A Natural Handwriting Algorithm for Tablets

Kim Arvin S. Silvoza, Ryan A. Blonna, Rowel O. Atienza
Electrical and Electronics Engineering Institute, University of the Philippines-Diliman
kim.silvoza@gmail.com, ryanblonna3@gmail.com, rowel@eee.upd.edu.ph

Abstract—**Note-taking applications today have features that imitate the behavior of real handwriting such as smooth strokes, responsive interface, and stroke thinning. However, the problem is that the writing tools of these applications do not fully simulate the feel of their real counterpart. Some of these applications have flaws like unnatural pen thinning effect, pixelation of strokes, and lack of highlighter blending. To solve these, an iPad application with a custom smooth writing algorithm was developed. It was compared to five commercially available note-taking programs in the App Store.**

***Keywords*—smooth, natural, handwriting, tablet, note-taking, apps, mobile**

I. Introduction

IN the past, people had used pen and paper as their primary tools for writing. With the recent advent of electronic tablets, this trend slowly began to change. Note-taking applications for touchscreen devices such as Bamboo Paper[1] and Notability[2] were developed to facilitate convenient note-taking.

To simulate the traditional handwriting experience as much as possible, these applications employ various techniques such as line thinning, object blending, and line anti-aliasing. While these programs attempt to create an interface to replace pen and paper, it is currently very difficult simulate a sufficiently accurate pen-and-paper writing experience. As such, documentation on the algorithms involved to create such an interface is scarce, especially for the iOS. To this end, this paper documents an iPad application with a custom pen-and-paper writing interface.

To document the application's implementation, a cubic Bézier interpolation algorithm that allows the rendering of smoothly flowing lines on virtual paper is described. Next, a function that relates the instantaneous line width to the current stylus speed to facilitate natural line thinning is discussed. And finally, two representations of line segments are documented, along with a fast edge anti-aliasing method.

For comparative analysis, the iPad application was tested against five commercially available note-taking programs available in the App Store - namely Bamboo Paper[1], Note Taker HD[3], Notability[2], Noteshelf[4], and UPAD[5].

II. Related Work

A. Note-taking Applications on the iPad

The application review was limited to applications for the iOS platform, since at the time of this project's conception, note-taking applications were more common on the said platform than in others. Five applications - namely (1) Bamboo Paper, (2) UPAD, (3) Noteshelf, (4) Note Taker HD, and (5) Notability - were reviewed based on subjective observations of their pen-and-paper writing interface.

The criteria for this review were (1) writing tool performance, (2) natural ink thinning, (3) stroke anti-aliasing, and (4) highlighter blending. The first criterion, pen tool performance, was measured by the responsiveness and fluidity of each stroke. Next, the quality of natural ink thinning was observed as the realism of perceived line thinning during and after each stylus stroke. The third criterion, stroke anti-aliasing, was taken as the degree of pixelation along the edges of each stroke. And finally, highlighter blending was seen as the presence of a darkening effect when one or more highlighter strokes were drawn in the same virtual space.

Bamboo Paper, UPAD, and Noteshelf exhibited excellent natural ink thinning and the best pen tool performances out of the five. These three applications, however, showed significant stroke aliasing, especially upon zooming in on the strokes.

On the other hand, both Notability and Note Taker HD effectively eliminated stroke aliasing. The pen tool performances of these two applications were noticeably inferior to the first three, and it was interesting to note that Note Taker HD did not exhibit any form of natural ink thinning at all. None of the five applications featured highlighter blending.

By observation, good stroke anti-aliasing produced a marked decrease in pen tool performance, and to some degree, natural ink thinning. However, it is evident that these applications use effective algorithms to accurately simulate pen-and-paper writing. In order to achieve a comparable effect, several line interpolation algorithms were reviewed.

B. Line interpolation

According to Kamermans, there are two basic drawing primitives: the straight line and the curved line[6]. In real handwriting, these primitives are created on the fly while stroking. However, for computers, it is necessary to provide a mathematical function that represents the object you have to draw. Straight lines can be represented by simple linear functions, while smooth curves can be represented using a series of interconnected Bézier curves.

In generating Bézier curves, a minimum of three points are needed: the start point, the end point, and the control points. The control point is responsible for the curvature of the generated stroke. Note that the curve does not pass through any of the control points as seen in Figure 1.

R. A. Blonna and K. S. Silvoza are with the UP EEEI. The adviser for this project is Professor Rowel Atienza.

MOBILWARE 2013, November 11-12, Bologna, Italy
Copyright © 2014 EAI
DOI 10.4108/icst.mobilware.2013.254278

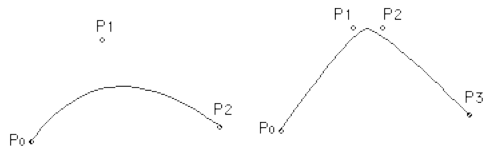

Figure 1. Bézier curves sample

The number of control points depends on the kind of curve to be generated. For quadratic curves (the left image in Figure 1), one control point (*p1*) and two end points (*p0* and *p2*) are needed. On the other hand, cubic Bézier curves (the right image in Figure 1) need two control points (*p1* and *p2*) and two end points (*p0* and *p*3). A characteristic of Bézier curves is that they are parametric curves. This means that every dimension (in this case, x and y) has its own function. Thus, x and y are independent of each other like in normal functions but are dependent on an external variable (in this case, t). Equations 1 and 2 represent cubic Bézier curves. Variable t ranges from 0 to 1 inclusive. Variables x_0, x_1, x_2, x_3 and y_0, y_1, y_2, y_3 represent the x and y coordinates of the start point, two control points, and end point respectively.

$$x(t) = (1-t)^3 x_0 + 3(1-t)^2 t x_1 + 3(1-t) t^2 x_2 + t^3 x_3 \quad (1)$$

$$y(t) = (1-t)^3 y_0 + 3(1-t)^2 t y_1 + 3(1-t) t^2 y_2 + t^3 y_3 \quad (2)$$

To generate the curve given the start, end, and control points, t must be scanned from 0 to 1. The more t values are generated, the higher the accuracy of the generated curve. The points generated from plugging in values would then be connected and a cubic Bézier curve is formed. If these curves are generated dynamically while the stylus moves across the screen, smooth curves will be drawn. However, smooth curves are not enough to make the strokes look realistic. Stroke pixelation must be addressed and this is where anti-aliasing comes in.

C. Overdraw

Overdrawing[7] is a smoothing method where an anti-aliased line is drawn over the outline of the object. The anti-aliased line is then blended with the edge of the object. This removes the discontinuity between the background and the object. Shown in Figure 2 is a sample of an object without and with overdraw.

The advantage of using overdraw is that it is easy to implement and relatively fast. This is because the application does not need to calculate the correct color and alpha value for each pixel in the edge of the object. It only has to obtain the outline of the object. Quartz2D, an advanced two-dimensional rendering engine used in iOS applications, has functions for obtaining the outline of an object. It also has functions that can be used to blend the overdraw with the object, thus making overdraw relatively easy to implement.

The disadvantage of using overdraws is that one side of each discontinuity edge is dilated by a pixel or two. Due to added processing, the application will also run slower than without overdraws at all. However, this approach succeeds in greatly reducing edge pixelation and improving rendering quality.

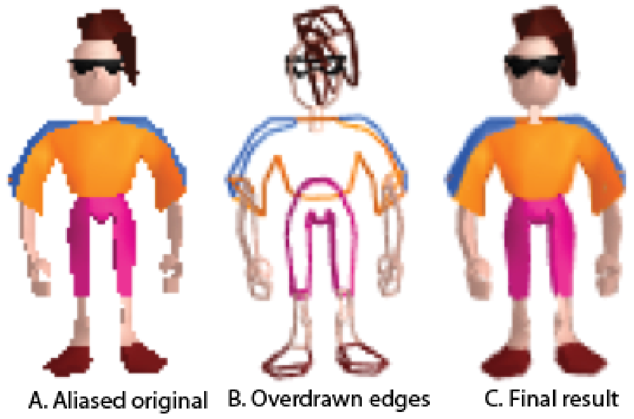

Figure 2. Images without and with overdraw[7]

D. Line Thinning

Line thinning is another aspect of a physical pen stroke. It is observed as an effect of the pressure and speed applied to the pen during writing. Since current Apple tablets do not support pressure sensing, this behaviour cannot be fully reproduced on an iPad. There is little existing documentation in the public domain on line thinning in drawing applications.

III. Smooth Writing Algorithm

A. Bézier Curves

As stated earlier, the iPad can only recognize stylus handwriting as a discrete set of points. It is the responsibility of the software to interconnect these points to produce a fluid curve. One approach is to connect the points using cubic Bézier curves. The cubic Bézier curve was chosen since it strikes a good balance between aesthetics and algorithmic speed. Higher degree Bézier curves produce more fluid-looking curves, but have more control points. Calculating a Bézier curve with many control points will decrease application responsiveness. Cubic Bézier curves still incur noticeable lag during processing, due to the fact that the algorithm requires four input points, two of which are endpoints and the other two control points. To increase perceived responsiveness, intermediate lines are rendered between each input point, and then replaced by the final Bézier curve when a total of four input points are detected by the iPad.

The following algorithm was used to create a smoothly flowing line with cubic Bézier curves.

1) The first point is detected and stored.
2) The second point is detected and connected to the first point with a straight line.
3) A third point is detected. The second and third points are connected.

4) When a fourth point is detected, the straight lines are discarded and a cubic Bézier curve is drawn.

Shown in Figure 3 is a graphical illustration of the algorithm.

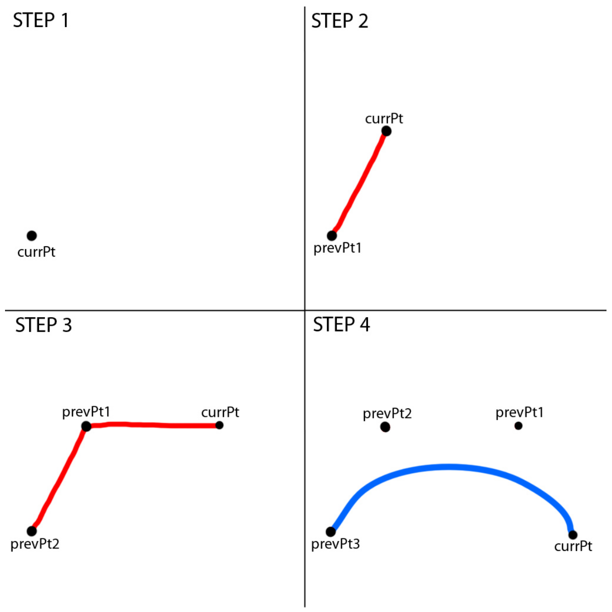

Figure 3. Cubic Bézier curve algorithm

Since the stroke does not necessarily have to pass through all four points, the second and third input points are used as control points for the cubic Bézier curve. The first and fourth input points are used as the start and end points, respectively, of the Bézier curve. However, simply connecting two separate Bézier curves will give the stroke a disjointed appearance at the junction points. This undesirable effect is due to the difference in slopes of the two connected Bézier curves as shown in Figure 4.

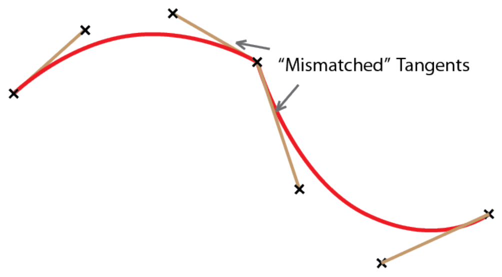

Figure 4. Disjointed cubic Bézier curves

To solve this, Khan suggested that the junction point must be adjusted such that the two Bézier curves appear to be one continuous curve (the slopes at the common point are equal)[8]. To do this, the junction point is re-calculated to be the midpoint between the second control point of the first Bézier curve and the first control point of the second Bézier curve. This assures that the three points are collinear. Shown in Figure 5 is a method to determine the new junction point of the Bézier curve.

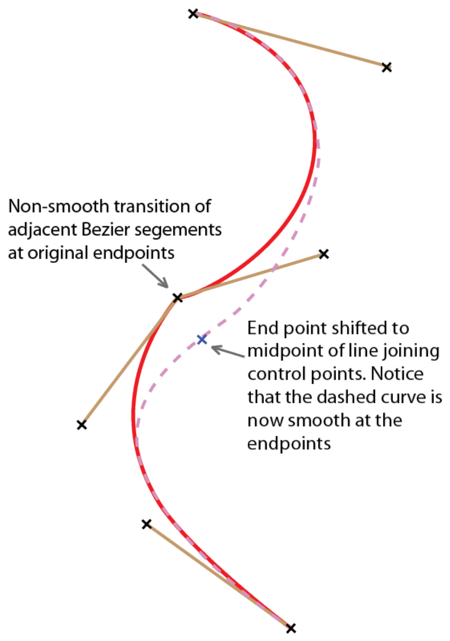
Non-smooth transition of adjacent Bezier segements at original endpoints
End point shifted to midpoint of line joining control points. Notice that the dashed curve is now smooth at the endpoints

Figure 5. Adjustment of the junction point of the two Bézier curves

B. Pen Width vs Speed Algorithm

The next step in achieving a natural pen-and-paper handwriting experience is to emulate pen-and-paper physics. Physics dictates that if the pen tip moves across the paper slowly, the pen width should be relatively constant. Furthermore, as the pen tip speed increases, the pen width should thin out.

For this effect, the application applies an algorithm that relates the speed of the stylus and the perceived stylus stroke width. The stylus speed is approximated by calculating the distance between the current detected location and previous detected location of the stylus (*strokeDist*) on the iPad display. A limiting value (*distLimit*) dictates the maximum distance between a pair of points for which line thinning has a perceivable effect. It should be noted that without this limit, the stroke width would thin out indefinitely. The algorithm also has a control mechanism based on the previous width (*prevPenWidth*) and a constant feedback factor (α). There is a set minimum for the stroke width that also takes into account the size of the overdraw. Due to the overdraw, which has a set width of two pixels, the change in stroke width is less noticeable. Therefore, the minimum stroke width was set to 10% of the maximum stroke width (*setPenWidth*). Other ratios like 20% and 40% were also tested, but because of the softening effect of overdraw, setting the percentage higher than 10% results in inperceivable the stroke thinning.

Based on the conditions stated, the pen width vs speed functions were derived as seen in Equations 3 and 4.

$$calcPenWidth = \alpha \left(\frac{distLimit - strokeDist}{distLimit} \right) (setPenWidth) + (1 - \alpha)(prevPenWidth) \tag{3}$$

$$calcPenWidth = MAX\left(calcPenWidth, 0.1\left(setPenWidth\right)\right) \quad (4)$$

As seen in equation 3, the first term of the right side of equation uses the current detected speed of the stylus to calculate the appropriate width of the pen. The last term serves as the feedback since it takes into account the previous width of the pen. To make things simple, α was set to 0.5 to perform arithmetic averaging. It is noted that further studies could be done to get a better value for α. The value of *distLimit* is set to a constant 70 pixels. Other values for *distLimit* were also tested, but it was observed that setting *distLimit* to lower than 70 made the pen width change too abruptly, and setting it higher than 70 made the pen width change so gradually that it was barely noticeable. Note that most of the calculations from the *distLimit* are based on subjective observations, and as such, further experiments involving stroke modeling could be performed to determine a better value for *distLimit*. Equation 4 ensures that the calculated pen width (*calcPenWidth*) does not go below 10% of the *setPenWidth*.

C. Polygon Strokes

Even with the stroke width v.s. stylus speed algorithm, the intersection between two strokes of different widths is still noticeable. To solve this problem, the polygon strokes algorithm was implemented. Polygons are interconnected with each other in such a way that there is a smooth transition between strokes. This solution was utilized by Zablocki[9] in his paint brush application. Shown in Figure 6 is the polygon he used in constructing the stroke.

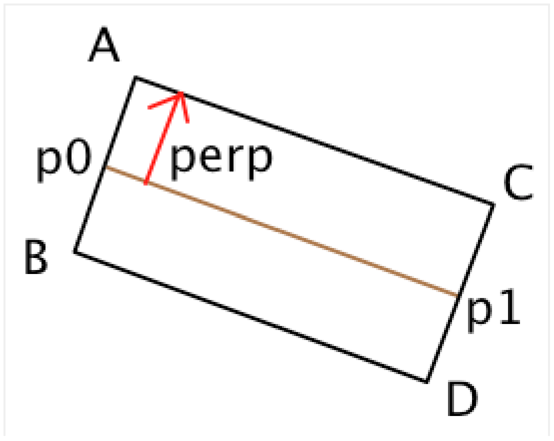

Figure 6. Polygon stroke method[9]

The points *p0* and *p1* represent the previous and current location of the stylus. The distance between these two points is proportional to the current speed of the stylus. This affects the shape of Figure 6 by relating the speed with the length of *AB* or *CD*. *AB* represents the previous width of the pen stroke while *CD* represents the current width of the stroke. These widths are related to the previous and current speed of the pen respectively.

This algorithm is sufficient for strokes that do not abruptly change direction. For strokes with significant differences in the angles between inter-connecting segments, the polygon stroke algorithm outputs strokes with noticeable discontinuity at the joints. To get around this, round caps were added to both ends of each segment making up a stroke. This technique provides an easy way to output rounded joints.

To approximate the Bézier curve polygon, the following steps were implemented:

1) Divide the cubic Bézier curve into several line segments. The longer the Bézier curve, the more segments are formed.
2) The difference between the previous pen width (*prevPenWidth*) and current pen width (*currPenWidth*) is divided by the number of segments formed in 1. This value will be the change in stroke width for each segment. The value will be negative if the stroke is thinning out, or positive if it thickening.
3) Draw the first line segment (containing the starting point) with its width equal to the previous width of the pen stroke.
4) Draw the succeeding segments with their widths equal to the width of the previously drawn segment, plus the change in the stroke width per segment.
5) Repeat this process until all segments are drawn. The last segment should have a stroke width equal to the calculated current stroke width.

Shown in Figure 7 is the step by step illustration on how the cubic Bézier polygon is formed.

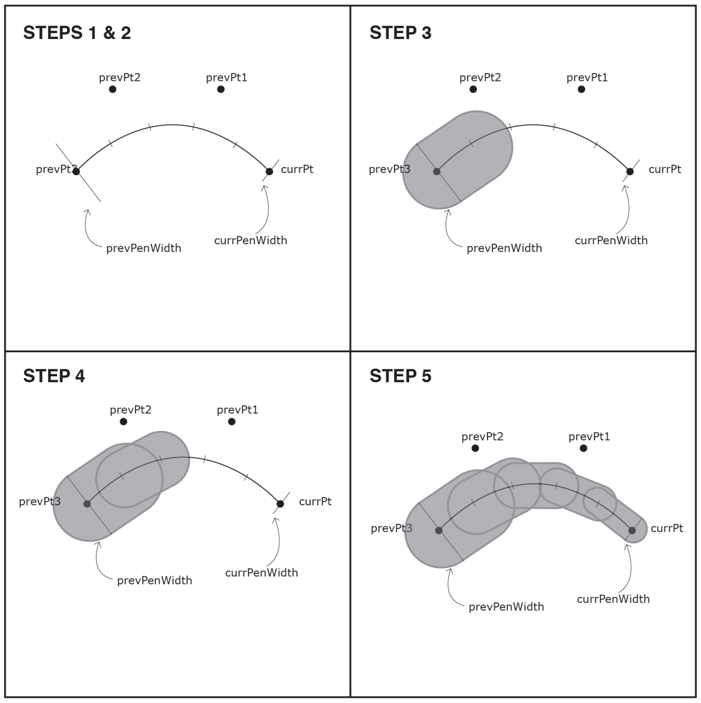

Figure 7. Interconnected polygons representing a stroke

D. Overdraws

One of the features of Quartz2D is that it can anti-alias objects in real time. This feature makes Quartz2D easy to use, since there is no need to write the algorithms for anti-aliasing from scratch. However, Quartz2D's form of anti-aliasing does not completely eliminate pixelation along line edges. Edge pixelation can be observed as staircase-like anomalies along the edges of a line. Since the strokes are opaque, this staircase pattern is noticeable. This effect creates undesirable visual jitters in the stroke.

The best way to eliminate pixelation is to hide it. In the iPad application, this is done by adding overdraws to the strokes in real-time. The overdraws are translucent overlays rendered around the edges of the stroke. These overlays make the color change from the center of the stroke to the stroke background more gradual, making pixelation less visible. During writing, two overdraw strokes are added: a one-pixel wide overdraw with 50% alpha and a two-pixel wide overdraw with 25% alpha. Shown below in Figure 8 are pen strokes without and with overdraw, respectively.

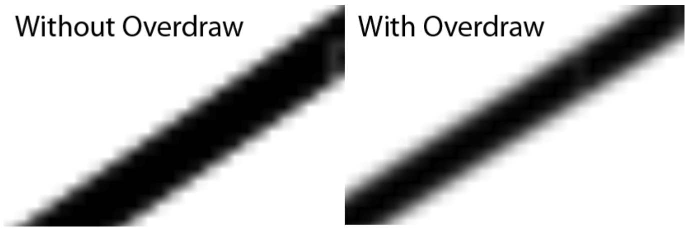

Figure 8. Without v.s. with overdraw

As seen in Figure 8 above, the edges of the stroke to the right is more blurred as compared to the one on the left. This is the effect of the overdraw algorithm. The visual jitters are less visible because the edges of the pen strokes are smeared out.

To verify the effect of overdraws, a row of pixels was obtained from each image, and then processed using the open-source program Octave. The color values were then plotted as shown in the Figure 9 below.

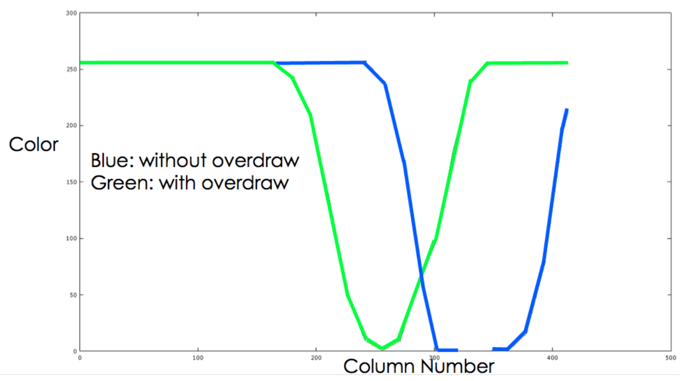

Figure 9. Color values plot

As seen in the graph, a stroke with overdraw has a more gradual change in color as compared to one without overdraw. This gradual change in color is what we see as the blur effect along the edges of a stroke.

IV. Experimental Results

To confirm that the application achieved its objectives, a comparative analysis was done. The application, named Writability, was tested compared against five commercially available note-taking applications, namely, Bamboo Paper[1], Notability[2], UPAD[5], Noteshelf[4], and Note Taker HD[3]. Penultimate[10], another popular iOS note-taking application, was not included in the analysis due to its lack of a highlighter tool. Testing was done on plain white virtual and real writing pads. Exactly thirty inviduals participated in the test.

For the initial part of the test, the testers were asked to draw a pattern using a real pen and paper. The pattern was then drawn using the five previously mentioned applications and Writeability. Shown in Figures 10 and 11 is the testing pattern for the pen and highlighter tool drawn using Writability.

For the virtualised pen tool, the the tester was asked to write all the alphanumeric characters, and then four lines in quick succession. The alphanumeric writing test was used since it is observed that most iPad users write notes using the English alphabet, while the line drawing test was performed in order to verify the ink thinning effect.

For the virtualised highlighter tool, four horizontal strokes, four vertical strokes, and a back and forth stroke were chosen as the test cases. The first two tests were used since highlighters are usually used to draw straight paths. The back and forth stroke was chosen as a test case to verify highlighter blending quality.

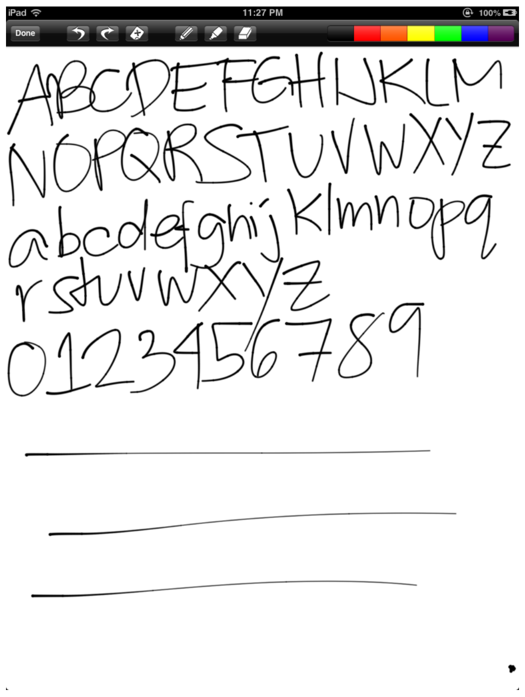

Figure 10. Pen testing pattern on Writability

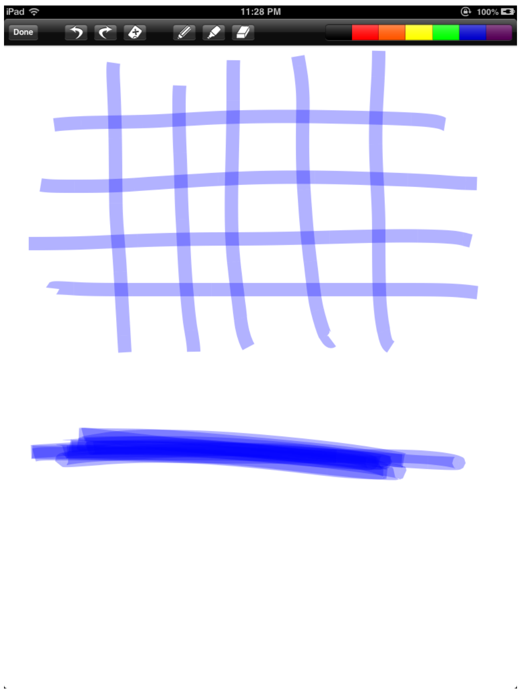

Figure 11. Highlighter testing pattern on Writability

After drawing the patterns in each application, the tester was asked to rate the application in different criteria, with ratings ranging from 1 - 10 (10 being the highest). The two features of Writeability that were tested were the pen and the highlighter tools. These criteria were further divided into different sub-criteria. Shown in Table I are the different criteria and their corresponding descriptions. A comment section was added at the end of the tester form for the tester's additional thoughts on his experiences with each application.

Table I. TESTING CRITERIA

Aspects	Criteria	Description
Pen	Responsiveness	Is there no lag between the stylus and the ink trail?
	Accuracy	How close is what is written on the tablet compared to what you intend to write?
	Smoothness of Curves	Are the curves not edgy (like straight lines joined together instead of a smooth curve)?
	Smoothness of Strokes	Are there stray pixels (zigzags) in the edges of the strokes?
	Thinning Effect	When you write fast, does it thin? Is the thinning realistic?
	Similarity to Real-Life	How similar is the pen tool compared to a real pen?
Highlighter	Responsiveness	How responsive is the app when you use the highlighter tool?
	Accuracy	How close is what is written on the tablet compared to what you intend to write?
	Smoothness of Curves	Are the curves not edgy (like straight lines joined together instead of a smooth curve)?
	Smoothness of Strokes	Are there stray pixels (zigzags) in the edges of the strokes?
	Similarity to Real-Life	How similar is it to a real highlighter?

Shown in Figures 12, 13, and 14 are the pen tool performance, highlighter tool performance, and total performance ratings, respectively.

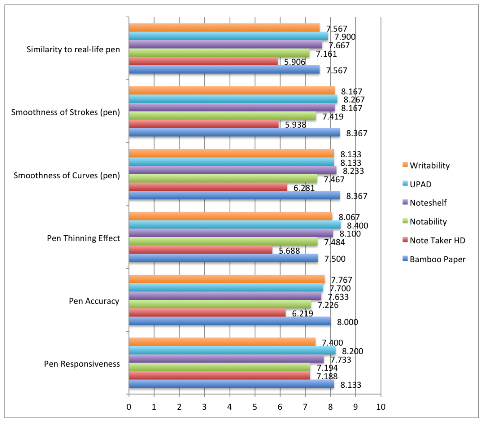

Figure 12. Pen Tool Rating Summary

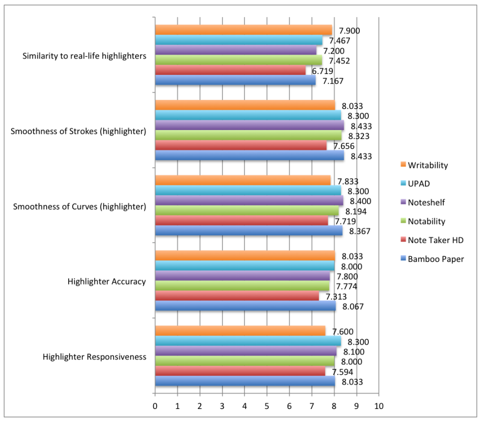

Figure 13. Highlighter Rating Summary

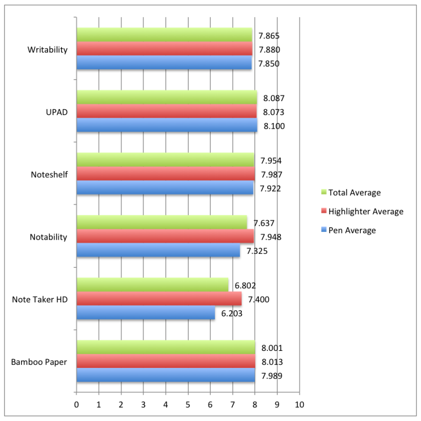

Figure 14. Total Summary

As seen in the Figure 12, Writability obtained high marks in terms of stroke performance. It was confirmed that the cubic Bézier curve algorithm was sufficient to create fluid strokes, while overdraws effectively eliminated pixelation. It also obtained a good score for natural ink thinning, because the strokes thinned out more naturally in comparison to other applications. In terms of the highlighter, Writability obtained the highest mark in terms of similarity to a real highlighter as shown in Figure 13. This is due to the fact that it was the only application that exhibited highlighter blending. For both the pen and highlighter, accuracy was high because Writeability did not render lines that the user did not intend to write. Based on the overall scores in Figure 14, Writability's writing tools was able to get fourth place out of the total of six that applications tested. However, as can be seen in the results, there is still room for improvement.

It was observed that both tools of Writeability were relatively slow to respond during writing. The perceived delay during the transition from straight lines to the final cubic Bézier curve was noticeable, resulting in low scores in terms of responsiveness. Also, the ink thinning effect of Writability was barely noticeable when the writing was small. This is because the length of the strokes in small writing are not long enough for the stroke thinning algorithm to produce a noticeable effect. This is a possible reason for the low ranking in terms of stroke thinning. With regards to the highlighter tool, it ranked only fifth in terms of smoothness, since there were visible anomalies in the highlighter strokes.

V. Conclusions and Recommendations

Based on the results of the testing, Writability fulfills its objective of providing a competitive natural pen-and-paper handwriting interface. The rendered strokes are fluid and stroke pixelation is not significant. It also has a natural ink thinning effect, and is able to simulate highlighter blending.

However, there is still room for improvement. The main flaw of Writability is that it does not strike the correct balance between responsiveness and aethetics. One solution to improve responsiveness during writing is to include a predictive algorithm to anticipate touch points. Another solution is to implement the same algorithms on the GPU using OpenGL ES, which should result and faster rendering.

In terms of natural ink thinning, the stroke width v.s. stylus speed algorithm could be made more context-sensitive in order to achieve ink thinning for small writing.

The highlighter tool could be further improved by removing the anomalies that appear in the strokes. This can be done by improving the polygon stroke algorithm directly, or using a transparency layer in a two-stage rendering process.

VI. Acknowledgements

This research is supported by the Office of the Vice Chancellor for Research and Development (OVCRD) of the University of the Philippines Diliman. Special thanks to Ms. Angelica Acierto for providing invaluable insight during the development of this project.

References

[1] Wacom. (2013, Feb.) Bamboo Paper. [Online]. Available: http://itunes.apple.com/us/app/bamboo-paper-notebook/id443131313?mt=8

[2] G. L. Inc. (2012, Dec.) Notability. [Online]. Available: https://itunes.apple.com/ph/app/notability-take-notes-annotate/id360593530?mt=8

[3] S. Garden. (2012, Oct.) Note Taker HD. [Online]. Available: http://itunes.apple.com/us/app/note-taker-hd/id366572045?mt=8

[4] R. Krishna. (2012, Oct.) Noteshelf. [Online]. Available: http://www.fluidtouch.biz/noteshelf/

[5] PockySoft. (2013, Jan.) Upad. [Online]. Available: https://itunes.apple.com/en/app/upad/id401643317?mt=8

[6] M. Kamermans. (2011, Dec.) Bezier curves - a primer. [Online]. Available: http://processingjs.nihongoresources.com/bezierinfo/

[7] P. Sander, H. Hoppe, J. Synder, and S. Gortler, "Discontinuity Edge Overdraw," in *Proceedings of the 2001 symposium on Interactive 3D Graphics*, 2001, pp. 167–174.

[8] A. Khan. (2012, Oct.) Smooth Freehand Drawing on iOS. [Online]. Available: http://mobile.tutsplus.com/tutorials/iphone/ios-sdk_freehand-drawing/

[9] K. Zablocki. (2012, Apr.) Drawing smooth lines with cocos2d ios inspired by paper. [Online]. Available: http://www.merowing.info/2012/04/drawing-smooth-lines-with-cocos2d-ios-inspired-by-paper/

[10] Evernote. (2013, May) Penultimate. [Online]. Available: http://itunes.apple.com/us/app/penultimate/id354098826?mt=8