Towards Intelligent Motion Inferencing in Mathematical Sketching

Salman Cheema
School of EECS
University of Central Florida
salmanc@cs.ucf.edu

Joseph J. LaViola Jr.
School of EECS
University of Central Florida
jjl@cs.ucf.edu

ABSTRACT

We present a new approach for creating dynamic illustrations to assist in the understanding of concepts in physics and mathematics using pen-based interaction. Our approach builds upon mathematical sketching by combining the ability to make associations between handwritten mathematics and free-form drawings with an underlying physics engine. This combination lets users create animations without having to directly specify object behavior with position functions through time, yet still supports writing the mathematics needed to formulate a problem. This functionality significantly expands the capabilities of mathematical sketching to support a wider variety of dynamic illustrations. We describe our approach to creating this mathematical sketching/physics engine fusion and discuss how it provides a foundation for using mathematical sketching in intelligent tutoring systems.

Author Keywords

Pen-based Interfaces, Mathematical Sketching, Sketch Parsing, Sketch Inferencing

ACM Classification Keywords

H.5.2 Information Interfaces and Presentation: User Interfaces—*Interaction styles*; I.2.3 Artificial Intelligence: Deduction and Theorem Proving—*Inference engines*

General Terms

Design, Human Factors

INTRODUCTION

Diagrams are a crucial part of many scientific disciplines. They aid learning by presenting concepts in a visual form [9, 10]. To solve problems, students often sketch a diagram using pencil and paper. The diagram usually includes initial conditions provided in the problem statement. However, such diagrams are static and serve only as a starting point to solving the problem. The answer may be a number or a function that does not necessarily provide much insight into the underlying concepts. By animating the drawing in a meaningful way, better insight and understanding can be imparted to students. Mathematical sketching is an approach that provides users the ability to animate these diagrams on a pen-based computer by associating them with handwritten mathematics to govern their behavior [5].

Our underlying research goal is to use mathematical sketching as the foundation for intelligent tutoring systems for mathematics and physics. To reach this goal, mathematical sketching needs to evolve to include a firm understanding of the problem, its solution, and a user's input. It can then provide appropriate feedback (using dynamic illustrations) on whether a user's solution is correct or not. In many cases, the information users enter will be insufficient to make a proper animation. Although the current mathematical sketching implementation supports a wide variety of dynamic illustrations [4], it is significantly limited because users must directly specify how objects behave with position and/or rotation functions of time. An inspection of [9, 10] show problems students are asked to solve rarely conform to the dynamic illustration creation scheme provided by mathematical sketching. Thus, mathematical sketching needs to be broadened to include inferencing capabilities to make a proper dynamic illustration, given information that is only indirectly or partially related to providing a behavioral specification for the animation.

In this paper, we present an approach that moves mathematical sketching towards the ability to infer proper dynamic illustrations from incomplete specifications. We combine the ability to make associations between handwritten mathematics and free-form drawings with an underlying physics engine. This combination provides mathematical sketching with more flexibility to support animations without having to directly specify object behavior with position/rotation functions through time, yet still supports writing the mathematics needed to formulate a problem.

RELATED WORK

Systems for recognizing and animating diagrams in terms of basic primitive shapes have been developed by Alvarado [1] and Kara [3]. They allow the representation of a range of problems in specific domains such as mechanical design and vibratory systems by using a simulation backend. However, they are limited in scope because they do not allow the user to write mathematics to govern animation behavior. MathPad^2 [6] sought to overcome these limitations by

IUI'10, February 7–10, 2010, Hong Kong, China.

having users write down the mathematics to govern all aspects of a diagram's animation, but it is also limited in the range of problems that it can represent (e.g. it would be hard to model a collision resolution problem in MathPad[2]). Our approach combines the best elements of [1, 3] with mathematical sketching, providing animations that will work when users provide just the diagram, the diagram with a full mathematical specification, or the diagram with a partial mathematical specification[1].

SYSTEM OVERVIEW

In our system, users can create a dynamic illustration by sketching a diagram and writing down initial conditions and any equations as part of a particular problem. User input takes the form of writing with a stylus on a tablet PC. After drawing the diagram and writing down initial conditions, users can request an analysis of the diagram.

The system attempts to analyze the diagram in terms of its components: convex polygons, circles, springs, and wires. The ink for recognized diagram components is replaced by rectified components. This lets the user know that the system has correctly interpreted the diagram. As the system needs to perform diagram animation with incomplete information, realistic values are assigned to recognized components' attributes based on appearance and position. For example, a shape's default mass is assigned proportional to its enclosed area, in order to provide realistic animated behavior because larger objects are perceived to have more mass.

Users can replace default attributes with proper initial conditions by writing mathematical expressions and making associations with recognized diagram components. To make an association, a lasso gesture is used to select one or more expressions followed by a tap gesture. The user can associate either a constant expression (e.g. $m = 5$) or an equation (e.g. $v(x) = sin(t)$) with any relevant diagram component. The system applies the force of gravity to all components by default. Users may associate any number of constant/variable forces such as push, drag, and reaction forces. Erroneous associations are ignored automatically. For example, it does not make sense to associate velocity equations with wires and springs. Similarly, associating spring constant values with a shape is meaningless. At any time, users can view associations by hovering the stylus over a recognized component.

Allowing users to modify the behavior of the system in this manner has several benefits. First, users can modify default values and behavior as needed for a given problem. Second, users should be able to debug associations and fix errors. Errors can exist in initial conditions, causing incorrect animation that conflicts with the user's intuition. We provide a reset mode that lets users correct and alter existing associations. Finally, our approach allows users to experiment with different initial values and gain better insight into the working of underlying concepts. Consider a projectile problem. It may be instructive to experiment with changing initial velocity or the magnitude of the drag force to see the effects on the range of the projectile.

[1]A simple example of a partial specification would be writing the appropriate force equations for an object and having the underlying physics engine fill in the details to ensure a plausible animation. The user would not have to write down a numerical routine to solve the differential equation.

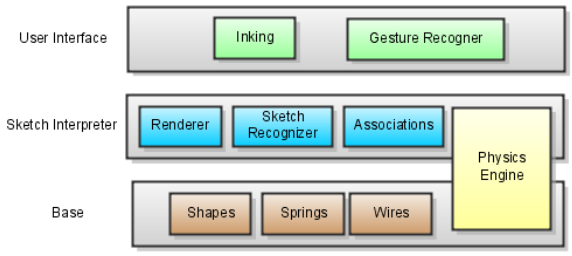

Figure 1. Overview of our system components.

In keeping with our goal to emulate pencil and paper, existing associations are preserved along different system modes. If users want to alter a 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 the user's part and lets her focus on the problem at hand rather than being encumbered by the user interface.

SYSTEM DESIGN

We use a layered approach for flexibility. A custom 2D rigid body physics engine [7] is used for updating and animating diagram components. See Figure 1 for an overview of the system components.

User Interface

The user interface provides facilities for inking to let the user sketch diagrams and write mathematics in a natural manner. A gesture recognition module recognizes three gestures: lasso, scribble-erase, and tap. The gesture set is limited to provide a simple and accessible user interface. The lasso gesture can be used to select both ink and recognized diagram components. Users can drag and reposition the selection anywhere on the screen. Similarly, the scribble-erase gesture is used to erase ink and diagram components. If users select mathematics and tap a recognized diagram component, an association is made. If no math is selected, the tap gesture will cause a recognized shape to become immobile. We provide options to save/load ink in order to easily recall an old problem for revision or clarification. Instant recognition feedback is provided by means of an online mathematics recognizer [8]. Recognition errors can be fixed by using the scribble erase gesture. The feedback results are also used to make the association which improves performance by not doing unnecessary recognition and minimizes recognition errors by using correctly recognized mathematics.

Sketch Interpreter

The sketch interpreter has components to perform diagram recognition, manage associations, render content, and update the animation. A large part of the physics engine resides

in this layer because association information is maintained here. The associations affect initial values and behavior of the physics engine. Therefore the physics engine cannot be decoupled from this layer.

Diagram Recognition

Proper parsing of a diagram is necessary to convert a problem into components that can be animated by the physics engine. From a usability perspective, it is critical that the symbols used to represent diagram components be intuitive and obvious to a physics student. We use circles, convex polygons, springs and wires as basic diagram components. We believe that it is possible to model a wide variety of elementary physics problems using these basic components. This section describes our methodology for recognizing them.

A cusp detector is used to recognize shapes where shapes must be closed strokes. A shape is classified as a polygon if it has more than 2 cusps, otherwise it is classified as a circle. A stroke is a spring if it is not a closed stroke and has three or more self intersections. We employ a line segment intersection test for counting self intersections. Wires are relatively straight lines.

An ink stroke is either a diagram component or part of a mathematical expression. The distinction is important because it is possible to misclassify parts of a mathematical expression as diagram components (e.g. zeros as circles or symbols with self intersections as springs). We use a few simple rules to address these problems. Shapes are separated from mathematics by ensuring that all convex diagram components enclose a minimum area. For springs and wires, at least one end of both must be attached to a shape. Hence recognition proceeds in the following order: shapes, springs, wires, and mathematics.

Associations Management

Associations between mathematics and diagram components are used by the physics engine at runtime to animate a diagram. Associations can include both constant expressions and equations. Constants can modify almost all attributes of a diagram component such as mass, velocity, angular velocity, orientation, position, acceleration, and forces. Constants are applied once, at the start of the animation. It is important to have a mechanism to undo them, because the user can reset the animation and start over with different parameters.

It is possible to write equations for several attributes of diagram components. The physics engine populates the appropriate set of equations with their parameter values at runtime and evaluates them to guide the animation of the diagram. When evaluating equations with errors, any unrecognized parameter is assigned a value of zero. This can yield an incorrect animation but also serves as feedback to the user indicating some error in input is causing abnormal behavior.

Animation of Diagram Components

Diagram components are animated by a custom 2D Rigid Body physics engine [7]. The default animation behavior of all diagram components depends on the standard equations of motion. The engine updates every movable component's position by computing net acceleration and then integrating twice for position. Collision detection and resolution are performed after the update. This step may also involve the computation of rest forces for components that are in resting contact [2]. Lastly, a post processing step is applied that infers unspecified circumstances. Important attributes such as forces and velocity are rendered using arrows. The length of each arrow changes in proportion to the magnitude of the quantity it represents, which lets users observe how important quantities change due to specified initial conditions.

The physics engine has a very open ended design. Much of its behavior can be altered by users. Users may move diagram components by lassoing and dragging. Attributes of diagram components can be altered by associating constants/equations. Users can also specify position or velocity equations as a replacement for standard forces or acceleration equations. In such cases, the system is able to infer what needs to be done to update components' positions. If only velocity is specified, it infers the need to integrate once to update position. If the position equation is specified, it infers that no integration is required. The net force is always computed for display purposes.

Interesting scenarios arise when the default behavior is overridden by user-specified behavior. Consider the following example. Two objects X and Y are part of the diagram. X is moving under standard equations of motion. Y is moving under user-defined equations that specify how its position changes with time. An important question from an animation perspective is what happens if X and Y collide? Clearly the position equations for Y no longer applies. How should the user-defined behavior be changed in order to produce a realistic animation? Our approach is to revert to the standard equations of motion.

Inferencing Unspecified Events

This subsystem is best illustrated by an example. Consider an object attached to some wires. Suppose that the combined tension in all wires is not enough to inhibit motion. How should the system proceed with animation? Clearly the object should move, causing some wires to break. Our approach is to start breaking the wires most opposite to the direction of motion, providing a convincing animation.

Such cases are not specified by users. To incorporate such unseen events into the animation, we have built a simple inferencing system into our physics engine. Inferencing is done after collision resolution. Currently it is limited to detecting if an object is in equilibrium (i.e., will it move?). Such scenarios can occur in a variety of situations involving wires, springs, and inclined plane problems.

EXAMPLE SCENARIO: FREEFALL

Consider a free-fall problem. A student is asked to write down the drag force acting on the body with a coefficient of 0.5 (see Figure 2). The student draws a ball and writes an expression for aerodynamic drag. Upon analysis, the system recognizes and replaces the ball with a circle. The user

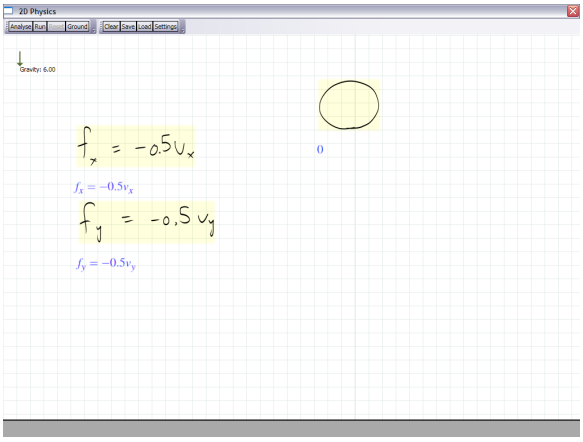

Figure 2. A diagram exploring free fall with aerodynamic drag.

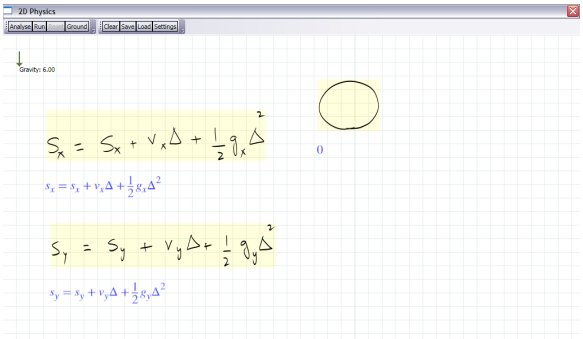

Figure 3. An altered freefall diagram that specifies how the position of the ball changes instead of giving the initial conditions.

associates the drag force with the circle. When run, the animation shows how the velocity and the drag force changes as the ball falls towards the ground. The ball will bounce a few times, losing momentum with each bounce. Eventually it will come to a complete stop on the ground. Note that the student was not asked to find the position equations for the ball, just write an expression for the drag force. In this case, the system infers from the drawing and the mathematics what forces are applied to the ball and that it must fill in the details to ensure a proper animation.

The system also supports the more traditional mathematical sketch creation mechanism. Suppose the student is asked to write down the ball's position equations. In this case, the system detects from the written mathematics that it cannot use the standard equations of motion because position equations have been defined. Instead, it uses the given equations to update the position of the ball. When the ball collides with something while moving, the specified equations become obsolete. The system detects this and the motion of the ball reverts back to the standard equations of motion.

CONCLUSION

We have presented a system that combines mathematical sketching with an underling physics engine to allow creation of a wider variety of dynamic illustrations to understand physics and mathematics concepts. This fusion provides a mechanism to infer how to make a proper animation given different levels of granularity of user input, from diagram only to complete behavioral specification. Currently our system understands the relationship between acceleration, velocity, and position/orientation. We plan to extend this by building diagrammatic reasoning components into the system. Although we have made strides toward creating a mathematical sketching system suitable for intelligent tutoring, there is still a significant amount of work that needs to be done to make a general solution. Our system needs to develop an understanding of concepts related to the $F = ma$ equation, such as work and kinetic energy, power, momentum, and impulse among others. An evaluation of the performance of system components, such as the shape and gesture recognizers, will help us improve our system. We also plan to get feedback regarding the system's usability and effectiveness by conducting a user study among university physics students.

ACKNOWLEDGEMENTS

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. 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
4. 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.
5. LaViola, J. *Mathematical Sketching: A New Approach to Creating and Exploring Dynamic Illustrations* , PhD Thesis, Brown University, 2005
6. 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.
7. Millington, I. *Game Physics Engine Developement* , Morgan Kaufmann, March 2007
8. StarPad. *http://pen.cs.brown.edu/starpad.html*, 2009.
9. Varberg, D. And Purcell, E.J. *Calculus with Analytical Geometry* , Prentice Hall, 1992
10. Young, H.D. *University Physics* , Addison-Wesley Publishing Company, 1992