EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling (2010)
M. Alexa and E. Do (Editors)

A Pen-Based Tool for Visualizing Vector Mathematics

Jared N. Bott and Joseph J. LaViola Jr.

University of Central Florida, School of EECS, Orlando, FL, USA

Abstract

We present VectorPad, a novel, pen-based application for three-dimensional vector mathematics visualization. VectorPad allows users to define vectors and perform mathematical operations on them through the recognition of handwritten mathematics. The user interface consists of a sketching area, where the user can write vector definitions and operations such as addition, subtraction, scalar multiplication, and cross product, and a 3D graph for visualization. Vectors are visualized dynamically on the graph, which can be manipulated by the user. We also performed a short, informal user study evaluating the user interface and visualizations of VectorPad. Results from the study show that visualizations were generally well liked but the application needs to provide a more comprehensive set of visualization tools as well as refinement to some of the animations.

Categories and Subject Descriptors (according to ACM CCS): User Interfaces [H.5.2]: Interaction styles—User Interfaces [H.5.2]: Graphical user interfaces—

1. Introduction

Visualization of three-dimensional vector operations can be very helpful in understanding vector mathematics. Entering 3D vector mathematics when using traditional WIMP-based (Window, Icon, Menu, Pointing device) [Shn86] techniques can be confusing and typically requires special knowledge of a one-dimensional input language. In addition, current tools for visualizing 3D vector mathematics are often troublesome and unintuitive. Often, they focus on support for one or two mathematical operations and do not provide significant insight into more complicated mathematical equations (e.g., equations that are composed of several vector operations). Given the complexity of vector mathematics, the ability to visually explore these operations has the potential to give users valuable insight into their nature. Thus, our goal is to provide a tool that provides the user with a simple way to visualize 3D vector operations with the ease of entering mathematics with pencil and paper.

To deal with these issues, we developed VectorPad, a novel application prototype that provides users with the ability to enter handwritten vector mathematics equations using a pen-based interface. These equations are then solved and evaluated using a mathematics engine, and the results are dynamically displayed on a 3D graph. Users can edit their mathematical equations and see how these changes affect the mathematical operations.

2. Related Work

Mathematics and pen-based computing has been explored in a number of ways. First, there are a variety of systems that use a stylus simply as a mathematics input system. In [Fat04], Fateman examines a number of pen-based mathematical input systems. PenCalc is another mathematical calculation system that uses handwriting as its input [CY01]. The Freehand Formula Entry System (FFES) is a mathematical formula editor that uses handwriting as its input [SNA01]. FFES also includes the ability to generate LaTeXfrom its input. Commercially, there are systems available such as Microsoft's Equation Editor, which can export to MathML.

Another type of pen-based mathematical system is mathematical sketching. First developed in MathPad^2 [LZ04], mathematical sketching is a pen-based approach for mathematics problem solving [Lav05] where users writes mathematics and supporting diagrams to create simple animations. These animations give the user the ability to use their intuition of the mathematics and other domain knowledge to verify that the mathematics is correct. Similarly, there are interactive computation systems that take pen-based mathematics as input and solve the mathematics. MathBrush is one such system, and uses a similar system architecture as VectorPad, containing a character recognizer, a structural parser, a typesetting system, and an interface with a compu-

DOI: 10.2312/SBM/SBM10/103-110

tational engine [LLM*08]. Zeleznik et. al. developed MathPaper [ZMLL08], a gestural system for pen-based mathematics, using a variety of gestures similar to MathPad^2 . MathPaper supports multiple computational engines, as well as a set of notations for controlling computational assistance. Microsoft Math [Mic09] is a commercial system that, among other things, allows the user to input mathematics using handwriting and solve the expressions. With VectorPad, we wanted to explicitly combine pen-computing and vector algebra to explore an area not fully realized in other work.

In terms of vector algebra visualization, three-dimensional vectors are generally visualized using a Cartesian coordinate system, particularly in math and physics textbooks. Often times 3D vector visualization is explored in the context of geometry visualization [KSW00, Mal04]. Most computer algebra systems provide some sort of graphing ability, such as MathEdge [MW97], which uses Maple as its computational backend. Tront's VectorPad [Tro09] is close in spirit to our application in that it is a pen-based system for vector visualization that allows the user to draw vectors and perform some mathematics using them. However, Tront's work focuses on 2D vector mathematics, does not provide animations, and does not let the user write down the vectors using mathematical notation. There are many small applications available on the Internet for basic vector mathematics visualization, particularly in two dimensions, which have been studied and surveyed in numerous papers [CZ00, Cat06, EH04]. However, these applets have a narrow focus in their visualizations and supported mathematics in comparison to VectorPad.

3. User Interface

3.1. Sketching Area

VectorPad was designed to have an interaction model similar to pen and paper. To this end, we provide a sketching area for the user to write mathematics as they normally would using pen and paper (Figure 1). Using a stylus, the user inputs mathematics through writing and performing a variety of gestures. A combination of three basic gestures are used, a lasso gesture, a tap gesture, and a scribble gesture. Since we use a reduced gesture set, similar combinations of gestures were chosen to perform similar actions in order to aid the user in remembering the gestures and their actions.

Each stroke the user writes is passed to an online mathematical handwriting recognizer from [ZMLL08], which recognizes the individual characters and the underlying mathematical structure. The results from the mathematical recognizer are displayed below the user's writing in real-time, using a mathematical typesetting system (see Figure 2). After a character is recognized, alternate recognition results are displayed on a toolbar at the bottom of the screen. When the user makes a lasso gesture around any set of strokes, the alternates toolbar is populated with a list of alternate recognition results for the enclosed strokes. This approach is similar

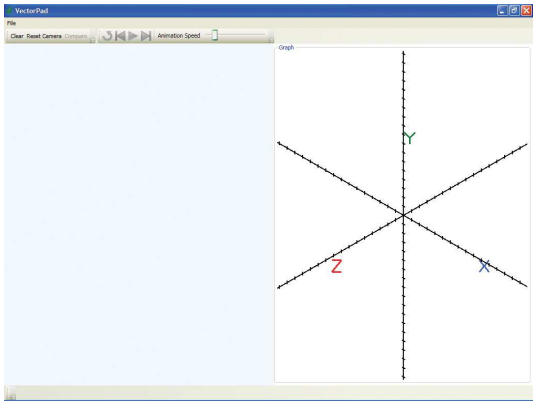

Figure 1: VectorPad's main window: on the left is the sketching area, on the right is the graph.

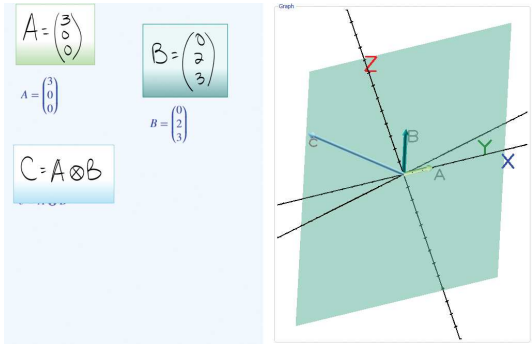

Figure 2: Three vectors are shown on the graph, A, B, and C. C is the cross product of A and B. The plane made by A and B is shown in a greenish-blue hue. C is shown as a purple arrow.

to a suggestive interface [IH07]. The lasso gesture used in combination with a tap gesture (lasso-tap) groups a set of strokes into an equation, as in [LZ04].

After an equation is created, it can be edited through the addition or removal of strokes. To remove strokes, a user makes a scribble gesture upon the strokes. VectorPad's graph seamlessly reflects changes made to the equations written on the sketching area. The results from one equation can also be used in another and the dependent equation automatically receives updates from the equation it depends upon.

3.2. Graph

The graph in VectorPad is implemented using a 3D viewport with an orthographic camera, which we chose because it provides the perspective that is commonly used in mathematics and physics textbooks. A three-dimensional Cartesian coordinate system is used, with the origin at the visual center of the graph. This system is presented as a set of three number lines with tick marks every one unit of measurement (see Figure 1). At the outer edge of the positive side of each

number line is a colored text label with the axis's name (X, Y, or Z). We think that this style works well as it should be familiar to users since it is modeled upon a common visualization.

After performing the gesture to recognize an equation, vectors are dynamically illustrated on the graph. Each vector is visually represented by a colored arrow and the equation is highlighted with the arrow's color; this highlight helps the user to understand the bounds of the equation, both for editing the equation and writing other equations. When a mathematical operation is recognized, the operation is animated on the graph; each operation has its own animation, designed to illustrate in an intuitive manner.

The graph is controlled using a trackball model, where the user moves the stylus throughout the graph to rotate it [CMS88] (see Figure 2). The graph can also be zoomed by pressing the stylus button and moving the stylus up to zoom in, and down to zoom out. When a new equation is graphed, the graph is automatically zoomed in or out to fit the various arrows onto the screen, so that they are fully visible.

A user might want to only see the arrows for a specific equation, so we provide a gesture combination to hide specific arrows: a lasso-double-tap. The lasso-double-tap gesture is performed by lassoing around an equation and tapping twice within the lassoed area. This causes the arrows to be removed from the graph and they are not used in calculations for automatic zooming. Conversely, the user might want to zoom in to a specific equation, which can be accomplished through by double-tapping on the equation.

Numerous labels are on the graph, for example the number lines are labeled with their axis name, and each arrow with its name. This text is visually two-dimensional, but has a three-dimensional location within the graph. In order to increase legibility of the text, as the graph is rotated by the user, the text rotates to stay facing the camera. We also wanted users to be able to see information about the vectors that is not readily apparent from the graph, like the normal of the vector, its length, and its equation. Clicking on an arrow brings up a small visual overlay on the graph that shows its equation, its vector value, length, and normal.

3.3. Visualization Constructs

VectorPad supports several mathematical operations: vector addition and subtraction, scalar multiplication and division, cross product, dot product, and length. When an operation results in a vector answer, a new arrow is added to the graph. These result arrows are colored one of several pink hues to differentiate them from arrows not derived in this manner. For vector-result operations, the operand arrows are moved around the graph and leave behind semi-transparent "shadows." The animations typically show the operands moving in some manner so that the user can see how the operation affects the two operands. Then, an arrow for the result is added to the graph. When an animation has finished, any arrows that have moved return to their original positions and the shadow copies are removed. This process shows the user the operation separate from its final representation and allows them to focus upon the operation before examining the result. The length of an animation can also be changed, allowing the user to examine how the operation works at their own pace.

3.3.1. Addition and Subtraction

Addition is an operation that has two operands and results in a vector answer, thus, the addition animation has two component arrows and a result arrow. The animation for addition takes the arrow for the second operand and translates it such that its origin is no longer the origin of the graph, but the endpoint of the first component arrow. This places the endpoint of the first arrow where the result's endpoint will be. Once the second arrow has finished translating, a new arrow is created.

The subtraction animation is similar to addition, except that the second arrow first rotates to its vector's opposite (i.e. its negative vector). We designed this animation for addition and subtraction because it shows the user the common visualization of these operations in vector mathematics, a triangle made of the operands and the result.

3.3.2. Scalar Multiplication and Division

Scalar multiplication and division share the same animation style. These operations have two operands, one a vector and the other a scalar, and the result is a vector. In scalar multiplication, all three components of the vector operand are multiplied by the scalar operand; in scalar division, all the vector components are divided by the scalar operand. Since there is only one vector operand, there is only one component arrow to be animated. We created two animations for these operations. In the first variant, the component arrow scales to the size specified by the operation. In the second animation, which we termed the "stacking" animation, the component arrow is stacked end over end the number of times specified by the scalar value. With these animations, we hoped to show the relationship between the vector and scalar operands.

3.3.3. Cross Product

The cross product operation has two operands, both vectors, and the result is also a vector; a cross product of two vectors results in a vector that is perpendicular to both operands. For this operation, we also created two animations. In the first animation, the first component arrow rotates to the second component arrow and then rotates to the direction specified by the resulting perpendicular vector. The second animation is similar to the first, except that a semi-transparent plane is added to illustrate the plane made by the two component arrows. We created the second animation after some users expressed that they were used to seeing a plane in the illustration of a cross product operation.

3.3.4. Dot Product

Like the previous operations, a dot product operation (also known as the scalar product) has two operands, both vectors. Unlike the previous operations the result of a dot product is a scalar value: the like vector components of the two operands are multiplied and summed together. This scalar result led to a decision to display text results, since a scalar is difficult to graphically represent. Another important aspect of the animation comes from the dot products' relation to the angle between the two vectors. By taking the length of the first operand and multiplying it by the cosine of the angle between the two vectors, the first vector is scalar projected onto the second vector. This is often illustrated when showing the result of a dot product, so we felt it beneficial to include it in the animation. We draw a distinctly colored arrow to show this projection.

3.3.5. Complex Operations

VectorPad allows users to create complex equations utilizing multiple operations. To make it easier for users to understand a complex equation, we divide them into a series of simple animations. The complex animation for $D = A \otimes B \otimes C$ is show in Figure 3. In the list of animations that VectorPad maintains, each animation for a subequation appears as a normal animation, so that a user can repeat it as they like.

3.4. User Controls

VectorPad has several user controls designed to aid the user in understanding the vector mathematics and the visualizations we constructed. We considered the ability to replay animations important while designing VectorPad, because a user might not understand an animation on their initial viewing. VectorPad thus has a set of VCR-like controls to give the user the ability to playback animations. A slider control and four VCR buttons are included: play, backward, forward, and replay. The slider controls the length of time an animation plays for, from 0.5 seconds to 10 seconds, with a default value of 2 seconds, which we found to be a good intermediate length and gives users sufficient time to examine the operation. The replay button plays the last animation, allowing the user to easily repeat the animation until they understand it. The backward and forward buttons move through the list of animations and the play button plays the currently selected animation. In addition, VectorPad has a compare button that brings up a second window with two graphs to allow the user to compare more easily visual changes to equation parameters.

4. Architecture

The architecture of VectorPad is divided into three primary components, one for the sketching area (the InkHandler), one for equations and other mathematics (the EquationHandler), and one for the graph (the GraphicsHandler).

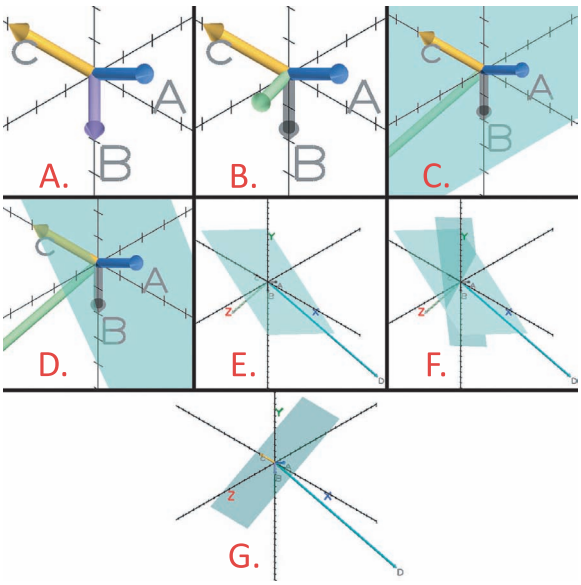

Figure 3: Two cross product animations. The arrows for A (blue), B (pink), and C (yellow) are shown in Subfigure A. In Subfigure B., the arrow for B is rotating to C and has been colored green. Subfigure C. shows an intermediate arrow scaling to its final length and the plane between B and C is rotating into place. Subfigure D. shows another intermediate step right before the plane has finished rotating. In Subfigure E. the plane between B and C is in place and an arrow for D (turquoise) has scaled and rotated to its final position. Subfigure F. shows an intermediate state where the plane between A and an intermediate arrow is rotating to its final position. Subfigure G. shows that plane in its final position.

4.1. InkHandler

The InkHandler deals directly with user input; it consists of a mathematical handwriting recognizer and a gesture recognizer, and directly interfaces with the EquationHandler and GraphicsHandler. For mathematical handwriting recognition, we chose to use the recognizer developed in MathPaper [ZMLL08] because it provided mathematical typesetting and was easily modified to fit our application requirements. For gestures, the InkHandler recognizes combinations of three gestures, tap, scribble, and lasso, and then passes the recognition of them on to the other components. A gesture is used to group strokes into equations for two reasons. First, the creation of arrows and animations based upon incomplete equations would likely be quite distracting and confusing to users. Second, the math recognizer had difficulties in differentiating closely spaced equations.

4.2. EquationHandler

4.2.1. Identifier

We designed components to internally represent important aspects of mathematical equations. The most base component of these is an Identifier, which represents a variable in an equation. Identifiers can be divided into two types in relation to how they are used within an individual equation, primary and secondary. Primary Identifiers are those whose value is being assigned (the left hand side of the equation), while secondary Identifiers are other Identifiers used in the equation. For example, in $A = [3,5,7]^T$, there is one Identifier "A", which is a primary Identifier, and has a vector value of $[3,5,7]^T$. In $C = A + B$, there are three Identifiers, A, B, and C, where C is the primary Identifier, and A and B are secondary Identifiers.

4.2.2. Equation

The next most specific component, which represents a user-input equation, is the equation object, the main organizational unit of VectorPad. The primary purpose of an equation object is to be a collection of the Identifiers, code, graphical models, and other objects that are created when an equation is written by the user.

Conceptually, there are two kinds of equation objects: assignment equations, those that simply define an Identifier with some value (e.g., $A = [3,5,7]^T$), and mathematical equations, those that define an Identifier through a mathematical operation (e.g. $C = A + B$). When a user writes an equation and performs a lasso gesture, an equation object is created, and all the strokes enclosed by the gesture are stored in the equation object. Recognition results from the math recognizer are also stored in the equation object in the form of a MathML structure.

4.2.3. Code Generator

When an equation object is created, the EquationHandler interfaces with the InkHandler and stores recognition results. To convert those results from MathML into a format that the mathematical engine can execute, we implemented a series of components to convert from one format to the other and generate the code that we need to execute. This Code Generator must also replace a diverse set of mathematical/typographical symbols with the specific set of symbols accepted by our computational engine, MATLAB.

4.3. GraphicsHandler

Variables are represented graphically by 3D arrows (as described in 3.2), which are implemented by the Arrow component. We implemented the arrow model as a cylindrical tube and a conical tip, and chose a set of distinct colors from which to color the Arrows. We also have a second set of colors, all shades of pink, for Arrows created for primary Identifiers that are from mathematical equation objects.

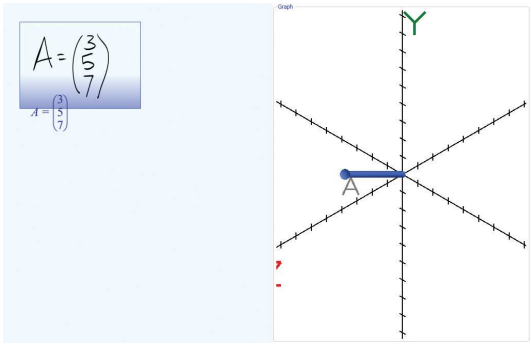

Figure 4: An Arrow has been created for Identifier A. This Arrow is colored blue, as is the box showing the bounds of the equation object.

After executing an equation object's code, an Arrow is created for the primary Identifier of the equation object. As discussed in Section 3, animations are created for mathematical operations; these animations manipulate the Arrows representing the secondary Identifiers in an equation object.

4.4. Process

In this section, we discuss the processes of VectorPad. The InkHandler first recognizes handwriting, performs recognition upon it, and passes the recognition results to the EquationHandler when a lasso-tap gesture is recognized. The EquationHandler then creates equation objects, and the recognition results are analyzed for mathematical operations. Finally, GraphicsHandler takes over, creating Arrows and animations as necessary.

4.4.1. Creating An Equation Object

When a user writes in the sketching area, the InkHandler operates and each stroke is tested to see if it is a gesture; once it has been determined that it is not, the stroke is passed to the math recognizer. Results are typeset and displayed on the sketching area. After a user has written their equation, they lasso-tap around it, and the recognition results for all the encircled strokes are pulled from the math recognizer in the form of MathML. Finally, the component strokes are grouped into an equation object which also stores the recognition results.

4.4.2. Converting to Computational Engine Code

Next, the EquationHandler begins its task and the MathML is passed to the Code Generator. Aside from generating executable code, during the Code Generator's processing, variables are identified, Identifiers are created, and secondary Identifiers are generated. The code that is generated is in a form from which it is easy to create animations.

4.4.3. Getting Results

Now the code can be executed by the computational engine and the results retrieved. For an equation object, we retrieve

the results only for the primary Identifiers for which results are not already known. Each result is retrieved in the form of an array, the dimensions of the array are examined to determine the Identifier's type (scalar or vector), and the value is stored.

4.4.4. Graphing and Animating

Next, the code for these Identifiers is examined to determine if it is an assignment or a mathematical statement. From here, the GraphicsHandler is in operation. A data structure with information for creating Arrows and animations is created from the Identifiers, and the structure is added to a queue of pending graph operations.

In the case of an assignment, an arrow is created upon the graph with its origin at (0,0,0) and its end at the vector value of the Identifier. The new arrow is assigned a color upon creation to help the user to differentiate between the various arrows. The arrow is added to the equation object and the object's strokes are highlighted with the arrow's color. When the arrow is added to the graph, it appears in its final position, with no animations showing its creation. Initially, VectorPad animated the creation of an arrow, showing it "growing" and rotating to its final size and rotation, but test users generally did not like having this animation, as they found it confusing and distracting, so it was removed. Once the Arrow has been added to the graph, the graph is zoomed so that the Arrow mostly fills the graph (see 4.4.5). At this time, the axes are also extended if the arrow extends past the current bounds of the axes.

For a mathematical animation, we first look at the mathematical operation to be animated. In general, animations move the arrows associated with the secondary Identifiers and leave behind "shadows" to show the user where the arrows originally were located. Again, the axes are extended to be longer than any arrow, if necessary. After the animation has completed the camera is zoomed to fit all the arrows on screen.

4.4.5. Fitting the Camera

When a new model is added to the graph, or an object is updated, VectorPad zooms the camera to ensure that the relevant models are visible on the graph and that they are maximized visually. The models that we are concerned with are arrows, planes, and text labels that are not used to label an axis. Fitting the camera to show all the models on the graph is a complicated procedure. For each model, its 2D coordinates on the graph must be computed, which is simple for arrows and planes, but problematic for text labels. Unfortunately, the approach we used for obtaining the 2D coordinates for arrows and planes in the 3D viewport did not work for our text labels, due to an implementation issue. In the method we devised, we look at the collection of 3D points that is used to create a text object's component letters. We took this collection, converting all the points to 2D by using a matrix that converts the viewport's 3D coordinates to 2D screen coordinates, and then finding the top-most, bottom-most, left-most, and right-most points to create a bounding box.

Once we have the 2D bounding boxes for all the relevant models, a bounding box containing all the models is created. A fit test is performed in which the bounds of the box are compared with the graph's rendered dimensions to determine if the models are all on-screen, and if so, whether the camera can zoom in more. If we determine that more zoom is needed, we iteratively zoom in and perform the same fit test until it is found that we have zoomed in too much. At that point we iteratively zoom out and perform the fit test until we have zoomed out too much. Lastly, we zoom out a small amount to give a bit of room around the edges of the graph. If the original fit test tells us that the models do not fit on the screen, the same sort of process is performed, but zooming out instead of in.

We felt that a user might want to zoom to the models for a specific equation object; when the InkHandler recognizes a double-tap in the bounds of an equation object, the GraphicsHandler starts the camera fitting process, but only for the models associated with the equation object. If the user wants to invoke fitting for all the models on the graph, they can double-tap outside of all the equation objects.

4.4.6. Preventing Text Occlusion

With several arrows in the graph area, text labels are fairly likely to occlude each other. We prevent this from occurring by moving one of the two occluding texts. The approach that worked best is the same as in 4.4.5, taking the texts' point collections, converting them to 2D bounding boxes, and comparing the bounding boxes. If the boxes intersect, one text is iteratively moved until the bounding boxes no longer intersect. We move the text in the direction of the normal of its original position ($\vec{p}_{norm} = \frac{\vec{p}}{\|\vec{p}\|}$). The text is moved in increments of one-quarter the norm of the direction ($\vec{d} = \frac{\vec{p}_{norm}}{4}$). Each time a text label is added to the graph, each text is checked against each other text. The same process occurs every time the graph is rotated. It is important that each text's bounding box is calculated using its original intended position for the first test of each testing session, otherwise texts will start to stray as the graph is rotated.

4.4.7. Updating An Equation

Updating an equation begins with one of two actions, erasing an ink stroke, or adding an ink stroke in the bounds of an established equation object. After an update to an equation object, any equation objects that are dependent upon it are updated. First, the graphics objects associated with the dependent equation object are removed from the graph. Next, the new Identifier values are given to the computational engine. The final steps are essentially the same as creating an

equation object. With a chain of dependent equation objects, this update process may occur several times.

When the InkHandler gesture recognizer detects a scribble gesture, it tests to see what strokes it intersects with. Any intersecting stroke is marked to be removed from the sketching area and from any equation object that contains it. Once the strokes to be removed have been collected, they are passed to the EquationHandler, which determines which equation object owns each stroke. Next, each stroke is removed from its owning equation. The last step checks whether all the strokes from an equation object have been removed. If so, the equation object and its arrows are removed from the system, and any equation objects that are dependent upon the equation object's primary Identifier are updated.

We consider an equation object to be orphaned when it is a dependent equation object, and the equation object upon which it depends has been erased (or is orphaned itself). An orphaned equation object does not have any arrows on the graph, because its primary Identifier is considered to have an unknown value. When we determine that there is an orphaned equation object, its strokes are colored a deep red and the user is notified that it cannot be graphed until the missing dependency has been resolved.

When a new stroke is made on the sketching area, the InkHandler performs a hit test against all the recognized equation objects to determine if the stroke belongs to an existing equation object, or to a new, unrecognized equation object. If the stroke is mostly contained within the bounds of an equation object, it is added to that object and new recognition results are generated. Old graphical models are cleared from the graph and the same process as creating an equation object is performed.

5. Informal User Study

We conducted an informal, preliminary user study with six participants to obtain some feedback on VectorPad's design and user interface. Participants were generally versed in vector mathematics and linear algebra. First, we demonstrated the use of VectorPad to the participants, showing them the mathematics VectorPad recognizes and how to use the various gestures. Participants were then free to play with VectorPad, although we asked them to try all the different animations. Once participants were done using VectorPad, we asked them about their experience, such as which cross product animation they preferred, whether they liked each animation, and for their comments on various aspects of the user interface.

5.1. Results

We asked participants whether they found the different arrow colors useful. While most participants said yes, one participant stated that he did not use the colors to identify the different arrows, but rather the labels. However, he did say that he liked having the arrows different colors. Another participant said that she "liked the shaded rectangles with the same color as the arrow," because they helped to distinguish which arrow represented which Identifier. Similarly, one participant responded that she would like the labels to be the same color as the arrow, which she said would make it easier to identify which label was for which arrow.

Participants liked how the graph zoomed in on arrows and animations, as it gave them a better view of the models. Most participants never turned this feature off, even though they were made aware of it during the demonstration and during their usage of VectorPad. We think that this shows that most users would not turn off the feature, at least initially; in other words, care should be taken in the design of such a feature, since users won't turn it off, even when it causes them problems.

Five of the six participants responded that the animation for the addition operation made sense and was clear. The last participant replied that he understood it, but that it did not fit his mental model for vector addition. While the participants felt that the addition animation was clear, there were some problems with the subtraction animation. The primary issue was with the second arrow's rotation to its negative position. This rotation is animated while the arrow is translating to the end of the first arrow and consequently three participants felt that it was less than clear what was occurring. One participant commented that it could be made clearer by translating and rotating at different times, while another said that rotating the arrow was not needed.

One participant who had a minimal background in vector mathematics found the second scalar multiplication animation, the stacking animation, easier to understand, as it better conveyed how the scalar operand affected the vector operand. Participants did not otherwise display much of a preference for one of the animations over the other, but several did express that they felt that the stacking animation would be better for novices who did not understand scalar multiplication.

For cross product, most of the participants preferred the animation that showed the plane made by the two operands. One participant said that it "makes sense" and shows the way people usually visualize it. However, another participant felt that the plane would rarely be needed and that it should be optional. Most participants seemed ambivalent about the dot product animation. Showing the scalar projection was not the most natural idea for the participants. Comments mainly focused upon the text results, with one participant commenting that the "text could make it cluttered quickly."

We had several suggestions for VectorPad from the participants. Some participants suggested that we show the answer to each equation on the sketching area in text form. Currently, to determine the value of a vector, a user must click on its arrow to see the information box that pops open.

We explored having the vector values on the sketching area in early versions of VectorPad, but removed it after negative user feedback.

From the user study we found that we should have provided a separate means for the user to identify the positive and negative directions on the number lines. While each axis is labeled on its positive side with the axis's name, this was not immediately clear to the participants. Additionally, when the graph was zoomed in far enough, the labels were not visible, so users would have no way to identify the positive and negative directions for the axes. Similarly, it can be hard to visually identify or verify the length of any arrow along any axis, particularly for vectors with non-zero components along all three axes. One participant suggested that showing the X, Y, and Z components for each arrow would help users to perform such visual identification.

Participant feedback gave us several avenues for future work. While participants were happy with the default camera position, it did not always provide an ideal view of the vectors and animations. We would like to investigate methods for automatically positioning the camera for specific vectors, particularly during animations. Second, we want to see how to present multiple vectors when there is a large disparity in their lengths. Finally, we would like to explore methods for recognizing when an equation has been completed.

6. Conclusions

We have presented VectorPad, a pen-based tool for dynamic visualization of 3D vector mathematics. VectorPad provides support for a number of mathematical operations on vectors, including addition, scalar multiplication, and cross product, and visualizes these operations using a 3D graph. Users input vectors using pen-based techniques and VectorPad recognizes the user's handwriting using character and mathematics recognition. We also conducted an informal user study, where we found the animations we implemented were generally liked by the participants. However, we also found that the participants wanted tools to help them to further understand and view the visualizations, such as automatic camera rotation. We also identified a number of areas for future work including extending VectorPad to support vector Calculus and further refinement of our dynamic visualizations.

7. Acknowledgements

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

References

[Cat06] CATALOGLU E.: Open source software in teaching physics: A case study on vector algebra and visual representations. *The Turkish Online Journal of Educational Technology - TOJET 5*, 1 (January 2006). 2

[CMS88] CHEN M., MOUNTFORD S. J., SELLEN A.: A study in interactive 3-d rotation using 2-d control devices. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1988), ACM, pp. 121–129. 3

[CY01] CHAN K.-F., YEUNG D.-Y.: Pencalc: a novel application of on-line mathematical expression recognition technology. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on* (2001), pp. 774–778. 1

[CZ00] CROWE D., ZAND H.: Computers and undergraduate mathematics 3: Internet resources. *Computers & Education 35*, 2 (2000), 123–147. 2

[EH04] EASON R., HEATH G.: Paintbrush of discovery: Using java applets to enhancemathematics education. *Primus 14*, 1 (2004), 79–95. 2

[Fat04] FATEMAN R.: Handwriting + speech for computer entry of mathematics. http://www.eecs.berkeley.edu/~fateman/papers/voice+hand.pdf , 2004. 1

[IH07] IGARASHI T., HUGHES J. F.: A suggestive interface for 3d drawing. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses* (New York, NY, USA, 2007), ACM, p. 20. 2

[KSW00] KAUFMANN H., SCHMALSTIEG D., WAGNER M.: Construct3d: A virtual reality application for mathematics and geometry education. *Education and Information Technologies 5*, 4 (2000), 263–276. 2

[Lav05] LAVIOLA JR. J. J.: *Mathematical sketching: a new approach to creating and exploring dynamic illustrations*. PhD thesis, Brown University, Providence, RI, USA, 2005. 1

[LLM*08] LABAHN G., LANK E., MACLEAN S., MARZOUK M., TAUSKY D.: Mathbrush: A system for doing math on pen-based devices. In *DAS '08: Proceedings of the 2008 The Eighth IAPR International Workshop on Document Analysis Systems* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 599–606. 2

[LZ04] LAVIOLA JR. J. J., ZELEZNIK R. C.: Mathpad^2: a system for the creation and exploration of mathematical sketches. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), ACM, pp. 432–440. 1, 2

[Mal04] MALKOWSKY E.: Visualization and animation in mathematics and physics. In *Proceedings of Institute of Mathematics of NAS of Ukraine* (2004), vol. 50. 2

[Mic09] Microsoft: Math. Computer program, July 2009. http://www.microsoft.com/math . 2

[MW97] MCCABE M., WATSON J.: From mathedge to mathwise: The cutting 'edge of interactive learning and assessment in mathematics. In *3rd International Conference on Technology in Mathematics Teaching* (October 1997). 2

[Shn86] SHNEIDERMAN B.: *Designing the user interface: strategies for effective human-computer interaction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986. 1

[SNA01] SMITHIES S., NOVINS K., ARVO J.: Equation entry and editing via handwriting and gesture recognition. *Behaviour and Information Technology 20*, 1 (January 2001), 53–67. 1

[Tro09] TRONT J.: Vectorpad. Computer program, July 2009. http://filebox.ece.vt.edu/~jgtront/tabletpc/vector_pad.html . 2

[ZMLL08] ZELEZNIK R., MILLER T., LI C., LAVIOLA JR. J. J.: Mathpaper: Mathematical sketching with fluid support for interactive computation. In *SG '08: Proceedings of the 9th international symposium on Smart Graphics* (Berlin, Heidelberg, 2008), Springer-Verlag, pp. 20–32. 2, 4