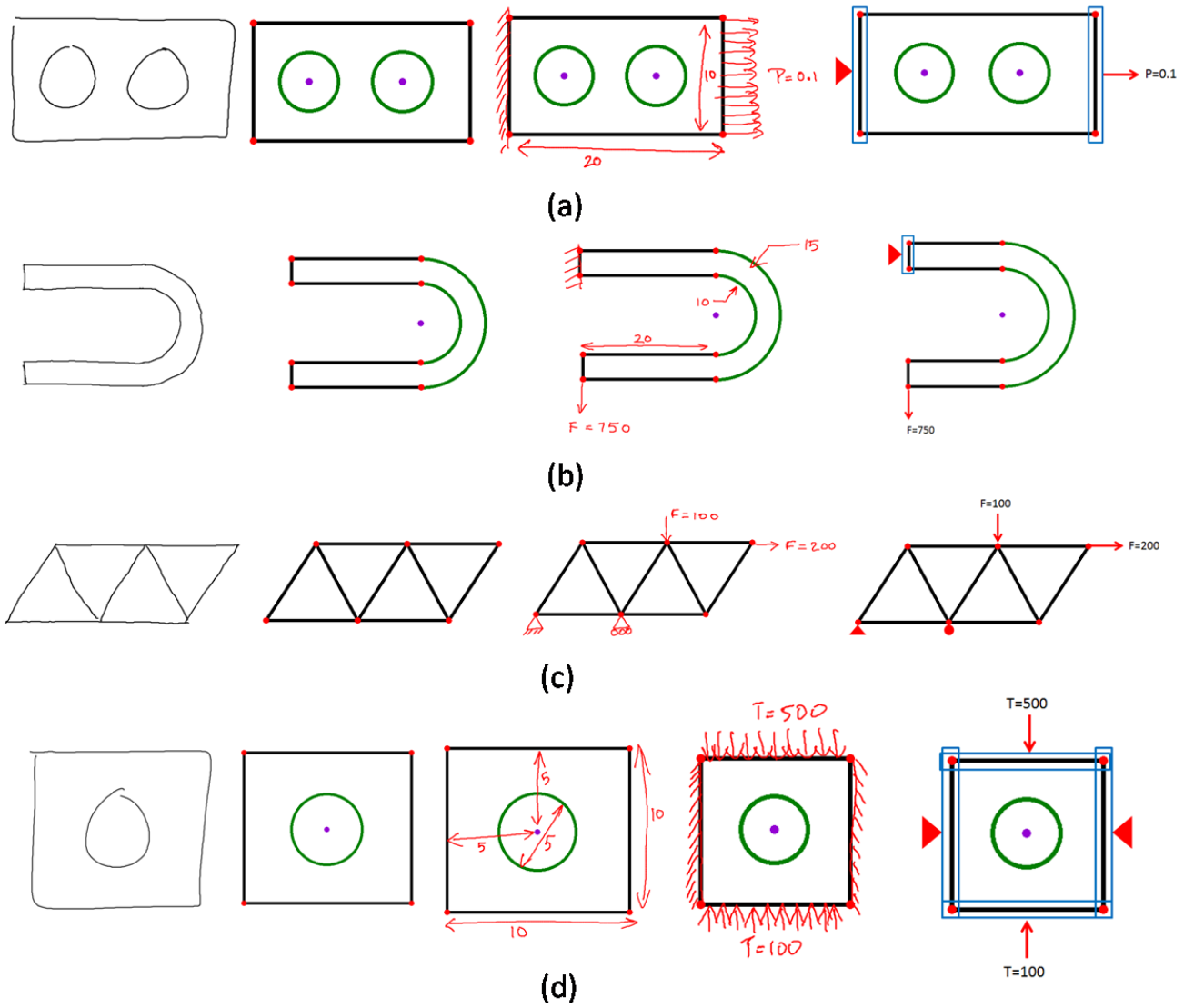


Fig. 12. Sample sketches from participants for the four problems. From left to right, each sub image inset in figures (a) - (d) shows the freehand sketch drawn in geometry mode, the beautified sketch, sketch with input symbols and text drawn in symbol mode, sketch after text & symbol recognition and context interpretation, in that order.

are: ε = 210000 MPa and ν = 0.3. The radius of inner arc = 10 cm and outer arc = 15cm. The arcs are concentric and the length of a horizontal line segment = 20cm. The top left edge is fully constrained and a point load = 750N acts downward at the bottom left point. Use PLANE 42 element in plane stress and a mesh size of 0.5.

Problem 3: Plot the displacement vector sum for the truss in Fig. 11(c) consisting of 6 joints and 9 links. Here, links L1, L2, L4 and L5 are horizontal (parallel to X-axis). Similarly, (L6, L8 and L3) and (L7 and L9) are parallel. Node N1 is fully constrained, while Node N2 is constrained only in the y-direction and free in the x-direction. A load of 100N acts along negative y-direction on N5 and a load of 200N acts along the positive x-direction on N4. The material properties are same as previous examples. Use LINK element with cross sectional area of 0.5 square units.

Problem 4: Plot the temperature contour plot for the shape in Fig. 11(d) and boundary conditions: A square plate (width = 10 and thickness = 1) with a circular hole (diameter = 5) at the center. The top end of the plate is constrained at a temperature = 500 degrees Celsius and the bottom edge at 100 degrees Celsius. The left and right edges are maintained at zero degrees Celsius (fully constrained). The thermal conductivity (k) of the material is 10 W/mC. Use PLANE 55 element in plane stress and a uniform mesh size of 0.25.

Beautification	Total
Total number of strokes	88
Total number of interactions	15
Number of critical points segmented correctly	236
Total number of critical points	238
Critical points segmentation accuracy (%)	**99.16**
Number of segments recognized correctly	173
Total number of primitives	174
Primitives recognition accuracy (%)	**99.43**

Fig. 13. Results of User study - Beautification.

8 Results and Discussion

The six participants all solved the four problems within the allocated time, providing a total of 24 sketches. Some of the sample sketches drawn by the participants are shown in Fig. 12. Each row represents the work flow snapshots taken by the system for each of the problems. Figure 13 summarizes the results obtained after beautification in geometry mode. A total of 88 geometry strokes and 13 interactions were recorded for the geometry part of the problems. The input strokes comprised of both single-primitive and multi-primitive types. A single-primitive stroke means a stroke can represent only one kind of primitive i.e. a line, an arc or a circle. On the other hand, a multi-primitive stroke can represent any number and any kind of primitives connected together. In all, 101 operations were required for successfully completing the geometry for all students and all problems. In contrast, if the system allowed only single-primitive strokes to be input to create the geometry, then the minimum number of total strokes required would be equal to 168, approximately 40% more number of strokes for geometry creation. In addition this number does not reflect the number of operations that would be required to specify geometry constraints. Our system correctly segmented 236 critical points out of 238 with 99.2% accuracy and correctly recognized 173 out of 174 primitives, achieving 99.4% primitive recognition accuracy. These results indicate that we have a robust beautification algorithm and the participants were able to draw the given shapes successfully and at the same time with minimal interactions and lesser time.

Figure 14 shows the results of text and symbol recognition algorithm implemented in the system. Our system correctly clustered and recognized 223 out of 228 stroke-groups with an accuracy of 97.8%. The various symbols recognized and their individual accuracies across both problems and types are shown in Figure 14. Problem 4 had 5 dimensions, 2 boundary conditions and 2 loading (temperature constraints) conditions, which when specified in a single iteration can lead to a crowded sketch and a high chance for overlapping symbols. In one such instance, the fully constrained condition on the left end of the symbol overlapped with the temperature constraint on the bottom edge. The participant had to manually delete the overlapping stroke(s) and process it again. To avoid over crowdedness, 3 of the 6 participants resorted to two iterations of context interpretation, where they specified all the dimensional constraints in the first iteration and all the loading and boundary conditions in the second iteration. This process is similar to traditional finite element systems where users usually finish the problem geometry before specifying other constraints.

Figure 15 shows the results of various contexts interpreted in the user study. Our system correctly interpreted 129 out of 132 contexts in the sketch with an accuracy of 98.5%. The misinterpreted contexts were due to the overlapped symbols as explained in the previous section. The results suggest that our recognition and interpretation algorithms work robustly for the domain of static finite element analysis.

A two-factor analysis of variance test was performed to see if there were any variations in results by problem or by user for the four response variables namely, critical point segmentation accuracy, primitive recognition accuracy, symbol recognition accuracy and context interpretation accuracy. The test showed no significant main effect for the problem factor, $F(3,15) = 3.29, p > 0.05$ and no significant main effect for the user factor, $F(5,15) = 2.90, p > 0.05$ for each of the response variables.

At the end of the user study, each participant was asked if they a) liked the interface and b) had any suggestions for improvement. All of the participants reported that the system was easy to use and expressed a positive attitude towards drawing using freehand sketching. The participants were very appreciative that the system could infer the implicit constraints automatically and satisfy them simultaneously without the need for manually specifying them. Of the six participants, four of them suggested that the system infer symmetry and expand the geometric constraints set that can be either detected automatically or specified manually like equal radii constraints and equal lengths. For example, in even a relatively simple

Symbols	Recognition Accuracy		
	CR	T	%
Fully constrained	29	30	96.67
Roller	6	6	100
Load	35	36	97.22
Dimension	58	60	96.67
Text	95	96	98.96
Total	**223**	**228**	**97.81**

Fig. 14. Results of User Study - Symbol and Text Recognition. (Legend: CR - total number of correctly recognized symbols and T - total number of symbols)

Contexts	Recognition accuracy		
	CI	T	%
Dimensions	60	60	100
Loading conditions	35	36	97.22
Boundary conditions	35	36	97.22
Total Accuracy (%)	**130**	**132**	**98.48**

Fig. 15. Results of User Study - Context Interpretation. (Legend: CI - total number of correctly interpreted contexts and T - total number of contexts)

geometry like in Problem-4, five dimensional constraints were required to construct the square and place a circle at the center. A possible solution is to modify some of the dimensions directly in the left tree. This particular system is currently best suited for exploratory studies in early design where the actual dimensions are not that important in comparison to the shape; for in-classroom demonstrations, where the location of stress concentration or the deflection of a truss member is the focal point of discussion rather than the actual values.

9 Conclusions

In this paper, we described *FEAsy*, a sketch-based interface that integrated freehand sketching with finite element analysis. We presented a beautification method that transforms ambiguous freehand input to more formal structured representations considering the spatial relationships implied in the freehand sketches. We also described algorithms for symbol and text recognition, and interpretation of various contexts in finite element domain. The results from the pilot study indicate that our algorithms are efficient and robust. However, more elaborate studies with a large sample size have to be done to see if such sketch-based interfaces are really a viable alternative. Our immediate future work is to integrate a finite element solver and provide visualization capabilities in the system making it a unified tool for finite element analysis.

Acknowledgements

This work was partly supported by the NSF IIS (Award No. 1422341) as well as the Donald W. Feddersen Chaired Professorship from Purdue School of Mechanical Engineering. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Ullman, D. G., Wood, S., and Craig, D., 1990. "The importance of drawing in the mechanical design process". *Computer and Graphics*, **14**(2), pp. 263–274.

[2] Yang, M. C., and Cham, J. G., 2007. "An analysis of sketching skill and its role in early stage engineering design". *Journal of Mechanical Design*, **129**(5), pp. 476–482.

[3] Shpitalni, M., and Lipson, H., 1997. "Classification of sketch strokes and corner detection using conic sections and adaptive clustering". *ASME Journal of Mechanical Design*, **119**, pp. 131–135.

[4] Yang, M., 2009. "Observations on concept generation and sketching in engineering design". *Research in Engineering Design*, **20**, pp. 1–11.

[5] Igarashi, T., Kawachiya, S., Tanaka, H., and Matsuoka, S., 1998. "Pegasus: a drawing system for rapid geometric design". In CHI '98: CHI 98 conference summary on Human factors in computing systems, ACM, pp. 24–25.

[6] Plimmer, B., and Grundy, J., 2005. "Beautifying sketching-based design tool content: issues and experiences". In AUIC '05: Proceedings of the Sixth Australasian conference on User interface, Australian Computer Society, Inc., pp. 31–38.

[7] Paulson, B., and Hammond, T., 2008. "Paleosketch: accurate primitive sketch recognition and beautification". In IUI '08: Proceedings of the 13th international conference on Intelligent user interfaces, ACM, pp. 1–10.

[8] Pu, J., and Ramani, K., 2007. "Implicit geometric constraint detection in freehand sketches using relative shape histogram". In SBIM '07: Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling, ACM, pp. 107–113.

[9] Veselova, O., and Davis, R., 2006. "Perceptually based learning of shape descriptions for sketch recognition". In ACM SIGGRAPH 2006 Courses, ACM, p. 28.

[10] Murugappan, S., and Ramani, K., 2009. "Feasy: a sketch-based interface integrating structural analysis in early design". In ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, pp. 743–752.

[11] Forbus, K. D., Lockwood, K., Klenk, M., Tomai, E., and Usher, J., 2004. "Open-domain sketch understanding: The nusketch approach". In AAAI Fall Symposium on Making Pen-based Interaction Intelligent and Natural, October, AAAI Press, pp. 58–63.

[12] Landay, J. A., and Myers, B. A., 2001. "Sketching interfaces: Toward more human interface design". *Computer*, **34**(3), pp. 56–64.

[13] Lin, J., Newman, M. W., Hong, J. I., and Landay, J. A., 2001. "Denim: an informal tool for early stage web site design". In CHI '01 extended abstracts on Human factors in computing systems, ACM, pp. 205–206.

[14] McCrae, J., and Singh, K., 2011. "Neatening sketched strokes using piecewise french curves". In Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling, ACM, pp. 141–148.

[15] McCrae, J., and Singh, K., 2009. "Sketching piecewise clothoid curves". *Computers & Graphics*, **33**(4), pp. 452–461.

[16] Sezgin, T. M., Stahovich, T., and Davis, R., 2001. "Sketch based interfaces: early processing for sketch understanding". In PUI '01: Proceedings of the 2001 workshop on Perceptive user interfaces, ACM, pp. 1–8.

[17] Calhoun, C., Stahovich, T. F., Kurtoglu, T., and Kara, L. B., 2002. "Recognizing multi-stroke symbols". In in 2002 AAAI Spring Symposium - Sketch Understanding, (Palo Alto CA, 2002, AAAI Press, pp. 15–23.

[18] Thiel, Y., Singh, K., and Balakrishnan, R., 2011. "Elasticurves: Exploiting stroke dynamics and inertia for the real-time neatening of sketched 2d curves". In Proceedings of the 24th annual ACM symposium on User interface software and technology, ACM, pp. 383–392.

[19] Dae Hyun Kim, M.-J. K., 2006. "A curvature estimation for pen input segmentation in sketch-based modeling". *Computer-Aided Design*, **38**(3), pp. 238–248.

[20] Aaron Wolin, Brian D. Eoff, T. A. H., 2008. "Shortstraw: A simple and effective corner finder for polylines". In Proceedings of Eurographics 2008 - Sketch-Based Interfaces and Modeling (SBIM).

[21] Hse, H., Shilman, M., and Newton, A. R., 2004. "Robust sketched symbol fragmentation using templates". In IUI '04: Proceedings of the 9th international conference on Intelligent user interfaces, ACM, pp. 156–160.

[22] Leslie Gennari, Levent Burak Kara, T. F. S. K. S., 2005. "Combining geometry and domain knowledge to interpret hand-drawn diagrams". *Computers & Graphics*, **29**(4), August, pp. 547–562.

[23] Zhang, X., Song, J., Dai, G., and Lyu, M., 2006. "Extraction of line segments and circular arcs from freehand strokes based on segmental homogeneity features". *Systems, Man, and Cybernetics, Part B, IEEE Transactions on*, **36**(2), April, pp. 300–311.

[24] Xiong, Y., and Jr., J. J. L., 2010. "A shortstraw-based algorithm for corner finding in sketch-based interfaces". *Computers & Graphics*, **34**(5), pp. 513 – 527. CAD/GRAPHICS 2009; Extended papers from the 2009 Sketch-Based Interfaces and Modeling Conference; Vision, Modeling & Visualization.

[25] Taele, P., and Hammond, T., 2014. "Developing sketch recognition and interaction techniques for intelligent surfaceless sketching user interfaces". In Proceedings of the companion publication of the 19th international conference on Intelligent User Interfaces, ACM, pp. 53–56.

[26] Babu, S. S. S., Jaiswal, P., Esfahani, E. T., and Rai, R. “Sketching in air: A single stroke classification framework”.
[27] Wang, C., 2007. “Drawing on air: Input techniques for controlled 3d line illustration”. *Visualization and Computer Graphics, IEEE Transactions on*, **13**(5), pp. 1067–1081.
[28] L.Eggli, H.ChingYao, B.Bruderlin, and Elber, G., February 1997. “Inferring 3d models from freehand sketches and constraints”. *Computer-Aided Design*, **29**, pp. 101–112(12).
[29] Zeleznik, R. C., Herndon, K. P., and Hughes, J. F., 1996. “Sketch: an interface for sketching 3d scenes”. In SIGGRAPH ’96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, ACM, pp. 163–170.
[30] Arisoy, E. B., Orbay, G., and Kara, L. B., 2014. “Predictive modeling for 2d form design”. In *Computational Modeling of Objects Presented in Images. Fundamentals, Methods, and Applications*. Springer, pp. 286–291.
[31] Damm, C. H., Hansen, K. M., and Thomsen, M., 2000. “Tool support for cooperative object-oriented design: gesture based modelling on an electronic whiteboard”. In CHI ’00: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM, pp. 518–525.
[32] Plimmer, B., and Apperley, M., 2004. “Interacting with sketched interface designs: an evaluation study”. In CHI ’04: CHI ’04 extended abstracts on Human factors in computing systems, ACM, pp. 1337–1340.
[33] Lee, Y. J., Zitnick, C. L., and Cohen, M. F., 2011. “Shadowdraw: real-time user guidance for freehand drawing”. In ACM Transactions on Graphics (TOG), Vol. 30, ACM, p. 27.
[34] Benjamin, W., Chandrasegaran, S., Ramanujan, D., Elmqvist, N., Vishwanathan, S., and Ramani, K., 2014. “Juxtapoze: supporting serendipity and creative expression in clipart compositions”. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, pp. 341–350.
[35] Gross, M. D., and Do, E. Y.-L., 1996. “Ambiguous intentions: a paper-like interface for creative design”. In UIST ’96: Proceedings of the 9th annual ACM symposium on User interface software and technology, ACM, pp. 183–192.
[36] Landay, J. A., and Myers, B. A., 1995. “Interactive sketching for the early stages of user interface design”. In CHI ’95: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM, pp. 43–50.
[37] Bae, S.-H., Balakrishnan, R., and Singh, K., 2008. “Ilovesketch: as-natural-as-possible sketching system for creating 3d curve models”. In Proceedings of the 21st annual ACM symposium on User interface software and technology, ACM, pp. 151–160.
[38] Schmidt, R., Khan, A., Singh, K., and Kurtenbach, G., 2009. “Analytic drawing of 3d scaffolds”. In ACM Transactions on Graphics (TOG), Vol. 28, ACM, p. 149.
[39] Naya, F., Contero, M., Aleixos, N., and Company, P., 2007. “Parsketch: A sketch-based interface for a 2d parametric geometry editor.”. In HCI (2), J. A. Jacko, ed., Vol. 4551 of *Lecture Notes in Computer Science*, Springer, pp. 115–124.
[40] Laviola, J. J., and Zeleznik, R. C., 2006. “Mathpad2: a system for the creation and exploration of mathematical sketches”. In SIGGRAPH ’06: ACM SIGGRAPH 2006 Courses, ACM.
[41] Kara, L. B., Gennari, L., and Stahovich, T. F., 2008. “A sketch-based tool for analyzing vibratory mechanical systems”. *Journal of Mechanical Design*, **130**(10).
[42] de Silva, R., Bischel, D. T., Lee, W., Peterson, E. J., Calfee, R. C., and Stahovich, T. F., 2007. “Kirchhoff’s pen: a pen-based circuit analysis tutor”. In Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling, ACM, pp. 75–82.
[43] Hutchinson, T. C., Kuester, F., and Phair, M. E., 2007. “Sketching finite-element models within a unified two-dimensional framework”. *Journal of Computing in Civil Engineering*, **21**(3), pp. 175–186.
[44] Fonseca, M. J., and Jorge, J. A., 2001. “Experimental evaluation of an on-line scribble recognizer”. *Pattern Recogn. Lett.*, **22**(12), pp. 1311–1319.
[45] Kara, L. B., and Stahovich, T. F., 2005. “An image-based, trainable symbol recognizer for hand-drawn sketches”. *Computers & Graphics*, **29**(4), pp. 501 – 517.
[46] Johnson, G., Gross, M., Do, E. Y.-L., and Hong, J., 2012. “Sketch it, make it: sketching precise drawings for laser cutting”. In CHI’12 Extended Abstracts on Human Factors in Computing Systems, ACM, pp. 1079–1082.
[47] Wenyin, L., 2003. “On-line graphics recognition: State-of-the-art”. In GREC, pp. 291–304.
[48] Chen K.-Z., Zhang X.-W., O. Z.-Y. F. X.-A., January 2003. “Recognition of digital curves scanned from paper drawings using genetic algorithms”. *Pattern Recognition*, **36**, pp. 123–130(8).
[49] Murugappan, S., Sellamani, S., and Ramani, K., 2009. “Towards beautification of freehand sketches using suggestions”. In Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling, ACM, pp. 69–76.
[50] Ltd, L. http://www.ledas.com/products/lgs2d/, accessed on feb 25th 2009.
[51] Bouma, W., Fudos, I., Hoffmann, C., Cai, J., and Paige, R., 1995. “Geometric constraint solver”. *Computer-Aided Design*, **27**(6), pp. 487 – 501.
[52] Swigart, S., 2005. “Easily write custom gesture recognizers for your tablet pc applications”. *Tablet PC Technical Articles*.