Proceedings of the ASME 2009 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2009
August 30 - September 2, 2009, San Diego, CA, USA

DETC2009-87727

FEASY: A SKETCH-BASED INTERFACE INTEGRATING STRUCTURAL ANALYSIS IN EARLY DESIGN

Sundar Murugappan
School of Mechanical Engineering

Karthik Ramani*
School of Mechanical Engineering,
School of Electrical and Computer Engineering (by courtesy)

Purdue University, West Lafayette, IN 47906, U.S.A.

ABSTRACT

The potential advantages of freehand sketches have been widely recognized and exploited in many fields especially in engineering design and analysis. This is mainly because the freehand sketches are an efficient and natural way for users to visually communicate ideas. However, due to a lack of fundamental techniques for understanding them, sketch-based interfaces have not yet evolved as the preferred computing platform over traditional menu-based tools. In this paper, we address the specific challenge of transforming informal and ambiguous freehand inputs to more formalized and structured representations. We present a domain- independent, multi-stroke, multi- primitive beautification method which detects and uses the spatial relationships implied in the sketches. Spatial relationships are represented as geometric constraints and satisfied by a geometric constraint solver. To demonstrate the utility of this technique and also to build a natural working environment for structural analysis in early design, we have developed *FEAsy* (acronym for ***F****inite* ***E****lement* ***A****nalysis made ea****sy***) as shown in Fig. 1. This tool allows the users to transform, simulate and analyze their finite element models quickly and easily through freehand sketching, just as they would draw on paper. Further, we have also developed simple, domain specific rules-based algorithms for recognizing the commonly used symbols and for understanding the different contexts in finite element modeling. Finally, we illustrate the proposed approach with a few examples.

KEYWORDS Sketch-based interfaces, beautification, CAD, Finite Element Analysis

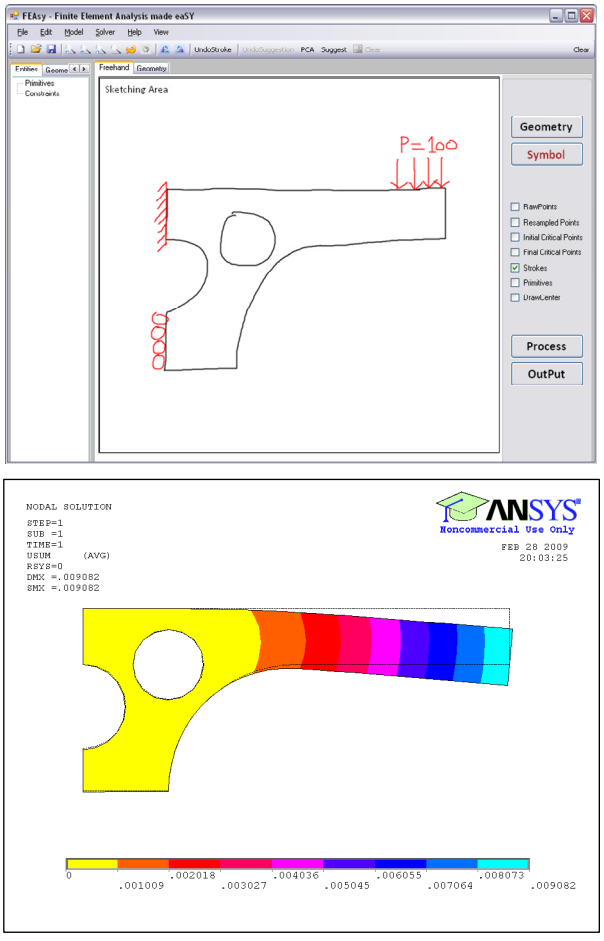


Figure 1. THE FEASY INTERFACE SHOWING A HAND-DRAWN SKETCH OF A 2D BRACKET EXAMPLE (TOP) AND DEFORMATION RESULTS IN ANSYS (BOTTOM).

1 INTRODUCTION

Over the past few years, there has been increased interest to support freehand sketching in user interfaces and tools for

* Professor and author of correspondence, Phone: (765) 427-7945, Email: ramani@purdue.edu.

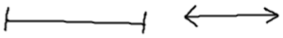
various applications in diverse domains such as Computer aided Design (CAD), simulation, computer animation and software design. The main motivation is based on the fact that freehand sketching is a natural, efficient and convenient way to visually represent ideas. Also, as sketch-based interfaces mimic the pen-paper paradigm of interaction, they provide a host of advantages over the traditional windows, Icons, menus and pointers (WIMP) style Graphical User Interfaces (GUI). The users can seamlessly and directly interact with the computer and require practically limited or almost no training, whereas in menu based interfaces, the users are forced to learn the system rather than the system having to learn the users' intentions. Further, studies in cognitive science have shown that sketching does not break the users thought process [1]. Due to this reason, sketches are particularly useful in early stages of design where their fluidity and ease of construction enable creativity and the rapid exploration of ideas [2]. Despite the potential benefits, sketch-based interfaces have not yet evolved as the computing platform of choice. One of the major requirements of such interfaces is the need for fundamental techniques to transform the informal and ambiguous freehand inputs to more formalized and structured representations. Such a transformation is a crucial step in sketch understanding, for assisting rapid creation and evaluation of new ideas and also in reduction of the total time and effort spent in creating drawings on a computer. This process is termed as *beautification*. Another such challenge is *symbol recognition*, the task of recognizing shapes and symbols.

Two of the main challenges that have hindered the development of a robust beautification system are: *Segmentation* - identification of critical points on the strokes, *Recognition* - classifying the segment between adjacent critical points as low level- geometric primitives (like lines, circles and arcs). Many of the current methods place many constraints on how users must draw shapes. For a natural sketch-based interface, there should not be any limitations on the users drawing style or any dependency on how a particular sketch is drawn (number or order of strokes or the number of primitives in the stroke). In this paper, we present a domain-independent, multi-stroke, multi-primitive beautification method that allows the users to draw in an unrestricted fashion and at the same time robustly copes with the imprecision and variation in freehand input.

When beautification is done on per-stroke level, it is termed as *local*. Local beautification often misses some important *global* information in sketches such as the spatial relationships between different primitives in a stroke and between strokes. These spatial relationships are usually represented as geometric constraints (like parallelism and tangency). Cognitive studies show that users attend preferentially to certain geometric features while drawing and recognizing shapes. Pu et al [3] highlight the role of such geometric constraints in beautification. Our method identifies these spatial relationships and uses them to drive beautification.

Further, to demonstrate the utility of this technique, we have developed a sketch-based interface for static finite element analysis. The primary rationale for the proposed tool is that the contemporary software tools for Finite Element Analysis (FEA) and simulation, like ANSYS® and SIMULIA® have a steep learning curve. Even, to solve simple one (1D) and two dimensional (2D) problems, the users have to follow a very tedious process. Usually, a FEA problem can be broken into four stages: 1) Initial setup which involves element and material selection, description, 2) creating geometry, 3) defining loading and boundary conditions and 4) meshing, solving and visualization of results. Among these steps, geometry creation and specifying load and boundary conditions are the crucial and time consuming process for the user. Also, users experience greater cognitive load causing deterioration in speed, attention focus, meta-cognitive control, correctness of problem solutions, and memory in using traditional systems [4]. This sketch-based tool would allow the users to create geometry as they would draw on paper with minimal constraints imposed on them.

Table 1. THE LIST OF FINITE ELEMENT SYMBOLS RECOGNIZED BY OUR SYSTEM

Symbols	Description
	Fully Constrained
	Roller
	Load (Force / pressure)
	Moment
	Dimension (Length or Diameter)
	Dimension (Radius)

Also, it is more intuitive to specify loading and boundary conditions through symbols and gestures as shown in Fig. 1 than through traditional menu-based input. Hence, we have developed a domain-specific algorithm for recognizing common symbols used in FEA. The various symbols recognized are shown in Table 1. Our ultimate goal is to create a unified framework for 1D and 2D finite element analysis that integrates all the four stages in one platform. As one step towards the goal, our system to date would allow the users to set-up the problem up to the simulation stage (i.e., except actual solving and visualization) as they would do on paper naturally and intuitively. The system would then export the model geometry, boundary conditions, loads, material and element description and meshing parameters in a unified data file as a set of commands (like a script) specific to the FEA software. The users can then run this file to get the desired results. A few examples explaining the procedure for ANSYS

is shown in section 8. Similarly it can be extended to other tools.

In addition to design, this tool can also be used in engineering education. 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. This can also be used as a potential tutoring tool for teaching assistants and as a lecture-aid for instructors.

The remainder of this paper is organized as follows. Section 2 provides some background and related work. This is followed by the description of the system in section 3. In section 4, we present the beautification algorithm and in section 5, the symbol recognition. Section 6 explains the sketch interpretation followed by finite element integration in section 7. Results and discussion are presented in section 8. We conclude with the limitations of our current implementation, ways to address them and future work in section 9.

2 RELATED WORK

2.1 Beautification – Segmentation and Recognition

Much of the earlier work [5–8], assumed that each pen stroke represented a single primitive such as a line segment or a curve. Despite their simplicity, the strategy based on single primitive/stroke usually results in a less natural interaction because of the constraints imposed on the users drawing freedom. For a natural freehand sketching interface, users should be allowed to draw multiple primitives with a single stroke and segmentation algorithm should take care of finding the critical points. Taking advantage of the interactive nature of sketching, Sezgin et al [9] and Calhoun et al [10] 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. Also, the speed data is dependent on recording the sketching activity in real time. Kim and Kim [11] proposed new metrics based on curvature - local convexity and local monoticity for segmentation. Aaron et al [12] 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 [9] and [11]. Other approaches to segmentation utilized artificial intelligence, such as the template-based approach [13], conic section fitting [2], and domain-specific knowledge [14]. 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 [2], otherwise, the segmentation performance will be affected. Also, the variance of freehand sketches makes it difficult to define unified models or rules for sketch segmentation utilizing different applications.

Recognition is usually performed at two levels: the composite (higher) or symbol level and primitive (low) level. Calhoun et al [10], Shpitalni et al [2] and Zhang et al [15] used least-squares based method for recognition. In [16], Qin's classification was based on the curve's linearity, convexity and complexity. For a detailed description of different techniques we refer to [17]. Our beautification method improves upon the technique described in [12]. We address its drawbacks i.e., recognizing corners at heavily obtuse angles. Also, we identify arcs and circles in addition to only the line segments. Further, our method can be easily extended to support other geometric primitives like splines. We explain our method in detail in section 4.3.

For symbol recognition, Fonseca et al. [18] 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 aggregate features of the pen strokes, it might be difficult to differentiate between similar shapes. Kara et al. [19] described a hand-drawn symbol recognizer based on a multi-layer image recognition scheme. It can learn new symbol definitions from a single prototype example and is insensitive to translation, rotation, and uniform scaling. However, this method requires training and is sensitive to non-uniform scaling. Veselova et al [20] used results from perceptual studies to build a system capable of learning descriptions of hand-drawn symbols which are invariant to rotation and scaling. Accurately recognizing symbols, independent of the domain is a challenging problem itself and not the focus of this research. Hence, for our needs to reliably recognize finite element domain specific symbols, independent of translation, rotation and scaling, we developed a rules-based approach using results from perceptual studies like [20].

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 [21] In CAD based applications like [22] and [23], the user has to draw objects in pieces, reducing the sense of natural sketching. Our system allows drawing strokes with multiple primitives without any restriction on the user. Sketch based interfaces have also been used in early software design [24, 25]; user-interface design [26]. Gesture- based systems have been explored in 2D pen-based applications [27, 28] where input strokes are converted or replaced with predefined primitives. CALI [18] is an online recognition library of simple gestures to create and edit 2D shapes for diagrams. ParSketch [29] is sketch-based interface for editing 2D parametric geometry. MathPad [30] is a tool for solving mathematics problems. Kara et al developed a sketch-based system for vibratory mechanical systems. STRAT [31] is a pen-based tool for students to learn standard truss analysis. Hutchinson et al. [32] developed a unified framework for finite element analysis. They used an existing freehand sketch recognition interface. However, the users still had to navigate through a lot of menus to specify input. Also, the beautification method is not robust, constraining the users' drawing freedom. Moreover, the system does not address the problems related to the ambiguous nature of freehand input. We have developed methods for resolving ambiguity through interaction, described in section 4.6. Krichoff's pen [33] is a pen-based tutoring system that teaches students to apply Kirchhoff's voltage law and current law.

3 SYSTEM OVERVIEW

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 [17]. Our sketch-based interface allows the user to draw in a natural way just as they would do on paper. The user can draw freely and there is no restriction on how a particular shape or symbol is drawn and text is written. One of the important challenges in such a sketch-based interface is to robustly classify the pen-strokes as geometry, text and gestures, which in itself an active area of research and not the focus of this work. Lineogrammer [34] provides heuristics for modelessly disambiguating between text, geometry elements and command gestures. For example, to distinguish between text and geometry, one of the criteria, they used is that the size of text must be small (< 2 cm) and (or) the gap between continuous strokes is less than 500ms. We use similar heuristics to distinguish between hand-written text, gestures and geometry elements. However, both the FEA problem's geometry and symbols are composed of low-level primitives like lines and arcs. Also, to maintain the naturalness of the interface, the stroke should be able to represent any number of shape primitives connected together.

The symbols or geometry can be specified in a single stroke or multiple strokes and there should be no requirements that the parts of symbol/geometry be drawn in the same order. Hence to accommodate these requirements and simplify recognition, we ask the user to switch pen colors (i.e. switch modes) while specifying geometry or symbols. Text can be input in any mode. In addition to reducing ambiguity, multi-color would provide visual clarity to the sketch drawn. The low-level primitives recognized in the current implementation of our system include lines, circles, and circular arcs.

Sketches can be created in our system using any of a variety of devices that closely mimic the pen-paper paradigm. We use 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. The system uses Microsoft Tablet PC SDK to capture the user input strokes and for handwriting recognition. The users can view their sketch either as raw pen strokes and (or) as primitives.

An example of the sketching process is shown in Fig. 1. The user starts sketching in 'geometry' mode and the input stroke is first classified as text or geometry. If the system classifies it as text, then the screen is updated with the recognized text label. Or else, each input stroke is 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. For this purpose, we have integrated LGS2D [35] in our system. The output from the solver is the beautified version of the input which is updated on the screen. The user can continue sketching or switch modes. The strokes input in 'symbol' mode are processed similarly as in 'geometry' mode.

When the user selects the 'Process' button for processing, the symbols are then sent to the 'symbol recognition' module for recognition. Then, the symbols, text and geometry are grouped and associated with each other for correct interpretation of context.

For example in Fig. 2, the *load* (downward single-arrow symbol) is associated with *F=100* (text) and point P_3 implying that a force of 100 units in negative 'y direction is applied at P_3. Similarly, the double-headed arrow is associated with *20* units and line L_1 implying that a dimension constraint equal to 20 for length of L_1 is added to the geometry and satisfied by the geometry constraint solver. Once the sketch is complete and processed, the user selects the 'OUTPUT' button, which starts the 'FE integration' causing a dialog to pop-up, where the user specifies the material properties, element description and meshing parameters (if any). The user uses a scratching gesture to indicate if the area needs to be meshed. When the dialog is closed successfully, the sketch is exported as a unified file suitable for import in ANSYS.

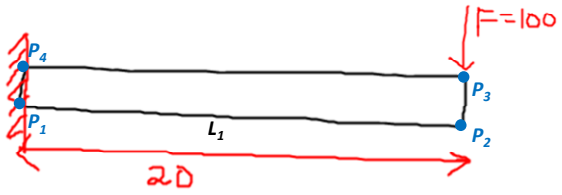


Figure 2. A FREEHAND SKETCH WITH THE PROBLEM GEOMETRY (IN BLACK) AND SYMBOLS (IN RED)

Common interpretation errors can be easily corrected in our system. Correction of errors in beautification is explained in section 4.6. When a symbol is recognized incorrectly, the user can explicitly mark it by holding down a button on the stylus and circling it. A pop-up menu will open, where the user can choose from a list of possible classifications. As the FEA domain is small, the number of misclassifications will be a minimum. When text is misinterpreted as geometry, drawing a line across by holding a button on the stylus will change it to text.

4 BEAUTIFICATION

Beautification aims at simplifying the input where the various points of the strokes are interpreted and represented in a more meaningful manner. Our approach to transforming the input to formalized representations (i.e. beautification) is based on the architecture shown in Fig. 3. There are five steps in the pipeline namely - resampling, segmentation, recognition, merging and geometry constraint solving. An example of the user drawn freehand stroke is shown in Fig. 3(a). Figure 3(b) shows the raw points (blue circles) as sampled by the hardware and Fig. 3(c), 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. 3(d). Then, the segments between the adjacent critical points are recognized and fit with primitives (Fig. 3(e)). The status of the freehand sketch after merging is shown in Fig. 3(f) and finally the sketch is beautified considering the geometric constraints (Fig. 3(g)). The various 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.

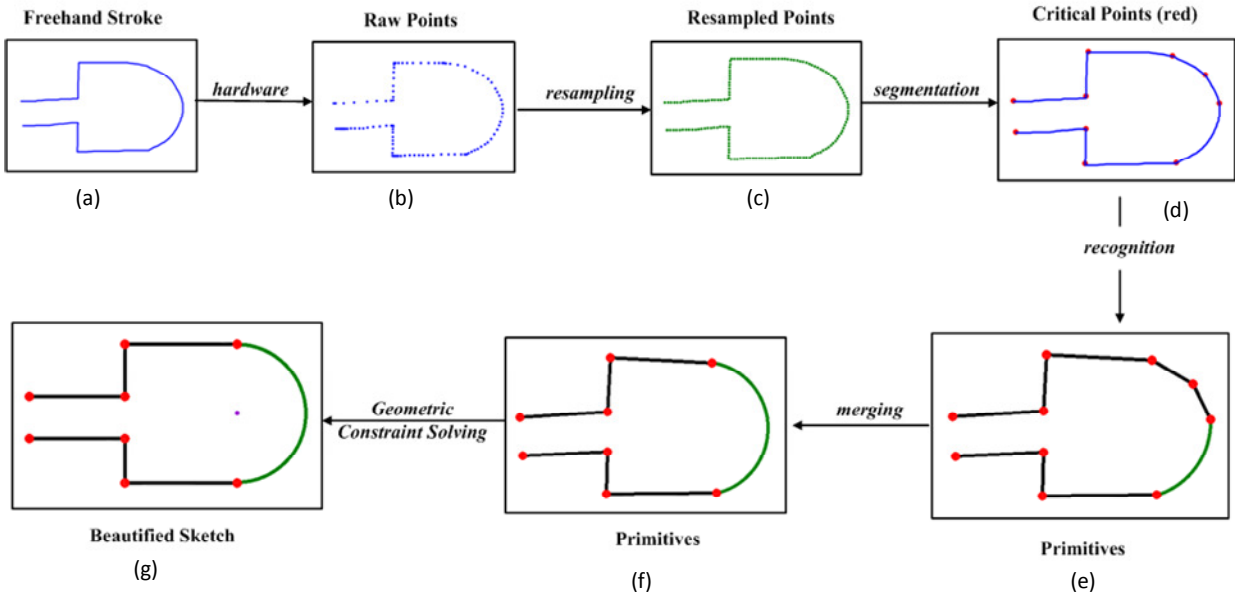


Figure 3. BEAUTIFICATION OF A FREEHAND STROKE.

4.1 Stroke Resampling

Freehand input is prone to contain some noise. Sezgin [36] identified two main sources - imprecise motor control and digitization. As the interface places no constraint on the users' drawing freedom, we ignore the noise due to the slight tremble of hand or poor drawing skills of the user. We consider the input as it is and all input points are considered as genuine. However, 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. To achieve uniform sampling, we resample the points of the input stroke such that they are evenly spaced. We used a fixed interspacing distance, I_d 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 3(c) shows the result of resampling for the stroke (Fig 3a) using our method. Uniform resampling is important for the segmentation algorithm to work efficiently.

4.2 Segmentation

In our system, a single freehand stroke can represent any number of primitives connected together. The task of 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 [12], which works well for strokes composed of only line segments. One of the drawbacks of this method is that the algorithm often misses identifying the 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.

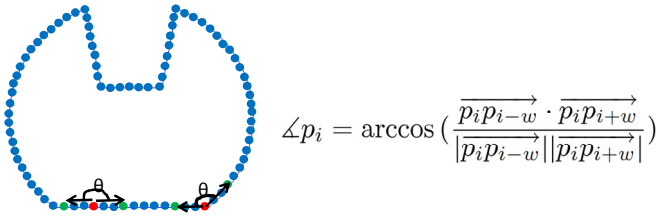


Figure 4. CHORD ANGLE' COMPUTATION.

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 4 shows the computation of the chord angle. The blue circles represent the resampled points and *'θ'* represents the `chord angle' computed using the formula inset in the Fig. 4. 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. 3(a), the initial set of critical points obtained is shown in Fig. 3(d). By default, the start and end points of a stroke are considered as critical points. Using a window of uniformly spaced points to compute the curvature (chord angle), smoothes out the noise, if any in the input stroke. The larger the window, the larger the smoothing effect resulting in missed critical points. Like [10], 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. The recognition method is based on least squares analysis [37], 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 location. For example see Fig. 5. To prevent such discontinuities between adjacent primitives of the stroke, we fix the endpoints of the primitives (as it is) of the original segment and then perform the analysis. Figure 5(c) shows the result of our recognition algorithm.

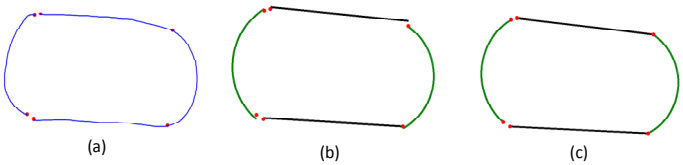


Figure 5. RECOGNITION OF SEGMENTS.

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.

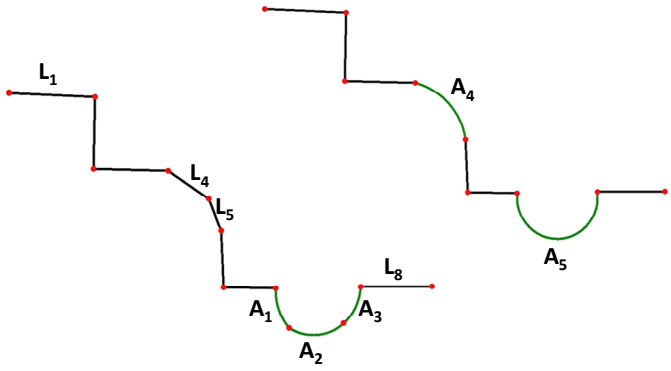


Figure 6. THE MERGING ROUTINE.

For every i^{th} segment, we try merging it with $i\text{-}1^{st}$ 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 Fig. 6, the two lines (L_4 and L_5) are merged into an arc (A_4) and the three arcs (A_1, A_2 and A_3) yield one arc (A_5).

4.5 Geometry Constraint Solving

Pu and Ramani highlight the importance of geometric constraints in beautification in [3]. Geometric constraints are widely used and an integral part in many design related applications, such as drawing programs, CAD tools and graphical user interfaces [3].

Table 2. IMPLICIT GEOMETRIC CONSTRAINTS INFERRED FOR BEAUTIFICATION

	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

In current drawing systems, specifying geometric constraints is a difficult, time consuming and a tedious task. Also, the users need to undergo prior training before using the system. Novice users must be made aware of geometric constraints and how these can be used to create what they want. To reduce this cognitive overload and to effectively support the use of geometric constraints in freehand sketching, the system infers and satisfies the constraints automatically without much intervention from user. 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) *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. Table 2 lists the the different constraints inferred in our system.

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 (or left click using a mouse), 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. However, the start and end points of a stroke cannot be removed by clicking unless the stroke

represents an arc. In such a case, clicking on one of the end points, converts the arc to a circle.

Errors in recognition correspond to primitive misclassification. An input stroke drawn by holding down a button on the stylus (or both left and right mouse button together) 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. This gesture does nothing for a circle because a line (circle) can never be misinterpreted as a circle (line) and moreover adding a critical point to a circle breaks it into two arcs. 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

The strokes classified as symbols are first simplified and represented as low-level geometric primitives by the beautification module. This simplification helps in reducing the variability that may exist in comparison to correctly identify complex symbols. 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 Table 1.

Table 3. THE DIFFERENT VARIATIONS OF (A) '*ROLLER*' AND (B) *FULLY CONSTRAINED* SYMBOLS.

(a)		**Roller**
(b)		**Fully Constrained**

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. The various symbols are configurations of these low-level primitives. Also, some symbols like 'Roller' have different variations (Table. 3a), where there is a difference in the number of circles drawn. A good symbol recognition algorithm must account for these variations. 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, in Table 3, the 'fully constrained' symbol is different from 'roller' symbol, as it can be distinguished with the presence of circles. In this case, the number of circles does not matter in differentiating between them. 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.

6 SKETCH INTERPRETATION

The sketch is 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. 8(a), the 'load' symbol, 'F=1000' and the top middle node combine together to imply the meaning that a load of 1000 units is applied on the node (point) in negative y-direction. The various context observed in finite element analysis can be classified into three categories, namely *Loading Conditions*, *Boundary Conditions* and *Geometric constraints.* Accordingly, the various symbols (Table 1) fall into these categories. We use this classification information and spatial proximity reasoning to create rules for understanding the different contexts in the sketch. Applied loads in the system are either point-loads or uniform loads which can be both forces and pressure (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 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.

7 FE INTEGRATION

When the user presses the 'Output' button, the system shows a dialog box, where the user enters the material information, element type and description, and mesh size for finite element analysis. Finite element solvers like ANSYS do not require any units, but assume that the input is consistent. Our system also behaves along the similar lines. Hence, it is on the part of the user to make sure that the input units are consistent. Our current implementation of the system supports two types of elements which are commonly used in structural and static finite element analysis. They are 1) Two dimensional- spar element, a uni-axial tension-compression element with two degrees of freedom at each node, i.e., translations in the nodal x- and y-directions. This element is used for modeling trusses. The equivalent element in ANSYS is LINK1, 2) Two dimensional structural solid element, for modeling solid structures. The equivalent element in ANSYS is PLANE42. The element can be used as a plane stress, plane strain or an axisymmetric element. The element is defined by four nodes having two degrees of freedom at each node: translations in the nodal x and y directions. The element properties include Young's modules ε, and Poisson's ratio ν, as well as geometric information, such as cross-sectional area. Specifying the mesh size is optional and it is enabled only for

structural solid elements. Similarly, the users can also specify what results they wish see after the analysis, like for example the displacement sum of nodal solution. Currently, the system allows the users to choose from von misses stress, reaction forces and deflections. On closing this dialog successfully, the system exports the input i.e., model geometry, boundary conditions, loads, material, element and meshing information to a unified file specific for ANSYS. This file can then be run in ANSYS to yield the desired results.

8 RESULTS AND DISCUSSION

In this section, we present some examples generated using our system. First, we show a simple example of a two dimensional cantilever beam with a point-load applied at its end to illustrate the sketching process. The user starts by sketching a rectangle with one input stroke in geometry mode as shown in Fig. 7(a). When the user lifts the pen, the system beautifies the input and the result is shown in Fig. 7(b). Then, the user switches to 'symbol' mode to specify the boundary conditions, loads and dimensions. On pressing the 'process' button, the system processes the input and the result is shown in Fig. 7(c). The user can repeat this process to make changes or can proceed further. When the user presses the 'output' button, the system opens the integration dialog. The user can choose from a library of materials or modify existing material or specify new properties. For this problem, the user chose steel with ε = 30e6 and ν = 0.3. Similarly, the element type (PLANE 42) and other parameters are input. On closing the dialog, the system exports the input as a set of commands specific to ANSYS shown in Fig. 7(e). The result (displacement vector sum plot) obtained on running this file in ANSYS is shown in Fig. 7(d). This model is an ideal example for students to start learning finite element analysis.

Figure 1 shows an example of a three dimensional bracket, modeled as a two–dimensional problem with uniform thickness = 0.5 inches. The initial freehand sketch (without beautified geometry) and the final ANSYS output are shown at the top and bottom of Fig.1 respectively.

To illustrate an example with two-dimensional spar element (LINK1), a truss problem is presented. Figure 8 shows the freehand sketch, output command file and the generated results in ANSYS. (The dimensions are not shown explicitly to avoid cluttering the scene). As this model does not require any area meshing, the geometry is modeled as a set of nodes and elements between them instead of key points and lines. The material properties used are the same as the cantilever beam with the input cross-sectional area = 0.5 sq. inches.

9 CONCLUSION

In this paper, we addressed the specific challenge of transforming ambiguous and informal freehand input to more formalized representations i.e., beautification. In this paper, we addressed the specific challenge of transforming ambiguous and informal freehand input to more formalized representations i.e., beautification. To this extent, we created a beautification method that is not domain specific, supports multi-primitives like lines, arcs and circles, and considers the spatial relationships implied in the freehand sketches.

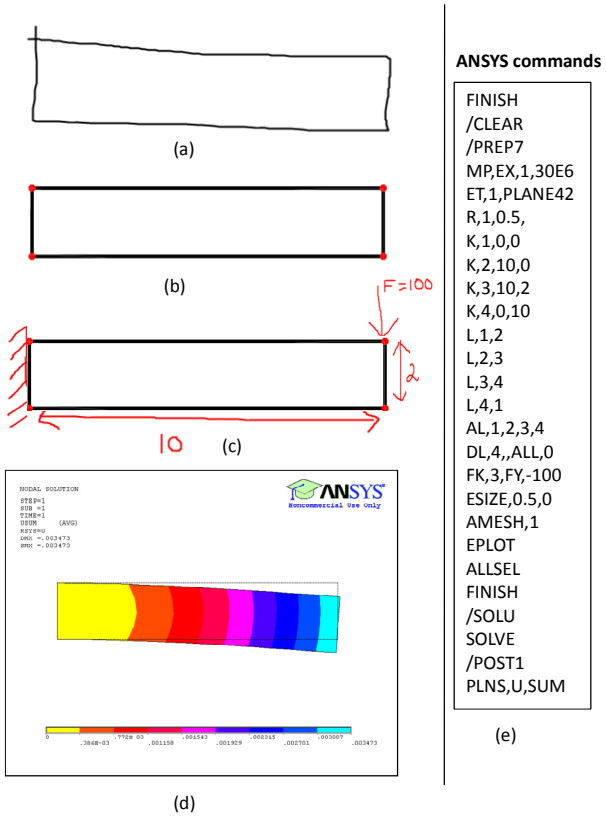


Figure 7. TWO DIMENSIONAL CANTILEVER BEAM SUPPORTED BY A POINT LOAD - 100LBS AT THE END

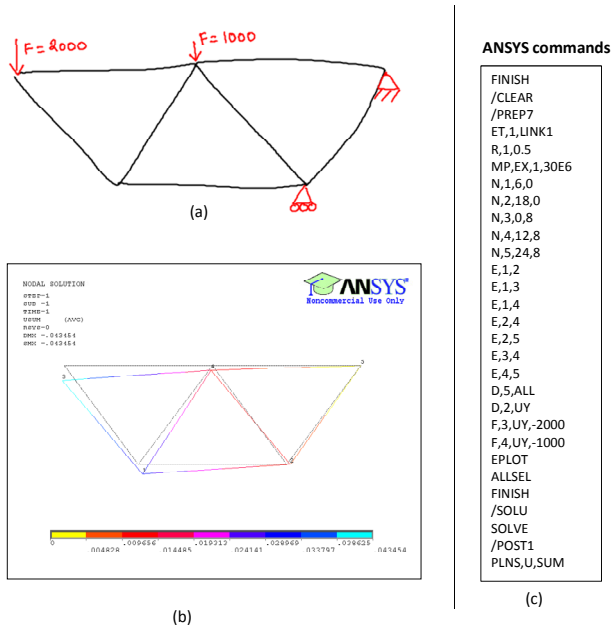


Figure 8. TRUSS ANALYSIS WITH TWO POINT LOADS - 1000LBS AND 2000LBS.

To demonstrate the utility of this method and also to create a natural working environment for designers, we developed *FEAsy*, a sketch-based interface that integrated freehand sketching with finite element analysis. We showed the capabilities of the system and also how this system can be used to evaluate ideas especially in early design and also as a potential learning tool for students. Most notably, all models were created quickly and in a more natural manner just as one would draw on paper. Anyone familiar with the finite element analysis domain can learn the system quickly with very little training. The current implementation of the system, exports the model geometry, loading and boundary conditions into a unified file that can be run in ANSYS to visualize results. Our ultimate goal is to create a unified framework from sketching to visualization of results. Hence, our immediate future work is to integrate a finite element solver and provide visualization capabilities in the system. Future improvements include supporting other types of analysis (apart from structural) and also to extend to three dimensions.

10 ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation Division of Information and Intelligent Systems (NSF IIS) under Grant No. 0535156. This work was done in collaboration with PARC (formerly Xerox PARC). We would like to acknowledge Dr. Eric Saund for his suggestions during this work. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

11 REFERENCES

[1] Davis, R., 2002. “Sketch understanding in design: Overview of work at the MIT AI lab”. In Sketch Understanding, Papers from the 2002 AAAI Spring Symposium, AAAI Press, pp. 24–31.

[2] 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.

[3] 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.

[4] Oviatt, S., Arthur, A., and Cohen, J., 2006. “Quiet interfaces that help students think”. In UIST ’06: Proceedings of the 19th annual ACM symposium on User interface software and technology, ACM, pp. 191–200.

[5] 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.

[6] 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.

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

[8] 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: CHI ’01 extended abstracts on Human factors in computing systems, ACM, pp. 205–206.

[9] 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.

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

[11] Kim, D.H, Kim, M.-J., 2006. “A curvature estimation for pen input segmentation in sketch-based modeling”. Computer-Aided Design, 38(3), March, pp. 238–248.

[12] Wolin, A, Eoff, B.D., Hammond. T.A, 2008. “Shortstraw: A simple and effective corner finder for polylines”. In Proceedings of Eurographics 2008 - Sketch-Based Interfaces and Modeling (SBIM).

[13] 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.

[14] 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.

[15] 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.

[16] Qin, S. F., Wright, D. K., and Jordanov, I. N., 2001. “On-line segmentation of freehand sketches by knowledge-based nonlinear thresholding operations”. Pattern Recognition, 34(10), pp. 1885–1893.

[17] Wenyin, L., 2003. “On-line graphics recognition: State-of-the-art”. In GREC, pp. 291–304.

[18] Fonseca, M. J., and Jorge, J. A., 2001. “Experimental evaluation of an on-line scribble recognizer”. Pattern Recogn. Lett., 22(12), pp. 1311–1319.

[19] 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.

[20] Veselova, O., and Davis, R., 2006. “Perceptually based learning of shape descriptions for sketch recognition”. In SIGGRAPH ’06: ACM SIGGRAPH 2006 Courses, ACM, p. 28.

[21] 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.

[22] Eggli,L., ChingYao,H., Bruderlin,B., and Elber, G., February 1997. “Inferring 3d models from freehand sketches and constraints”.Computer-Aided Design, 29, pp. 101–112(12).

[23] 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.

[24] 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.

[25] Chen, Q., Grundy, J., and Hosking, J., 2003. “An e-whiteboard application to support early design-stage sketching of uml diagrams”. In HCC ’03: Proceedings of the 2003 IEEE Symposium on Human Centric Computing Languages and Environments, IEEE Computer Society, pp. 219–226.

[26] 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.

[27] 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.

[28] Landay, J. A., andMyers, 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 Press/Addison-Wesley Publishing Co., pp. 43–50.

[29] 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.

[30] 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.

[31] Peschel, J., and Hammond, T., 2008. “Strat: A sketched-truss recognition and analysis tool”. In Proceedings of the Fourteenth International Conference on Distributed Multimedia Systems.

[32] 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.

[33] 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 SBIM ’07: Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling, ACM, pp. 75–82.

[34] Zeleznik, R. C., Bragdon, A., Liu, C.-C., and Forsberg, A.,2008. “Lineogrammer: creating diagrams by drawing”. In UIST ’08: Proceedings of the 21st annual ACMsymposium on User interface software and technology, ACM, pp. 161–170.

[35] Ledas Ltd, http://www.ledas.com/products/lgs2d/, accessed on feb 25th 2009.

[36] Sezgin, T. M., and Davis, R., 2006. “Scale-space based feature point detection for digital ink”. In SIGGRAPH ’06: ACM SIGGRAPH 2006 Courses, ACM, p. 29.

[37] Chen, K.-Z., Zhang, X.-W., Ou, Z.-Y., Feng, X.-A., January 2003.“Recognition of digital curves scanned from paper drawings using genetic algorithms”. Pattern Recognition, 36, pp. 123–130(8).