

Development of educational software for beam loading analysis using pen-based user interfaces

Yong S. Suh

Department of Mechanical Engineering, California State University, Sacramento, 6000 J Street, Sacramento, CA, U.S.A.

(Manuscript Received September 6, 2013; Revised November 11, 2013; Accepted November 11, 2013)

Abstract

Most engineering software tools use typical menu-based user interfaces, and they may not be suitable for learning tools because the solution processes are hidden and students can only see the results. An educational tool for simple beam analyses is developed using a pen-based user interface with a computer so students can write and sketch by hand. The geometry of beam sections is sketched, and a shape matching technique is used to recognize the sketch. Various beam loads are added by sketching gestures or writing singularity functions. Students sketch the distributions of the loadings by sketching the graphs, and they are automatically checked and the system provides aids in grading the graphs. Students receive interactive graphical feedback for better learning experiences while they are working on solving the problems.

Keywords: Beam loading analysis; Pen-based interface; Education software; Shape matching

1. Introduction

Shafts and beams are commonly used in mechanical components, building structures, and bridges. Many mechanical and civil engineering courses include the topic of analyzing the stresses and deflections of simple beams and shafts under various lateral loadings. [1-3] A simple beam is a straight beam with a constant section, and the analyses are relatively easy to solve by hand without complex digital computer simulations. An example of a simple beam-loading problem is shown in Figure 1. Students need to calculate the shear force, bending moments, and deflections along the beam, and identify the location and magnitudes of their maximum values.

Although the problems can be solved analytically on paper, the computational steps are usually tedious and prone to error. There are many commercial [4-6] and noncommercial [7, 8] software programs available for the analyses. There are also many systems [9-17] designed to be used as educational tools for the beam and structural analyses. Most of them are usually designed to generate solutions quickly and automate the processes for ease of use. Although effective in obtaining solutions, they may not be the best educational tools for learning the concept. Most of all, the user interfaces of the

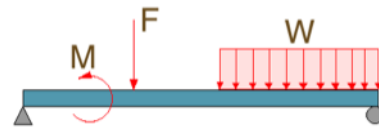


Figure 1. A statically determinate beam under loads.

current educational software may not be ideal for educational purposes. They use the WIMP (Windows, Icons, Menus, and Pointers) based graphical user interfaces (GUI), in which users choose the loading conditions and parameters from the menus or icons or dialog boxes. This user interface deviates from the traditional learning mode that is using a pen on paper. Recent studies [18, 19] show that the students will learn better in their learning when the user interface is closer to the familiar work practice. According to Oviatt et al. [19], using a graphical user interface (GUI) increases students' cognitive load, causing distractions to learning. They conclude that learning performance is best with interfaces similar to the existing work practice of using pen-and-paper. In the traditional method, students learn by taking their time in drawing beams, marking loading symbols, writing and solving equations by hand. The magnitudes of loadings, shear forces, moments, and deflections along the beam are also drawn graphically by hand, based on the hand calculations. The

^{*}Corresponding author. Tel.: +1 916-278-6162 Fax.: +1 916-278-7713

E-mail address: ysuh@csus.edu

© 2014 Society of CAD/CAM Engineers & Techno-Press

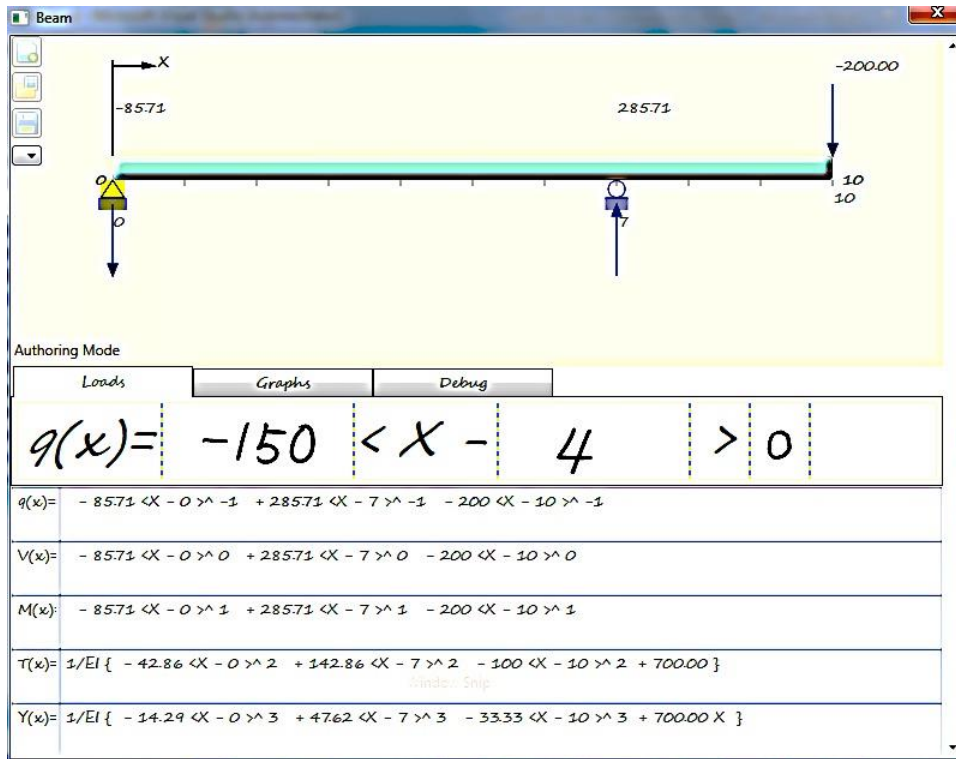


Figure 2. UI for writing singularity functions.

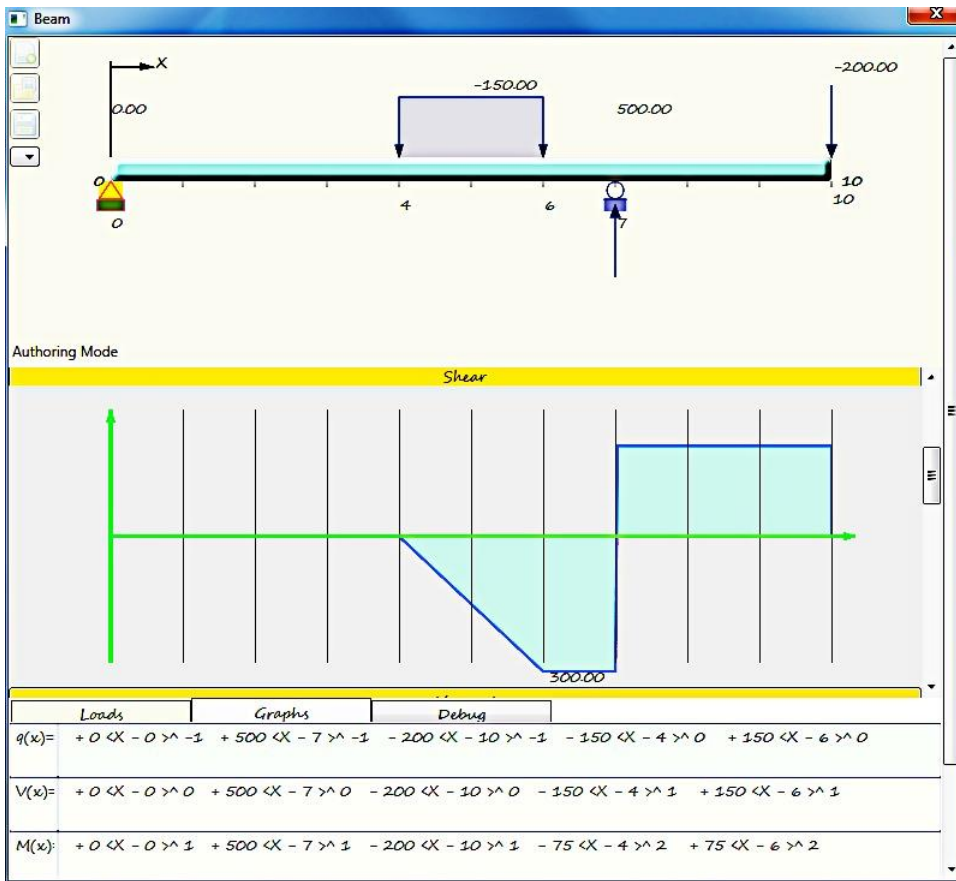


Figure 3. UI for sketching graphs.

WIMP-based user interface is limited in bringing such manual interfaces to the students.

On the other hand, computer-based educational tools have advantages over the traditional pen-and-paper method in that they can give interactive feedback to the students, automate grading for instructors, and provide an easy way of archiving students' works and progresses.

A pen-based computer user interface can be a good alternative tool by taking the advantages of both the approaches. In this paper, the design and implementation of an educational software tool for the beam loading analysis using a pen-based interface is described. Although there are many alternate devices with the pen strokes interfaces available, the system described in this paper is implemented on a tablet PC.

2. Beam analysis using singularity function

Although there are a few different approaches, the beam problems can be systematically (and programmatically) solved by using the *singularity functions* [1]. A singularity function, $f_n(x)$, is expressed and defined as in Equation (1). Where $n > -3$ is an integer, and x is the distance along the beam and a is the value of x where the function begins. The Dirac delta, δ , is a unit impulse function. A mathematical model of a load to a beam can be represented by a combination of one or more singularity functions. The singularity function is chosen because it is not only easy to write and solve by hand, but also easy to convert to computer algorithms.

$$f_n(x) = \langle x - a \rangle^n = \begin{cases} \delta(x - a) & n < 0 \\ 0 & n \geq 0, x < a \\ (x - a)^n & n \geq 0, x \geq a \end{cases}$$

$$\int_{-\infty}^x \langle x - a \rangle^n dx = \begin{cases} \langle x - a \rangle^{-1} & n = -2 \\ \langle x - a \rangle^0 & n = -1 \\ \langle x - a \rangle^{n+1} & n \geq 0 \end{cases} \frac{1}{n+1} \quad (1)$$

To add a certain load to a beam, students are required to write the singularity functions of the load in question. An alternative way is to draw a symbolic sketch of the load to the beam, and the symbol is recognized and converted to the corresponding one or more singularity functions. The system performs all computations based on the singularity functions internally. Detailed descriptions on solving the beam problem using the singularity functions can be found in [1-3].

3. User interface design

Whenever it is applicable and appropriate, pen-based user interfaces such as stroke-gestures, sketching, and handwriting are used in this system. Figure 2 shows a screen shot of the system in defining the loads on a beam. The upper half of the window, the *beam canvas* area, displays a beam. Users can add loads to the beam by sketching the corresponding symbols

above the beam. The system recognizes the strokes as gesture inputs, and the matching load symbol is recognized. A corresponding singularity function of the recognized symbol is automatically generated and added to the function list below. For compound loads, multiple singularity functions are generated.

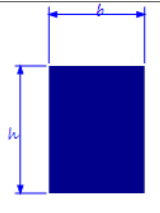

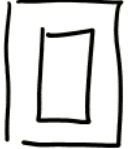
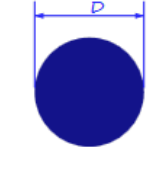
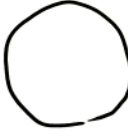
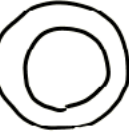
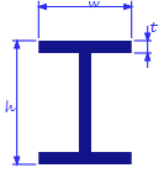
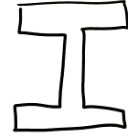

Conversely, users can also write singularity functions on the function input panel in the middle below the beam panel. Because all the singularity functions share a fixed format, users only write parameters (numbers) for better accuracy of recognizing the handwritings. As a user makes a checkmark gesture after finishing writing the parameters, the system recognizes the singularity function. At the same time, the load symbol for the recognized loading is also automatically drawn on the beam canvas as shown in Figure 2.

Once all the loads are added to the beam, students are required to sketch graphs for the various states of the beam due to the loads. Figure 3 displays the system in the graph-drawing mode. If a user selects the Graphs tab in the middle of the window, a graph canvas replaces the function input panel so students can sketch graphs. Grids are displayed for guidelines, and students can draw loading, shear, moment, slope, and deflection of the beam. Figure 3 displays the distribution of the shear force along a beam under a set of loads generated by the system. In the learning mode, however, the graph is not displayed and students are required to sketch the curves.

4. Setting beam properties

The Young's modulus of the beam material is required to calculate the deflections of a beam under loads. The moment

Table 1 Beam section templates.

	Parameters	Sketched sections	
Rectangle			
Circle			
I-Beam			

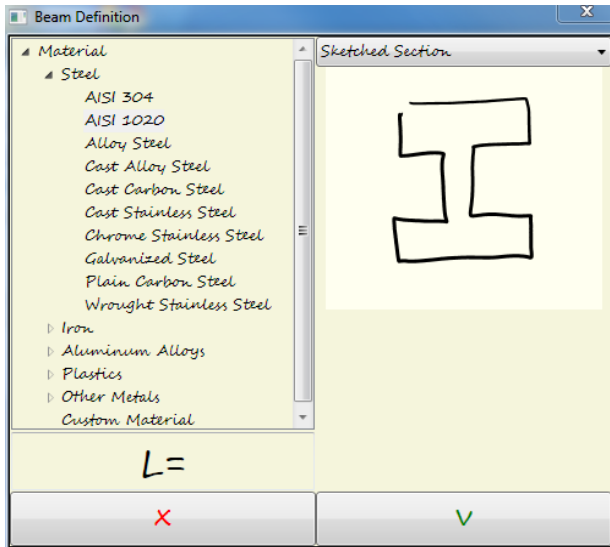


Figure 4. Setting beam properties.

of inertia of the section geometry is also required. In the following subsections, pen-based user interfaces for assigning the beam properties are described.

4.1 Setting material properties

Users can specify the material by selecting one from a list of pre-defined materials in the beam definition dialog window as shown in Figure 4. Custom materials not on the list can be added by writing the value of the Young's modulus.

4.2 Beam section geometry

The section geometry of a beam is sketched by the user on the right-hand side of beam definition dialog as shown in Figure 4. The system recognizes the sketch and finds the correct beam section in the beam section database. Currently, only the following set of pre-defined section geometry has been implemented for a concept proof as listed in Table 1.

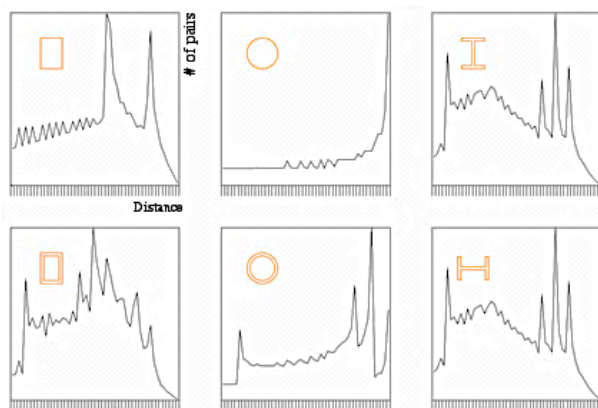


Figure 5. Histograms of section templates.

Additional shapes can be added to the database without modifying the algorithm.

Once a user sketches a beam section and draws a checkmark gesture, the system searches for a matching geometry from a section database. The following section describes the algorithm in detail.

4.3 Shape matching algorithm

Among various shape matching algorithms available, the shape histogram method [20] is modified and used as a shape descriptor in this application. In the shape histogram method, a sufficient number of points on a shape are randomly chosen, and the Euclidean distances between an exhaustive combination of pairs of the points is recorded. The distances are then normalized, and similar distances are grouped and put into a predefined number of bins. In this application, the normalization is performed by simply dividing the distances by the longest distance. A histogram is then created by counting the number of point-pairs in each bin. Histograms of typical beam-section shapes are shown in Figure 5. In these histograms, 50 bins are arranged horizontally in the order of distances, and the corresponding numbers of point-pairs in the bins are plotted in the vertical axis. Notice the distinctive characteristic of the histogram of each shape, and the similarity of the histograms of the two I-beams even if they are in different orientations.

Histograms for various shapes are generated and stored as templates in a database. When a user sketches a shape, its histogram is created and compared with all the template histograms in the database. The shape is then recognized as the one that matches best.

There are many methods to measure the similarities among histograms [20-22], and we use the Minkowski L_2 Norm (Euclidean distance) method [20] in which a Probability Density Function, D , is computed for discrete sets of f and g

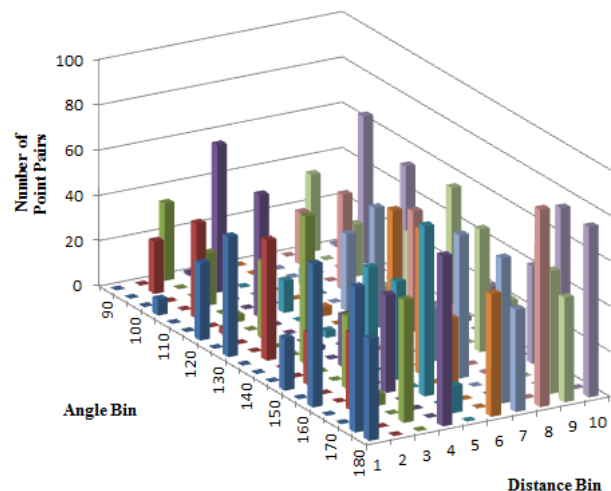


Figure 6. Distance-angle histogram.

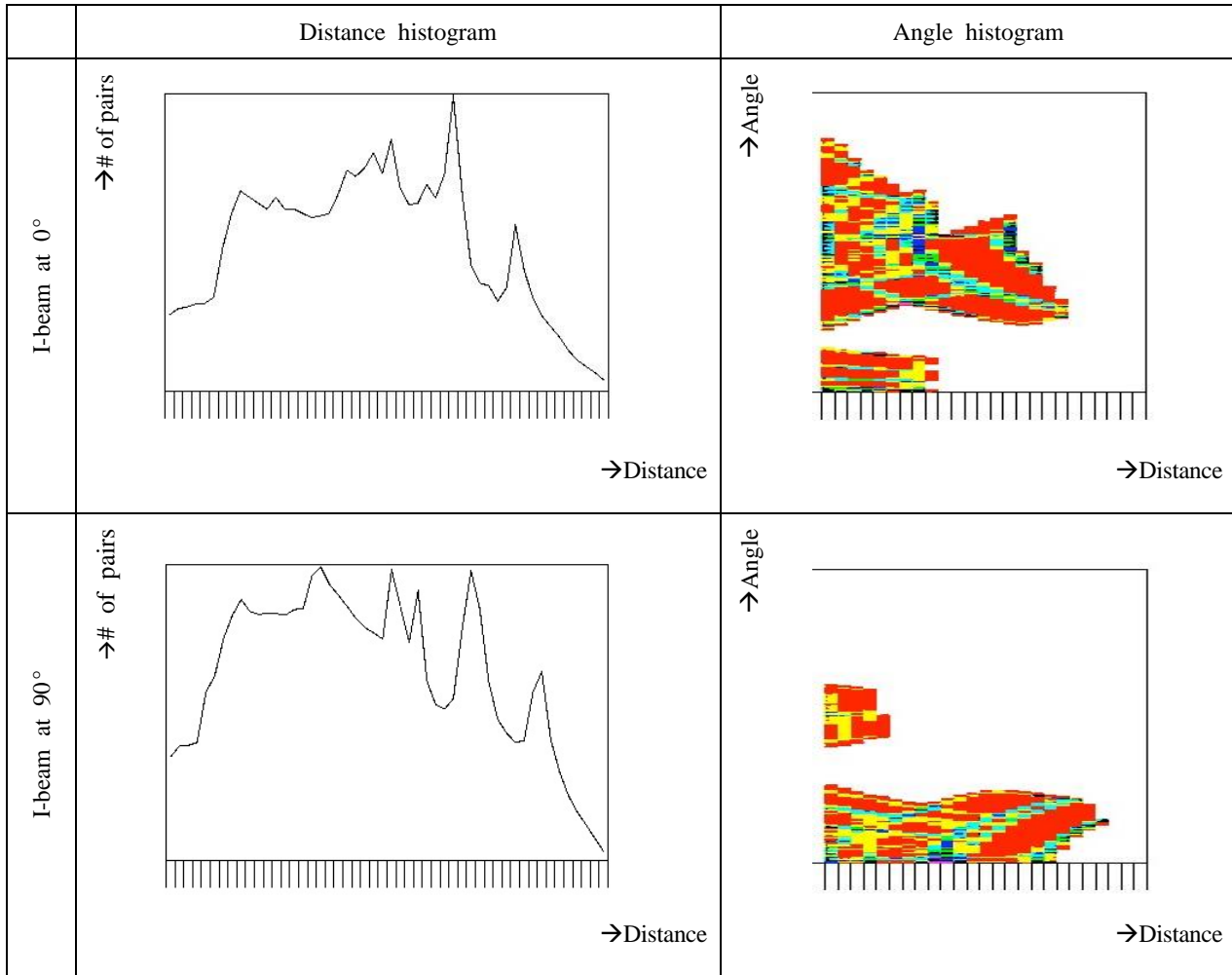


Figure 7. Histograms of I-beam in two orientations.

as follows:

$$D(f, g) = \sqrt{\sum_{i=1}^n (f_i - g_i)^2} \tag{2}$$

In general, shape descriptors are invariant to the orientations and sizes of the shapes such that similar shapes can be found regardless of their orientations and sizes. In this application to the beam section, however, the orientation is important because the value of the moment of inertia is dependent on the orientation. On the other hand, the matching algorithm must be tolerant to certain misalignments in the orientations as the sketch is drawn by hand. Therefore, we need to modify the shape descriptor that distinguishes *significant* differences but is invariant to *slight* differences in the orientation. It also needs to be invariant to any size differences.

To distinguish the orientations of similar shapes, a two-

dimensional histogram is introduced. In addition to the distance histogram described above, we also compute the angles of the point-pairs with respect to a reference line (e.g., horizontal X axis). Assuming that we can capture the orientation of a shape by checking the angles of the particular point-pairs that are far apart (long enough distances between the two points), we compute the angles only for the point-pairs whose distances are longer than a threshold distance to reduce the computation time. We choose the half of the diagonal distance of the bounding box of the shape as the threshold. For each distance bin in the histogram, angles of the point-pairs in the bin are computed and put into sub-bins. We use 5 degrees as the size of the sub-bins (second direction of the two dimensional array), so there may be at maximum 36 (=180/5) sub-bins in each distance bin. We use a Hash data structure to save storage space and speed storing and processing the 2-dimensional histogram. Figure 6 illustrates the two-dimensional histogram. Experiments show that the tool performs fast enough to be successfully used as an interac-

tively by users.

Figure 7 shows an example of the histograms of an I-beam that are in different orientations. In the angle histogram, the horizontal axis is the distance bins (larger than the threshold value), and the vertical axis represents the angle bins from 0° to 180° . The color code represents the number of point-pairs in each angle bin. Notice that the distance histograms are similar for both I-beams regardless of the orientation. But the angle histograms are different each other and distinguishable.

The similarity of two histograms is measured by a score that is computed by a linear combination of the scores of the distance histogram and the angle histograms as follows:

$$S = C_1 D_d + C_2 D_a \quad (3)$$

We set C_1 and C_2 to be 0.5, and D_d and D_a are computed using Equation (2).

Once a matching shape is recognized, the Beam Definition dialog shows the selected section shape and the users need to write the parameters specifying the exact size of the section as shown in Figure 8.

5. Beam supports

Currently, the system only allows determinate systems with up to two supports per beam. Users can add a fixed support (constraining the beam both horizontally and vertically) by sketching a triangle, and add a non-fixed support (constraining the beam vertically) by sketching a circle as shown in Figure 9. A cantilever beam has one support and a moment at one end of the beam.

As a support is recognized, users are required to write the location (X coordinate) of the support as shown in Figures 9

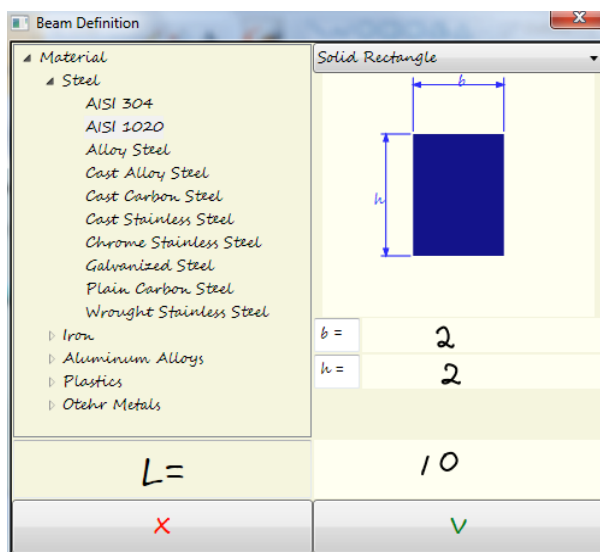


Figure 8. Entering parameters for section geometry.

Table 2 Loads.

Load	Symbol	Gesture	Singularity Fn
Moment			$m\langle x-a \rangle^{-2}$
Unit impulse			$m\langle x-a \rangle^{-1}$
Unit step			$m\langle x-a \rangle^0$
Ramp			$m\langle x-a \rangle^{-1}$

(a) and (b). The sketched gestures are replaced by the support symbols as shown in Figure 9(c). The reaction forces are automatically computed whenever the user adds or modifies the loads to the beam, which may or may not be displayed to the students. A support can be edited at any time by tapping on the symbol with a stylus pen.

6. Beam Loads

6.1 Types of loads

The loads can be added to a beam by sketching gestures on the beam canvas, or writing singularity functions corresponding to the loads. The types of the loads supported are summarized in Table 2.

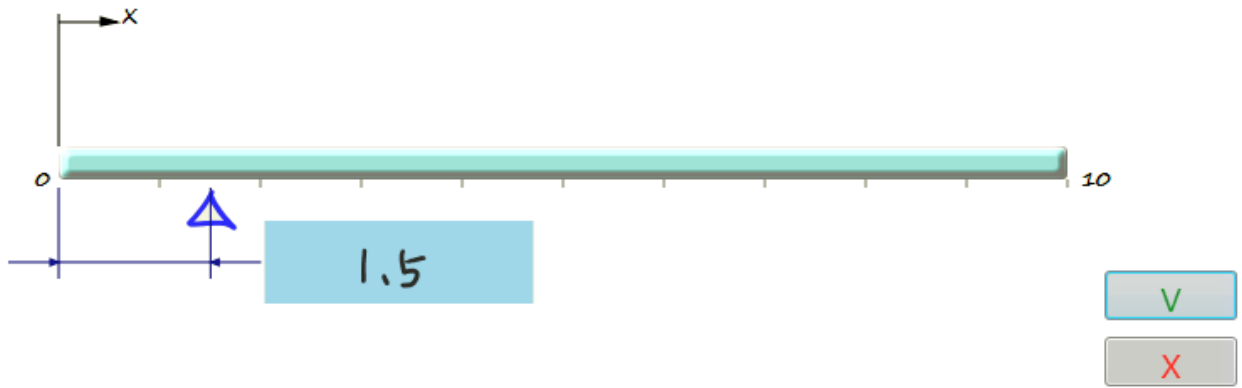
6.2 Adding loads by gesture input

The load symbols are sketched in the beam canvas area. Instead of using symbol recognition [23, 24], a gesture-based interface [25] is used. Table 2 illustrates the loads and their gestures. The gestures are recognized by the first one or two strokes of the actual symbols. As a user sketches a gesture, the system prompts the user to write load parameters as shown in Figure 10(a).

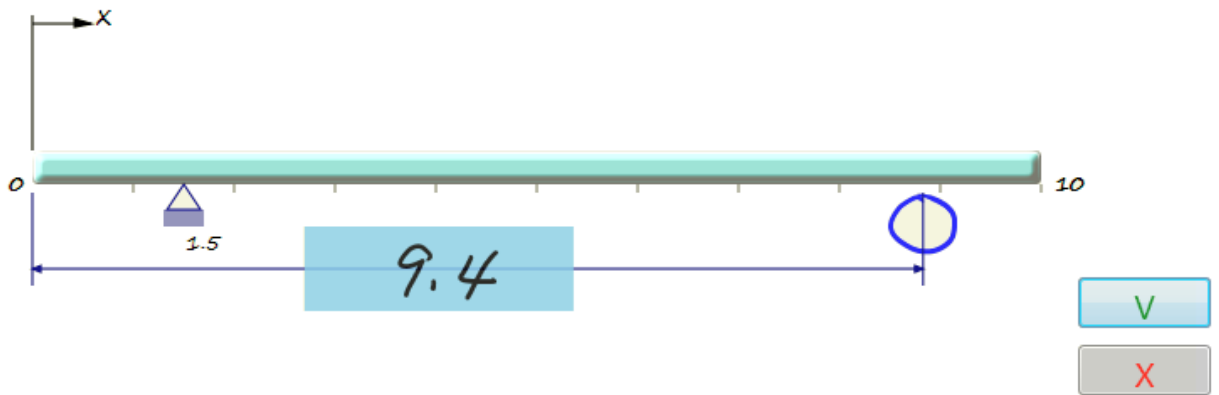
The user writes the values for the location and the magnitude of the load, and then the gesture strokes are replaced by the symbol of the recognized load at the correct location as shown in Figure 10(b). At the same time, corresponding singularity function(s) is (are) also created and displayed. Notice in that the reaction force is automatically recalculated after the load is added as illustrated in Figure 10.

6.3 Adding loads by writing singularity functions

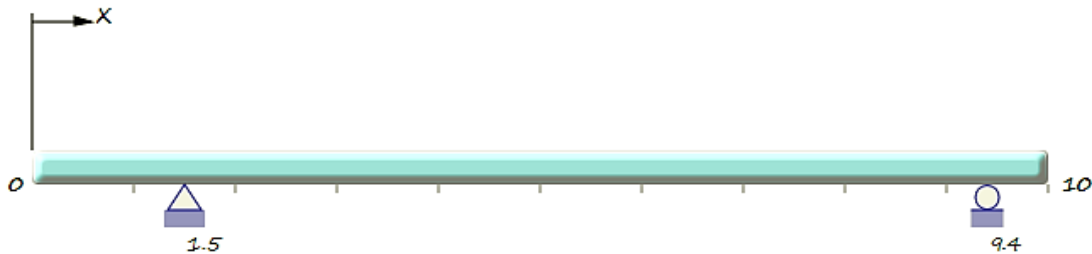
Alternatively, users can also write singularity functions to



(a)



(b)



(c)

Figure 9. Adding supports to a beam: (a) adding a fixed support, (b) adding a non-fixed support, (c) recognized supports.

add loads using the *load canvas* panel. For the robust recognition of equations, users will write each parameter of the singularity function in each compartment as shown in Figure 11.

After writing a singularity function, a user makes a checkmark gesture to let the system recognize the load. Then the corresponding symbol for the load is added to the beam graphics area. Compound loads can be represented by a combination of multiple singularity functions.

7. Sketching loading graphs

Each time a new load is added, the reaction forces at the supports are updated, and the distributions of the shear forces, moments, and deflections along the beam are recalculated. The graphics diagrams of the loads can be displayed with the maximum values. Figure 3 shows a shear force diagram displayed for the beam with the loads.

The diagrams will not be displayed in the learning mode, however. Instead, students must sketch the diagram by hand,

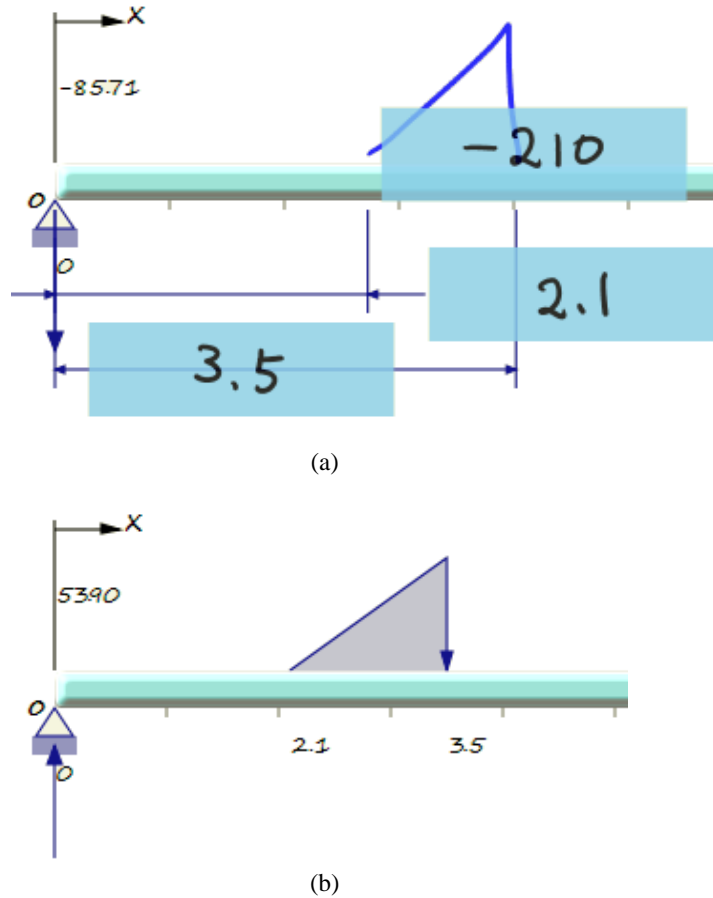


Figure 10. Adding a ramp load by gesture input.

$$q(x) = -210 < x - 2.3 > -1$$

Figure 11. Writing a singularity function.

and the correctness of the sketched diagrams is checked and graded automatically.

7.1 Grading the graphs

Students' sketched diagrams cannot be directly compared with the computer-generated diagrams as the vertical scale is arbitrary and the sketch quality may not be compatible. Hence, the free-hand sketches must be checked qualitatively not quantitatively. Let the X-axis of a coordinate system be along the beam and the Y axis be vertical as illustrated in Figure 3. The hand-drawn diagrams are checked for their correctness based on the following criteria:

- Correct local and global maxima values in Y-axis. Students write the global maximum value on the diagram canvas.

- Correct relative Y values at the junction points where the loading conditions change.
 - Curve types – linear, monotonic, convex, or concave.
- The analyses of the hand-drawn graphs are performed by the following steps.

1. Find the corner points of the strokes.
 2. Split the strokes at the corner points. Divide the graph into multiple spans at the characteristic points including the corner points. Find the Y values at the junctions.
 3. For each span, classify the geometry of the curve. Find the maxima points in each span.
 4. Find the global maximum (and minimum) Y value and its X location is checked.
 5. Evaluate/grade the graphs based on the findings.
- The following subsections describe the steps in detail.

7.2 Splitting graphs at characteristic points

In the pen-based tablet interface, the computer takes the movement of a stylus pen into a stroke structure that is composed of a series of ordered points. A continuous movement without lifting the pen creates one stroke. Hence, a stroke may contain multiple curves and lines as users draw them

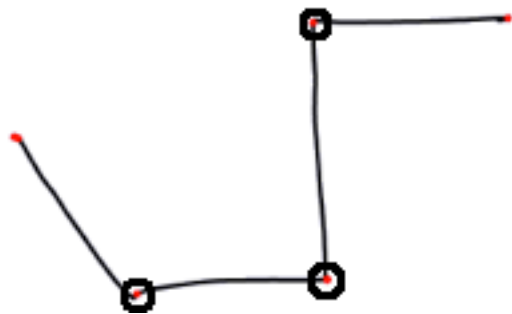


Figure 12. Segmentation of a graph diagram.

without lifting the pen. The stroke needs to be broken down into multiple segments so that each segment can be represented by a continuous polynomial equation (thus, singularity function). This can be done by finding the corner points in the stroke, and splitting the stroke at the points.

Unfortunately, the corner detection problem of freehand drawing is not trivial, and many different segmentation algorithms [26-30] have been proposed. The sliding rectangle method [31] is modified and adopted in the system. Before applying the corner detection algorithm, the input strokes must be re-sampled so that the points in the strokes are uniformly distributed. Figure 12 shows an example of corner points identified from a hand-drawn graph.

Once the corner points are detected and the strokes are split into multiple segments, the following analyses are performed for each segment.

7.3 Identifying curve geometry

For each segment span of the hand-drawn graph, the intended geometry is recognized and classified into one of the following categories.

A stroke is identified as a line by fitting the points to a line with the least squares method. If the error is smaller than a threshold value, the segment is identified as a straight line. If a segment is identified as a non-linear curve, check if the curve is monotonic using the following simple algorithm shown in Figure 13.

7.4 A rubric for grading graphs

After the sketched graph is analyzed, it is graded based on the followings:

1. Check the location of the overall maximum (or minimum) point of the graph. Students may write the magnitude of the maximum value on the graph canvas. Also, the X location of the maximum value along the beam needs to be checked.
2. Check the spans of the segments. Find and check the locations of the characteristic points such as corner points and intercepts. The relative Y values are checked by ordering the points along to the Y-axis. Check if the orders are all correct.

Table 3. Geometry classification.

Line	Increasing	
	Decreasing	
	Horizontal	
	Vertical	
Non-linear curve	Monotonous	Increasing
		Decreasing
	Non-monotonous	

```

IncreasingSet = 0
DecreasingSet = 0
Point pt0 = strokePoints[0];
For i=1 to i< number of points
    Point pt1 = strokePoints[i]
    If ( pt0.Y < pt1.Y )
        IncreasingSet ++
    Else
        DecreasingSet++
End
Threshold = 95% of all points
If (IncreasingSet > Threshold)
    Return "monotonically increasing"
Else if (DecreasingSet > Threshold)
    Return "monotonically decreasing"
Else
    Return "Non-monotonous"
End If
    
```

Figure 13. Checking monotonicity of a curved stroke.

3. Check the geometric type of the segments according to the classifications in Table 3.

8. Implementation

The software is implemented on a tablet PC running Microsoft Windows operating system. The Windows Presentation Foundation (WPF) is used for the graphics and user interfaces with the Microsoft Visual Studio using C# language. For the handwriting and gesture recognitions, Microsoft WPF InkAnalysis module is used. Since the success rates of the recognizers will generally follow those of Microsoft's, any statistical analysis of success rates is omitted in this work.

9. Conclusions

A design and implementation of an educational software tool for teaching beam analysis is presented. Based on recent studies, students learn effectively when the user interface is similar to the familiar work practice. Except the basic universal interfaces such as opening and saving files, most user interactions are performed using pen-based interfaces that are similar to solving problems on paper with a pen, a method

familiar to students. The pen-based interfaces let students take time writing and solving equations and drawing graphs. The computer-based learning tool also enables students to receive immediate feedback on their mistakes and automatic grading. Various known techniques of the pen-based interfaces such as corner detection, gesture and handwriting recognitions are used in the program. To realize the pen-based interface in the beam loading analysis, a shape-matching algorithm is modified to recognize sketched beam section geometry. This paper is focused on the design and implementation of the software, and the user studies and the effectiveness of the pedagogical approach are left for further studies.

References

- [1] Norton RL. Machine design: an integrated approach. 3rd ed. New Jersey: Prentice Hall; 2006. 984 p.
- [2] Shigley J, Mischke C, Budynas R. Mechanical engineering design. 7th ed. New York: McGraw-Hill; 2003. 1056 p.
- [3] Huston, R.L. Practical stress analysis in engineering design, 3rd edition. CRC Press; 2012. 664 p.
- [4] Company P, Piquer A, Contero M. On the evolution of geometrical reconstruction as a core technology to sketch-based modeling. In: Eurographics Workshop on Sketch-Based Interfaces and Modeling; 2004 August 30-31; Grenoble, France; p.97-106.
- [5] Alvarado C. Sketch recognition user interfaces: guidelines for design and development. In: Proceedings of AAAI Fall Symposium on Pen-Based Interfaces; 2004 October 21-24; Washington D.C.; p. 8-14.
- [6] Varley PAC, Suzuki H. Interpreting line drawings of objects with k-vertices. In: Geometric Modeling and Processing; 2004 April 13-15; Beijing, China; p. 249-258.
- [7] Devenport W, Kapania R, Rojiani K, Singh K [Internet]. Beam view - Java applets for engineering education. Available from: <http://www.engapplets.vt.edu/statics/BeamView/BeamView.htm>.
- [8] Getze R [Internet]. Beamboy. Available from: <http://www.geocities.com/richgetze/>.
- [9] Dempsey KM, Kane JJ, Kurtz JP. Beamtool: Interactive beam analysis for today's student and tomorrow's engineer. Computer Applications in Engineering Education. 2005; 13(4): 293-305.
- [10] Devenport W, Kapania R, Rojiani K, Singh K [Internet]. EngAPPLETS. Virginia Tech. Available from: <http://www.engapplets.vt.edu>
- [11] Alvarez-Caldas C, Garcia JLSR, B. Abella BM, Gonzalez AQ. Educational software to design shafts and analyze them by FEM. Computer Applications in Engineering Education. 2007; 15(1): 99-106.
- [12] Barretto SFA, Piazzalunga R, Ribeiro VG. A web-based 2d structural analysis educational software. Computer Applications in Engineering Education. 2003; 11(2): 83-92.
- [13] Hillsman VS, Tomovic MM. Beam deflections via computer algebra systems. In: Proceedings of the Frontiers in Education Conference; 1995 Nov 1-4; Atlanta, GA; p. 4b4.1-4b4.4.
- [14] Maiorana C, Sgarbossa L, Salomon V. New methodologies in teaching e-structural mechanics using www. Computer Applications in Engineering Education. 2008; 16(8): 189-210.
- [15] T. W. C. Hu [Internet]. Enhanced student learning in engineering courses with cas technology. Available from: <http://celt.ust.hk/tlsymp07/pdf/R13-ThomasHu.pdf>
- [16] JEI-Rimawi J, El-Hamalawi A. Dynamic strategy for teaching structural analysis. Computer Applications in Engineering Education. 2002; 10 (4): 194-203.
- [17] Murugappan S, Ramani K. Feasy: a sketch-based interface integrating structural analysis in early design. In: ASME 2009 International Design Engineering Technical Conferences & Computers in Information in Engineering Conference, IDETC/CIE; 2009; San Diego, CA.
- [18] Da Silva R, Bischel DT, Lee W, Peterson EJ, Calfree RC, Stahovich TF. Kirchhoff's pen: a pen-based circuit analysis tutor. In: EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling. 2007; Riverside, CA; p.75-82.
- [19] Oviatt S, Arthur A, Cohen J. Quiet interfaces that help student think. In: The 19th annual ACM Symposium on User Interface Software and Technology; 2006; Montreux, Switzerland: p. 191-200.
- [20] Osada R, Funkhouser T, Chazelle B, Dobkin D. Shape distributions. ACM Transactions on Graphics (TOG); 2002; 21(4): 807-832.
- [21] Zhang T, Peng Q. Shape distribution-based 3d shape retrieval methods: review and evaluation. Computer-Aided Design and Applications. 2009; 6(5): 721-735.
- [22] Lp CY, Lapadat D, Sieger L, Regli WC. Using shape distributions to compare solid models. In: ACM Symposium on Solid and Physical Modeling; 2002; Saarbrucken, Germany; p. 273-280.
- [23] Kara LB, Stahovich TF. An image-based, trainable symbol recognizer for hand-drawn sketches. Computers & Graphics. 2005; 29(4): 501-517.
- [24] Kara LB, Stahovich TF. Hierarchical parsing and recognition of hand-sketched diagrams. In: 17th ACM User Interface Software Technology (UIST); 2004; Santa Fe, NM; p.13-22.
- [25] Landay JA, Rowe LA, Rowe JA, Michiels J. Visual similarity of pen gestures. In: SIGCHI Conference on Human Factors in Computing Systems; 2000; The Hague, The Netherlands; p. 360-367.
- [26] Kim DH, Kim MJ. A curvature estimation for pen input segmentation in sketch-based modeling. Computer Aided Design. 2006; 38(3): 238-248.
- [27] Stahovich TF. Segmentation of pen strokes using pen speed. In: AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural; 2004; Arlington, VA; p. 174.
- [28] Qin SF, Wright DK, Jordanov IN. On-line segmentation of freehand sketches by knowledge-based nonlinear thresholding operations. Pattern Recognition. 2001; 34(10): 1885-1893.
- [29] Sezgin TM. Feature point detection and curve approximation for early processing of free-hand sketches. Department of

Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge; 2001.

- [30] Wolin A, Eoff B, Hammond T. Shortstraw: a simple and effective corner finder for polylines. In: Eurographics 2008-Sketch-Based Interfaces and Modeling (SBIM); 2008; Annecy, France; p. 33-40.
- [31] Masood A, Sarfraz M. Corner detection by sliding rectangles along planar curves. *Computers & Graphics*. 2007; 31: 440-448.