Applying Mathematical Sketching to Sketch-Based Physics Tutoring Software

Salman Cheema and Joseph J. LaViola Jr.

University of Central Florida
School of EECS
Orlando, FL 32816
salmanc,jjl@cs.ucf.edu

Abstract. We present a prototype sketch-based physics tutoring system that combines mathematical sketching, an interaction paradigm that supports construction of dynamic illustrations using the association of handwritten mathematical expressions with drawings to govern animation behavior, and a custom physics engine. We highlight key features of our core system that focus on correcting approximately drawn sketches and maintaining the correspondence between imprecise handwritten drawings and precise mathematical specifications. We describe the behavior and design of the system in detail and finally, present two example scenarios illustrating its possible uses in an educational setting.

1 Introduction

Mathematics and physics teachers often use diagrams to present scientific concepts in visual form and as an initial step in problem solving [12, 13]. In addition, to solve a physics or mathematics problem, students usually condense the information given in the problem statement by drawing a diagram. Students annotate these initial diagrams with mathematical expressions to make links between abstract concepts and concrete diagram elements. Often, the answer to a problem is numeric or symbolic and it is unclear how it would affect the drawing, requiring students to rely on their imagination to visualize the underlying concepts in action. To alleviate this issue, we postulate that providing meaningful animation of student-drawn sketches based on the associated mathematics used to solve a problem is required in order to impart better understanding of physics concepts. Such animation can also help students in intuitive verification of their answers to problems. Our research goal is to construct sketch-based intelligent tutoring systems for mathematics and physics that capture the essence of pen and paper diagrams while leveraging the power of computation by providing meaningful animation to enhance student learning.

We have developed a prototype physics tutoring system that is based on mathematical sketching [7], an interaction paradigm that supports construction of dynamic illustrations using the association of mathematical expressions with drawings to govern animation behavior. We took initial steps toward this goal with a proof-of-concept system in [3]. Our current prototype more fully integrates

the advantages of mathematical sketching with an underlying physics engine and leverages domain knowledge more efficiently to help it infer how to animate a sketch at different levels of mathematical specification.

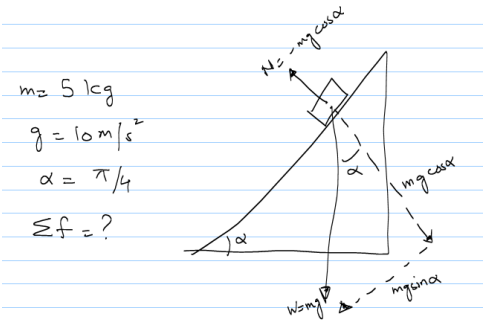

Fig. 1. A typical inclined plane diagram drawn by a student.

Hand drawn sketches are often approximate in nature. For example, Figure 1 represents an inclined plane problem. The triangle drawn by the user is approximately right-angled. Likewise, the inscribed angle α is not exactly $\pi/4$. Such approximations are acceptable with pen and paper diagrams because the user relies on his imagination to see concepts in action. However, these inaccuracies can cause problems for the underlying physics engine in [3] due to ambiguity between the precise mathematical specifications and the imprecise drawings. A sketch-based tutoring system must be able to correct such diagrams (i.e., perform rectification) to allow students to sketch diagrams in a natural manner. In this paper, we describe the drawing rectification features that enable our tutoring system to deal with approximate sketches. We present a discussion of its feedback mechanisms that allow users to more directly observe changes in diagram state over time. We also highlight scenarios where understanding the user's intent based on physics domain knowledge allows us to present a better dynamic illustration.

2 Related Work

Pen-based systems for understanding hand-drawn sketches and providing visual feedback have been developed in the past for a number of specific goals. Alvarado [1] and Kara [5] have constructed systems for recognizing and animating diagrams in specific domains such as mechanical design and vibratory systems. Both utilize a simulation back-end to facilitate animation of problems understood in terms of basic primitive shapes. However, they are limited in scope because they do not allow the user to write mathematics to govern animation behavior. MathPad^2 [8] provides a better mechanism for animating sketches by allowing

users to write down mathematics to describe aspects of animation. MathPad^2 is limited in its scope because users must specify all aspects of animation.

Existing physics tutoring systems, such as the Andes Physics Tutoring System [11], provide step by step guidance in problem solving, but rely on WIMP interfaces and do not provide users with the ability to write down their solutions and sketches as if using pen and paper. Newton's Pen [9] is a pen-based tutoring system that aids student learning in a few selected classes of physics problems. It does not permit free-form sketching of diagrams and does not include the ability to make associations in order to perform visual linkages between mathematics and diagram. PenProof [4] is a pen-based geometry theorem proving system that is able to recognize and understand the correspondence between geometric constructs and proof steps. This system is able to leverage the correspondence between geometry and proof steps to provide visual feedback to the user. At the same time, its limitation is that users are forced to disambiguate between geometry and proof steps explicitly and must rely on the system to associate proof steps with the figure.

3 System Features

Users sketch diagrams and write mathematics by means of a stylus on a tablet computer. Recognized diagrams can be annotated by making associations with mathematical expressions. Expressions used to make associations can either denote initial conditions or mathematical equations governing behavior.

The system parses a sketch when instructed to do so. In the past, we experimented with real-time sketch recognition but discarded it because it did not allow flexibility in terms of making/changing associations. On-demand parsing of the user's sketch is more natural because in a pen and paper scenario, students first sketch the diagram and then annotate it with mathematical expressions. The system is capable of recognizing the following diagram components: convex shapes(Circles,Polygons), springs, and wires. Realistic values are assigned to each component's attributes upon recognition based on the component's spatial characteristics. For example, a shape's initial mass is assigned proportional to its enclosed area. This approach allows us to animate sketches realistically even when the user does not specify any initial values. Assignment of initial attributes in this manner also allows the user interface to be flexible because it does not necessitate the input of all initial conditions by the user. Users can alter attributes by writing mathematical expressions to reflect proper initial conditions and associating them with diagram components. A lasso gesture is used to select expression(s). The association is completed by tapping the desired component. Existing associations can be viewed by hovering the stylus over a recognized component.

Mathematical expressions can either be constant expressions (e.g. $f_1 = (0, 5)$) or equations (e.g. $v_x = k\sin\theta$). Equations can denote scalar quantities (e.g. magnitude of normal force due to an incline) or vector quantities. Vector quantities can be specified in terms of x and y component equations or as a single vector

equation. When x and y components are not specified, we use physics domain knowledge to determine if the variable in the left-hand side of the equation denotes a vector or a scalar. Equations may be written in terms of numbers (e.g. $f_1 = -0.5v_A$) or in terms of symbolic constants (e.g. $f_1 = -\mu v$ where $\mu = -0.5$). The system also uses domain knowledge to establish possible errors regarding associations and diagram components. This takes an understanding of physical properties of recognized diagram components. For example, it is illogical to associate an expression specifying a spring constant (e.g. $k = 0.8$) with a shape or to associate a tension with a spring.

Writing and associating mathematics allows users to apply initial values and modify behavior as needed for a given problem and also provides intuitive feedback. Recognition errors or mistakes in equations can cause incorrect animation that conflicts with the user's intuition. A reset mode is provided that lets users debug and correct existing associations. This allows users to experiment with different initial values and gain better insight into the working of underlying concepts. We also allow a user to directly observe an attribute on a movable shape with respect to time. Observable attributes are velocity, acceleration, net force, all forces, x-displacement, y-displacement and position. When a particular attribute is under observation, most other information on the shape's animation is replaced by the selected attribute in bold. The user can also choose to view a real-time graph of a several selected attribute at any time.

In keeping with our research goal to capture and enhance the essence of problem solving using pen and paper, the system has the capability to preserve existing associations along different system modes. If users want to alter part of an expression that is already associated with a diagram component, they do not have to make the association again. We believe that this approach minimizes unnecessary work on a user's part and lets him focus on the problem at hand rather than being encumbered by the user interface.

4 System Design

Figure 2 shows a high level view of the various components of the system. The following sections will highlight salient aspects of important components.

4.1 User Input

Sketches and written mathematics are acquired as digital ink strokes. A gesture recognition module recognizes three gestures: lasso, scribble-erase, and tap. The gesture set is limited thus to reduce interface complexity. The lasso gesture is used to select both ink and diagram components. Users can drag and reposition the selection on the screen. The scribble-erase gesture is used to erase ink and diagram components. If mathematical expression(s) are selected and a tap gesture is performed on a diagram component, an association is made. If no mathematics is selected, the tap gesture will cause a recognized shape to

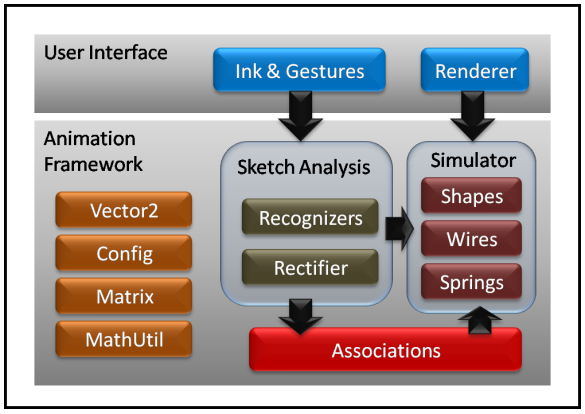

Fig. 2. Overview of system components.

become immobile. The interface includes options to save/load ink. Instant recognition feedback for mathematical expressions is provided by means of an online mathematics recognizer [14]. Mathematical recognition errors can be corrected by erasing the offending part of the expression by the scribble erase gesture and rewriting it. The feedback results are also used to make the association which improves performance by avoiding unnecessary work and minimizes recognition errors by using only correctly recognized mathematics.

4.2 Sketch Recognition

Sketch recognition is the first step in converting a user's sketch into components that will be animated by the underlying physics engine. All ink strokes are filtered and re-sampled prior to recognition to remove noise and ensure equal spacing of stroke points. We use a custom cusp detector to count the number of cusps in a stroke. A threshold is applied to the distance between the last and first cusp of the stroke to check for closure. A closed stroke is classified as a polygon if it has more than 2 cusps, otherwise it is possibly a circle. As a measure of circularity, we compute the standard deviation of the angle subtended at the stroke's centroid by each line segment in the stroke. Since the points are evenly spaced, the deviation will be extremely low for circular shapes. Strokes that have 2 cusps and are not closed are possibly springs or wires. A stroke is a spring if it has three or more self intersections. A line segment intersection test is employed for counting self intersections in the ink stroke. Wires are simply approximate straight lines. Any ink strokes that do not meet the above criteria are considered to be mathematics.

Every ink stroke is classified as either a diagram component or part of a mathematical expression. It is possible to mis-classify parts of a mathematical expression as diagram components (e.g. zeros as circles). We use the following rules to disambiguate between diagram components and mathematics. Shapes

are disambiguated by ensuring that all convex components enclose a minimum area. For springs and wires, the end point of each must be attached to a shape. Hence recognition proceeds in the following order: Shapes, Springs, Wires, and lastly mathematics.

4.3 Associations

Associations between mathematics and diagram components are used to modify aspects of the animation. Associated mathematical expressions are either constants or equations. Constants can modify attributes of a diagram component such as mass, velocity, acceleration, etc. Constants are applied once, at the start of the animation. The values of associated constants can be changed in reset mode.

Equations can be similarly used to modify attributes of diagram components. During each animation step, existing equations are populated with their parameter values and evaluated to guide the animation. When evaluating equations with mathematical errors (e.g. $f = -\mu v_A$ is valid only if μ is specified), any unrecognized/unspecified parameters are assigned zero. This ultimately serves as visual feedback indicating an error in input being the cause of abnormal animation. In [3], equations could either use values associated with a single component or use global values (e.g. gravity and time). We have extended this scheme to include symbolic constants associated with any component in the diagram. Such a mechanism is necessary to solve the inclined plane problem discussed in Section 6.1.

4.4 Physics Engine

Recognized diagrams are animated by a custom 2D physics engine that is based on principles described in [10]. The default animation behavior of all diagram components depends on the standard equations of motion. Each shape's position is updated by computing net acceleration and performing numerical integration twice for position. Collision detection/resolution are performed after the position update. Earlier we used the method proposed by [2] to deal with resting/sliding contacts. But upon experimentation, we found that keeping track of every shape's net motion via exponential smoothing defined by

$$\begin{aligned} Motion_{frame} &= \|v\|^2 + \omega^2 \\ Motion_{net} &= (1-\alpha)Motion_{net} + \alpha Motion_{frame} \end{aligned}$$

where $\alpha = 0.8$ works fairly well and is faster.

After collision processing, a inference step is applied to deal with unspecified circumstances such as whether wires will break or not. Important attributes of shapes (e.g. velocity) are rendered as arrows. To provide visual feedback regarding how the magnitudes and directions of important quantities change over time, the length of rendered arrows are adjusted in proportion to the magnitude of the attribute represented.

Attributes of recognized components can be altered by making associations with written mathematics. It is also possible for a user to specify position or velocity update equations for a diagram component as replacement for standard equations of motion. If only velocity is associated, it is integrated once to update position. If the position is associated, no integration is required. Observable attributes (see Section 3) are computed in every frame even if there must be computed indirectly (e.g. compute velocity when position equations are associated).

Interesting situations arise when default behavior is overridden by custom behavior. Consider the following example. Shape A moving under standard equations of motion collides with shape B moving under user-defined position update equations. It is clear, that after the collision is resolved, the user-defined position update equation for B no longer applies. When such objects collide, the user defined equations are disabled and the physics engine takes over to resolve the collision. After the collision, the collided objects' position is updated by the computation of net force and acceleration and double integration with respect to time.

5 Correction of Approximate Sketches

Correction of approximate sketches can take three forms. First, components may be corrected individually to conform to a user's intended sketch. Second, groups of shapes may need to be corrected. For example, the user may intend for one shape to rest on top of the other, but his actual sketch places them a small distance apart. During animation, the underlying physics engine will not treat the two shapes as if they were in contact. Lastly, making an association may change the appearance of a diagram component. Consider an inclined plane problem. The user draws a triangle and then writes an expression for the incline angle. When the association is made, the triangle should be adjusted to reflect the correct incline angle. This is necessary because the underlying physics engine will resolve contacts between shapes based on their spatial characteristics. For example, an incline with actual incline angle $\pi/4$ is associated with the following mathematical expression $\theta = \pi/6$. If a ball is placed on the incline, the user will calculate the parallel and perpendicular components of weight in terms of $\theta = \pi/6$. When the association is made and the animation run, the ball will fly off the incline instead of sliding down the incline. This happens because the direction of the net force will be slightly divergent from the incline instead of parallel to it.

5.1 Shape Correction

Sketched polygons can have approximately horizontal and vertical edges and may thus need individual correction. Each edge of a polygon is examined to see if its slope is nearly vertical or horizontal. Approximate vertical or horizontal edges of a polygon are thus corrected. In [3], users were required to draw polygon

vertices in anti-clockwise order. This constraint has now been lifted by examining the sign of the cross product between adjacent polygon edges. If any of the cross products is positive, then the ordering of vertices is reversed to ensure anti-clockwise winding. This allows users to sketch polygons in an unconstrained manner.

It is possible for the user to draw two or more shapes close to each other with the intent that the shapes be in physical contact. Upon recognition, the shapes may be a small distance apart. To correct these situations, the minimum distances between all pairs of shapes are computed. If the distance between two shapes is below a threshold value, then they are moved into contact. Between circles, movement occurs along the axis connecting the centers. If either shape in a pair is a polygon, then a vector corresponding to the minimum distance is computed. This is always perpendicular to exactly one edge of one of the polygons. This vector serves as the axis along which movement occurs. The direction of movement is from the shape with lower mass toward the more massive shape.

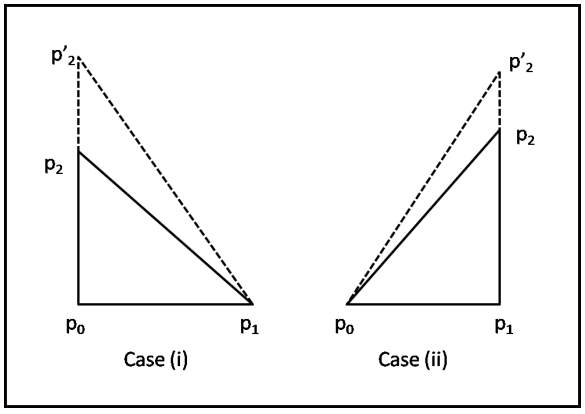

Fig. 3. Two cases for Triangle Rectification

Associating an angle with a right-angled triangle changes the incline of the triangle. Two possible cases are show in Figure 3. The only difference is whether the incline is uphill or downhill. In case (i), the new vertex p'_2 is computed as

1. Rotate p_0 clockwise by θ about p_1 to get p'_0
2. Compute direction $\boldsymbol{v} = \frac{p'_0 - p_1}{\|p'_0 - p_1\|}$
3. Compute $p'_2 = p_1 + k\boldsymbol{v}$, where $k = \frac{\|p_0 - p_1\|^2}{\boldsymbol{v} . \frac{p_0 - p_1}{\|p_0 - p_1\|}}$

In case (ii), the new vertex p'_2 is computed as

1. Rotate p_1 anticlockwise by θ about p_0 to get p'_1
2. Compute direction $\boldsymbol{v} = \frac{p'_0 - p_1}{\|p'_0 - p_1\|}$

3. Compute $p_2' = p_0 + k\boldsymbol{v}$, where $k = \frac{\|p_0 - p_1\|^2}{\boldsymbol{v} \cdot \frac{p_1 - p_0}{\|p_1 - p_0\|}}$

5.2 Wire and Spring Correction

Wire and spring correction can also take three forms. The first case relates to the length of the wire/spring. During sketching, users may accidentally draw them as ending within a shape rather than attached to its boundary. In order to capture the user's intent, the correction mechanism clips the endpoint of the wire/spring against the shape it is attached to. This ensures that forces due to wires/springs act at their proper locations and result in a correct animation. Secondly, if the distance between the endpoint of a wire/spring and a polygon vertex is below a threshold, the endpoint is aligned with the vertex to conform to the user's intent.

The last case relates to the angle that wires/springs may make with horizontal/vertical axes. If an association involves an angle, the system determines if the association is made near the start or the end of the wire/spring. An angle association completed near the end shape implies that the angle with the horizontal axis is being altered. An angle association completed near the starting point of the wire/spring indicates that the angle being altered is the angle with the vertical axis. We assume that wires/springs endpoints are always drawn in top-down ordering. Once the axis and the proper endpoint are determined, a rotation about the other endpoint is required to alter the wire/spring into the desired location in conformity with the user's intent.

6 Example Scenarios

6.1 Example 1: Inclined Plane

Figure 4 represents a problem that involves a ball rolling down an inclined plane. The information given to a student is the angle of the incline and the mass of the ball. The student is asked to work out the magnitude of the normal force acting on the ball. The student sketches a triangle to represent the inclined plane. He then draws a circle on the incline to represent the ball. From the given information he works out that the magnitude of the normal force acting on the ball is $f_1 = \|W\| \cos\theta$.

To verify the answer, the student decides to animate the sketch with the solution as input. Upon analysis, the incline and ball are replaced with corrected diagram components (triangle and circle). The triangle is made immobile by tapping it once. The student associates the expression for incline angle with the triangle. This causes the triangle to change appearance and the circle on it to be repositioned accordingly. The student associates the expressions for mass and the normal force (f_1) with the circle and animates the sketch.

If the computed answer is correct, the ball will have a resultant force that will cause it to roll down the incline. If the expression is wrong, then the ball will either bounce off the incline (normal force too low) or fly off (normal force

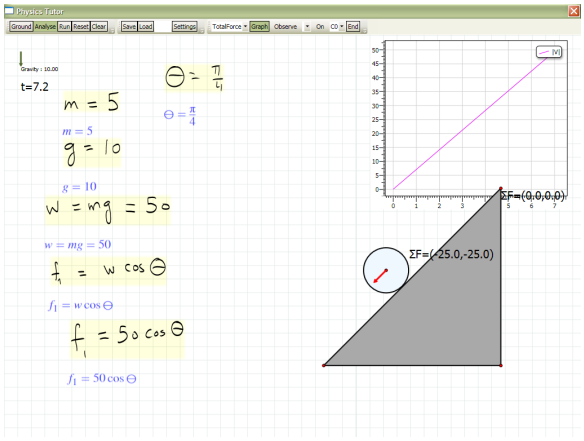

Fig. 4. An inclined plane problem.

too great). An incorrect animation therefore provides the user with hints about possible errors in his calculation. The student also chooses to view a graph of how the magnitude of the velocity changes with time for the ball. It show a linear increase in velocity with time, implying a constant acceleration. Notice that the student worked out the answer as he would on paper. The information associated with recognized diagram components is minimal and is a subset of the student's solution. The student did not have to specify the position update equations to govern the ball's motion down the incline. He simply worked out the magnitude of the normal force acting on the ball due to the inclined plane. Also, the expression derived by the student is just the magnitude of the normal force and by itself is insufficient to animate the diagram. The system infers from the shapes in contact and the association of the angle expression with the triangle that an inclined plane problem is being animated. The unit direction of the normal force is computed as perpendicular to the incline edge of the triangle and multiplied by the magnitude (associated by the user) to get the normal force. Note also that the expression for the normal force is written in terms of θ. Our previous implementation [3] did not allow writing of vector equations and would have required the specification of the normal force in terms of x and y coordinates as $f_x = W_x \cos\theta$ and $f_y = W_y \cos\theta$. The above equations are incorrect from a physics point of view and would only confuse a student should he have to use them to specify the animation's initial conditions. We have removed this cause of confusion in our current prototype.

6.2 Example 2: Wires Holding an object in Equilibrium

Figure 5 represents a problem where a box is being held in equilibrium by means of two wires attached at an angle to it. The student is given the mass of the box and the angle that each wire makes with the horizontal axis. He must calculate

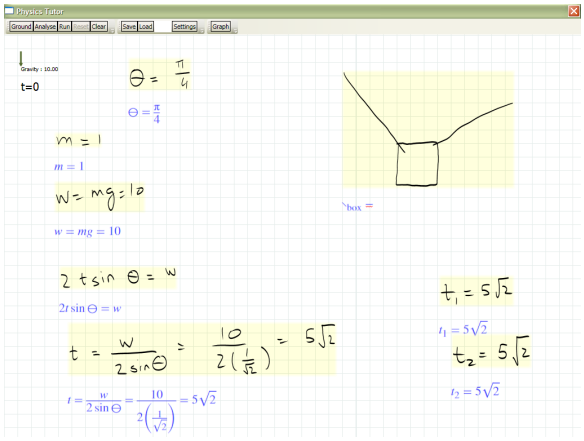

Fig. 5. An equilibrium problem modeled in our system.

the tension in each wire that will keep the box suspended in equilibrium. Upon reflection, the student realizes that the sum of the vertical components of tension in each wire must equal the weight of the box. With this insight, the student works out that the magnitude of the tension in each wire is $5\sqrt{2}$. As before, to verify the result, he instructs the system to analyze the sketch. The system recognizes and replaces the box with a square. The wires are clipped properly against the square's boundary. The student alters the mass of the square and associates the tension with each wire. Notice that the sketch is approximate and that the wires are not truly at an angle of $\pi/4$ with the horizontal. Upon making the association ($\theta = \pi/4$), the system alters each wire so that it is at an angle of $\pi/4$ with the horizontal.

When the animation is run, the system shows the box being held in a stationary position. If the student had computed the tension in any one of the wires incorrectly, the animation would have resulted in a net force on the box. The box's resulting motion would cause one or both wires to break, confirming to the student that his answer was incorrect.

7 Conclusion

We have presented a prototype sketch-based physics tutoring system that fuses mathematical sketching with an underlying physics engine. We have described key features of our prototype that allow it to deal with several cases of approximate user sketches through sketch rectification, enhance its capabilities in terms of accepting flexible input, provide visual feedback, and support associations between drawings and mathematics. Our current prototype is limited to simple linear kinematics and can only provide visual verification. In the future, we plan to include problems from other areas such as work, energy, rotational kinematics, simple harmonic motion, and planetary motion. We also plan to support

numerical verification of a user's answer to a problem. Other possible avenues of future work include implementation of a dimensionality analysis tool and a tool to convert between different systems of units. We plan to do a general study of the techniques and ways that university physics students employ to solve different types of physics problems. We also plan on conducting technical and user evaluations of our system at a suitably mature stage to help us get important feedback to improve student learning.

8 Acknowledgments

This work is supported in part by NSF CAREER Award IIS-0845921.

References

1. Alvarado, C.J. *A Natural Sketching Environment: Bringing the Computer into early stages of Mechanical Design* , Master's Thesis, Massachusetts Institute of Technology, 2000.
2. Baraff, D., And Witkin, A. *Physically Based Modeling: Principles and Practice* , Siggraph Course Notes, 1997.
3. Cheema, S., LaViola, J. *Towards Intelligent Motion Inferencing in Mathematical Sketching* , Proceedings of IUI 2010, 289-292, 2010.
4. Jiang, Y., Tian, F., Wang, H., Zhang, X., Wang, X. and Dai, G. *Intelligent Understanding of Handwritten Geometry Theorem Proving* , Proceedings of IUI 2010, 119-128, 2010.
5. Kara, L.B., Gennari, L., And Stahovich, T.F. *A Sketch-based Tool for Analyzing Vibratory Mechanical Systems* , Journal of Mechanical Design, Volume 130, Issue 10, 2008.
6. LaViola, J. *Advances in Mathematical Sketching: Moving Toward the Paradigm's Full Potential* , IEEE Computer Graphics and Applications, 27(1):38-48, January/February 2007.
7. LaViola, J. *Mathematical Sketching: A New Approach to Creating and Exploring Dynamic Illustrations* , PhD Thesis, Brown University, 2005.
8. LaViola, J. and R. Zeleznik. *MathPad^2: A System for the Creation and Exploration of Mathematical Sketches*, , ACM Transactions on Graphics (Proceedings of Siggraph 2004), 23(3):432-440, August 2004.
9. Lee, W., de Silva, R., Peterson, E.J., Calfee, R.C, Stahovich, T.F. *Newton's Pen: a pen based tutoring system for statics* , Proceedings of SBIM 2007, 59-66, 2007.
10. Millington, I. *Game Physics Engine Developement* , Morgan Kaufmann, March 2007.
11. VanLehn, K., Lynch, C., Schulze, K. Shapiro, J. A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., and Wintersgill, M. *The Andes physics tutoring system: Lessons Learned* , International Journal of Artificial Intelligence and Education, 15 (3), 1-47, 2005.
12. Varberg, D. And Purcell, E.J. *Calculus with Analytical Geometry* , Prentice Hall, 1992.
13. Young, H.D. *University Physics* , Addison-Wesley Publishing Company, 1992.
14. Zeleznik, R., Miller, T., Li, C., and LaViola, J. *MathPaper: Mathematical Sketching with Fluid Support for Interactive Computation* , Proceedings of Smart Graphics 2008, 20-32, 2008.