CSY4010 Computing Dissertation	
Submission Date	24 April 2016
Title	Towards the development of Image processing and analysis techniques to influence creation of structured typefaces from handwritten text images.
Student Name	James Mitchell
Student ID	10425907

The University of Northampton's Policy on Plagiarism & Mitigating Circumstances will be strictly implemented.

By submitting this assignment you are asserting that this submission is entirely your own individual work.

Student Signature	James Mitchell - 24/04/2016
Assessment Details and Feedback	

CSY 4010 Computing Dissertation Project Report

Towards the development of Image processing and analysis techniques to influence creation of structured typefaces from handwritten text images.

James Mitchell

10425907

A Interim report submitted in fulfilment of the
requirements for the degree of

BSc (Honours) Computing (Software Engineering)

The University of Northampton.

2016

LIST OF FIGURES

Figure 1: Proposed Functionality [1.0]
Figure 2: Design Methodology and Implementation Strategy [1.3]
Figure 3: Research and Development Methodology [2.1.1]
Figure 4: Anatomy of Typography [2.1.1]
Figure 5: Main Window Controller Dependencies [3.2]
Figure 6: Tool Window Controller Dependencies [3.2]
Figure 7: UI Tool Window Sequence Diagram [3.2]
Figure 8: Screen Dimension Constraints [4.1]
Figure 9: UI Image Upload Default [4.2]
Figure 10: UI Finder Dialog [4.2]
Figure 11: UI Error Feedback [4.2]
Figure 12: UI Success feedback [4.2]
Figure 13: UI Image processing tools [4.3]
Figure 14: UI Letterform Comparison [4.4]
Figure 15: UI Saving output [4.5]
Figure 16: UI Multi stepped guided output [4.5]
Figure 17: UI Storyboard [4.6]
Figure 18: Image Acquisition [5.1.1]
Figure 19: Image Crop Result [5.1.2]
Figure 20: Original image and desired output from preprocessing [5.2]
Figure 21: Threshold function pseudo code [5.2.2.1]
Figure 22: Threshold function implementation [5.2.2.1]
Figure 23: Neighborhood offset coordinates [5.2.3]
Figure 24: Neighborhood offset Cartesian coordinates [5.2.3]
Figure 25: Spatial Filter iteration 1 [5.2.3]
Figure 26: Spatial Filter iteration 2 [5.2.3]
Figure 27: Simple averaging filter pseudo code [5.2.3.1.1]
Figure 28: Simple averaging filter implementation. [5.2.3.1.1]
Figure 29: Original Image Gray scaled [5.2.3.1.1]
Figure 30: 5 * 5 Simple Averaging Filter [5.2.3.1.1]
Figure 31: Thresholded 30% [5.2.3.1.1]
Figure 32: Weighted averaging filter [5.2.3.1.2]
Figure 33: Weighted Averaging filter pseudo code. [5.2.3.1.2]
Figure 34: Weighted Averaging filter implementation [5.2.3.1.2]
Figure 35: Median Filter Pseudo Code [5.2.3.1.3]
Figure 36: Median Filter Implementation. [5.2.3.1.3]
Figure 37: Original Image Gray scaled [5.2.3.1.3]
Figure 38: 9 * 9 Median Filter [5.2.3.1.3]

Figure 39: Filtered Image Thresholded 40% [5.2.3.1.3]
Figure 40: Min/max Filter Implementation [5.2.3.1.4]
Figure 41: Original Image Gray scaled [5.2.3.1.4]
Figure 42: 11 * 11 Max/Min Filter [5.2.3.1.4]
Figure 43: Max/Min Filter Thresholded 20% [5.2.3.1.4]
Figure 44: Broken lines of an object [5.2.4]
Figure 45: Lines reconnected using combination of a Max Filter and dilation [5.2.4]
Figure 46: The original subject. [5.2.4]
Figure 47: Data removed from subject [5.2.4]
Figure 48: Structuring element overlapping a object in the subject [5.2.4.2]
Figure 49: Dilation Pseudo Code [5.2.4.2]
Figure 50: Dilation Implementation [5.2.4.2]
Figure 51: Dilation subject [5.2.4.2]
Figure 52: Dilation Result 3x3 structuring element [5.2.4.1]
Figure 53: Erosion Pseudo Code [5.2.4.2]
Figure 54: Erosion Implementation [5.2.4.2]
Figure 55: Erosion subject [5.2.4.2]
Figure 56: Erosion Result 3x3 structuring element [5.2.4.2]
Figure 57: Opening Implementation [5.2.4.3]
Figure 58: Opening subject [5.2.4.3]
Figure 59: Opening Result [5.2.4.3]
Figure 60: Erosion [5.2.4.3]
Figure 61: Closing Implementation [5.2.4.3]
Figure 62: Closing subject [5.2.4.3]
Figure 63: Closing Result [5.2.4.3]
Figure 64: Attaining Borders [5.2.4.4]
Figure 65: Image Difference [5.2.4.4]
Figure 66: Difference polarity switched [5.2.4.4]
Figure 67: Switching Polarity Implementation [5.2.4.4]
Figure 68: Comparison of CPU time (in seconds) Consumed by Different Parallel Thinning Algorithms. [5.2.4.5]
Figure 69: Zhang Suen outer loop [5.2.4.5.1]
Figure 70: Zhang Suen sub iteration 1 [5.2.4.5.1]
Figure 71: Zhang Suen sub iteration 2 [5.2.4.5.1]
Figure 72: Thinning input [5.2.4.5.1]
Figure 73: Thinning output [5.2.4.5.1]
Figure 74: Thinning output with new condition [5.2.4.5.1]
Figure 75: Subject [5.2.4.5.1]
Figure 76: Subject thinned with original condition [5.2.4.5.1]
Figure 77: Subject thinned with amended condition [5.2.4.5.1]
Figure 78: Moore Neighborhood [5.3.1.1]

Figure 79: Moore Neighbor Algorithm interpretation. [5.3.1.1]
Figure 80: Working of Moore's Neighbor tracing algorithm. [5.3.1.1]
Figure 81: Moore Neighbor Pseudo Code [5.3.1.1]
Figure 82: Moore Neighbor Implementation [5.3.1.1]
Figure 83: Moore Neighbor Implementation Continued [5.3.1.1]
Figure 84: Moore Neighbor Trace Loop Implementation
Figure 85: Reducing to distinct points.
Figure 86: Starting letterform subject
Figure 87: Thinned subject
Figure 88: Traced result
Figure 89: Thinned subject
Figure 90: Trace Failure Explanation
Figure 91: Projection profile pseudo code
Figure 92: Projection profile Implementation
Figure 93: Result of projection profile
Figure 94: Overlapping lines
Figure 95: Projection profile for analysis
Figure 96: User selecting an area to crop.
Figure 97: The cropped region.
Figure 98: (l-r) Subject Image, Averaging, Median, Min, Max filters
Figure 99: (l-r) Subject Image, Averaging, Median, Min, Max filters thresholded
Figure 100: (l-r) Subject Image, Erosion, Dilation, Opening, Closing
Figure 101: Broken Image.
Figure 102: Dilated Broken Image.
Figure 103: Thinning Evaluation.
Figure 104: Path Generated By PaintCode.
Figure 105: Contour Traced Border
Figure 106: Contour Traced Thinned Image
Figure 107: Projection Profile Line Segmentation
Figure 108: Projection Profile Letterform Analysis
Figure 109: Demonstration UI.

TABLE OF CONTENTS

1. Introduction.
    1.1. Background.
    1.1.1 Initial Research and Literature Review.
    1.1.2 Comparable Systems.
    1.1.3 Relevance.
    1.2 Project Aims and Objectives.
    1.2.1 Aims.
    1.2.2 Objectives.
    1.3 Methodology.
    1.3.1 Analysis and Design.
    1.3.2 System Design.
2. Requirements Engineering.
    2.1 Background Reading and Research.
    2.2 Comparable Products.
3. System Design.
4. Interface Design.
    4.1 Screen Dimension Constraints
    4.2 Image Upload.
    4.3 Image Processing tools
    4.4 Output presentation
    4.5 Saving output.
    4.6 UI Storyboard.
5. Implementation.
    5.1. Image Data Acquisition.
    5.2. Image Processing.
    5.2.1. Image Processing Considerations.
    5.2.2. Point Operations.
    5.2.3. Spatial Filters.
    5.2.4. Morphology.
6. Evaluation.
7. References.
8. Appendix.
    8.1 Source Code.
    8.2 Viva Presentation.

1. INTRODUCTION.

Personal computers provide users with multiple ways in which to create and share documents. Be it the writing of letters or emails, creation of presentations, and designing for print or the web. Each of these activities involve the use of one core element - typography. A building block of which is the typeface.

The creation of documents for the majority of users will be provided via the use of word processors, presentation tools, email clients and graphic design tools. These types applications all provide a limited set of of typefaces available to allow the user to personalise and distinguish their documents from that of others. Although the web provides services to buy and install fonts, these can be expensive, geared toward the professional graphic designer or web designer, and with near endless choice a time consuming venture.

The process of designing and implementing a typeface has a high bar for entry, takes many hours to create and many more hours to master. Furthermore type development is not a skill available to all, despite artistic ability or intent. This fact takes the potential for complete personal and visual expression away from even the most creative users.

Although tools do exist that remove an element of the required skill set, these range from the higher skill level in the case of fontself, through a template manipulation web tool called prototypo, to the lower end my-script-font. (see Comparable Systems). Despite the quality of these tools they fail to provide both a lower bar of entry and representative, personalised output.

The intention of this project is to design and implement an application that will lower the bar for typeface font creation taking the essence of a persons handwritten text in order to provide personalised and expressive fonts for the use of document creation.

Through the analysis of handwritten letterforms provided by the user in the form of a bitmap image, these images will be processed in order to remove noise, colour, and unwanted none-letterform artefacts/objects of the provided bitmap. On removal of unwanted data from the bitmap, this will be thresholded to produce a binary representation of the letterforms. It is then the intention to create a tool set of techniques in order to identify features of the handwritten letterforms such as: x- height, ascender and descender length, crossbar heights, counter size and shape, cursive exit and entry heights, and letter widths.

Information gleaned from the letterform analysis will then be applied to letterform templates to be transformed and translated to represent features of the input handwritten text.

Figure 1 - Proposed Functionality

1.2 PRODUCT AIMS AND OBJECTIVES.

1.2.1 AIMS.

The core aims of this project are: 1. Design and implement an application that will lower the bar of the average user to create typeface fonts. 2. Provide a simple, user-friendly and intuitive user interface.

In order to achieve the core aims the application will need to:

1. Receive input of handwritten text images.
2. Perform image processing and analysis techniques to influence output.
3. Compose typeface fonts influenced by image processing and analysis.
4. Provide formatted output for installation and/or sharing.

1.2.2 OBJECTIVES.

These four points provide natural segmentation into four stages of development:
1. Image Processing and Analysis.
2. Letterform Composition.
3. Font Formatting.
4. User Interface Development.

The project objectives allowing for achievement of these aims will be:

1.2.2.1 IMAGE PROCESSING.

1. Gain insight Image processing techniques including.
   1.1 Image Acquisition, Image preprocessing, Image Enhancement, Image Segmentation, Object recognition, Character Recognition.
2. Compile list of typographic features required for extraction from handwritten text images.
3. Analise image processing techniques best suited for extraction of typographic features from handwritten text images.
4. Collate, deconstruct and analyse required image processing algorithms to produce pseudo code.
5. Evaluate suitability of methods, testing algorithms with sample images. 6. Implement and test.

1.2.2.2 LETTERFORM COMPOSITION.

1. Understand, design, develop algorithms for creation and manipulation of bezier curves.
2. Develop letterform templates required for manipulation / translation.

1.2.2.3 FORMATTED OUTPUT.

1. Develop methods to convert manipulated templates to SVG format for output.

1.2.2.4 USER INTERFACE.

1. Create Wireframes and storyboards for various states of the user interface. 2. Develop prototype user interface to finalise state transitions, UI copy, and UI design.
2. Implement and testing.

1.3 METHODOLOGY.

Due to the nature and scale of this project it has not been possible to adopt a traditional waterfall methodology of collecting requirements and defining a specification before progressing to development.

Development of this project requires small iterations of: Research, Design, Implementation and Testing.

TODO: SOMETHING ABOUT AGILE!

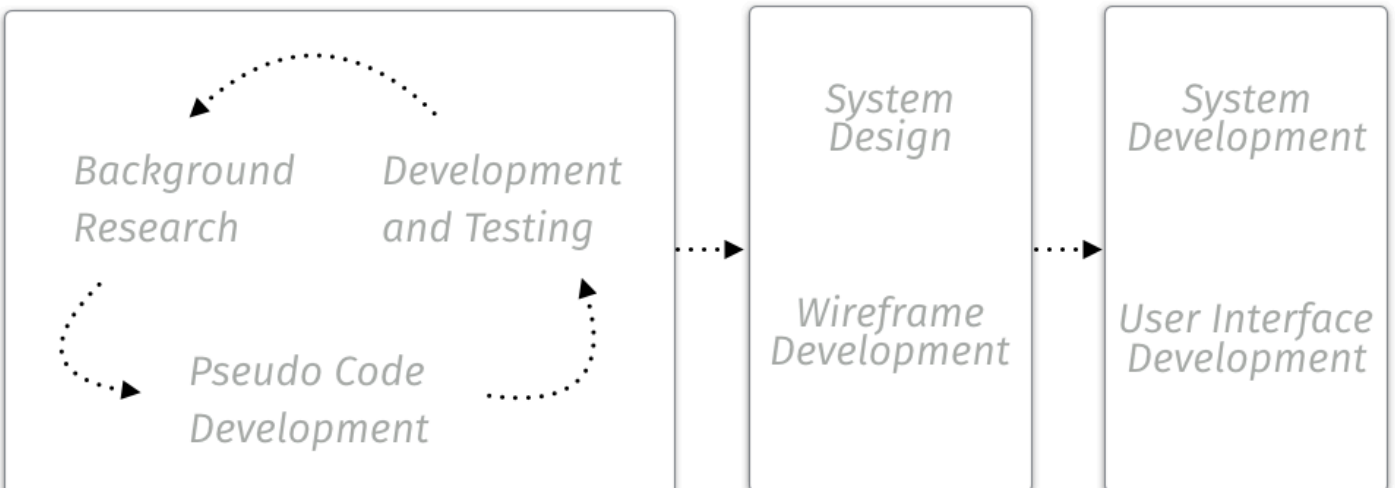


Figure 2: Design Methodology and Implementation Strategy

This practice will provide: -

A short feedback loop allowing for a more adaptive process. - Modular development of functionality allowing for extendibility or expansion.

An adaptive process is required as some image processing, analysis, and object recognition techniques are better suited to the the requirements of this project then others. Although initial research has been carried out to assume suitability of techniques, due to the high level of data contained within images and variety of sometimes random information received during the capture of digital images, testing and feedback may show poor suitability. In which case alternative methods will need to be sought.

Once the suitable collection of processing, analysis, recognition, and letterform construction tools has been finalised, a final system design can be actioned coupling together technologies and the user interface.

1.3.1 ANALYSIS AND DESIGN.

As noted in as the start of this section development of this project will follow the short cycle of:
- Background research,
- Pseudo code development,
- Implementation and testing.

Background research into techniques has been performed to determine suitability of image processing and analysis techniques. Where suitability is assumed correct formula and expressions provided are deconstructed into pseudo code before being implemented, tested, and analysed.

1.3.2 SYSTEM DESIGN.

Once the function of the system is complete the requirement will be to pull each tool set together and integrate into the User Interface. UML diagrams will provide the description of the system.

Unified Modeling Language (UML) will provide a significant tool set during the system design stage. UML offers documented standards allowing for a common language throughout the industry. As a result of standardisation and industry strength, supporting documentation for the use of UML is in abundance.

1.3.3 SYSTEM IMPLEMENTATION.

The final application of this project will be developed for installation on Mac OS X.

The default language of choice for OS X is Objective-C supported by the Cocoa Framework. The use of Objective-C and Cocoa will ensure this application will be native to the environment and be performant due to being a language and framework optimised for the system the application will run on.

In addition the Cocoa framework provides the Core Image interface potentially aiding the the Image Processing aspect with the functionality that Core Image provides in addition providing GPU rending if required.

Being a strict superset of the C programming language, Objective-C provides lower level control over memory management that could be important when dealing with resource hungry operations involved with image processing and analysis.

Development of the source code and UI will be undertaken within Xcode 7 development environment.

1.3.4 SYSTEM EVALUATION.

Due to the potentially subjective nature of whether or not an output typeface font is representative of the input handwritten text image, measurable evaluation will of success will be difficult. Therefore, should a successful realisation of this project occur, tests will be designed that presents testers with a sample of handwritten text along with a number of potential output font examples with only one example being the true representation. Success will therefore be measured on the testers ability to recognise a font from a handwritten example.

2.1 BACKGROUND READING.

Reading and research into the application to be implemented in this project will largely be undertaken iteratively as required. Sources have been gathered preemptively to ensure availability and allowing for timely and reactive cross reference. The following resources have been gathered:

IMAGE PROCESSING.

- Digital Image Processing - (Gonzalez and Woods, 2002)
- Digital Image Processing - (Castleman, 1995)
- Principles of digital image processing: Fundamental techniques - (Burger and Burge, 2014a)
- Principles of digital image processing: Core Algorthms - (Burger and Burge, 2014b)
- Principles of digital image processing: Advanced Methods - (Burger and Burge, 2014c)

TYPOGRAPHY AND TYPE DESIGN.

- Counterpunch - (Smeijers, 1996)
- Stop Stealing Sheep & find out how type works - (Spiekermann, 2013)
- The Geometry of Type - (Coles and Spiekermann, 2013)

2.1.1 INITIAL RESEARCH.

Initial background reading has been undertaken in order to understand the collection of image processing tools that will be required to start the implementation of this project. The core resource of this background reading has been Digital Image Processing (Gonzalez and Woods, 2002).

Through quick inquiry into this resource the following applicable chapters and sections have been singled out for deeper analysis along with the construction of figure 3 highlighting tool sets required for research and development in order to achieve the required letter form features. Figure 4 provides reference of typeface attributes detailed.

- 2 Digital Image Fundamentals.
    - 2.5 Some Basic Relationships Between Pixels.
- 3 Image Enhancement in the spatial Domain.
    - 3.3 Histogram Processing.
    - 3.6 Smoothing Spatial Filters.
    - 3.7 Sharpening Spatial Filters.
- 9 Morphological Image Processing.
    - 9.2 Dilation and Erosion.
    - 9.3 Opening and Closing.
- 10 Image Segmentation.
    - 10.1 Detection of Discontinuities.

    - 10.3 Thresholding
    - 10.5 Segmentation by Morphological Watersheds.
- 12 Object Recognition.
    - 12.2 Recognition Based on Decision-Theoretic Methods.
    - 12.3 Structural Methods.

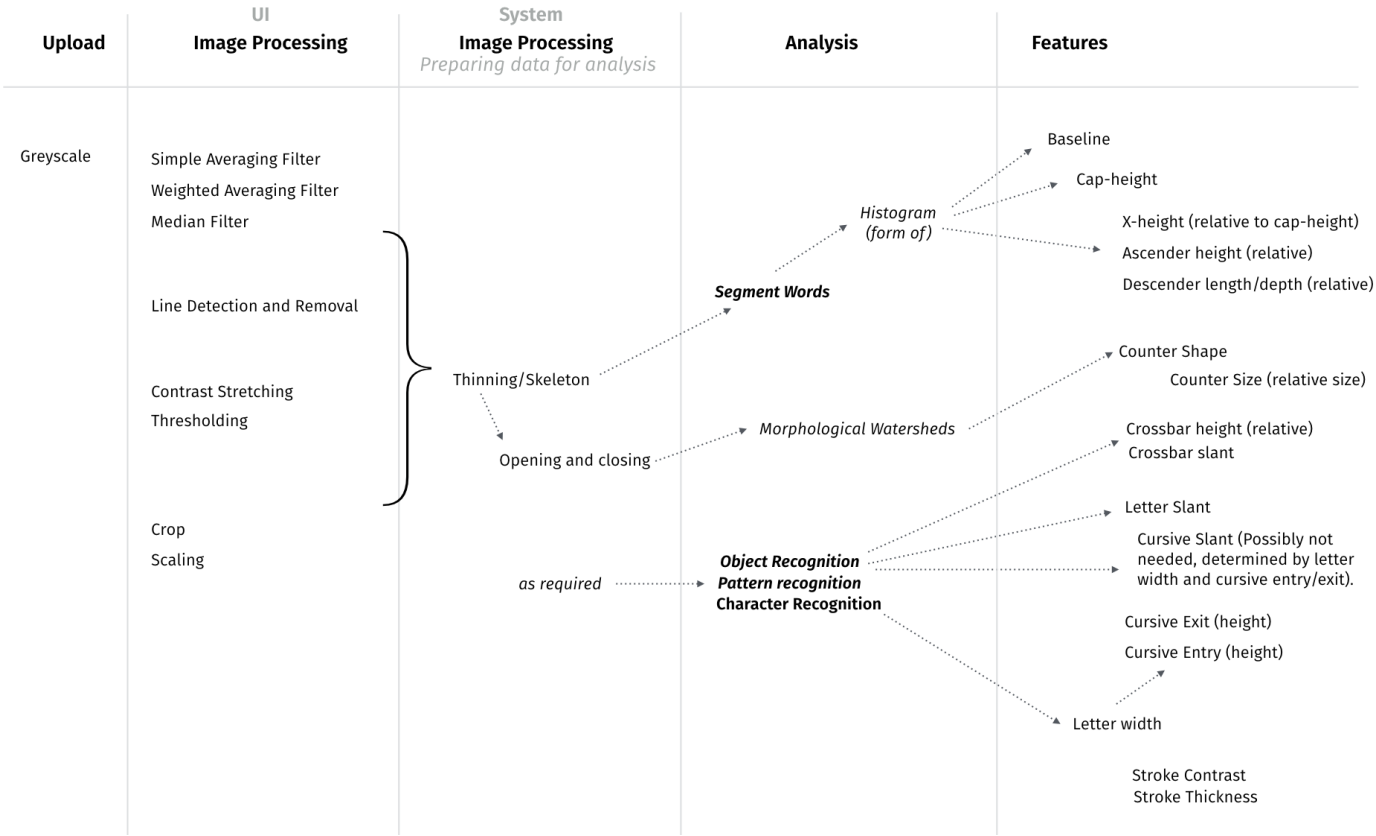


Figure 3: Research and Development Methodology

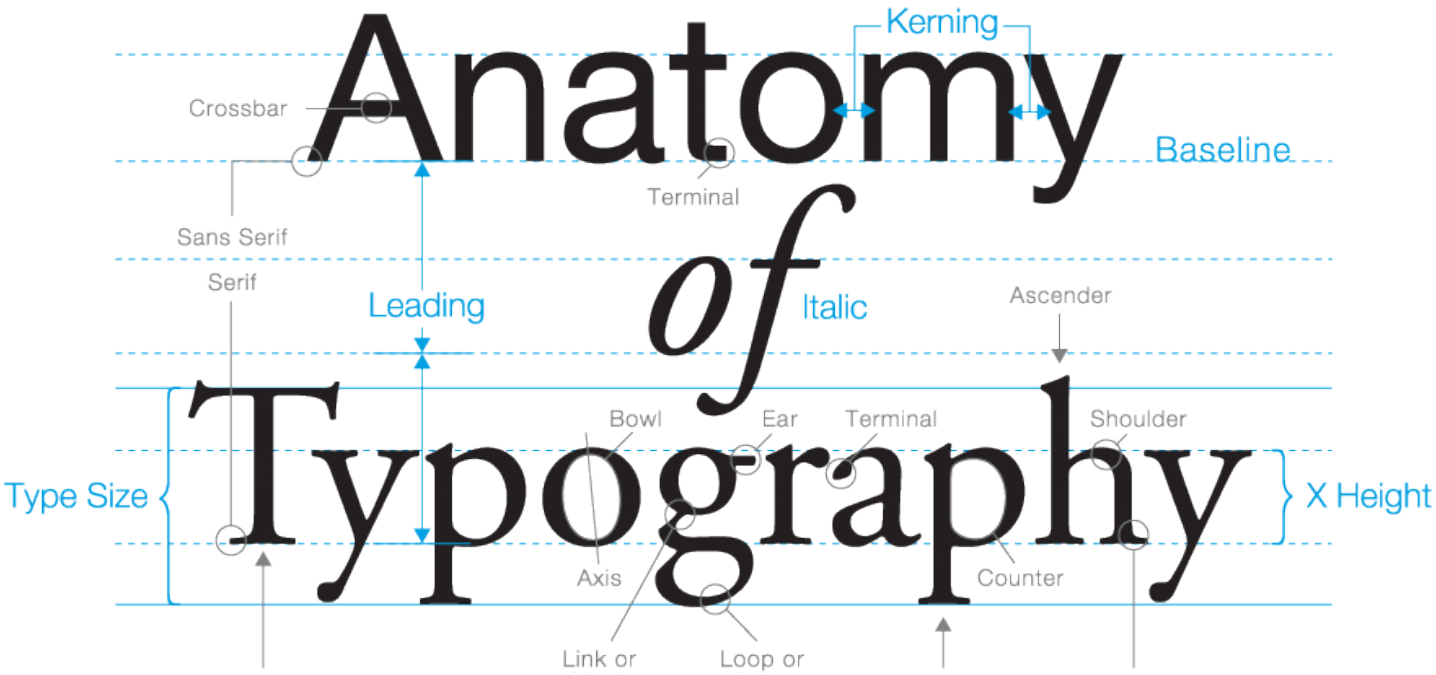


Figure 4: Anatomy of Typography (Designer Insights, 2012)

2.2 COMPARABLE PRODUCTS.

Inquiry into comparable systems has taken place with features and functionality being reviewed.

FONTSELF.

PLATFORM: ADOBE PHOTOSHOP/ILLUSTRATOR

- Requires the creation of each individual character.
- Aimed at high end professional letterers and graphic designers.
- Colour fonts can be created.
- Additional feature being designed called Fontself Catapult that will allow users to distribute fonts to market places such as MyFonts, Google Fonts, Font Squirrel, Typekit and Font Deck.
- Hand lettering can be scanned in and vectorised by Photoshop or Illustrator or created within Photoshop or Illustrator.

PROCESSES:

- Each prepared character is selected.
- The plug-in is run.
- Fontself creates each glyph and provides option to install font.

PROTOTYPO.

PLATFORM: WEB

Prototypo is a web application that allow for the creation of fonts from a provided template. Currently in beta, prototypo provides to typeface variants, Sans-Serif and Serif. With a font variant selected the user can selected individual glyphs from the template font set and type a string of text to allow for feedback when editing the selected glyph. Parameters are provided to allow for the manipulation of attributes for the selected glyphs, these parameters are split into three groups structural, style, and serif. While each attribute is manipulated, real time feedback is provided through the user provided phrase.

The structural attributes that can be changed include: X Height, Capital height, Ascender height, Descender length, Crossbar height, Character width, Slant, and Overshoot.

The style attributes that can be changed include: Thickness, Contrast, Extremity, Aperture, Top, Bottom, Curviness, and Optic Thickness. Serif manipulation includes: Width, Height, Curve, Arc, Terminal, Rotation."

MYSCRIPTFONT

PLATFORM: WEB

My Script font is an web application that provides an output font from an uploaded template image. The template requires the drawing of each individual letter form into the relative box of the template. Once complete the image is scanned by the use preferably in grey scale and uploaded to the My Script Font site with a jpg, png, pdf, jpeg, or tiff extension and of maximum 6 MB file size and maximum image dimension of 6500 x 6500 px. The font is then named by the user and output font format chosen. Once downloaded the created font can be installed on the local machine.

SUMMARY.

Each of the three tools reviewed provide interesting application for typeface creation each with varying required skill and quality of output. The high skill requirement of fontself takes this out of the average users range and the low bar for entry my-script-font requires a time-consuming, un-intuitive method of input and unrepresentative output. Prototypo carries a low skill requirement and providing generally good output however does not create a presentation of a written text.

3. SYSTEM DESIGN.

3.1 INITIAL SYSTEM DESIGN.

As this project does not follow a traditional software development methodology requiring research and experimentation during development a full system design cannot yet be determined.

However, to ensure control over the system architecture is maintained the following simple rules and guidelines have been determined.

1. Maintain a separation of Model, View, and Controller concerns.
2. Group concerns. Therefore, Image Processing methods will share a class, Image Analysis methods will share a class.
3. Where a concern forms its own group within a concern, this will create its own class. e.g. Morphology is an image processing toolkit, however presents a large sub set to make its own class.
4. Where a documented algorithm has been used this will create its own class. i.e. Thinning can be described as a Morphological process, however should many thinning algorithms be implemented the Morphology class would contain many “thin” methods. Therefore, each thinning algorithm would instead follow a thinning protocol. (Protocols are ~equivalent to interfaces in Java, therefore if this where a Java application, each thinning algorithm would implement a thinning interface).

3.2 FINAL SYSTEM DESIGN.

Figure . documents the implementation of the user interface stating the dependencies realised by the `MainWindowController`. The `MainWindowController` deals with the presentation of state dependent views as its primary concern.

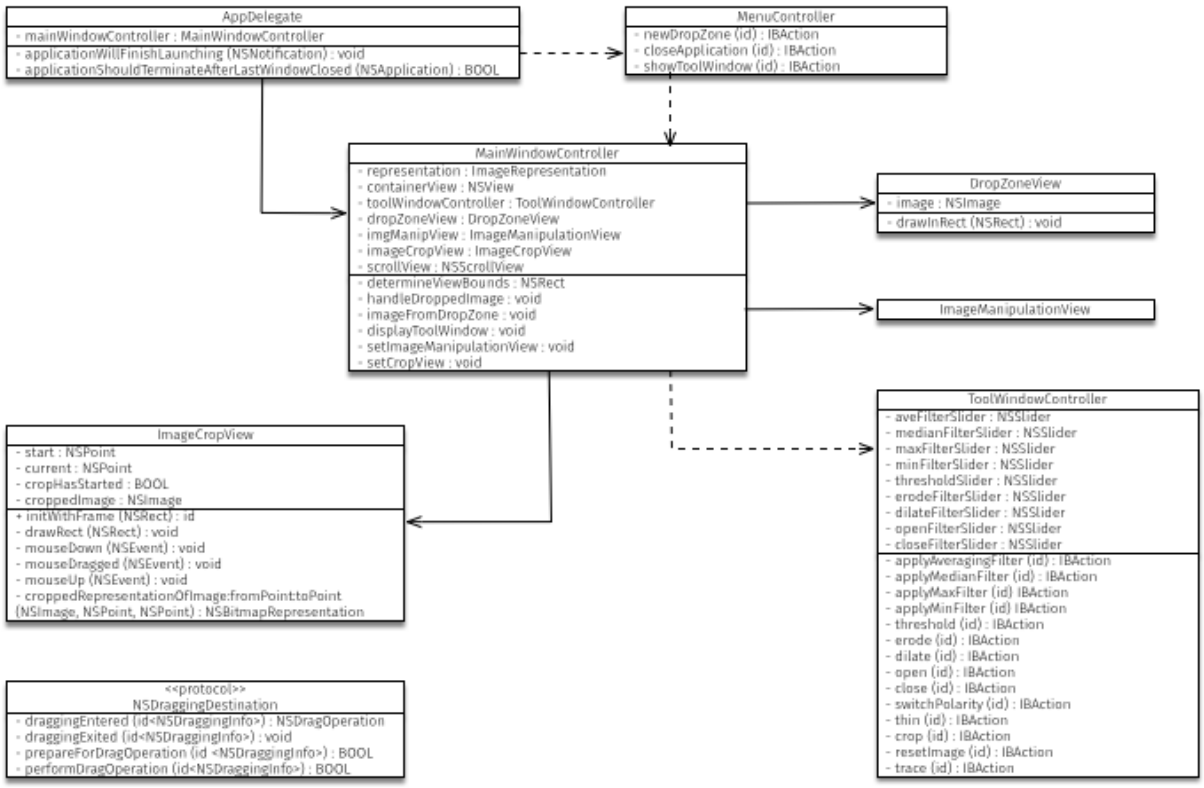


Figure 5: Main Window Controller Dependencies

Figure . details the implementation of the `ToolWindowController`. The primary function of the `ToolWindowController` is taking user action and calling the relevant processes.

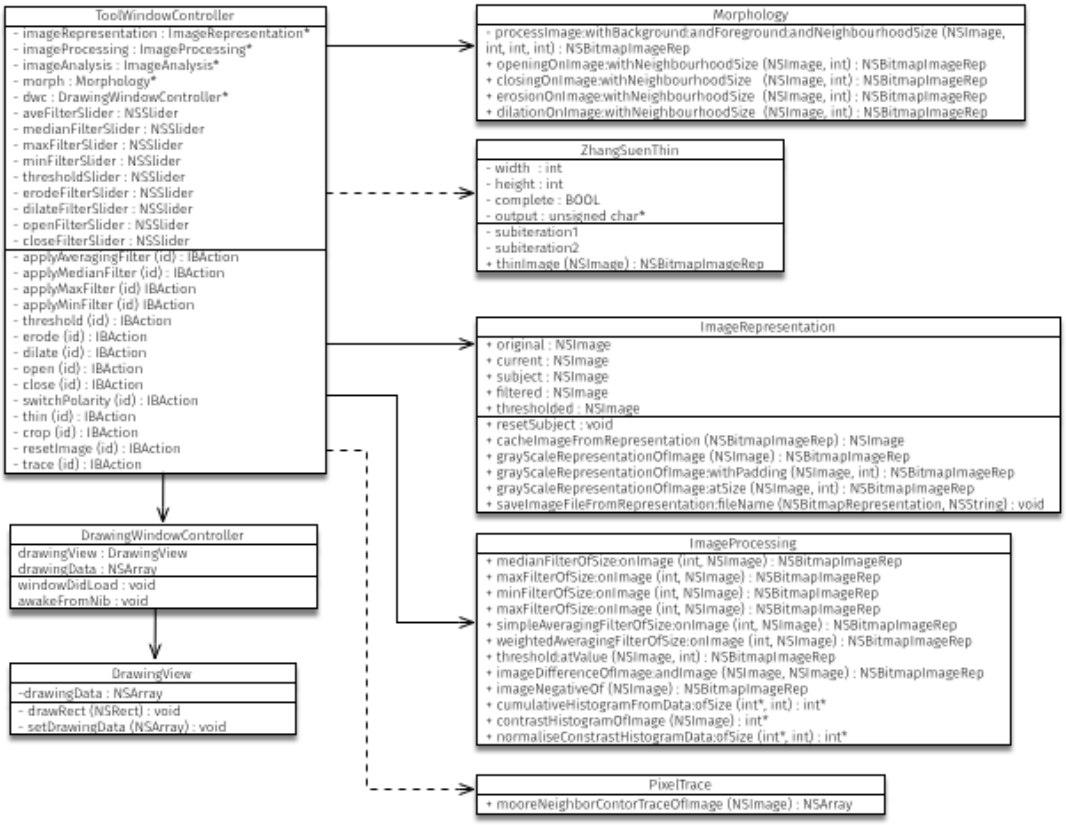


Figure 6: Tool Window Controller Dependencies

A UML Sequence Diagram has been constructed documenting the interaction between classes when dealing with user interaction. Figure .

Cocoa provides the `NSNotificationCenter` classes allowing for the sending of messages between classes (Apple, 2009). Use of this functionality has been considered for the design of this system to remove hard dependencies between window and view objects.

Equivalent to an implementation the Observer Design Pattern (Gamma et al., 1995), `NSNotificationCentre` provides the interface to create observers to be notified as a reaction to an event. For the system design observers set in `MainWindowController` are notified by `DropZoneView`, `ImageCropView`, and `ToolWindowController` as a result of user action. When notified of an event `MainWindowController` can act on the event. Significant to the design of this system, the implementation of observers means the three notifying classes do not need to maintain reference to `MainWindowController` removing dependencies between the classes. Therefore, should changes be made and the `MainWindowController` is replaced by a different controller the notifying classes are not required to change.

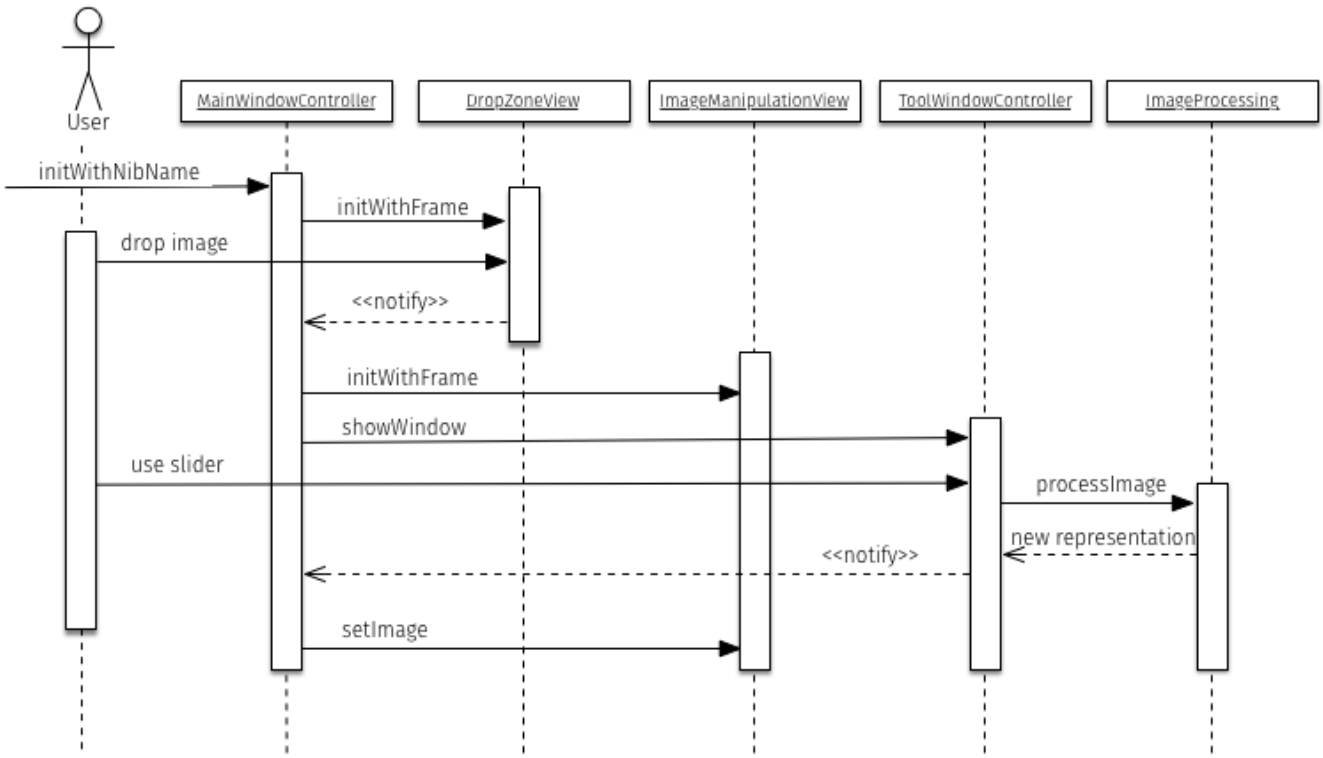


Figure 7: UI Tool Window Sequence Diagram

3.MainWindowDep.png (Figure 5: Main Window Controller Dependencies) [805x536]

4. INTERFACE DESIGN.

Initial interface designs have been completed in order to determine the look, feel, layout and minor interaction details required for users to complete the creation of their typeface.

4.1 SCREEN DIMENSION CONSTRAINTS.

Prior to the creation of detailed wire-frames the screen size constraints have been determined. Early versions of this application will be targeted towards MAC OSX, therefore to ensure all users from the range of Apples OSX devices the screen size of the smallest device, 11in Macbook Air, have been applied as the base constraint. All subsequent designs will therefore be required to fit these boundaries. The constraint values have been provided by Sketch.app (Bohemian Coding, 2016) which will generate a predefined “Artboard”.

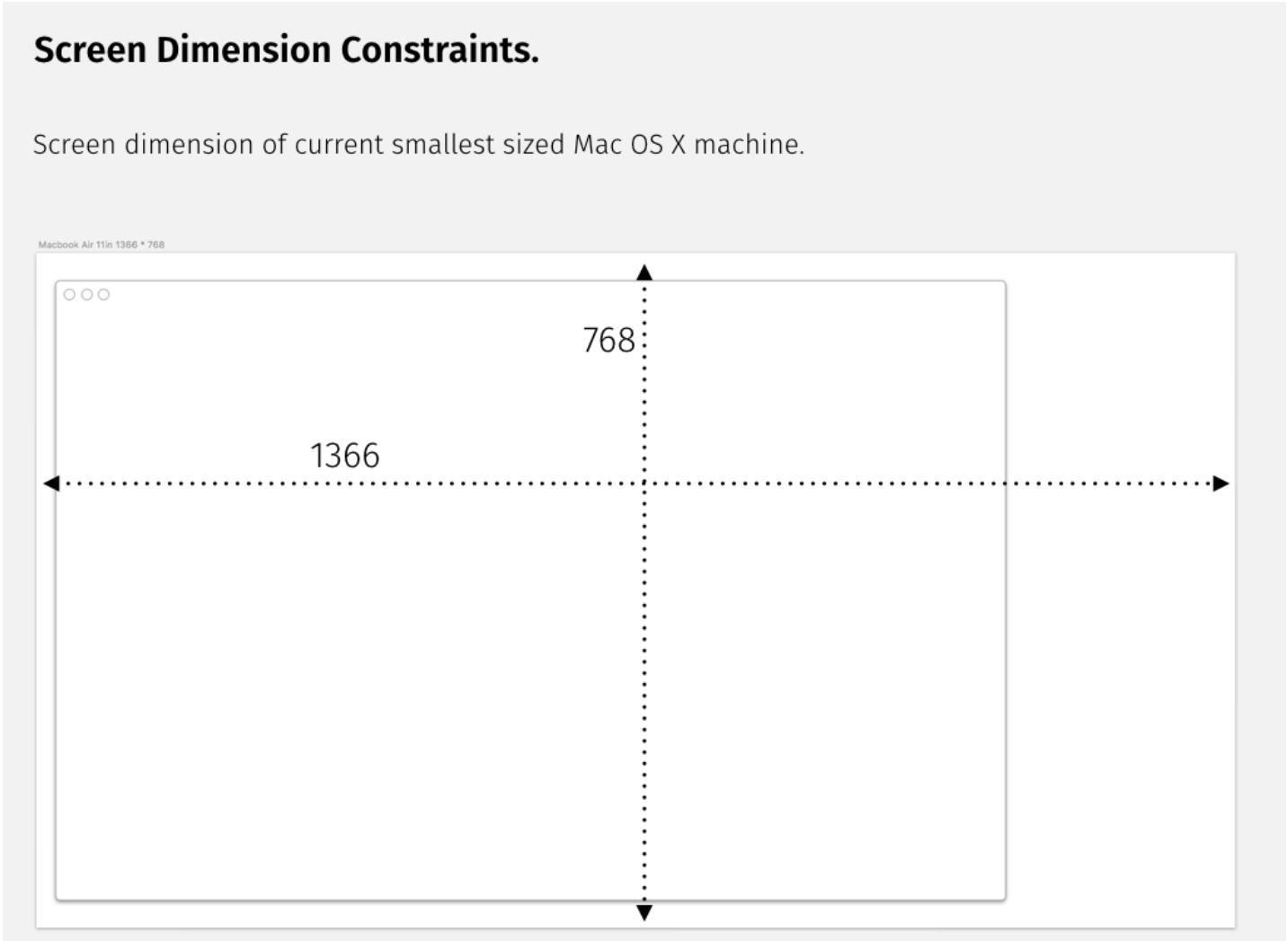


Figure 8: Screen Dimension Constraints

4.2 IMAGE UPLOAD.

The primary method of subject image upload will be through dropping an image onto the application window, simplifying the process. However, generally the interaction of drag and drop is favorable when items are dropped from the desktop or a least the viewable screen. Therefore, alternatives should been included. As Figure . there use of a Finder dialog has been considered. In addition future versions of this application could offer cloud support allowing from the taking of pictures from a device, sent to cloud and pulled into the desktop application.

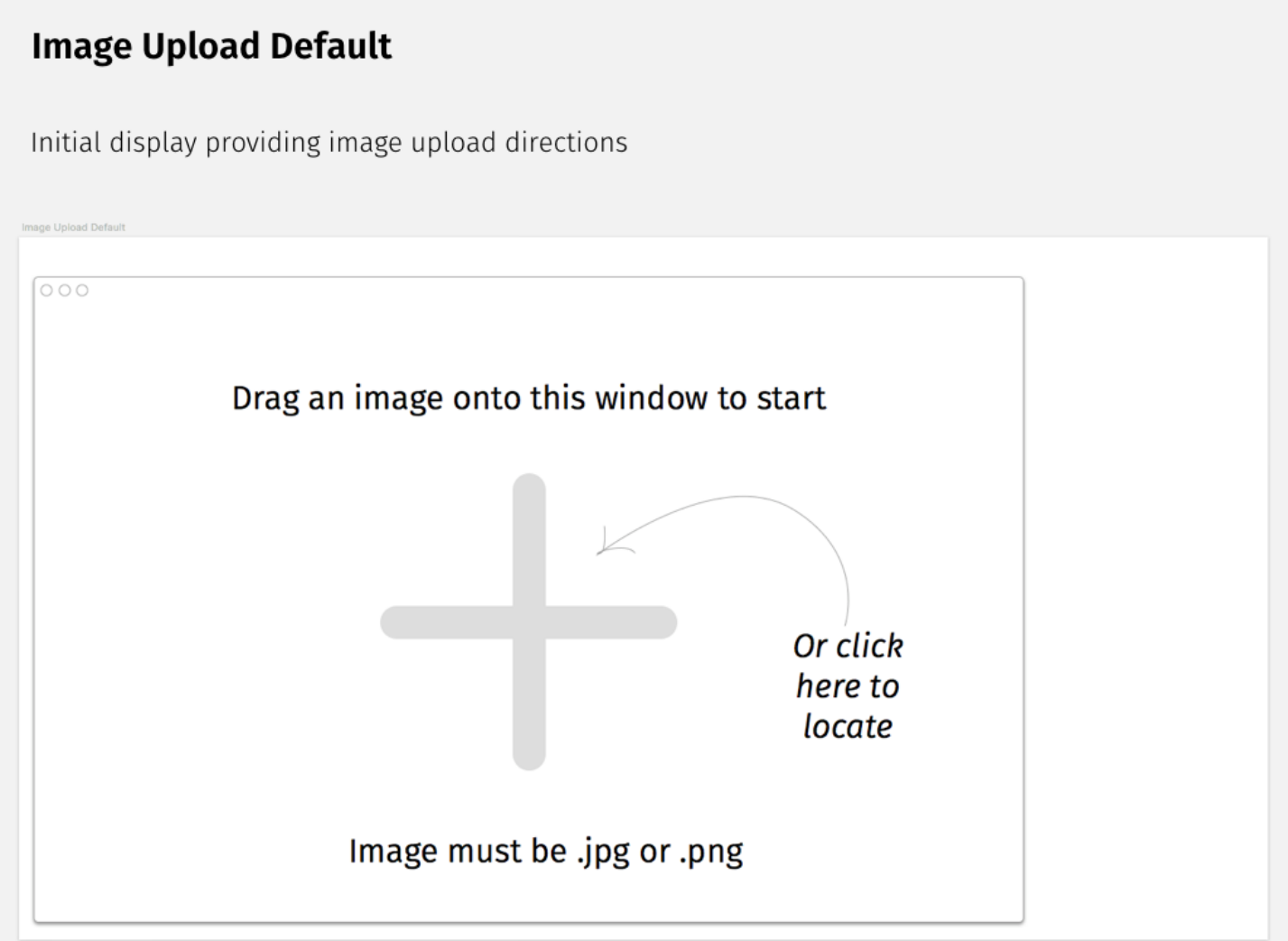


Figure 9: UI Image Upload Default

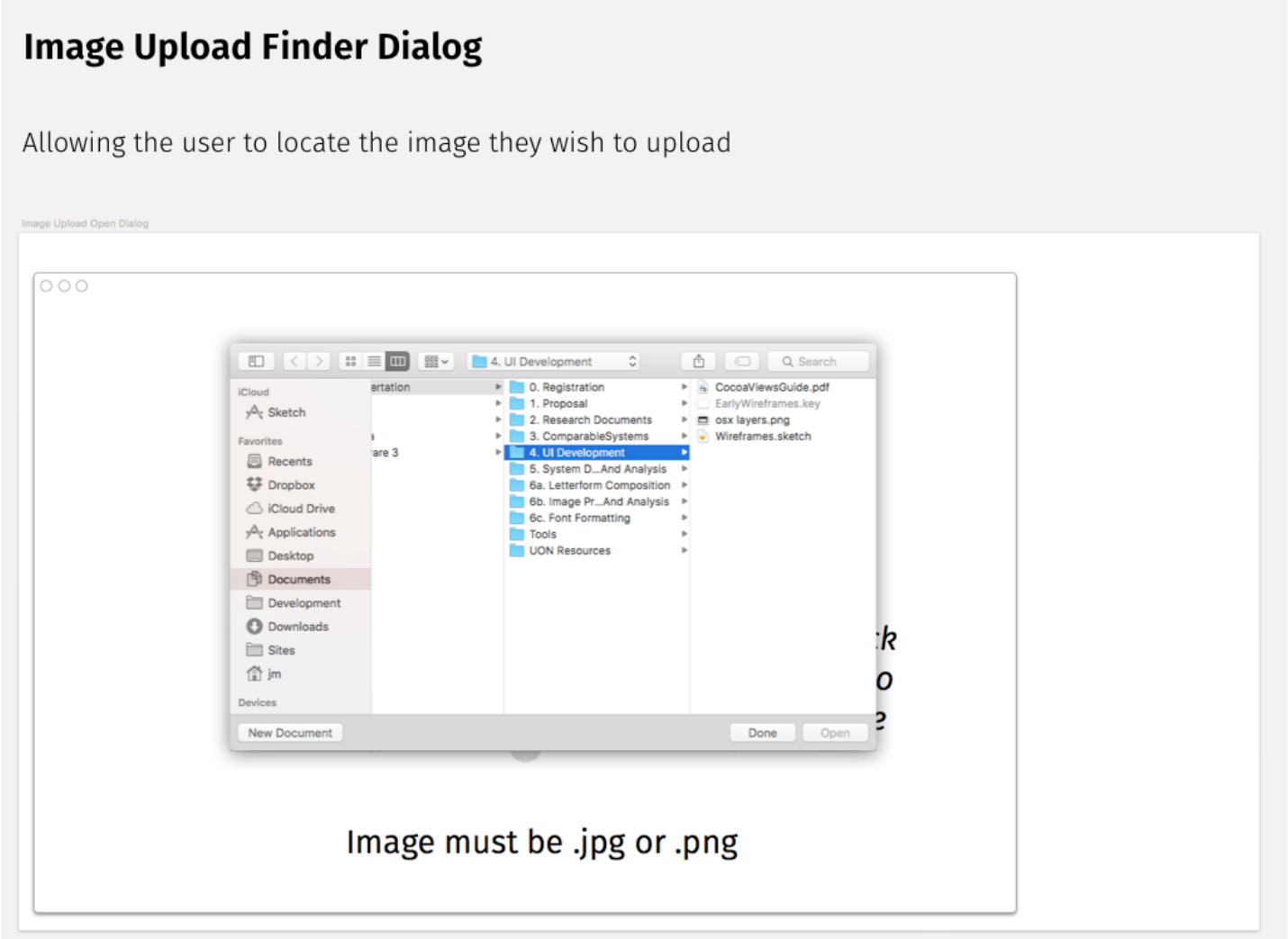


Figure 10: UI Finder Dialog

Implementation of the drag and drop interface would require the display of feed back to the user in the event an incorrect format is provided. In the case of a Finder dialog, the available list of formats would be reduced down to only the applicable formats.

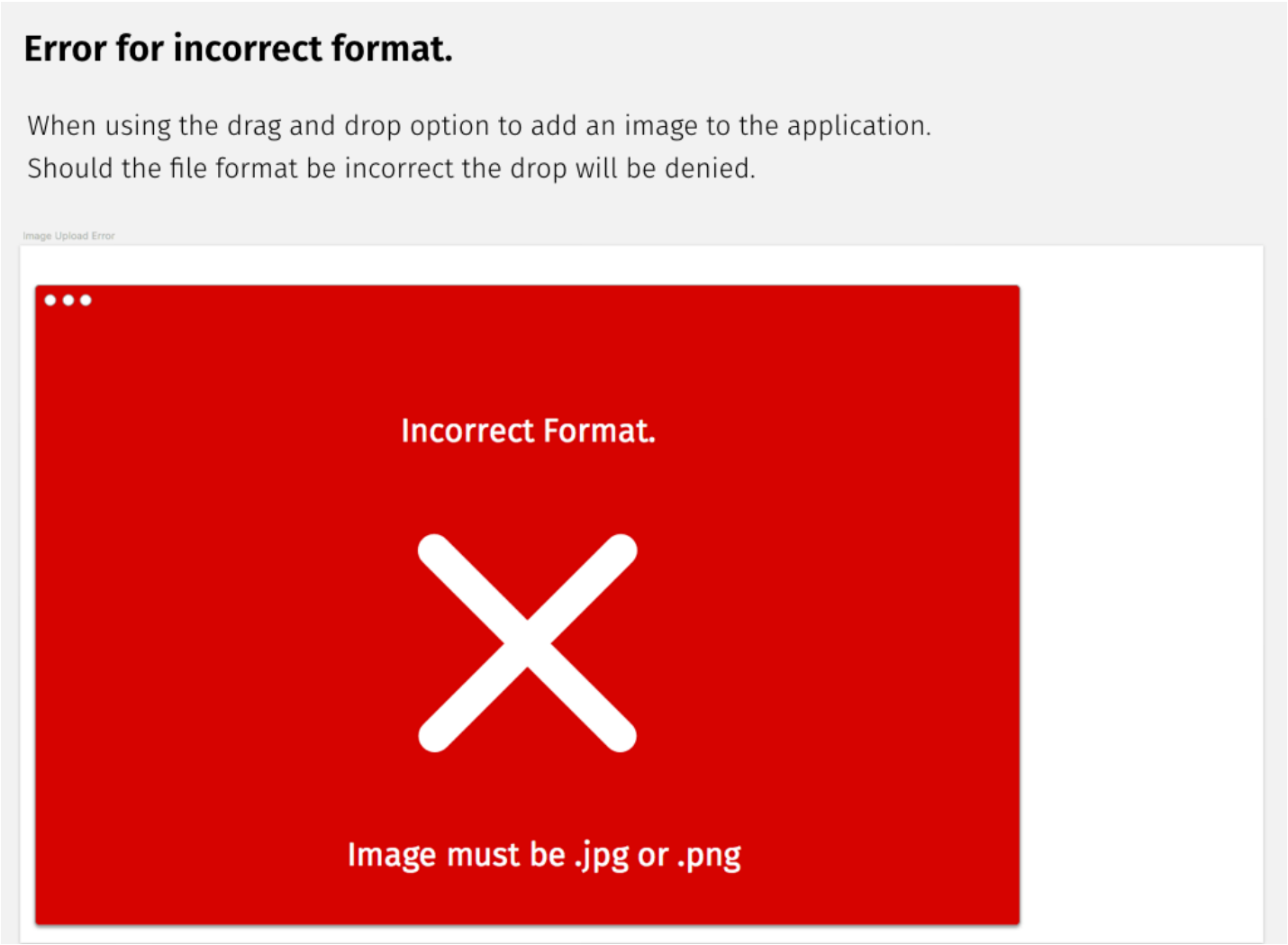


Figure 11: UI Error Feedback

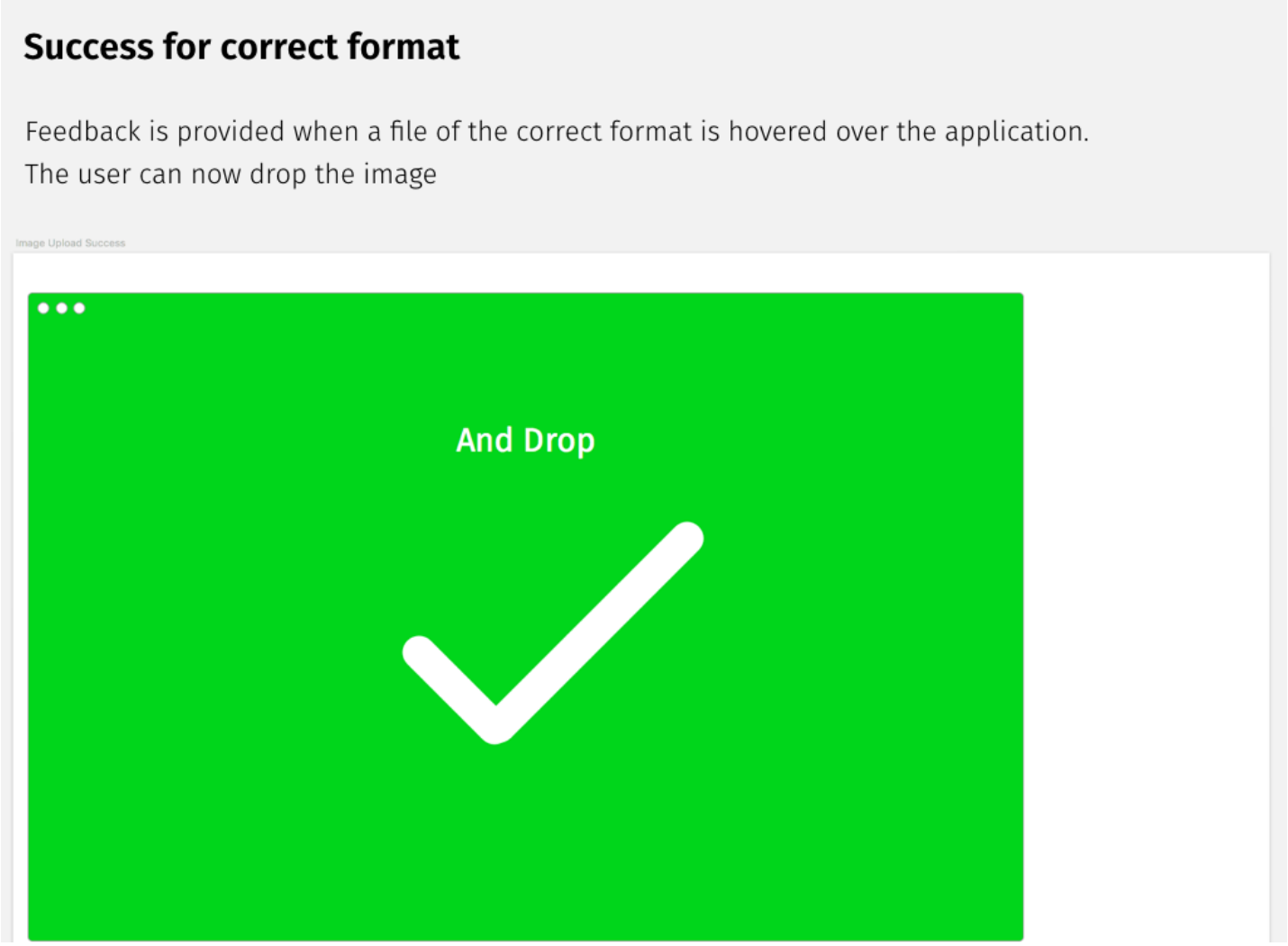


Figure 12: UI Success feedback

4.3 IMAGE PROCESSING TOOLS.

Due to the wide variety of image quality that maybe provided to the application, user interaction may be required during the Image Processing phases. Although the intension is to limit the required user interaction by programmatically fixing the image quality, as of version one of this application some interaction maybe required. Where values are required by the system, Slider controls would be used to alter the values pasted to the various image processing implementations.

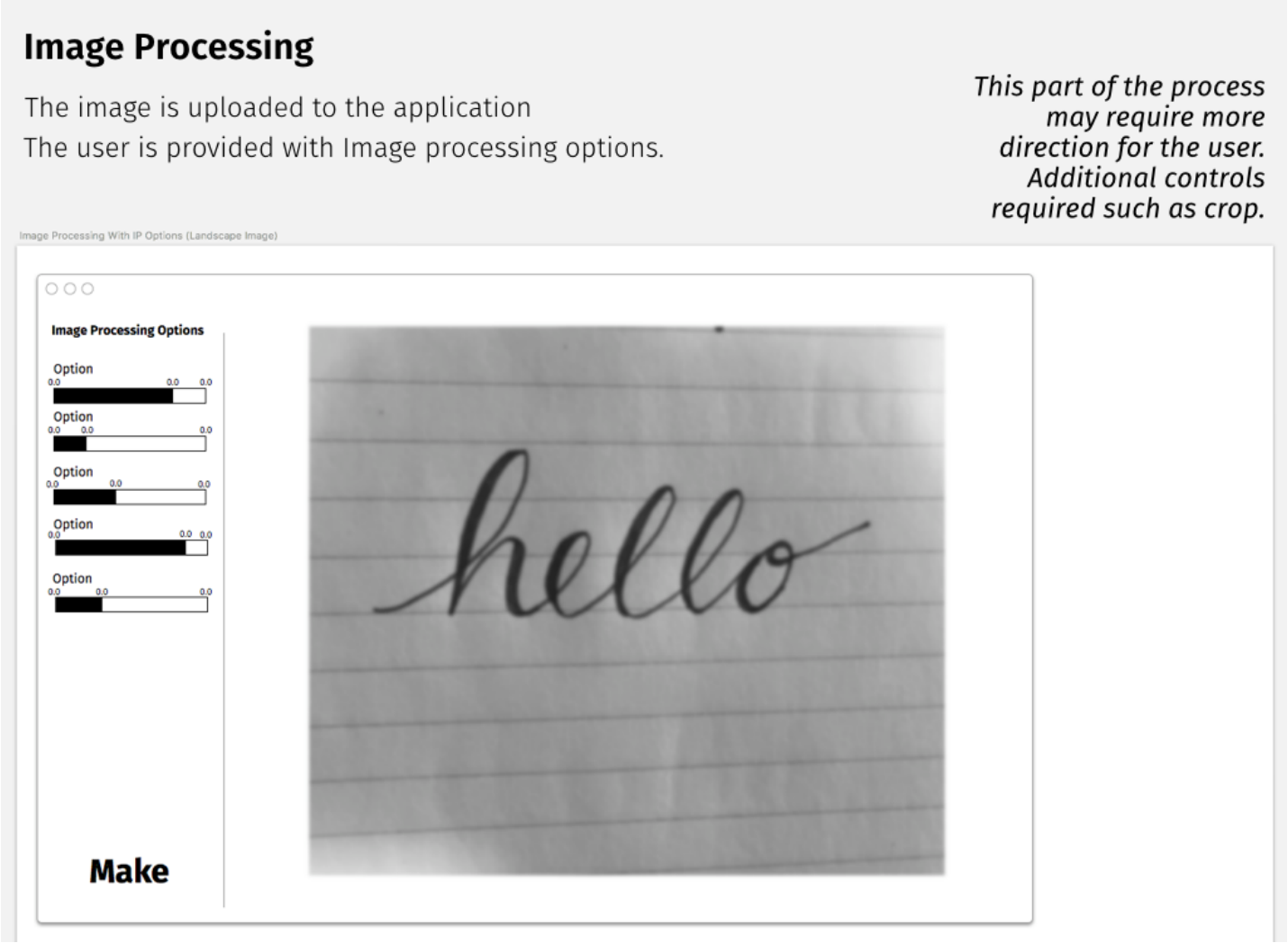


Figure 13: UI Image processing tools

4.4 OUTPUT PRESENTATION.

Once processing of an image is complete, displaying final forms to the user will allow for comparison and in future versions the ability for the user to grade output opening the possibility of a systems that learns.

Figure 14: UI Letterform Comparison

4.5 SAVING OUTPUT.

Two possibilities for the saving and installation of font files has been considered.

1) Simply allow the user to save the file using a Finder Dialog. Figure 15
2) A multi stepped process guiding the user through: naming a font, specifying fonts formats to generate, selection of licenses for sharing of the output typeface, and whether to install and/or download the files. Figure 16

For version one of this application the intention is to simply provide an SVG output, therefore option one would be acceptable.

Save Output

Save dialog provided on action.

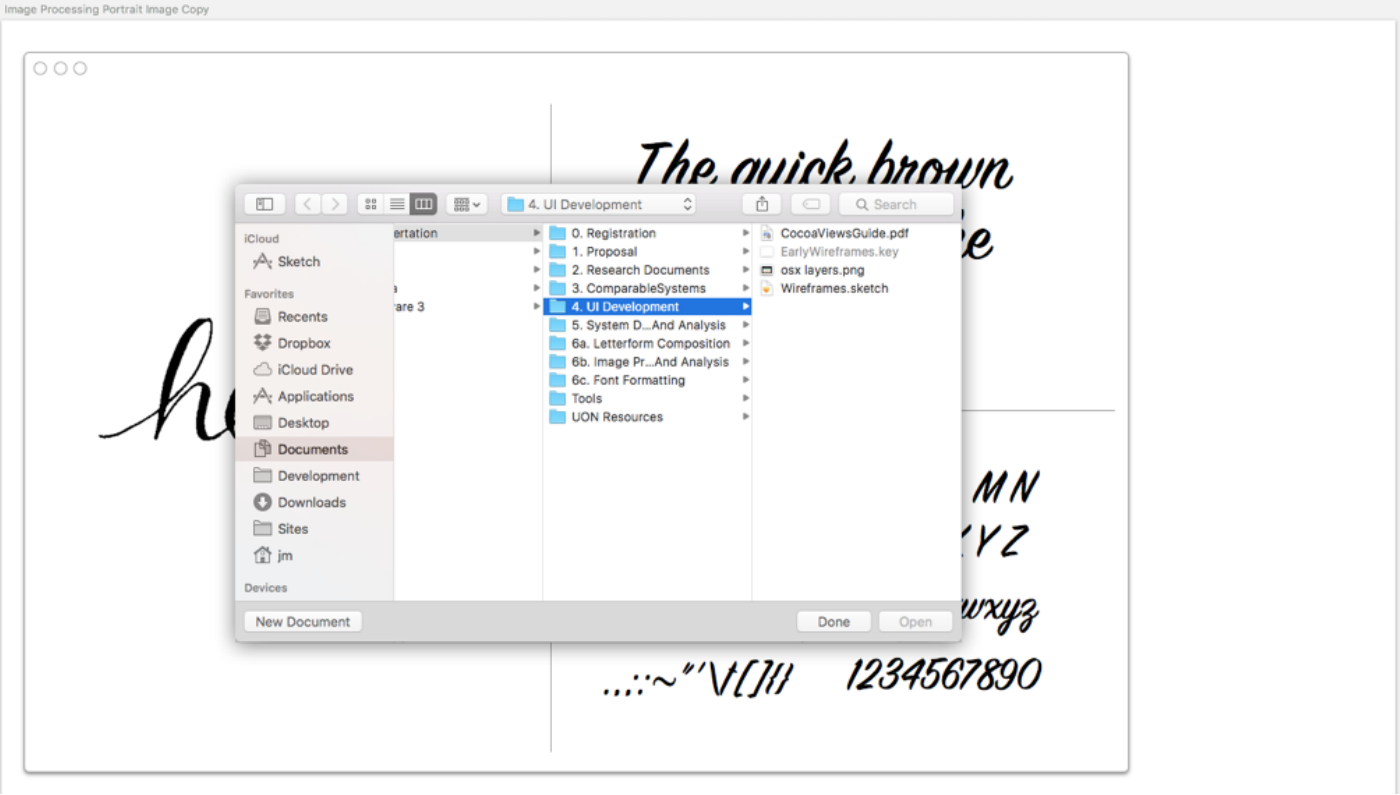


Figure 15: UI Saving output

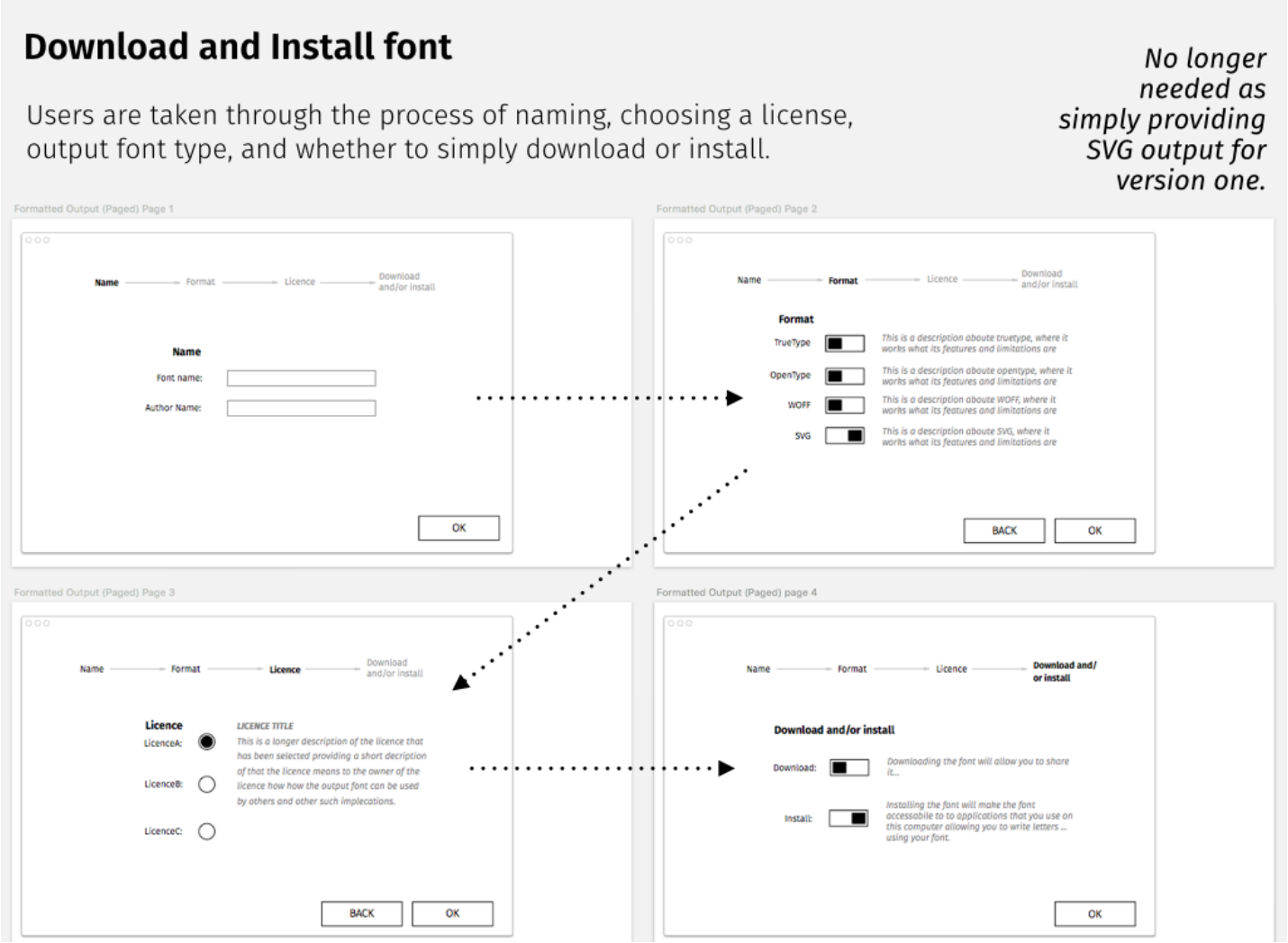


Figure 16: UI Multi stepped guided output

4.6 UI STORYBOARD.

Figure . details the a high level view of the full process required of for user interaction

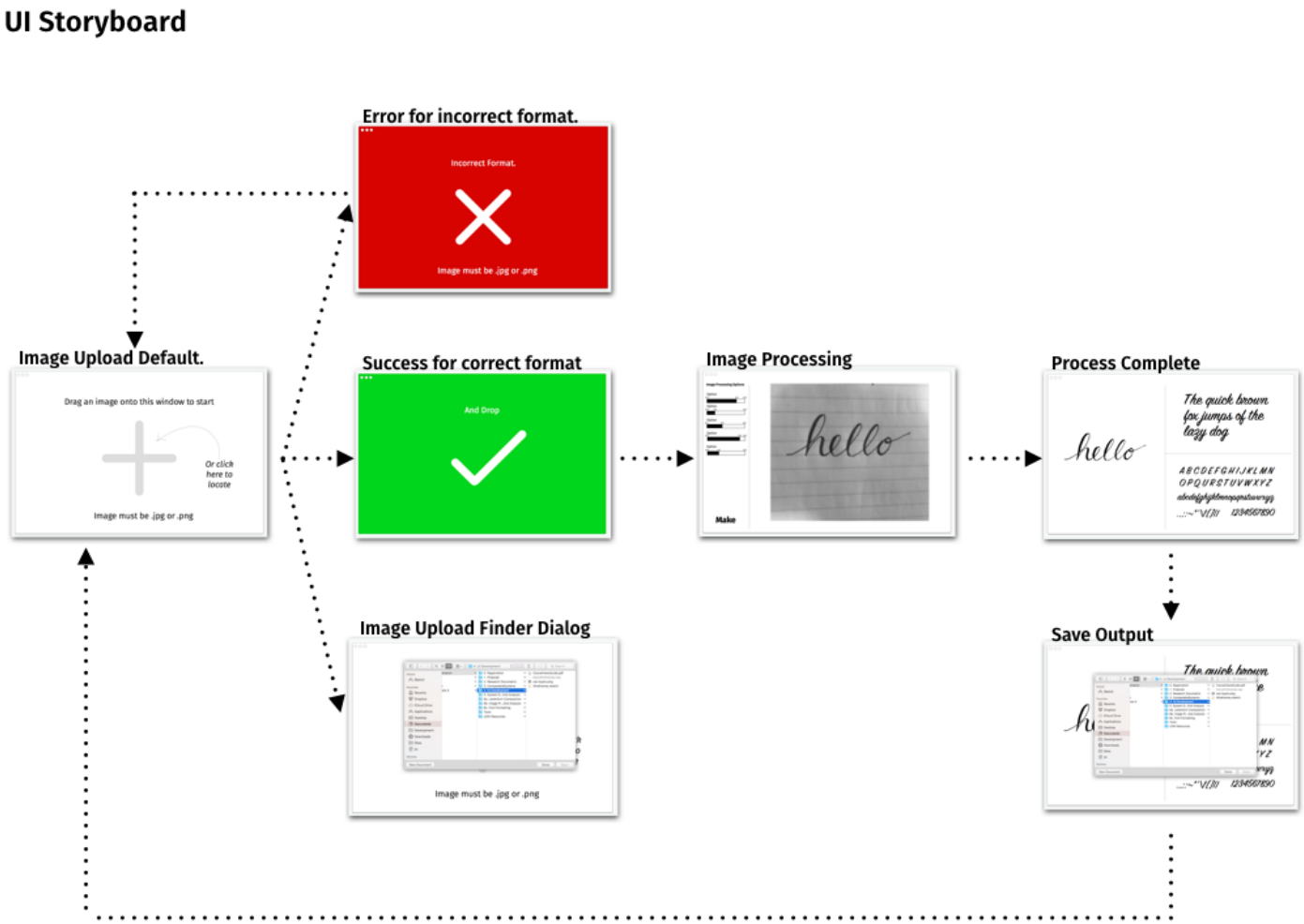


Figure 17: UI Storyboard

4.ui.001.png (Figure 8: Screen Dimension Constraints) [1024x768]

5.1 IMAGE DATA ACQUISITION.

5.1.1 USING COCOA TO RETRIEVE THE IMAGE DATA.

In order to facilitate the extraction of image data into a processable format of manipulation, the Cocoa framework provides the `NSBitmapImageRep` class with the `–initWithBitmapDataPlanes:pixelsWide:pixelsHigh:bitsPerSample:samplesPerPixel:hasAlpha:isPlanar:colorSpaceName:bitmapFormat:bytesPerRow:bitsPerPixel:` method.

As colour data is not required for the any produced output the `bitsPerSample`, and `bitsPerPixel` will be provided with only 8 bits, with the `samplesPerPixel` limited to 1. This limits the data to a 0 - 255 range of grey level values.

https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CocoaDrawingGuide/Images/Images.html#//apple_ref/doc/uid/TP40003290-CH208-BCIBBFGJ

(Apple, 2009) Cocoa Drawing guide presents the `NSGraphicsContext` class as a method of drawing an image to a newly created bitmap representation.

This has allowed of the creation of a gray scale representation, to which the acquired image is drawn (Figure 18).

```
    // ...
    NSString* file = [@"~/path/to/image.png" stringByExpandingTildeInPath];
    NSImage* image = [[NSImage alloc] initByReferencingFile:file];

    // ImageRepresentation.m
    NSBitmapImageRep *representation = [[NSBitmapImageRep alloc]
                                        initWithBitmapDataPlanes: NULL
                                        pixelsWide: image.size.width
                                        pixelsHigh: image.size.height
                                        bitsPerSample: 8
                                        samplesPerPixel: 1
                                        hasAlpha: NO
                                        isPlanar: NO
                                        colorSpaceName: NSCalibratedWhiteColorSpace
                                        bytesPerRow: image.size.width
                                        bitsPerPixel: 8];

    NSGraphicsContext *context = [NSGraphicsContext
graphicsContextWithBitmapImageRep:representation];
    [NSGraphicsContext saveGraphicsState];
    [NSGraphicsContext setCurrentContext:context];

        [image drawAtPoint:NSZeroPoint
                  fromRect:NSZeroRect
                 operation:NSCompositeCopy
                  fraction:1.0];

    [context flushGraphics];
    [NSGraphicsContext restoreGraphicsState];

    unsigned char* data = [representation bitmapData];
```

Figure 18: Image Acquisition

5.1.2 IMAGE CROP FUNCTIONALITY.

The addition of cropping functionality will:
1. Reduce the size of the representation requiring processing, therefore improving performance.
2. Reduce the variety of data that is required to be processed ensuring only relevant detail is analysed.
3. Allow the user to specify the exact regions of text for analysis.

A major trade off of adding the crop functionality is a lesser requirement for user interaction upon aspects of the image processing phase that could be considered advanced ability or knowledge requirement.

As an example, the following representation provided by the user contains: two varying regions of handwritten text a) and b), a region of texture resulting from the wooden table in the background c).

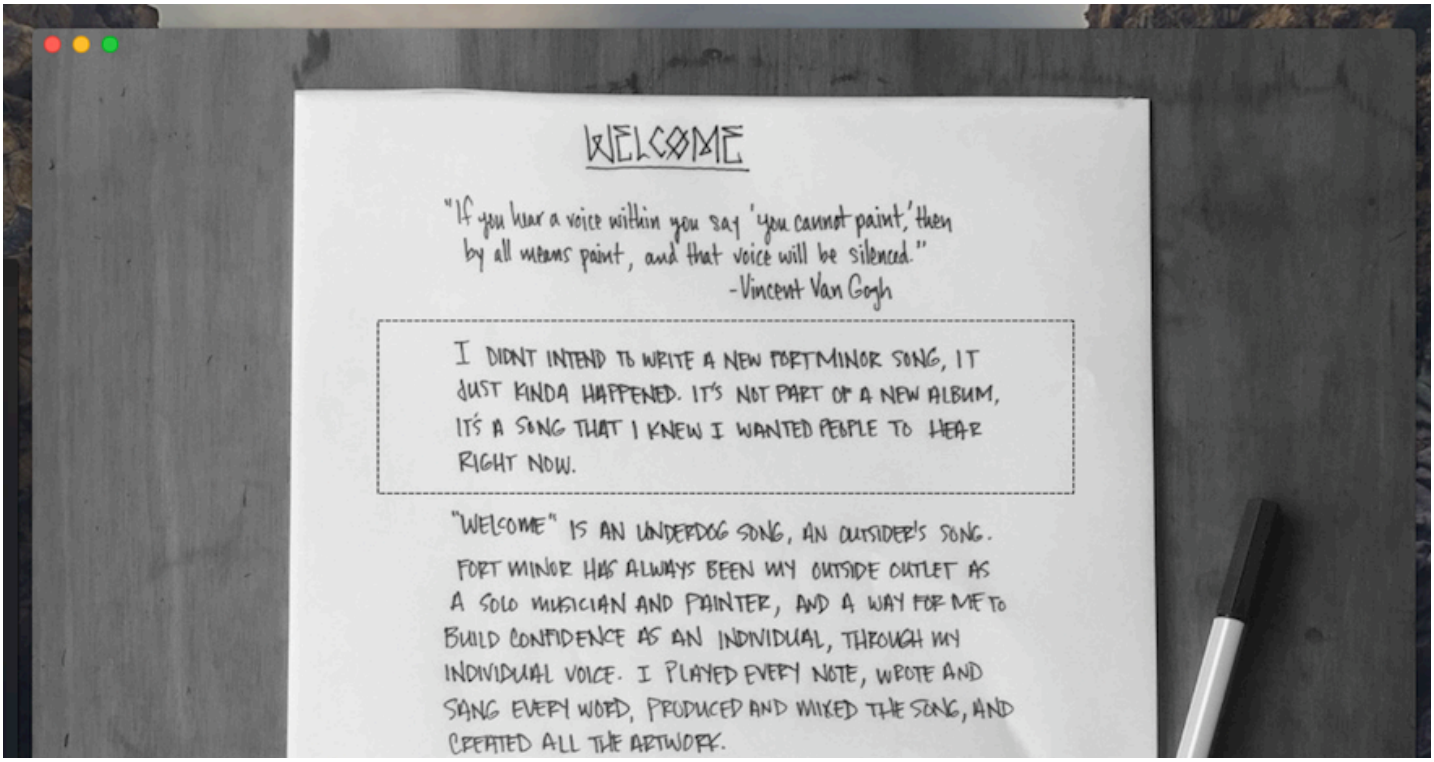


Figure 19: Image Crop Subject

Using the thresholding of the image as an example, the wide range of grey level values of the representation creates difficultly when trying to determine an optimal threshold value that will maintain a strong representation of the required data (the handwritten element of the supplied image) and remove unwanted data. This could potentially require the user to specify the threshold value.

As cropping functionality is common in popular applications such as: Microsoft Word or PowerPoint, and Photoshop, cropping of images likely already within a users knowledge, whereas thresholding functions would not be common an potentially difficult for the user to understand. In addition, a cropping function provides the further benefit by allowing the user more control over the selection of text in images where variance in handwriting style is apparent as in Figure 19.

5.2 IMAGE PROCESSING DEVELOPMENT.

The importance of the image processing phase is to create the best environment possible when capturing the stylistic features of the handwriting text. Ideally, the resultant data available prior to the image pre-processing phase will be a binary image containing representations of only pen marks of the subject made to represent text.

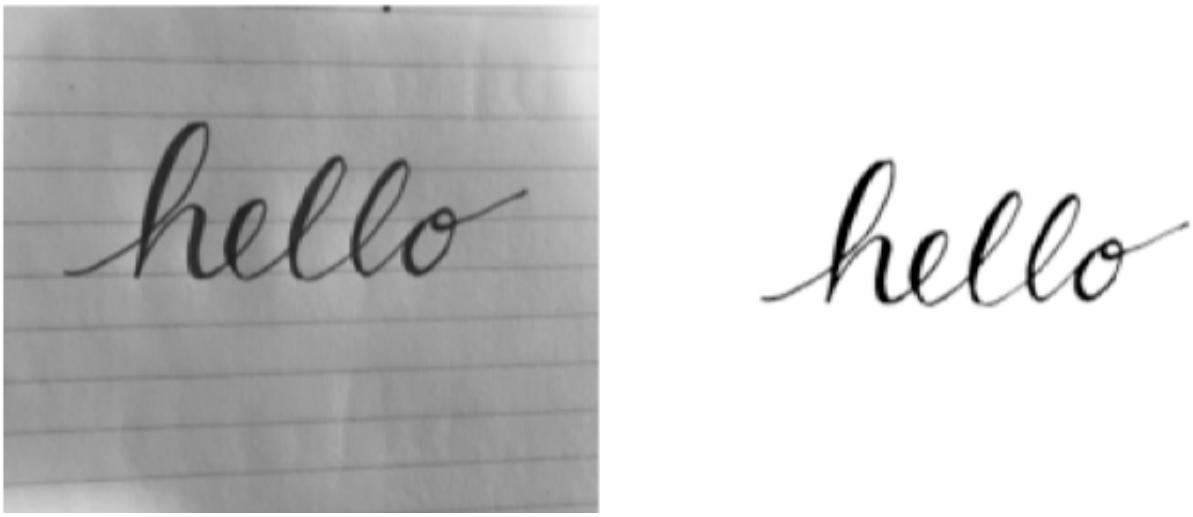


Figure 20: Original image and desired output from preprocessing

5.2.1 IMAGE PROCESSING CONSIDERATIONS.

Isolation of the text region by removing: - Any non-text regions of the image such as the table behind a photographed notepad of text.

Removal of all noise including:
1. Notepad lines that may appear on the image. (line detection)
2. Pen marks showing though thin paper. (smoothing filter, light blurring then thresholding)
3. Shadows on folded paper. (smoothing filter, blurring)
4. Impulse noise/Spotty artifacts due to poor image quality. (smoothing filter, light blurring)

Contrast enhancement.
- Increase of the dynamic range using contrast stretching (Gonzalez and Woods, 2002, p. 85)

Finally to create the binary representation.
- thresholding.

Dealing with low resolution images. Thresholding of low resolution image will can result in (jagged) lines of the pen marks. In addition where a user can crop an image maybe need that the image is scaled up.
- Scaling up of cropped images.

5.2.2 POINT OPERATIONS OPERATIONS.

Point operations simply perform a provided action on a specified pixel.

(Gonzalez and Woods, 2002, p. 77) provides the follow expression.

$$g(x, y) = T[f(x, y)]$$

Where $f(x, y)$ is the input and $g(x, y)$ is the output , with $T[\dots]$ being the function defined over neighborhood (x, y)

(Gonzalez and Woods, 2002, p. 77)

5.2.2.1 GLOBAL THRESHOLDING/HARD THRESHOLDING.

Global thresholding is the matter of applying the same threshold value to every pixel in the representation.

“The grey level value remains constant over the selected neighborhood”

(Castleman, 1995, p. 452)

(Gonzalez and Woods, 2002, p. 86) provides the follow expression to represent a threshold operation.

$$g(x, y) = \begin{cases} 1 & if \quad f(x, y) > T \\ 0 & if \quad f(x, y) \leq T \end{cases}$$

From this the following pseudo code can be developed.

```
// Pseudo Code

T = threshold

for each row y in representation
    for each column x in row
    if pixel(x, y) > T then
        pixel(x, y) greylevel = 1
    else
        pixel(x, y) greylevel = 1
```

Figure 21: Threshold function pseudo code

```
- (NSBitmapImageRep *) thresholdWithValue:(int)value
{

    NSBitmapImageRep *output = [self grayScaleRepresentationOfImage:self.image;
    unsigned char *data = [output bitmapData];

    for ( int y = 0; y < self.height; y++ )
    {
        for ( int x = 0; x < self.width; x++ )
        {
            int index = x + (y * self.width);
            if ( data[index] < value) {
                data[index] = 0;
            } else {
                data[index] = 255;
            }
        }
    }

    return output;
}
```

Figure 22: Threshold function implementation

5.2.3 SPATIAL FILTERS.

Formula:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(x,y) f(x+s, y+t)$$

Where:
$g(x,y)$ is the final image.
$w(s,t)$ is the filter or mask.
$f(x+s, y+t)$ is the pixel value of the input related to a postion on the mask or filter

(Gonzalez and Woods, 2002, p. 177)

example filter relative to x, y

```
w(-1,-1), w(-1,0), w(-1,-1)
w(0,-1),  w(0,0),  w(0,1)
w(1,-1),  w(1,0),  w(1,1)
```

Figure 23: Neighborhood offset coordinates

(Gonzalez and Woods, 2002, p. 177)

However as Cocoa on OS X uses the Cartesian coordinate system (Cocoa drawing Guide, 2012) this would be:

```
w(-1,1),  w(0,1),   w(1,1)
w(-1,0),  w(0,0),   w(1,0)
w(-1,-1), w(0,-1),  w(1,-1)
```

Figure 24: Neighborhood offset Cartesian coordinates

So the first iteration of the inner loop at f x = 5, y = 5

```
s = -1
t = -1

g(x, y) += f(x = 4, y = 4) * w(-1, -1)
```

Figure 25: Spatial Filter iteration 1

The second iteration at f x = 5, y = 5

```
s = -1
t = 0

g(x, y) += f(x = 4, y = 5) * w(-1, 0)
```

Figure 26: Spatial Filter iteration 2

5.2.3.1 SMOOTHING FILTERS.

When applying a filter to an image there needs to be consideration for edge pixels as the filter neighborhood is centred around each pixel with the neighborhood overlapping the edge of the image when computing the edge pixels.

Two options to consider are:

1. Padding the image by the length of the new filter to the edge of the filter or $\left|\frac{sizeoffilter}{2}\right|$ requiring the representation to be computed for each size of filter before processing the filter.
2. Pad the loop over the image by abs(size of the filter / 2) leaving the image only partially filtered and create artefacts around the edges of the image.

5.2.3.1.1 SIMPLE AVERAGING FILTER.

For a simple averaging filter each value of the filter is the same. Simply computed as $\frac{1}{sizeofthefilter}$ this value is then applied to the neighbourhood. Dependant on the size of the filter specified the range of `s` and `t` variable will change.

```
w = 1.0 / size

for each pixel x, y

for s = -1 to 1
  for t = -1 to 1

    val += pixel[x + s, y + t] * w

  end
end

pixel[x, y] = val
```

Figure 27: Simple averaging filter pseudo code.

As the use of a simple averaging filter will degrade the edges of pen marks, these will be used sparingly and only of a small filter size.

```
// main.m
NSBitmapImageRep *smoothed = [ip smoothWithSimpleAveragingFilterOfSize:5];


// IP.m
- (NSBitmapImageRep *) smoothWithSimpleAveragingFilterOfSize:(int)size
{

    // create a represenation of the original image
    NSBitmapImageRep *representation = [self
grayScaleRepresentationOfImage:self.image];
    unsigned char *original = [representation bitmapData];

    // create a representation that will store the smoothed image.
    NSBitmapImageRep *output = [self grayScaleRepresentationOfImage:self.image];
    unsigned char *smoothed = [output bitmapData];

    float weight = 1.0 / (float)(size * size); // e.g. 1/(3 * 3) = 0.111
    int padding = (size - 1) / 2.0;  // pad the image

    // iterate over each pixel of the image
```

```
    for ( int y = padding; y < self.height - padding; y++ ) {
        for (int x = padding; x < self.width - padding; x++) {

            // find the centre pixel.
            int centre = x + y * self.width;
            int val = 0;

            // iterate over the filter
            for (int s = -padding; s < (padding + 1); s++) {
                for (int t = -padding; t < (padding + 1); t++) {

                    // offset the current x, y
                    int index = (x + s) + ((y + t) * self.width);
                    // add the values
                    val += original[index] * weight;

                }
            }

            // reject values over 255 to prevent
            if ( val > 255 ) val = 255;
            // apply the new value to centre of the filter
            smoothed[centre] = val;
        }
    }

    return output;
}
```

Figure 28: Simple averaging filter implementation.

OUTPUT.

Figure 29: Original Image Grayscaled

*Figure 30: 5 * 5 Simple Averaging Filter*

Figure 31: Thresholded 30%

5.2.3.1.2 WEIGHTED AVERAGING FILTER.

The aim of a weighted averaging filter is to reduce the blurring in the noise removal process. With a filter that is more greatly weighted towards the centre and more importance given to the centre pixel (Gonzalez and Woods, 2002, p. 120). Therefore, when centre pixel is part of or the edge of an object of the image this will maintain a higher prominence.

For the filter:

```
1, 2, 1
2, 4, 2 x (1/16)
1, 2, 1
```

Figure 32: Weighted averaging filter (Gonzalez and Woods, 2002, p. 120)

```
filter = [1, 2, 1, 2, 4, 2, 1, 2, 1] * (1 / 16)
i = 0

for each pixel x, y

for s = -1 to 1
  for t = -1 to 1
    val += pixel[x + s, y + t] * filter[i++]
  end
end

pixel[x, y] = val
```

Figure 33: Weighted Averaging filter pseudo code.

```
// As with simpleAveragingFilter

int weights[9] = {1, 2, 1, 2, 4, 2, 1, 2, 1};
float filter[9];

for (int i = 0; i < 9; i++)
{
    filter[i] = (float)weights[i] / 16.0;
}

// As with simpleAveragingFilter

   val += original[index] * filter[i++];

// As with simpleAveragingFilter
```

Figure 34: Weighted Averaging filter implementation.

5.2.3.1.3 MEDIAN FILTER.

The median filter applies the *median* value of the neighborhood to origin pixel.

```
filter[size]
i = 0

for s = -1 to 1
  for t = -1 to 1
    filter[i++] = pixel[x + s, y + t]
  end
end

pixel[x, y] = medianValueFromArray:filter ofSize:size

// medianValueFromArray.

middle = size / 2
sort filter

return filter[middle]
```

Figure 35: Median Filter Pseudo Code

```
// main.m

NSBitmapImageRep *median = [ip reduceNoiseWithMedianFilterOfSize:9];


// IP.m
- (NSBitmapImageRep *) reduceNoiseWithMedianFilterOfSize:(int)size
{
    // create representations.
    // as with smoothWithSimpleAveragingFilterOfSize

    int padding = (size - 1) / 2.0;
    int filter[size * size];

    for ( int y = padding; y < self.height - padding; y++ ) {
        for (int x = padding; x < self.width - padding; x++) {

            int centre = x + y * self.width;
            int i = 0;

            for (int s = -padding; s < (padding + 1); s++) {
```

```
                for (int t = -padding; t < (padding + 1); t++) {

                    int index = (x + s) + ((y + t) * self.width);
                    filter[i++] = original[index];

                }
            }

            smoothed[centre] = [self getMedianFromArray:filter ofSize:size];
        }
    }

    return output;
}


- (int) getMedianFromArray:(int [])arr ofSize:(int)size
{
    int middle = (int)(size / 2);
    [self bubbleSort:arr ofSize:size];
    return arr[middle];
}
```

Figure 36: Median Filter Implementation.

Figure 37: Original Image Grayscaled

*Figure 38: 9 * 9 Median Filter*

Figure 39: Filtered Image Thresholded 40%

Results of the 9 * 9 Median filter show artefacts created on the upstroke of the “h” and lost information on the exit of the first “l” and on the tail of the “o”

5.2.3.1.4 MIN/MAX FILTER.

The max and min filters differ from the median filter in only one detail, the value set is either the maximum or minimum values of the collected neighborhood rather then the median.

```
// main.m
NSBitmapImageRep *max = [ip reduceNoiseWithMaxFilterOfSize:11];


// as with median filter

- (int) minFromArray:(int [])arr ofSize:(int)size
{
    [self bubbleSort:arr ofSize:size];
    return arr[0];
}
```

Figure 40: Min/max Filter Implementation

Figure 41: Original Image Gray scaled

*Figure 42: 11 * 11 Max/Min Filter*

Figure 43: Max/Min Filter Thresholded 20%

The result of the Max filter on the sample image shows the thickening of the notepad lines and distortion of the pen marks.

5.2.4 MORPHOLOGY.

Morphology is the process of changing the shape of objects of the image.

The application of morphological operations on provided subjects will create a toolkit for:

- re-connecting broken stokes in the subject.
    - Performed by dilation, or growing the object.
- removing data for the purpose of analysis.
    - Performed by erosion, shrinking of the object.

As displayed in figure 44 the image provided contains thin lines that when thresholded to a certain degree will disconnect from the object. Should tracing [5.3.1.1] be applied to this subject with the intention of capturing border information from the subject potentially vital information could be overlooked.

Figure 44: Broken lines of an object

By applying a combination of a Max filter and a dilation operation it is possible to connect the objects.

Figure 45: Lines reconnected using combination of a Max Filter and dilation

To enable the possibility to analyse the direction and/or degree of a letterform subject's slant, removing the detail of the lines with thinner contrast by eroding the subject may provide a reasonable environment for analysis (Figure:)

Figure 46: The original subject

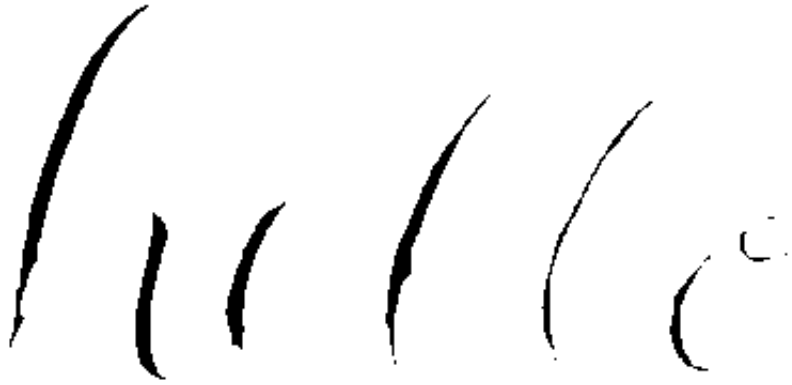


Figure 47: Data removed from subject

As with the application of smoothing filters[5.2.3] implementation of two selected morphological operations, erosion and dilation, a structuring element representing a neighborhood of pixels will be used. The structuring element for will be a binary representation. As described by the following:

$$H(i,j) \in 0,1$$

So, the structure H(i, j) is a member of the set {0, 1}

(Burger and Burge, 2014)

5.2.4.1 DILATION.

(Gonzalez and Woods, 2002, p. 524) describes the dilation of an image as:

> *"The dilation of A by B then is the set of all displacements z, such that A and B overlap by at least one element.*

Here B is the structuring element with A being the subject. The displacement z is the current position x, y in in the representation.

Therefore, where a structuring element built around a defined point *hits* or makes a connection to a foreground pixel, the defined point (centre pixel in this implementation) is set to a foreground pixel.

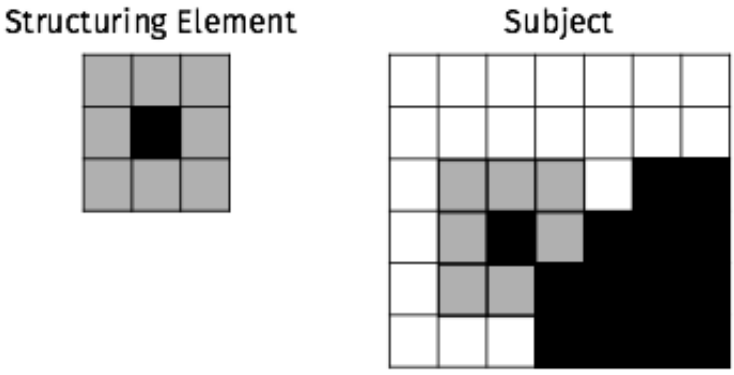


Figure 48: Structuring element overlapping a object in the subject

PSEUDO CODE.

As with the smoothing filters [5.2.3] implemented each pixel of the representation is evaluated, an offset or padding around the centre of the structuring element is defined in order to visit each offset of the structuring element. Where any element of the

```
padding = (size - 1) / 2.0

foreground = black
background = white

for y, x pixels in image

    for each offset from current x, y

        if input subject at offset == foreground
            output subject at centre of structuring element = foreground
        else
            output subject at centre of structuring element = background
```

Figure 49: Dilation Pseudo Code

IMPLEMENTATION.

Taking a thresholded image and creating two representation, an original for inspection and another for the output, iterate over each pixel. At each pixel visit each pixel in the offset and determine if the offset is of the foreground or background. Where a foreground pixel is found a *hit* is recorded and foreground is later applied to the centre point. Should there be no foreground pixels under the offset position the centre is set to the background.

```
// created representations of the original and output

int padding = (size - 1) / 2.0;

background = 0; // black
foreground = 255; // white

// for y, x

    for (int s = -padding; s < (padding + 1); s++) {
        for (int t = -padding; t < (padding + 1); t++) {

            int index = (x + s) + ((y + t) * width);

            if ( original[index] == foreground )
            {
                hits = YES;
            }
        }
    }

    processed[centre] = background;
    if ( hits )
    {
        processed[centre] = foreground;
    }
```

Figure 50: Dilation Implementation.

RESULT OF DILATION.

Figure 51: Dilation subject

Figure 52: Dilation Result 3x3 structuring element

5.2.4.2 EROSION.

> *"The white shapes make the background and the black shapes make the foreground. The background makes the foreground and the other way around. Change one and you change the other too.*

(Smeijers, 1996)

Therefore, while dilation is used to increase the foreground by the structuring element (effectively decreasing the background), erosion can be the mater of increasing the background by the structuring element. Al that is required is to reverse the inquiry target and the value that will be set at centre.

```
padding = (size - 1) / 2.0

foreground = white
background = black

for y, x pixels in image
    for each offset from current x, y

        if input subject at offset == foreground
            output subject at centre of structuring element = foreground
        else
            output subject at centre of structuring element = background
```

Figure 53: Erosion Pseudo Code

This allows for the creation of a method that can do both dilation and erosion by simply supplying the background and foreground values as parameters.

```
- (NSBitmapImageRep*) simpleDilationOfImage:(NSImage*)image
                       withNeighbourhoodSize:(int)size
{
    return [self processImage:image
                withBackground:255
                 andForeground:0
           andNeighbourhoodSize:size];
}


- (NSBitmapImageRep*) simpleErosionOfImage:(NSImage*)image
                      withNeighbourhoodSize:(int)size
{
    return [self processImage:image
                withBackground:0
                 andForeground:255
           andNeighbourhoodSize:size];
}

- (NSBitmapImageRep*) processImage:(NSImage *)image
                    withBackground:(int)background
                     andForeground:(int)foreground
              andNeighbourhoodSize:(int)size
{

 // as with dilation figure ...

}
```

Figure 54: Erosion Implementation

RESULT OF EROSION.

Figure 55: Erosion subject

Figure 56: Erosion Result 3x3 structuring element

5.2.4.3 OPENING AND CLOSING.

Opening operations is provide the function of Smoothing contours and eliminating small islands and sharp peaks. While Closing operations, smooths contours and fuses narrow breaks.

The equations for Erosion and Dilation are as follows.

- Erosion: $A \oplus B = \left\{ z | \left(\hat{B} \right)_z \cap A \neq \emptyset \right\}$
- Dilation: $A \ominus B = \left\{ z | (B)_z \subseteq A \right\}$

(Gonzalez and Woods, 2002, p. 528)

And the equations for opening an closing:

- Opening: $A \circ B = (A \ominus B) \oplus B$
- Closing: $A \bullet B = (A \oplus B) \ominus B$

Therefore, Opening is the application of Dilation followed by Erosion and Closing is Erosion followed by Dilation.

(Gonzalez and Woods, 2002, p. 528)

The implementation of these tools simply requires the calling of one method after the other.

OPENING IMPLEMENTATION

```
- (NSBitmapImageRep*) openingOnImage:(NSImage*)image
                withNeighbourhoodSize:(int)size
{
    NSImage* temp = [[NSImage alloc] initWithSize:image.size];
    NSBitmapImageRep* eroded = [self simpleErosionOfImage:image
                                     withNeighbourhoodSize:size];

    temp = [ImageRepresentation cacheImageFromRepresentation:eroded];
    NSBitmapImageRep* dilated = [self simpleDilationOfImage:temp
                                       withNeighbourhoodSize:size];
    return dilated;
}
```

Figure 57: Opening Implementation

RESULT OF OPENING.

Figure 58: Opening subject

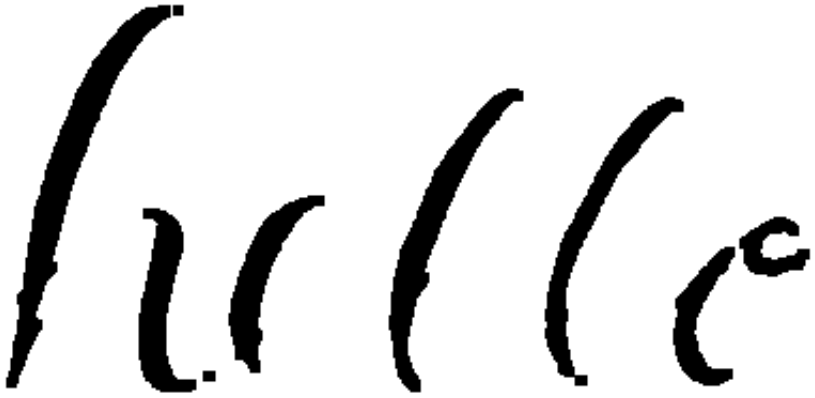


Figure 59: Opening Result

The result of opening (Figure: 59) shows that the remaining object are smoother that the result of using erosion alone (Figure: 60).

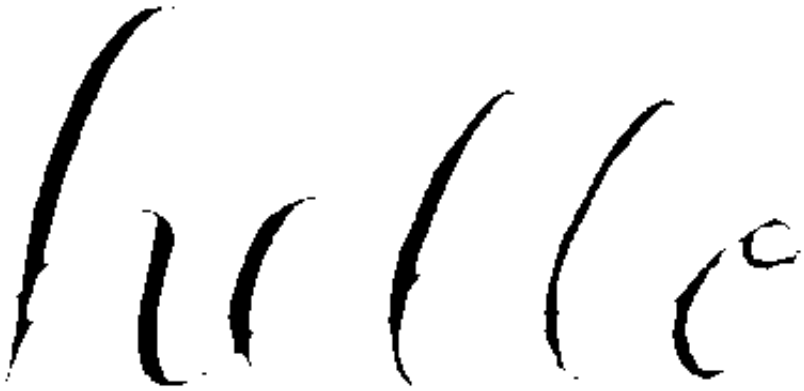


Figure 60: Erosion

CLOSING IMPLEMENTATION.

```
- (NSBitmapImageRep*) closingOnImage:(NSImage*)image
                withNeighbourhoodSize:(int)size
{
    NSImage* temp = [[NSImage alloc] initWithSize:image.size];
    NSBitmapImageRep* dilated = [self simpleDilationOfImage:image
                                       withNeighbourhoodSize:size];

    temp = [ImageRepresentation cacheImageFromRepresentation:dilated];
    NSBitmapImageRep* eroded = [self simpleErosionOfImage:temp
                                    withNeighbourhoodSize:size];
    return eroded;
}
```

Figure 61: Closing Implementation

CLOSING RESULT.

Figure 62: Closing subject

Figure 63: Closing Result

The result of Closing showing the strengthening of join points.

5.2.4.4 ATTAINING BORDERS.

Retrieving the border of a subject is possible by taking two subjects and eroding or dilating one of the subject before subtracting one image from the other.

(Gonzalez and Woods, 2002, p. 528)

Alternatively using a tracing algorithm. [5.3.1.1]

IMPLEMENTATION.

```
- (NSBitmapImageRep*) imageDifferenceOf:(NSImage*)image1
                                    and:(NSImage*)image2
{
    NSImage* outputImage = [[NSImage alloc] initWithSize:image1.size];

    NSBitmapImageRep* rep1 = [ImageRepresentation
grayScaleRepresentationOfImage:image1];
    NSBitmapImageRep* rep2 = [ImageRepresentation
grayScaleRepresentationOfImage:image2];
    NSBitmapImageRep* output = [ImageRepresentation
grayScaleRepresentationOfImage:outputImage];

    unsigned char *one = [rep1 bitmapData];
    unsigned char *two = [rep2 bitmapData];
    unsigned char *three = [output bitmapData];

    int width = image1.size.width;
    int height = image1.size.height;

    for ( int y = 0; y < height; y++ )
    {
        for (int x = 0; x < width; x++)
        {
            int index = x + (y * width);
            three[index] = one[index] - two[index];
        }
    }

    return output;
}
```

Figure 64: Attaining Borders

RESULT.

Figure 65: Image Difference

The result can then be negated to return the foreground and background colours to normal.

Figure 66: Difference polarity switched

```
- (NSBitmapImageRep*) imageNegativeOf:(NSImage*)image
{
    NSBitmapImageRep* output = [ImageRepresentation
grayScaleRepresentationOfImage:image];

    unsigned char* rep = [output bitmapData];

    int width = image.size.width;
    int height = image.size.height;

    for ( int y = 0; y < height; y++ )
    {
        for ( int x = 0; x < width; x++ )
        {
            int index = x + (y * width);

            int val = rep[index] - 255;

            if ( val < 0 )
            {
                val = val * -1;
            }

            rep[index] = val;
        }
    }
    return output;
}
```

Figure 67: Switching Polarity Implementation

5.2.4.5 THINNING.

PURPOSE.

Thinning of the representation will provide data for both the construction of the letterform template, and the recognition of letter slant, cursive connections, and counter shape, patterns.

BACKGROUND.

Thinning is the iterative erosion of a connected set down to a single pixel thickness without splitting the structure. (Burger and Burge, 2014)

Application of a thinning algorithm over a provided image will therefore preserve the required structure of the letterform representation, while removing features that are not required, such as contrast. In addition reducing the amount of data required to be processed.

Prior to thinning the representation will need to be thresholded to replicate a binary image.

(Lam, Lee, and Suen, 1992) provides explanation of two varying types of thinning algorithms, sequential and parallel.

> *“In a sequential algorithm, the pixels are examined for deletion in a fixed sequence in each iteration, and the deletion of p in the nth iteration depends on all the operations that have been performed so far.*

(Lam, Lee, and Suen, 1992)

Implementation of a sequential algorithm requires recording of each preceding operation, marking pixels for deletion on a given condition. At the end of an iteration marked pixels are scrutinized further before removal.

> *“In a parallel algorithm, the deletion of pixels in the nth would only depend on the result that remains after the (n - 1)th; therefore, all pixels can be examined independently in a parallel manner each iteration.*

(Lam, Lee, and Suen, 1992)

Parallel thinning algorithms pixels are removed during each iteration and therefore only act upon the image data left by the last iteration.

Due to performance concerns with the requirement to maintain reference to pixels marked or “flagged” for deletion in the implementation of a sequential thinning algorithm, a parallel thinning approached will be used in the initial implementation of this project.

The Zhang Suen thinning algorithm has been examined due to favorable processing speed, simplicity of computations required, and high level of explanation.

Pattern	Four-Step	Two-Step	Zhang Suen
B	0.765	0.678	0.454
Moving body	2.713	2.221	1.163

Figure 68: Comparison of CPU time (in seconds) Consumed by Different Parallel Thinning Algorithms. (Zhang and Suen, 1984)

5.2.4.5.1 ZHANG SUEN THINNING ALGORITHM.

The goal of the Zhang Suen algorithm is to remove all contour points and preserve the structure of the pattern. The deletion of each pixel depends on it self and that of its 8 adjacent / connected neighbors and is split into two sub iterations in order to maintain structure.

Zhang Suen define the following conditions for each iteration.

a) $2 \leq B(P_1) \leq 6$

The sum of the 8 (nonzero/black) neighbors of P_1 which are structure pixels is greater then or equal to 2 and less than or equal to 6.

b) $A(P_1) = 1$

Of the ordered set $P_2, P_3, P_4 \ldots P_8$, the sum of each 0 to 1 patterns found equals 1. Therefore in the pattern from P_2 to P_8 (0, 1, 0, 1, 0, 0, 0, 1) there are three 01 patterns and so P_1 would not be removed.

c) $P_2 \times P_4 \times P_6 = 0$
d) $P_4 \times P_6 \times P_8 = 0$

Or, one of P_2, P_4, or P_6 is equal to 0 (white) and one of P_4, P_6, or P_8 is equal to 0 (white).

Where all of these conditions are true, the pixel at the centre of the neighborhood P_1 is removed, or in the case of this application set to white.

Steps a) and b) are repeated in the second iteration, however c) and d) are changed:

c) $P_2 \times P_4 \times P_8 = 0$
d) $P_2 \times P_6 \times P_8 = 0$

(Zhang and Suen, 1984)

The following pseudo code has been developed from (Zhang and Suen, 1984) algorithm detailed.

```
while changes are still being made.
    do subiteration 1
    do subiteration 2
```

Figure 69: Zhang Suen outer loop

```
foreach pixel x, y

    if pixel(x, y) != black
        skip all other conditions.

    // a)
    if pixel(p2) == black, a++
    ...
    if pixel(p9) == black, a++
    conditionA = 2 <= a <= 6


    // b)
    if pixel(p2) == white and pixel(p3) == black, b++
    ...
    if pixel(p9) == white and pixel(p2) == black, b++
    conditionB = (b == 1)

    // c)
    conditionC (p2 == white or p4 == white or p6 == white)

    // d)
    conditionD (p4 == white or p6 == white or p8 == white)

    if conditionA and conditionB and conditionC and conditionD
        set pixel x, y to white
        change made.
```

Figure 70: Zhang Suen sub iteration 1

```
// as with sub iteration 1

    // c)
    conditionC (p2 == white or p4 == white or p6 == white)

    // d)
    conditionD (p2 == white or p6 == white or p8 == white)

    if conditionA and conditionB and conditionC and conditionD
        set pixel x, y to white
        change made.
```

Figure 71: Zhang Suen sub iteration 2

RESULT.

Implementation and testing of the Zhang Suen thinning algorithm detailed in their paper (Zhang and Suen, 1984) shows that stems or thin structrues of letterforms are removed completly with vital data is lost.

Figure 72: Thinning input

As shown below the stem of b, d h, i, j in this small sample has been removed.

Figure 73: Thinning output

The inaccuracies in (Zhang and Suen, 1984) have be detailed in (Chen and Hsu, 1988) showing that condition a) does not preserve all end points of structures. As discussed by Chen, and Hsu altering condition a) to $3 \leq B(P_1) \leq 6$ resolves this problem.

Figure 74: Thinning output with new condition

However, this change will effect different representations in varying ways. Whereas with the thinned typeface detailed above and the structure degraded, applying the original a) condition to a handwritten letterform and the same degradation is not observerd.

Application with the amended condition on the same image shows a high level of trailing artifacts.

Figure 75: Subject

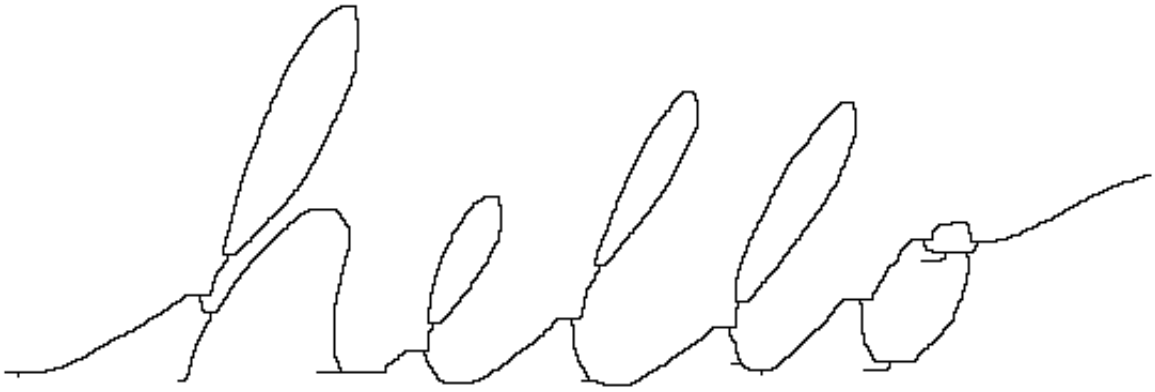


Figure 76: Subject thinned with original condition

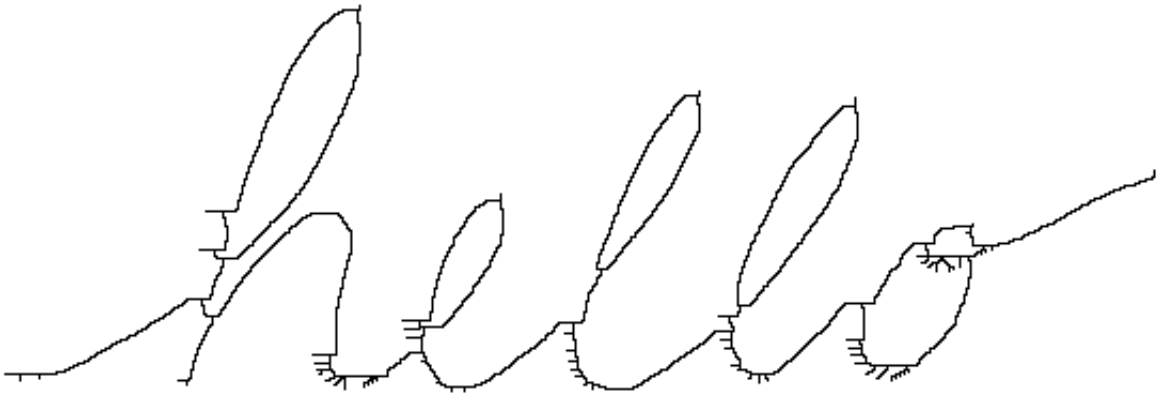


Figure 77: Subject thinned with amended condition

5.3 LETTERFORM TEMPLATE.

To ensure that the output represents a balanced and legible set of characters when constructed, data gleaned from the input samples, such as the x-height, cap-height, and slant, will be applied to a template of letterforms. This template can then be translated / manipulated to the specification provided by extracted attributes of the handwritten text.

The application of manipulating the template requires that:
- The template be in a vector format allowing for points to be adjusted rather then individual pixels.
- points of the letterform template are required to be relational / relative measures ensuring the balance is maintained throughout each character of the template and the subsequent output.
- Relative measures will need to be applied to the global letter set to account for the requirement of letter width and kerning to be of a consistent balance.

Due to the importance of the letter width and kerning on the presentation of a typed text the template will be based on a professionally developed typeface.

5.3.1 LETTER FORM POINT EXTRACTION.

The creation of the template will start with the extraction of point information from a skeletonised bitmap representation of a professional developed typeface.

5.3.1.1 CONTOUR AND LINE TRACING.

Inquiry into tracing algorithms has present 4 possible implementations.

- Moore-neighbor,
- Square tracing,
- Pavlidis algorithm,
- radial sweep.

Moore-neighbor, and Square tracing will ignore holes, however as the current requirement is to trace the contours of a skeletonised representation of letterforms the shape of inner holes will be equivalent to the outer shape.

The square tracing algorithm will fail to to trace an 8-connected structures that isn’t 4 connected.

Both the Moore-neighbor and the square tracing algorithms follow the same principle actions:

1. Find the start point.
2. Trace the contour of the image.
3. Stop at a given condition.

Where these algorithms differ is in action 2.

MOORE-NEIGHBOR ALGORITHM.

The Moore-neighbor algorithm named as such due to its use of the Moore neighborhood being the 8 connected neighbors (p1, p2 …, p8) around a point (p0).

```
p1  p2 p3
p8  p0 p4
p7  p6 p5
```

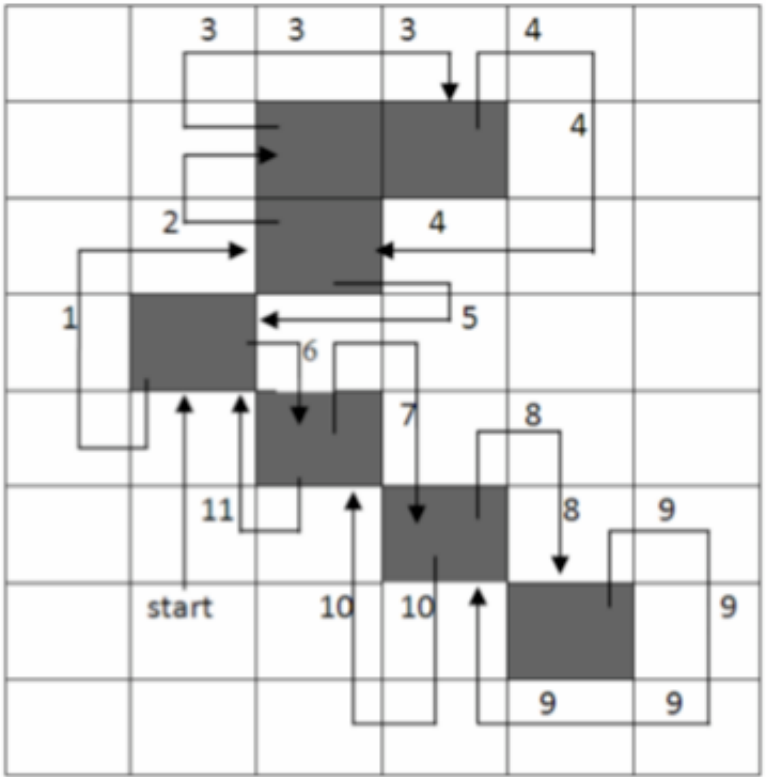
Figure 78: Moore Neighborhood

As a high-level representation of the algorithm the following presents the key steps as provided by the full algorithm in figure 79.

1. Iterate of the binary image until a black pixel is found.
2. Store the start pixel for later reference to be involved in the stopping criteria.
3. Set the start pixel as centre of Moore neighborhood
4. Go back to the pixel previous to the start pixel was found.
5. Move in a clockwise(or anticlockwise) direction around the centre pixel of the neighborhood considering each pixel.
6. Should the considered pixel be the start pixel, conciser exiting the procedure.
7. Where the considered pixel is black, make this the centre pixel and set the considered pixel to the previously considered pixel.
8. repeat 4.

Figure 79: Moore Neighbor Algorithm interpretation

Figure shows the 8 neighbors of p0 are ordered p1, p2, …, p8 and represents the order in which considered pixels around p0 are visited.



80: Working of Moore Neighbor tracing algorithm. (Samet and Hancer, 2012)

(Rajashekar Reddy, Amarnadh, and Bhaskar, 2012) details the following algorithm.

```
Input: A square tessellation, T, containing a connected component P of black cells.
Output: A sequence B (b 1 , b 2 ,..., b k ) of boundary pixels i.e. the contour.

Define (a) to be the Moore neighborhood of pixel a.
Let p denote the current boundary pixel.
Let c denote the current pixel under consideration i.e. c is in M (p).

Begin
Set B to be empty.

From bottom to top and left to right scan the cells of T until a black pixel, s, of
P is found.

    Insert s in B.
    Set the current boundary point p to s i.e. p=s
    Backtrack i.e. move to the pixel from which s was entered.

    Set c to be the next clockwise pixel in M(p).
        While c not equal to s
        do
            If c is black
                insert c in B
                set p=c
                backtrack (move the current pixel c to the pixel from which p was
entered)
            else
             advance the current pixel c to the next clockwise pixel in M(p)
        end while
End
```

Figure 81: Moore's Neighbor tracing algorithm (Rajashekar Reddy, Amarnadh, and Bhaskar, 2012)

From this algorithm the following pseudo code has been developed.

```
// Moore Neighborhood Offsets.
Offsets[] = {(-1,1), (0,1),  (1,1),
             (-1,0), (0,0),  (1,0)
             (-1,-1),(0,-1), (1,-1)}

traced = [];

for each pixel x, y
    if pixel == black
        matched? = yes
        start = pixel
        traced[0] = start
    else
        last = pixel

    if matched? exit loop

center = start
consideration = last
consideration offset = last - start.

while stopping criteria has not be met
    if consideration is black
        traced add consideration
        center = consideration
        consideration = current + next offset.
    else
        backtrack = consideration
        consideration = current + next offset.
```

Figure 81: Moore Neighbor Pseudo Code

IMPLEMENTATION.

The first part of the implementation of this algorithm represents step 1 and 2 detailed in table:

As with spatial filtered developed and detailed on 5.2, a representation is made from the provided subject. Each pixel of the representation is scrutinized in order to find the first black pixel, the coordinates of which are stored. At each unsuccessful pass, reference is maintained to the last unsuccessful pixel match, this will determine the first backtrack position.

```
- (NSArray*) mooreNeighborContourTraceOfImage:(NSImage*)image
{
    /* get the bitmap data from image representation. */

    int width = image.size.width;
    int height = image.size.height;
    int searchPixel = 0; // to find black pixels.
    BOOL match = NO;
    int x = 0, y = 0;

    NSPoint start, last;
    NSMutableArray* points = [[NSMutableArray alloc] init];

    for (y = 0; y < height; y++ )
    {
        for (x = 0; x < width; x++ )
        {
            index = x + y * width;

            if ( data[index] == searchPixel )
            {
                match = YES;
                start = NSMakePoint(x, y);
                NSValue *p = [NSValue valueWithPoint:start];
                [points addObject:p];
                break;
            } else {
                last = NSMakePoint(x, y);
            }

        }
        if ( match ) break;
    }
```

Figure 82: Moore Neighbor Implementation.

Once the first black pixels has been found, the starting values for the trace are determined. The `backtrackOffset` is calculated as the difference between the matched pixel and the last successful pixel and is used to find the current position in the p1, p2, p3, …, p8 offset order, this will determine the next offset around p0 that will be scrutinised. Here step 3 and 4 of table: are represented.

```
int next = 0;
NSPoint offsets[] = {NSMakePoint(-1, -1),
                     NSMakePoint(-1, 0),
                     NSMakePoint(-1, 1),
                     NSMakePoint(0, 1),
                     NSMakePoint(1, 1),
                     NSMakePoint(1, 0),
                     NSMakePoint(1, -1),
                     NSMakePoint(0, -1)};

NSPoint current = start;
NSPoint consider = last;
NSPoint backtrackPosition = last;
NSPoint backtrackOffset = NSMakePoint(last.x - start.x, last.y - start.y);

// find the position
for ( int i = 0; i < 8; i++ )
{
    if ( CGPointEqualToPoint(backtrackOffset, offsets[i]) )
    {
        next = i;
    }
}
```

Figure 83: Moore Neighbor Implementation Continued

The trace will now begin and continue until the stopping criteria is met, representing steps 4 and 5.

At each successful match of a black pixel the points of the pixel are collected. Although duplicate points are not wanted for the purpose of creating the letterform template, rather then query the collection at each successful match, the collection will be reduced to a ordered set before being returned. Collecting each point regardless of duplication also ensures the order in which pixels are collected will be maintained, this will be an important factor when reducing the collected points from a set of pixel coordinates to bezier curves.

```
BOOL run = YES;
while ( run )
{
    index = consider.x + consider.y * width;

    if ( data[index] == searchPixel )
    {
        // stopping criteria
```

```
            // stopping criteria
            if ( CGPointEqualToPoint(start, consider) )
            {
                run = NO;
                break;
            }

            // collect the point.
            NSValue *val = [[NSValue alloc] init];
            val = [NSValue valueWithPoint:consider];
            [points addObject:val];

            // reset the center
            current = consider;
            backtrackOffset = NSMakePoint(backtrackPosition.x - current.x,
backtrackPosition.y - current.y);

            // find the offset for the backtrack position
            for ( int i = 0; i < 8; i++ )
            {
                if ( CGPointEqualToPoint(backtrackOffset, offsets[i]) )
                {
                    next = i;
                    break;
                }
            }
            consider = NSMakePoint(current.x + offsets[next].x, current.y +
offsets[next].y);
        } else {
            // store the last position to concider
            backtrackPosition = consider;

            // move to the next offset
            next++;
            if ( next == 8 ) next = 0;
            consider = NSMakePoint(current.x + offsets[next].x, current.y +
offsets[next].y);
        }
    }
```

Figure 84: Moore Neighbor Trace Loop Implementation

In order to remove duplicate values `[NSOrderedSet orderedSetWithArray:points]` provides unique values while maintaining order.

```
    NSOrderedSet* set = [NSOrderedSet orderedSetWithArray:points];
    NSArray* distinctPoints = [set array];

    return distinctPoints;
```

Figure 85: Reducing to distinct points.

RESULT.

To test the result of this implementation a subject letterform from the typeface set that will be the basis of the letterform template is thresholded in order to remove aliasing. The subject is then thinned using the Zhang Suen algorithm to reduce the subject representation down to a single pixel width.

Figure 86: Starting letterform subject (thresholded to remove aliasing)

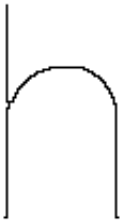


Figure 87: Thinned subject

The data produced from the trace has been drawn to the screen point by point using the `NSBezierPath` class of the Cocoa framework in order to understand the success of the trace.

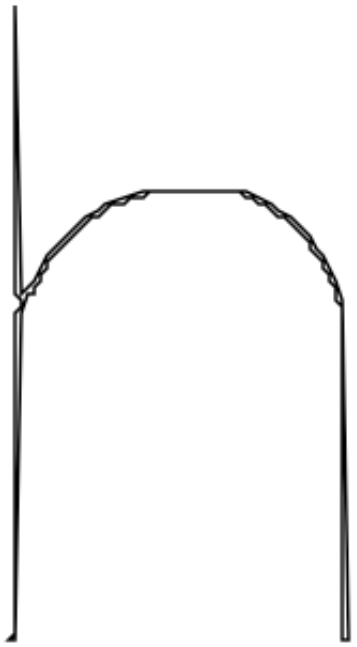


Figure 88: Traced result

As shown in figure 88 although the application of the tracing algorithms has correctly traced the thinned subject, the result has not been as desired. As stated … the desired was to produce a single line representation of the subject, this would then be used as a template from manipulation. However, due to failure to fully consider the results produced by the thinning of the original thresholded subject and data collected from the trace. As clearly displayed in Figure 89, the structure supplied by the thinning operation where curves and diagonal lines make up the structure these are joined by a two pixel thickness.

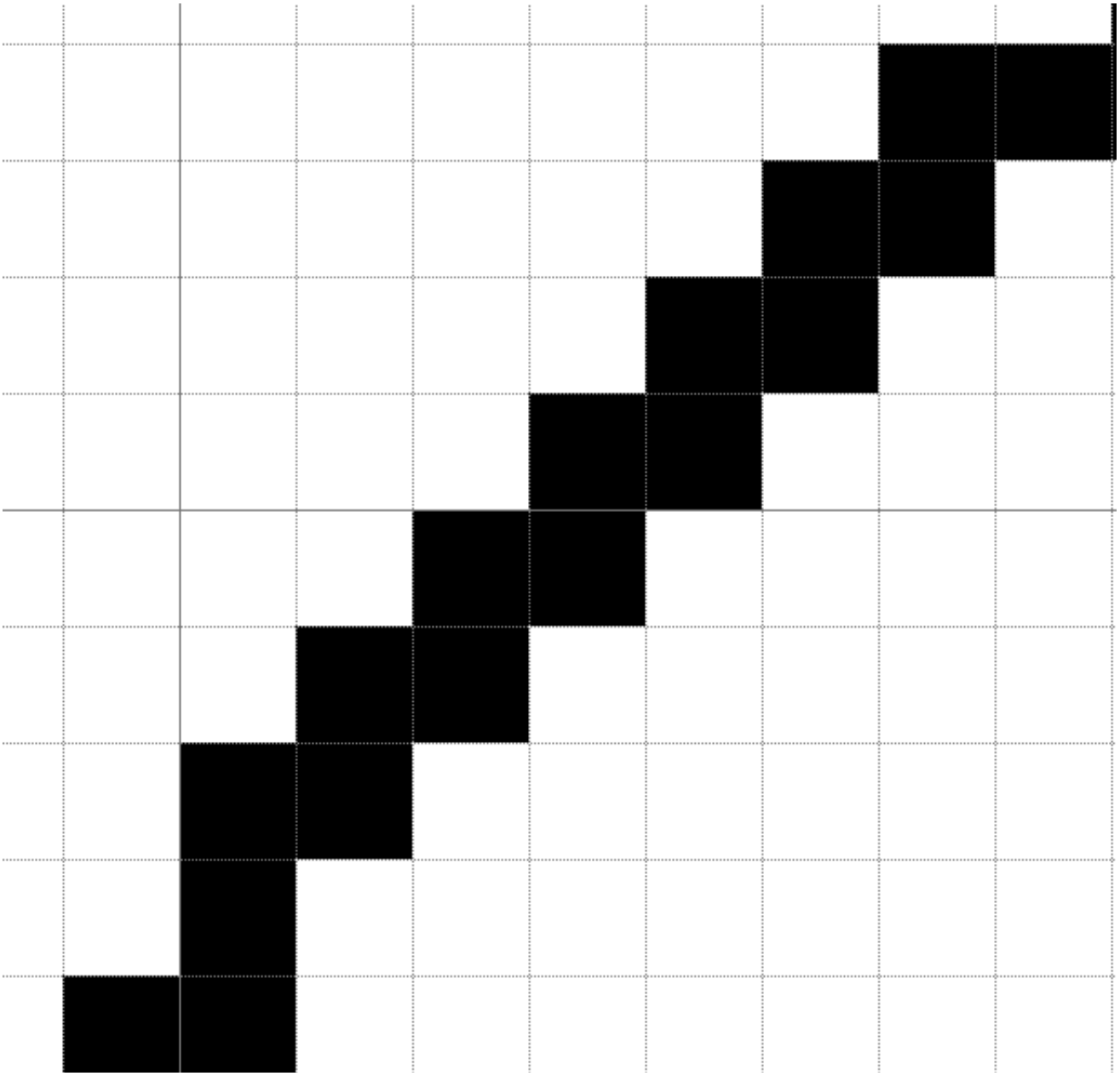
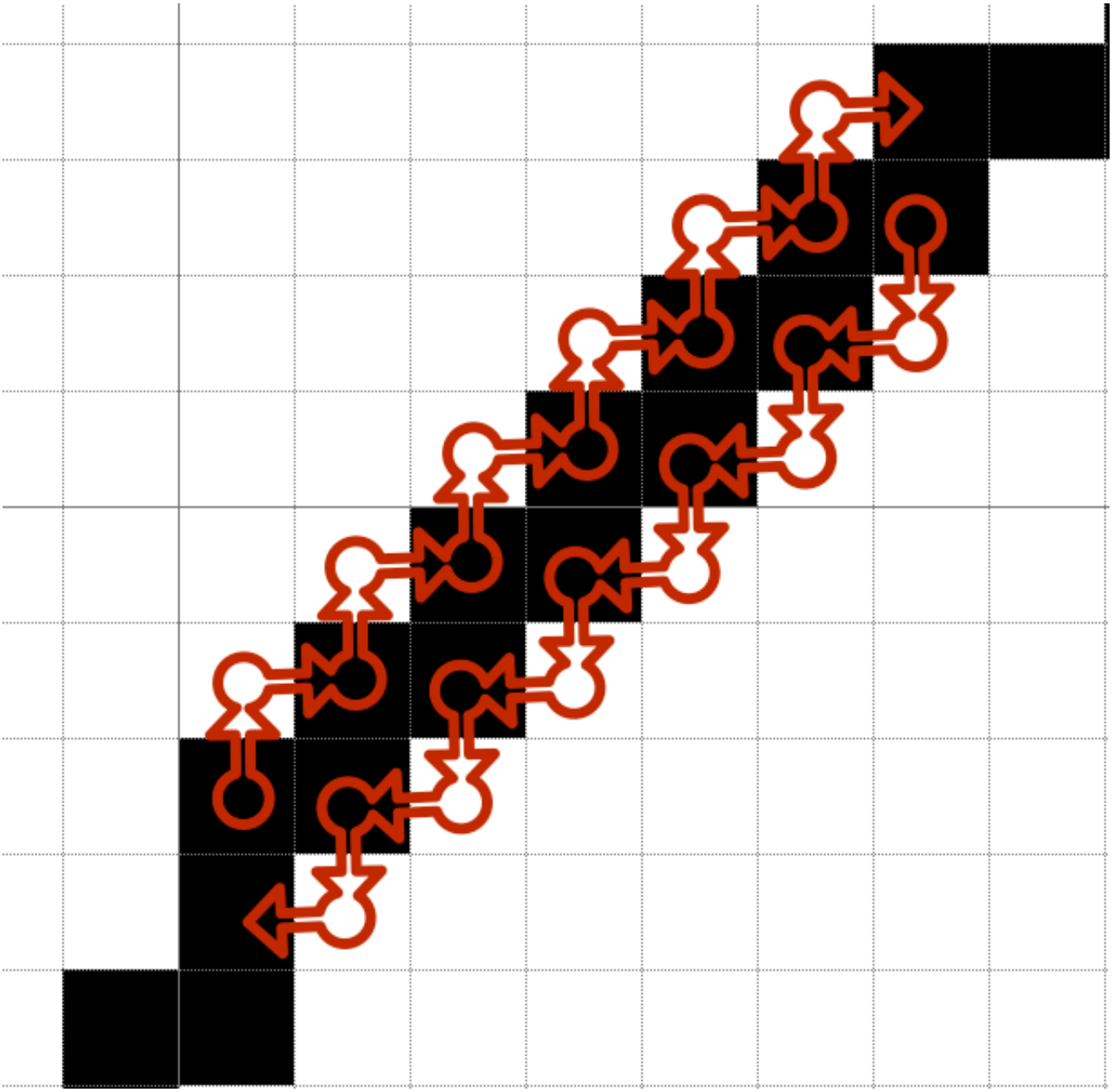


Figure 89: Thinned subject

Therefore, the tracing of along one side of the structure and then the other side of the structure would not meet on duplicate points, meaning both sets of points would be maintained after removal of duplicate points. (Figure 90)



5.4 IMAGE SEGMENTATION AND ANALYSIS

With the subject clear of noise and non-text objects and then thresholded steps need to be take in order to segment the subject into lines and individual words. This will then create the basis for establishing the key required letter form attributes, cap-height, ascender height, and x-height.

5.4.1 LINE SEGMENTATION.

In order to achieve the segmentation of lines of handwritten text, a method of “Projection-profile” has been attempted.

(Likforman-Sulem, Zahour, and Taconet, 2006) define the projection profile file as simply:

> *“summing pixel values along the horizontal axis for each y value.*

Therefore:

PSEUDO CODE.

```
for each row y

    set the counter to zero

    for each column x

        if pixel x, y is black.
            count++

    row(y) = count
```

Figure 91: Projection profile pseudo code

IMPLEMENTATION.

```
// create representation from image.


// create an array to capture values.
int* output = malloc(sizeof(int) * height);

// For each row
for ( int y = 0; y < height; y++ )
{
    int count = 0;

    // for each column
    for ( int x = 0; x < width; x++ )
    {
        int index = x + (y * width);

        // count each black pixel
        int t = input[index];
        if ( t == 0 )
        {
            count++;
        }
    }

    // at the end of the row cache the result.
    output[y] = count;
}
```

Figure 92: Projection profile Implementation

RESULT.

Result of projection profile has successfully provided the positions at which lines of text start and stop along the y-axis.

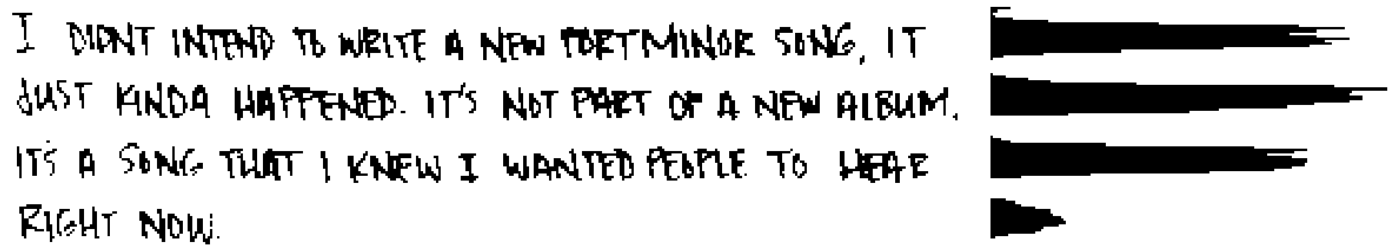


Figure 93: Result of projection profile

However, the above subject represents a clear separation between lines, largely due to each letter form being uppercase and therefore absent of descender strokes.

Applying the same method to a letter set of mixed case, overlapping has occurred and the separation less clear. In such a case an alternative method will need to be found.

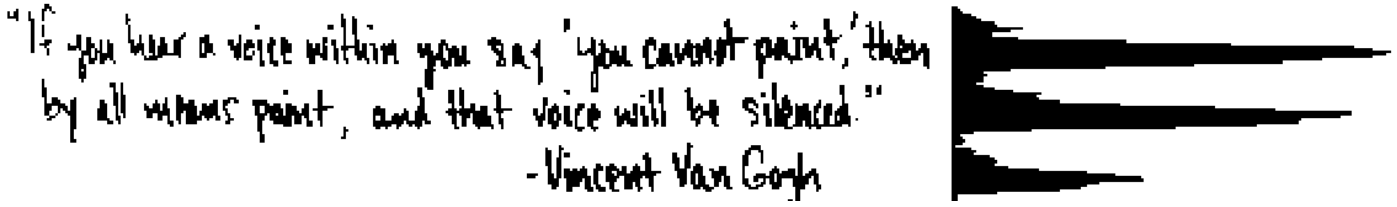


Figure 94: Overlapping lines

The use of projection profile of the purpose of analysis could however prove to be of worth. As displayed in Figure … and even Figure … it is possible to recognise the various points of change occur relative to the attributes sought.

Figure 95: Projection profile for analysis

6. EVALUATION.

The implemented features have been evaluated, determining their success as an implementation and as part of this project. Following feature evaluation, the project is evaluated from a software engineering perceptive.

6.1 FEATURE EVALUATION.

6.1.1 CROPPING IMAGES.

The cropping of images in order to remove non-text elements or textual elements that where unwanted has been successful. Although, in later versions if this application it is possible that non textual areas, such as the background table (Figure: 96), could possibly be removed programmatically, allowing the user to specify areas of text to analysis would remain. In addition, the cropping functionality implementation allowed for progress in other areas of the project that would otherwise have been held up by evaluating the removal of non text areas.

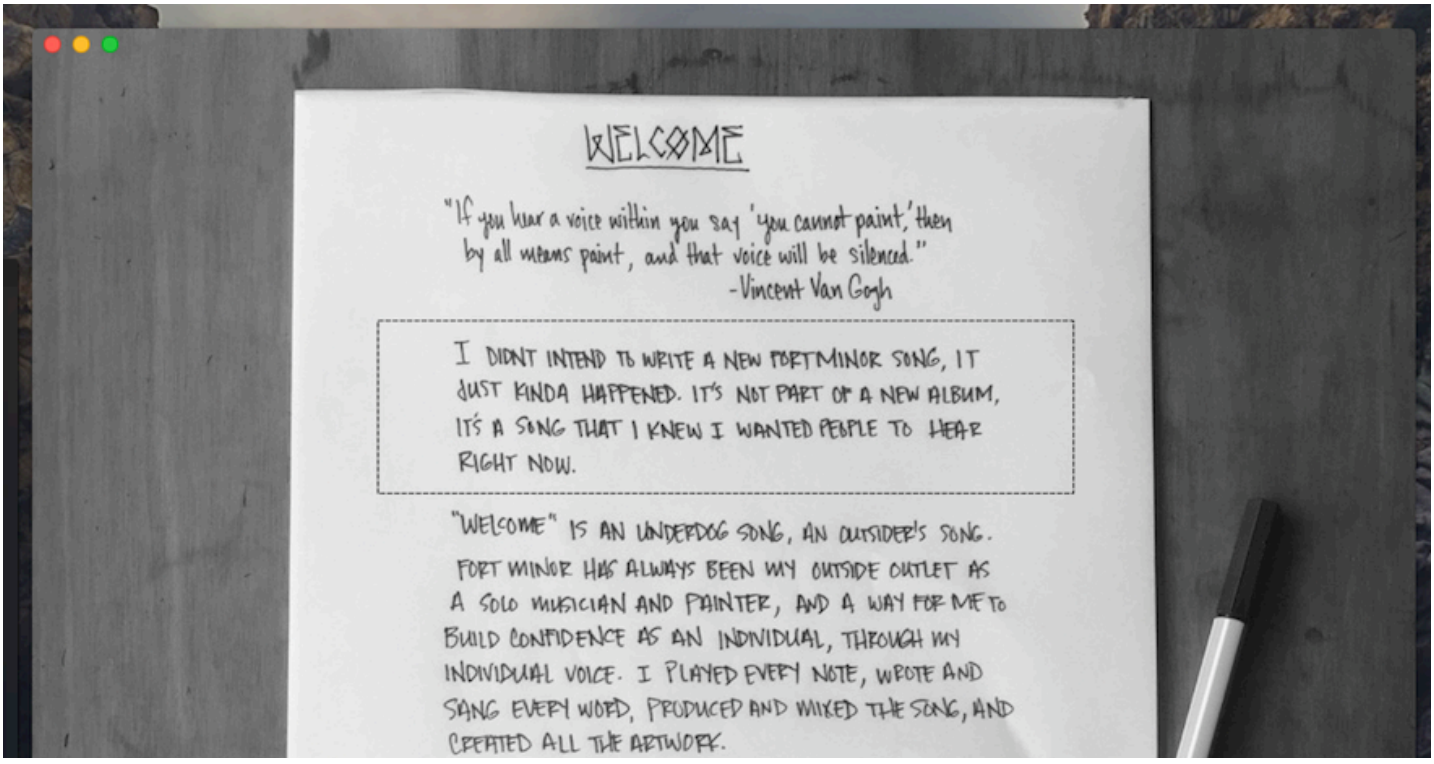


Figure 96: User selecting an area to crop.

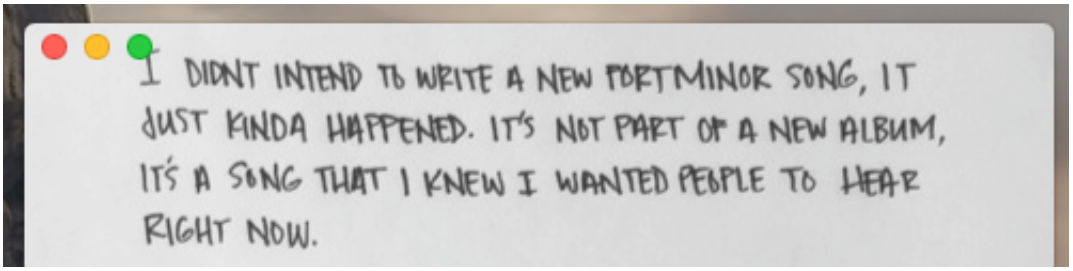


Figure 97: The cropped region

6.1.2 NOISE REMOVAL FEATURES AND THRESHOLDING.

Although the implementation of spatial filter for the purpose of noise removal has been reasonably successful, due to that subject matter of handwritten text being dark colours on light backgrounds, the removal of noise is a minor issue when a subject will be thresholded.

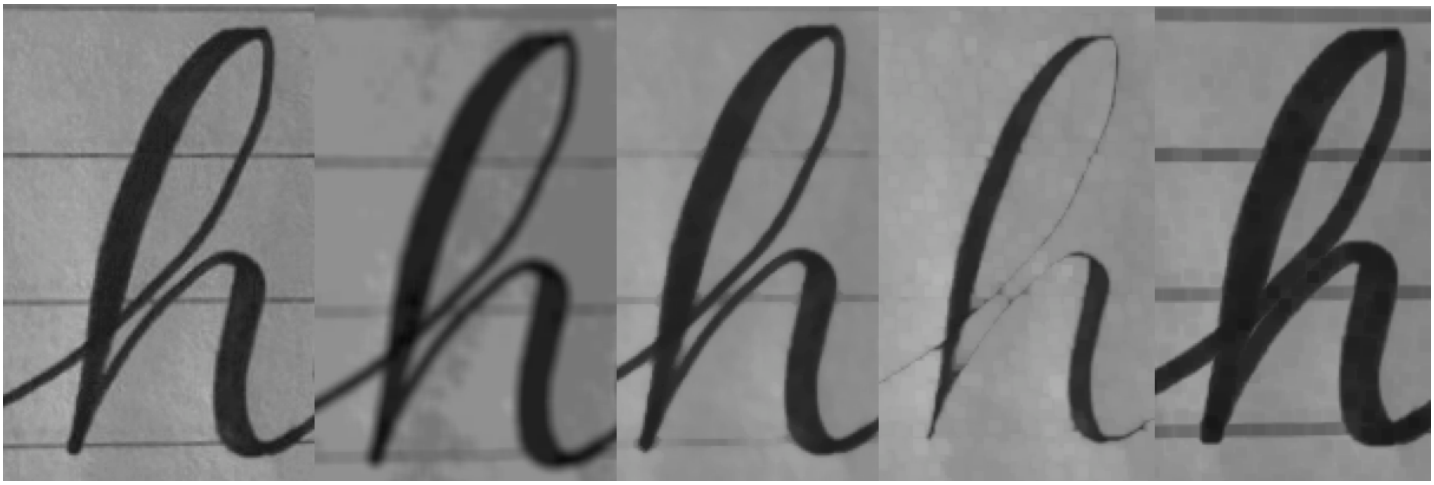


Figure 98: (l-r) Subject Image, Averaging, Median, Min, Max filters

With each of the examples of subject and the filters implemented (Figure: 98) and then thresholded to the same value (Figure: 99) from this limited set it is possible to see that output presenting the fewest artifact is the thresholded subject image, followed by the median filter (with the median filter showing artifact elsewhere [5.2.3.1.3]). Therefore, it may have been possible to forgo the the implementation of filters.

Figure 99: (l-r) Subject Image, Averaging, Median, Min, Max filters thresholded

6.1.3 MORPHOLOGY.

As with [6.1.2], the implementation of morphological operations has been successful, applying the operations to the subject image has shown minimal benefit over thresholding for creating a cleaner representation to work with.

Figure 100: (l-r) Subject Image, Erosion, Dilation, Opening, Closing

However, taking a broken subject (Figure: 101) and applying a dilation operation, broken lines are connected, and missing data in the object is filled

Figure 101: Broken Image

Figure 102: Dilated Broken Image

6.1.3.1 THINNING.

The requirement for thinning of the subject came from the desire to created a skeleton letterform template programmatically in addition would possibly have allowed for the analysis of curves and connections in the subject letter forms.

The results of applying the ZhangSuen algorithm to the subject where not as successful as initial hoped (Figure: 103) presenting both “Parasitic elements” and line distortion around the high contrast strokes of the subject.

The “Parasitic elements” presented could be remedied through the implementation of “Pruning” operations (Gonzalez and Woods, 2002, p.545), however, the distorted lines/shapes such as the entry to the “o” character and the join at the entry of the “h” would need to be straightened to present a more ideal representation of the subject.

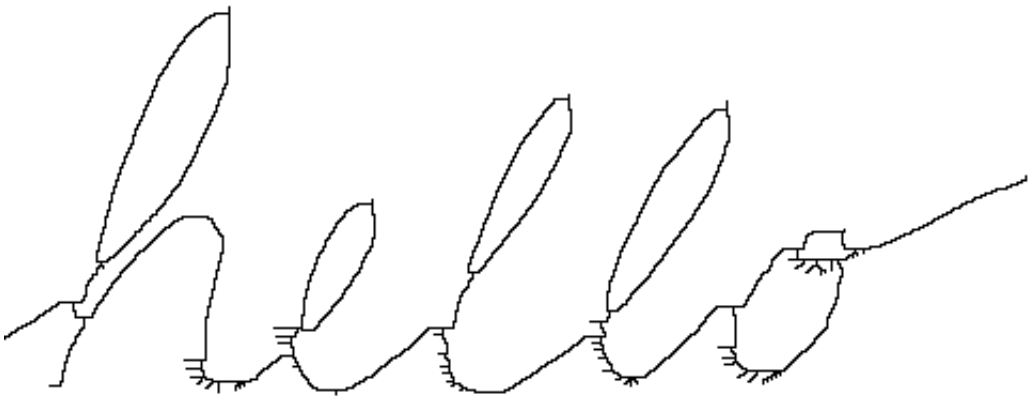


Figure 103: Thinning Evaluation

6.1.4 LETTERFORM COMPOSITION.

Two possible methods of creating a letterform template on which extracted attributes would be applied:

1. Programmatically extract forms from an available typeface.
2. Create the template by attaining point information from PaintCode.app (PixelCut, 2016)

PaintCode.app will take an SVG sample and generate the points presented in Objective-c code (Figure: 104)

Despite access to this application, the decision was made to apply method 1 and create the template by implementing a tracing algorithm in order to extract the paths of a bitmap representation of a typeface.

```
// Generated by PaintCode.App

  NSBezierPath* hPath = [NSBezierPath bezierPath];
  [hPath moveToPoint: NSMakePoint(0.06, 0)];
  [hPath lineToPoint: NSMakePoint(1.59, 0)];
  [hPath lineToPoint: NSMakePoint(1.59, 20.16)];
  [hPath curveToPoint: NSMakePoint(13.41, 32.69)
        controlPoint1: NSMakePoint(1.59, 27.53)
        controlPoint2: NSMakePoint(6.5, 32.69)];
  [hPath curveToPoint: NSMakePoint(24.47, 21.66)
        controlPoint1: NSMakePoint(19.97, 32.69)
        controlPoint2: NSMakePoint(24.47, 28.5)];
  [hPath lineToPoint: NSMakePoint(24.47, 0)];
  [hPath lineToPoint: NSMakePoint(26, 0)];
  [hPath lineToPoint: NSMakePoint(26, 21.72)];
  [hPath curveToPoint: NSMakePoint(13.59, 34.12)
        controlPoint1: NSMakePoint(26, 29.25)
        controlPoint2: NSMakePoint(20.97, 34.12)];
  [hPath curveToPoint: NSMakePoint(1.66, 25.69)
        controlPoint1: NSMakePoint(7.75, 34.12)
        controlPoint2: NSMakePoint(3.09, 30.66)];
  [hPath lineToPoint: NSMakePoint(1.59, 25.69)];
  [hPath lineToPoint: NSMakePoint(1.59, 47.06)];
  [hPath lineToPoint: NSMakePoint(0.06, 47.06)];
  [hPath lineToPoint: NSMakePoint(0.06, 0)];
  [hPath closePath];
  [hPath setMiterLimit: 4];
  [hPath setWindingRule: NSEvenOddWindingRule];
  [fillColor setFill];
  [hPath fill];
```

Figure 104: Path Generated By PaintCode

6.1.4.1 CONTOUR TRACING.

The use of a contour tracing algorithm [5.3.1.1] presented three issues in the creation of the template.

1. Inner shapes of character would be ignored.
2. Incorrect assumptions where made about the structure of a thinned representation [5.3.1.1].
3. Assigning control points at which attributes, extracted during analysis, would be applied to the template.

Despite these issues, the implementation of the Moore Neighbor algorithm was successful. Therefore, this was a specification failure due to not correctly understanding or evaluating the problem, rather then application of the algorithm.

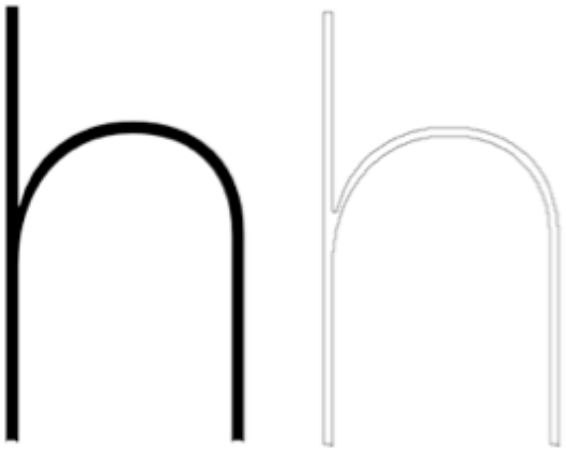


Figure 105: Contour Traced Border

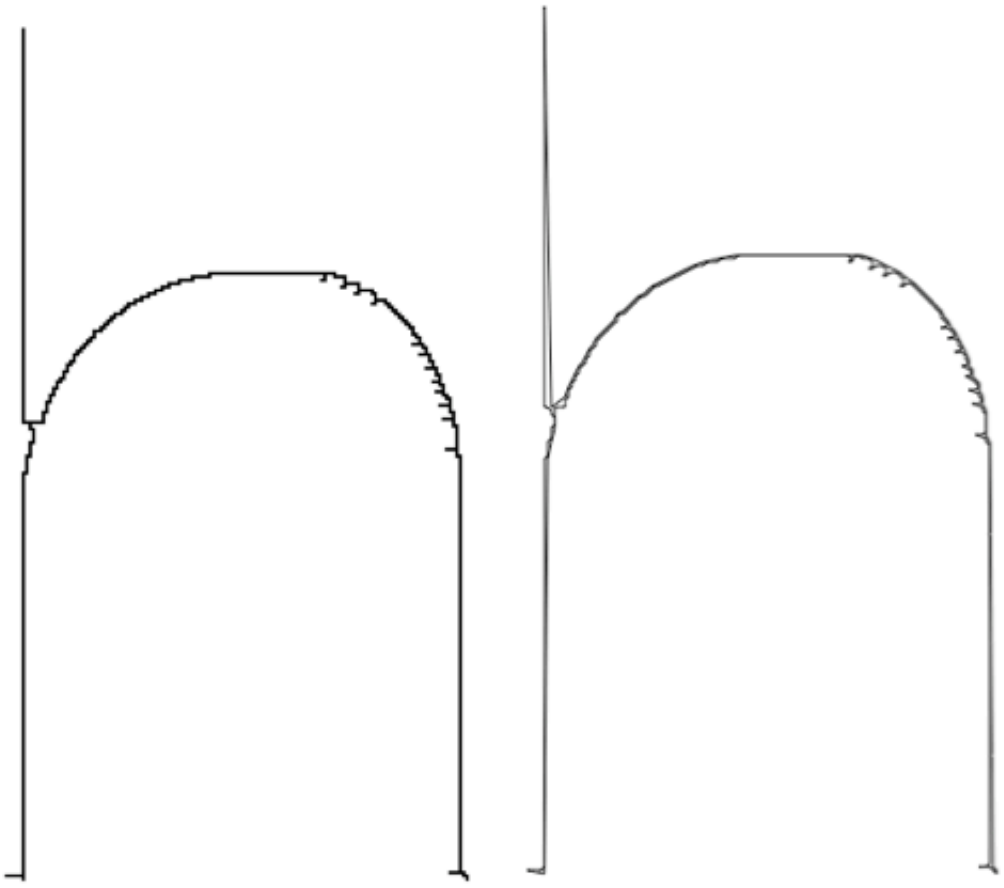


Figure 106: Contour Traced Thinned Image

6.1.5 IMAGE ANALYSIS.

Due to time constraints and the lack of a letterform template, only minimal analysis has been achieved.

6.1.5.1 PROJECTION PROFILE.

Projection profile provides a simple method of extracting four key letterform attributes: Cap-height, x-height, ascender height, descender length. However, the effectiveness of this method is yet to be fully determined as the results have not yet been fully analysed.

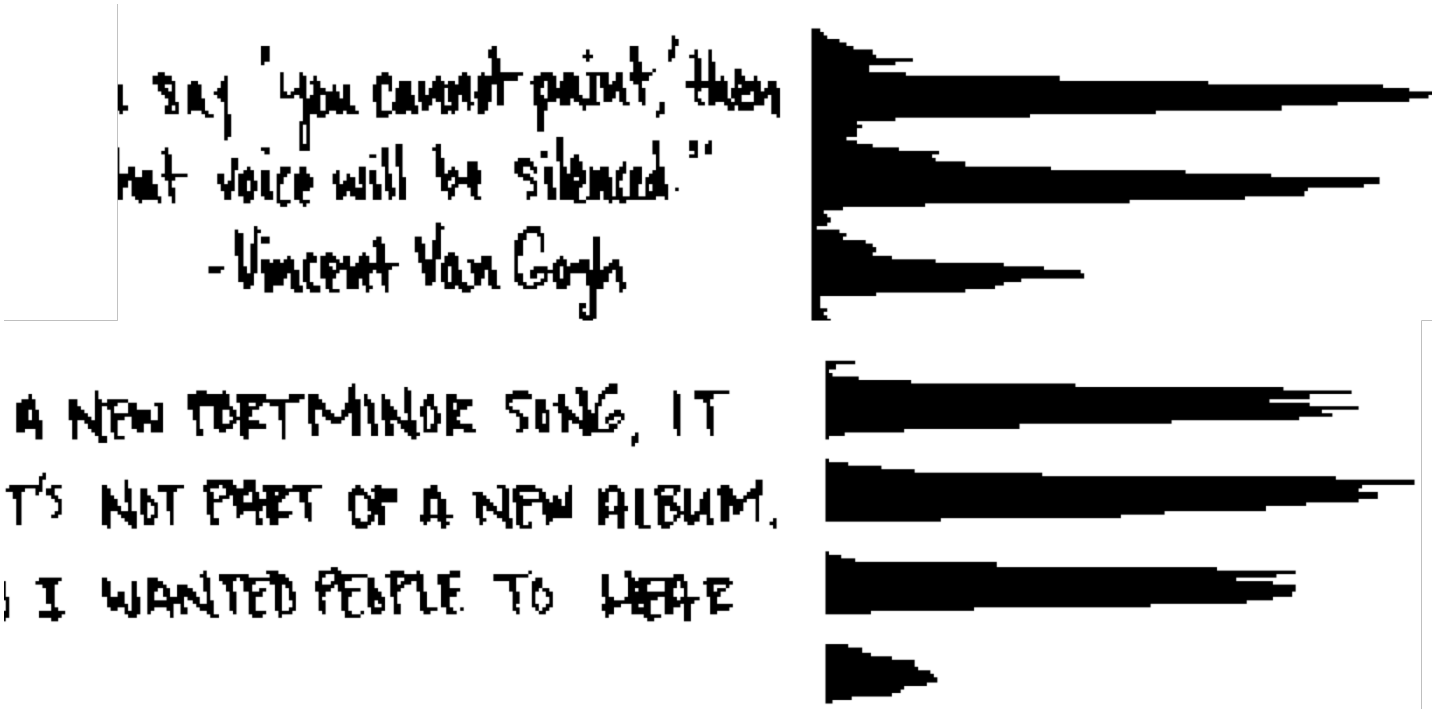


Figure 107: Projection Profile Line Segmentation

Figure 108: Projection Profile Letterform Analysis

6.2 PROJECT EVALUATION.

The inability to establish an output typeface from this implementation is due to failures in the project in three areas.

6.2.1 PROJECT FAILURES.

1. DELAY IN ESTABLISHING THE METHODOLOGY FOR RESEARCH AND DEVELOPMENT.

In the initial stages of this project attempts where made to understand the application of image processing in a detailed form. However, this had lead to an over abundance of information being consumed, therefore creating: a) bewilderment and excess noise, b) time spent learning tools that where not required, c) difficulty establishing the correct path of action, and d) difficultly understanding how the operations being researched would be translated to code.

This has meant development of the project began at a later then planned stage.

Only once development of the simpler operations such a spatial filters had began, did the correct methodology cycle of research, development, implementation, become clear. This methodology was then carried throughout the rest on the project.

2. THE INCORRECT ORDER OF RESEARCH FOCUS.

At an early stage, this project was broken down into 4 key sub components, Image Processing and analysis, Letterform Composition, Generated output, UI Development. Although breaking down the problem into these areas was correct, initial planning decided that the order of development would follow that of the order of operations within the system to complete the required task. This order being:

1. Image Pre-processing.
2. Image Analysis.
3. Template Creation.
4. Letterform Composition.
5. Generated Output

With UI Development being implemented in tandem.

In addition to establishing this order it was also determined that this would be a linear process of implementing 1–5 in order.

Both the order of implementation and the linear movement through the sub components lead to:

1. Being the first component, image pre-processing became the core focus of research.
2. Slow movement through the components.

Although a large part of this project would have had to be devoted to Image Processing. As discovered at a late stage and noted in this evaluation, the application of smoothing filters seems largely unneeded due to the subject mater and the effectiveness of even a global threshold function.

Ideally, the real focus should have been on Image Analysis and referencing image pre-processing techniques when required as well as a larger onus being have also been placed on the template creation component.

Therefore a non-liner approach with a greater focus on Image analysis was required.

3. INABILITY TO CORRECTLY ESTABLISH THE PROBLEM PRIOR TO IMPLEMENTATION OF SOME FEATURES.

In addition to notes about the requirement for smoothing filters, the effectiveness of a Moore neighbor tracing algorithm on a thinned items as a basis for the letterform template had been grossly mis-assumed. Although smoothing filters could still prove of worth to this project with the introduction of a wider verity of subject image quality, the application of the tracing algorithm for template creation displays a failure to simply analyse the pixel structure of a thinned image. [5.3.1.1]

Implementation of these features where time consuming and therefore reduced available time that could have been applied elsewhere.

6.2.2 PROJECT SUCCESSES.

Despite falling short of any generated output, each of the tools created are performant and achieve their own goals as required. Therefore, these are all successes in their own right and will provide application in future implementations.

The development of the demonstration UI provides a simple mechanism for apply the created tools to a supplied image. (Figure: 109)

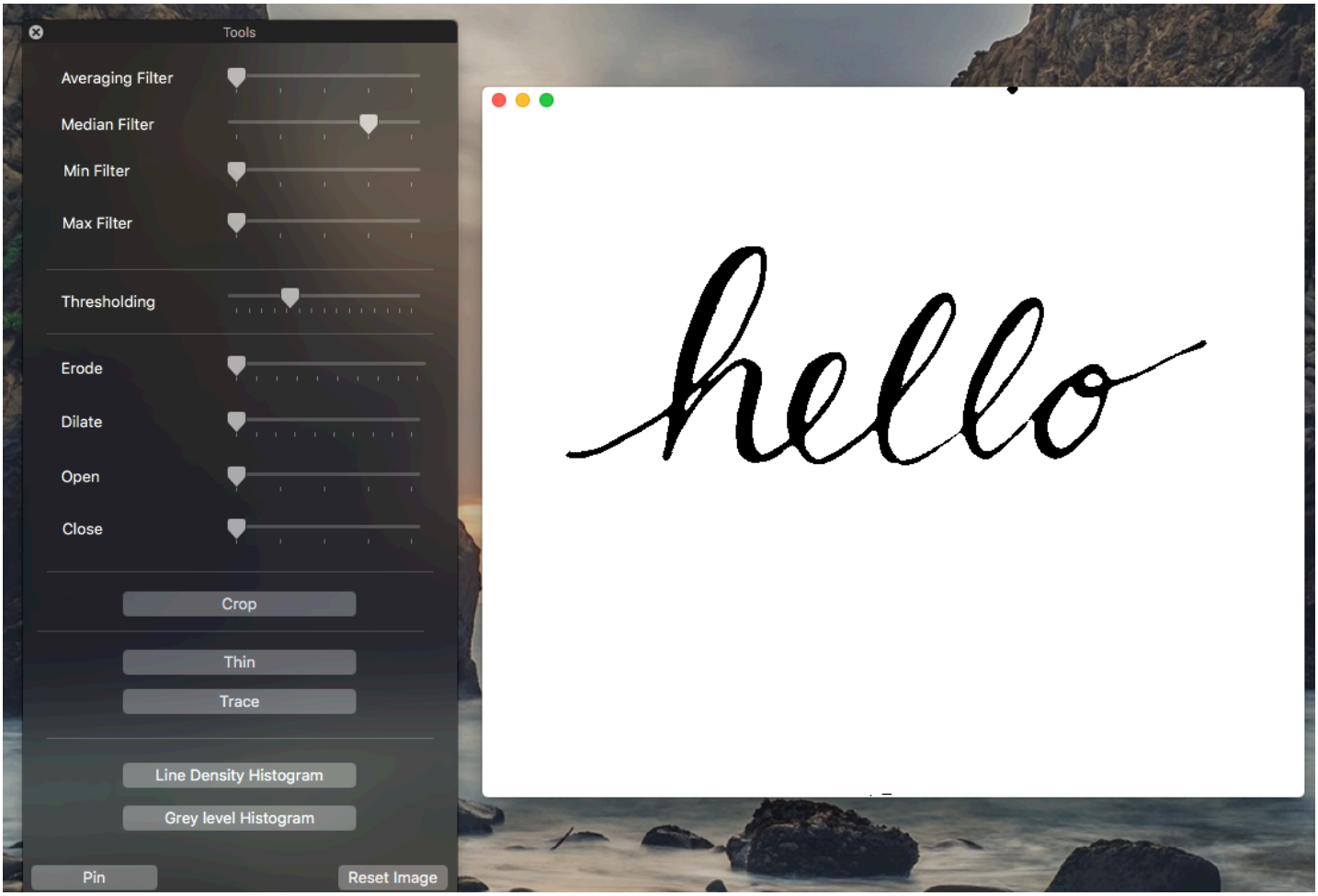


Figure 109: Demonstration UI

8.1 FINAL SOURCE CODE.

```
//
//  AppDelegate.h
//  ImageCropUI
//
//  Created by James Mitchell on 21/01/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import <Cocoa/Cocoa.h>

@class MainWindowController;

@interface AppDelegate : NSObject <NSApplicationDelegate>
{
    MainWindowController* mainWindowController;
}

- (IBAction) newDropZone:(id)sender;
- (IBAction) showToolWindow:(id)sender;

@end
```

```
//
//  AppDelegate.m
//  ImageCropUI
//
//  Created by James Mitchell on 21/01/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import "AppDelegate.h"
#import "MainWindowController.h"

@interface AppDelegate ()

@end

@implementation AppDelegate

- (void) applicationWillFinishLaunching:(NSNotification *)notification
{
    mainWindowController = [[MainWindowController alloc]
initWithWindowNibName:@"MainWindow"];
    [mainWindowController showWindow:self];
}

- (BOOL)applicationShouldTerminateAfterLastWindowClosed:(NSApplication *)sender
{
    return YES;
}

- (IBAction) newDropZone:(id)sender
{
    [mainWindowController changeToDropZoneController];
}

- (IBAction) showToolWindow:(id)sender
{
    [mainWindowController displayToolWindow];
}

@end
```

```
//
//  IntArrayUtil.h
//  ImageProcessingCLI
//
//  Created by James Mitchell on 06/02/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface IntArrayUtil : NSObject

+ (void) bubbleSort:(int *)arr ofSize:(int)size;
+ (int) getMedianFromArray:(int [])arr ofSize:(int)size;
+ (int) maxFromArray:(int [])arr ofSize:(int)size;
+ (int) minFromArray:(int [])arr ofSize:(int)size;
+ (int*) zeroArrayOfSize:(int)size;

@end
```

```
//
//  IntArrayUtil.m
//  ImageProcessingCLI
//
//  Created by James Mitchell on 06/02/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import "IntArrayUtil.h"

@implementation IntArrayUtil

#pragma mark -
#pragma mark Sorting

+ (void) sort:(int *)arr ofSize:(int)size
{
    [self bubbleSort:arr ofSize:size];
}

+ (void) bubbleSort:(int *)arr ofSize:(int)size
{
    BOOL swap;
    int temp;
```

```
    do {
        swap = NO;

        for ( int i = 0; i < (size - 1); i++ )
        {
            if ( arr[i] > arr[i + 1] )
            {
                temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
                swap = YES;
            }
        }
    } while (swap);
}

+ (int) maxFromArray:(int [])arr ofSize:(int)size
{
    int max = arr[0];
    for ( int i = 0; i < size; i++ )
    {
        int val = arr[i];
        if ( val > max )
        {
            max = val;
        }
    }

    return max;
}

+ (int) minFromArray:(int [])arr ofSize:(int)size
{
    int min = arr[0];
    for ( int i = 0; i < size; i++ )
    {
        int val = arr[i];
        if ( val < min )
        {
            min = val;
        }
    }

    return min;
}
```

```
+ (int) getMedianFromArray:(int [])arr ofSize:(int)size
{
    int middle = (int)(size / 2);

    [self bubbleSort:arr ofSize:size];

    return arr[middle];
}

+ (int*) zeroArrayOfSize:(int)size
{
    int* output = malloc(sizeof(int) * size);

    for ( int i = 0; i < size; i++ )
    {
        output[i] = 0;
    }
    return output;
}

@end
```

```
//
//  DrawingView.h
//  ImageCropUI
//
//  Created by James Mitchell on 14/04/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import <Cocoa/Cocoa.h>

@interface DrawingView : NSView
{
    NSArray* drawingData;
}

@property (nonatomic) NSArray* drawingData;

- (void) setDrawingData:(NSArray*)data;

@end
```

```
//
//  DrawingView.m
//  ImageCropUI
//
//  Created by James Mitchell on 14/04/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import "DrawingView.h"

@implementation DrawingView

@synthesize drawingData;

- (id) initWithFrame:(NSRect)frameRect
{
    self = [super initWithFrame:frameRect];

    if ( self )
    {
        // init code.
    }

    return self;
}

- (void) setDrawingData:(NSArray *)data
{
    drawingData = data;
}

- (void)drawRect:(NSRect)dirtyRect {
    [super drawRect:dirtyRect];

//    NSRect viewSize = self.bounds;

    [[NSColor whiteColor] setFill];
    NSRectFill(dirtyRect);

    if ( !drawingData ) return;

    // REFERENCE:
//developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CocoaDrawingGuide/Transforms/Transforms.html#//apple_ref/doc/uid/TP40003290-CH204-BCIHDAIJ
    NSRect frameRect = [self bounds];
    NSAffineTransform* flip = [NSAffineTransform transform];
```

```
    [flip translateXBy:0.0 yBy:frameRect.size.height];
    [flip scaleXBy:1.0 yBy:-1.0];
    [flip concat];

    NSColor* fillColor = [NSColor colorWithCalibratedRed: 0 green: 0 blue: 0 alpha:
1];
    NSBezierPath* path = [NSBezierPath bezierPath];

    NSPoint n;
    NSValue *value;

    value = [drawingData objectAtIndex:0];
    [value getValue:&n];
    [path moveToPoint:n];

    for ( int i = 0; i < [drawingData count]; i++ )
    {
        value = [drawingData objectAtIndex:i];
        [value getValue:&n];
        [path lineToPoint:n];
    }

//    [hPath curveToPoint: NSMakePoint(13.41, 32.69) controlPoint1:
NSMakePoint(1.59, 27.53) controlPoint2: NSMakePoint(6.5, 32.69)];
//    [hPath curveToPoint: NSMakePoint(24.47, 21.66) controlPoint1:
NSMakePoint(19.97, 32.69) controlPoint2: NSMakePoint(24.47, 28.5)];

    [path closePath];
    [path setWindingRule: NSEvenOddWindingRule];
    [path setLineWidth:0.5];
    [fillColor setStroke];
    [path stroke];
}

@end
```

```
//
//  ImageManipulationView.h
//  ImageCropUI
//
//  Created by James Mitchell on 09/04/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import <Cocoa/Cocoa.h>

@interface ImageManipulationView : NSImageView
{

}


@end
```

```
//
//  ImageManipulationView.m
//  ImageCropUI
//
//  Created by James Mitchell on 09/04/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import "ImageManipulationView.h"

@implementation ImageManipulationView

- (void)drawRect:(NSRect)dirtyRect {
    [super drawRect:dirtyRect];

    // Drawing code here.
    [[NSColor whiteColor] setFill];
}

@end
```

```
//
//  ImageCropView.h
//  ImageCropUI
//
//  Created by James Mitchell on 26/01/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import <Cocoa/Cocoa.h>

@interface ImageCropView : NSImageView
{
    NSPoint start;
    NSPoint current;
    BOOL cropHasStarted;
    NSImage* _croppedImage;
}

@property (nonatomic, strong) NSImage* croppedImage;

@end
```

```
//
//  ImageCropView.m
//  ImageCropUI
//
//  Created by James Mitchell on 26/01/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import "ImageCropView.h"

@implementation ImageCropView

@synthesize croppedImage = _croppedImage;

- (id) initWithFrame:(NSRect)frameRect
{
    self = [super initWithFrame:frameRect];

    if (self)
    {
        cropHasStarted = NO;
```

```
    }

    return self;
}


- (void)drawRect:(NSRect)dirtyRect {
    [super drawRect:dirtyRect];

    if ( cropHasStarted )
    {
        CGFloat width = current.x - start.x ;
        CGFloat height = current.y - start.y;
        NSRect rect = NSMakeRect(start.x, start.y, width, height);

        CGFloat pattern[] = {4.0, 1.0};
        NSBezierPath* path = [NSBezierPath bezierPathWithRect:rect];
        [path setLineDash:pattern count:sizeof(pattern) / sizeof(pattern[0])
phase:0];
        [path stroke];
    }
}


- (void) mouseDown:(NSEvent *)theEvent
{
    NSPoint windowLocation = [theEvent locationInWindow];
    NSPoint viewLocation = [self convertPoint:windowLocation fromView:nil];
    start = viewLocation;

    [self setNeedsDisplay:YES];
}

- (void) mouseDragged:(NSEvent *)theEvent
{
    NSPoint windowLocation = [theEvent locationInWindow];
    NSPoint viewLocation = [self convertPoint:windowLocation fromView:nil];
    current = viewLocation;

    cropHasStarted = YES;
    [self setNeedsDisplay:YES];
}

- (void) mouseUp:(NSEvent *)theEvent
{
    if ( !cropHasStarted ) return;
```

```
    NSSize cropSize;
    cropSize.width = (int)(current.x - start.x);
    cropSize.height = (int)(start.y - current.y);

    int temp;

    // normalise crop position points.
    // these are the positions within the view.
     if ( start.x > current.x )
     {
         temp = start.x;
         start.x = current.x;
         current.x = temp;
     }

    if ( start.y < current.y )
    {
        temp = start.y;
        start.y = current.y;
        current.y = temp;
    }

    _croppedImage = [[NSImage alloc] initWithSize:cropSize];
    [_croppedImage addRepresentation:[self croppedRepresentationOfImage:[self image]
                                                              fromPoint:start
                                                                toPoint:current]];

    [self setImage:_croppedImage];

    [[NSNotificationCenter defaultCenter] postNotificationName:@"ImageCropComplete"
                                                        object:nil];

    cropHasStarted = NO;
    [self setNeedsDisplay:YES];
}

- (NSBitmapImageRep *) croppedRepresentationOfImage:(NSImage *)image
                                          fromPoint:(NSPoint)from
                                            toPoint:(NSPoint)to
{
    int width = (int)(to.x - from.x);
    int height = (int)(from.y - to.y);

    NSRect newRect = NSMakeRect(from.x , to.y, width, height);

    NSBitmapImageRep *representation = [[NSBitmapImageRep alloc]
                                        initWithBitmapDataPlanes: NULL
```

```
                                        pixelsWide: width
                                        pixelsHigh: height
                                        bitsPerSample: 8
                                        samplesPerPixel: 4
                                        hasAlpha: YES
                                        isPlanar: NO
                                        colorSpaceName: NSCalibratedRGBColorSpace
                                        bytesPerRow: width * 4
                                        bitsPerPixel: 32];

    [image lockFocus];
    NSGraphicsContext *context = [NSGraphicsContext
graphicsContextWithBitmapImageRep:representation];
    [NSGraphicsContext saveGraphicsState];
    [NSGraphicsContext setCurrentContext:context];

    [image drawInRect:NSMakeRect(0, 0, width, height)
             fromRect:newRect
            operation:NSCompositeCopy
             fraction:1.0];

    [context flushGraphics];
    [NSGraphicsContext restoreGraphicsState];
    [image unlockFocus];

    return representation;
}

@end
```

```
//
//  DropZoneView.h
//  ImageCropUI
//
//  Created by James Mitchell on 26/01/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import <Cocoa/Cocoa.h>
#import "MainWindowController.h"

@interface DropZoneView : NSView <NSDraggingDestination>
{
    NSImage* image;
}

@property (nonatomic, strong) NSImage* image;

@property (assign) BOOL successDisplay;
@property (assign) BOOL defaultDisplay;
@property (assign) BOOL errorDisplay;

//- (NSString*) name;

@end
```

```
//
//  DropZoneView.m
//  ImageCropUI
//
//  Created by James Mitchell on 26/01/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import "DropZoneView.h"
#import "ImageRepresentation.h"

@class ImageRepresentation;

@implementation DropZoneView

@synthesize image;

- (id)initWithFrame:(NSRect)frameRect
{
```

```
    self = [super initWithFrame:frameRect];

    if (self)
    {
        [self registerForDraggedTypes:[NSArray
arrayWithObject:NSFilenamesPboardType]];
    }

    return self;
}


- (NSDragOperation) draggingEntered:(id<NSDraggingInfo>)sender
{
    // the pastboard and drag operation.
    NSPasteboard* pasteboard;
    NSDragOperation sourceDragInformation;

    // get the drag information from the sender.
    sourceDragInformation = [sender draggingSourceOperationMask];
    // get the pastebaord information from the sender.
    pasteboard = [sender draggingPasteboard];

    if ( [[pasteboard types] containsObject:NSFilenamesPboardType] )
    {
        if ( sourceDragInformation & NSDragOperationCopy )
        {

            self.successDisplay = YES;
            [self setNeedsDisplay:YES];

            return NSDragOperationCopy;
        }
    }

    return NSDragOperationNone;
}

- (void) draggingExited:(id<NSDraggingInfo>)sender
{
    self.defaultDisplay = YES;
    [self setNeedsDisplay:YES];
}

- (BOOL) prepareForDragOperation:(id <NSDraggingInfo>)sender
{
    // Apple
```

```
Docs[https://developer.apple.com/library/mac/samplecode/CocoaDragAndDrop/Listings/CocoaDragAndDrop_DragDropImageView_m.html]
    // Only interested in a sender that can create an image
    return [NSImage canInitWithPasteboard: [sender draggingPasteboard]];
}

// Handles drop data.
- (BOOL) performDragOperation:(id<NSDraggingInfo>)sender
{

    NSImage* droppedImage = [[NSImage alloc] initWithPasteboard:[sender
draggingPasteboard]];
    image = droppedImage;
    [[NSNotificationCenter defaultCenter]
postNotificationName:@"ImageUploadReciever" object:self];

    return YES;
}

- (void)drawRect:(NSRect)dirtyRect {
    [super drawRect:dirtyRect];

        [[NSColor greenColor] setFill];
        NSRectFill(dirtyRect);
        [super drawRect:dirtyRect];
}

@end
```

```
//
//  TracedWindowController.h
//  ImageCropUI
//
//  Created by James Mitchell on 14/04/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import <Cocoa/Cocoa.h>

@class DrawingView;

@interface DrawingWindowController : NSWindowController
{
    DrawingView* drawingView;
    IBOutlet NSView* containerView;
    NSArray* drawingData;
}

@property (nonatomic) NSArray* drawingData;
@property (nonatomic) NSView* drawingView;
@property (nonatomic) NSScrollView* scrollView;

@end
```

```
//
//  TracedWindowController.m
//  ImageCropUI
//
//  Created by James Mitchell on 14/04/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import "DrawingWindowController.h"
#import "DrawingView.h"

@implementation DrawingWindowController

@synthesize drawingData;
@synthesize drawingView;
@synthesize scrollView;

- (void)windowDidLoad {
```

```
    [super windowDidLoad];

//    drawingView = [[DrawingView alloc] initWithFrame:[self.window frame]];
//    [[self.window contentView] addSubview:drawingView];
//    [drawingView setNeedsDisplay:YES];

    if ( drawingData )
    {
        [drawingView setDrawingData:drawingData];
    }
}

- (void)awakeFromNib
{
//     REFERENCE: stackoverflow.com/questions/25250762/xcode-swift-window-without-title-bar-but-with-close-minimize-and-resize-but
    self.window.titleVisibility = NSWindowTitleHidden;
    self.window.titlebarAppearsTransparent = YES;
    self.window.styleMask |= NSFullSizeContentViewWindowMask;

    NSRect viewBounds = self.window.frame;

    drawingView = [[DrawingView alloc] initWithFrame:viewBounds];
    [drawingView setNeedsDisplay:YES];

    // add the new view.
    scrollView = [[NSScrollView alloc] initWithFrame:viewBounds];
    [scrollView setHasVerticalScroller:YES];
    [scrollView setHasHorizontalScroller:YES];
    [scrollView setDocumentView:drawingView];

    [containerView setBounds:viewBounds];
    [containerView addSubview:scrollView];
    [self.window setContentView:scrollView];
}

@end
```

```
//
//  ToolViewController.h
//  ImageCropUI
//
//  Created by James Mitchell on 08/04/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//
```

```
#import <Cocoa/Cocoa.h>

@class ImageRepresentation;
@class ImageProcessing;
@class ImageAnalysis;
@class Morphology;
@class DrawingWindowController;

@interface ToolWindowController : NSWindowController
{
    ImageRepresentation* representation;
    ImageProcessing* imageProcessing;
    ImageAnalysis* imageAnalysis;
    Morphology* morph;
    DrawingWindowController* dwc;

    IBOutlet NSSlider *aveFilterSlider;
    IBOutlet NSSlider *medianFilterSlider;
    IBOutlet NSSlider *maxFilterSlider;
    IBOutlet NSSlider *minFilterSlider;
    IBOutlet NSSlider *thresholdSlider;
    IBOutlet NSSlider *erodeFilterSlider;
    IBOutlet NSSlider *dilateFilterSlider;
    IBOutlet NSSlider *openFilterSlider;
    IBOutlet NSSlider *closeFilterSlider;
}

@property (nonatomic) ImageRepresentation* representation;

- (IBAction) applyAveragingFilter:(id)sender;
- (IBAction) applyMedianFilter:(id)sender;
- (IBAction) applyMaxFilter:(id)sender;
- (IBAction) applyMinFilter:(id)sender;
- (IBAction) threshold:(id)sender;
- (IBAction) erode:(id)sender;
- (IBAction) dilate:(id)sender;
- (IBAction) open:(id)sender;
- (IBAction) close:(id)sender;
- (IBAction) switchPolarity:(id)sender;
- (IBAction) thin:(id)sender;
- (IBAction) crop:(id)sender;
- (IBAction) resetImage:(id)sender;
- (IBAction) trace:(id)sender;

- (IBAction) lineDensityHistorgram:(id)sender;
- (IBAction) greylevelHistogram:(id)sender;
```

```
- (IBAction) pinCurrent:(id)sender;
- (void) resetToOriginal;

@end
```

```
//
//  ToolViewController.m
//  ImageCropUI
//
//  Created by James Mitchell on 08/04/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import "ToolWindowController.h"
#import "DrawingWindowController.h"
#import "ImageProcessing.h"
#import "ImageAnalysis.h"
#import "ImageRepresentation.h"
#import "Morphology.h"
#import "ZhangSuenThin.h"
#import "PixelTrace.h"
#import "IntArrayUtil.h"

@implementation ToolWindowController

@synthesize representation;

- (IBAction) applyAveragingFilter:(id)sender
{
    int filterSize = [sender intValue];

    if ( filterSize != 0 )
    {
        if ( !imageProcessing )
        {
            imageProcessing = [[ImageProcessing alloc] init];
        }

        // set other filters to 1
        [medianFilterSlider setIntValue:1];
        [maxFilterSlider setIntValue:1];
        [minFilterSlider setIntValue:1];

        // reset the filter.
```

```
        representation.filtered = nil;

        // apply the filter to the origianal image.
        NSImageRep* rep = [representation.subject.representations objectAtIndex:0];
        [representation.subject removeRepresentation:rep];
        [representation.subject addRepresentation:[imageProcessing
         simpleAveragingFilterOfSize:filterSize
onImage:representation.current]];
    } else {
        [representation setSubject:representation.current];
    }

    [[NSNotificationCenter defaultCenter]
postNotificationName:@"ImageUpdateReciever" object:self];
}

- (IBAction) applyMedianFilter:(id)sender
{
    int filterSize = [sender intValue];

    if ( filterSize != 1 )
    {
        if ( !imageProcessing )
        {
            imageProcessing = [[ImageProcessing alloc] init];
        }

        // set other filters to 1
        [aveFilterSlider setIntValue:1];
        [maxFilterSlider setIntValue:1];
        [minFilterSlider setIntValue:1];

        // reset the filter.
        representation.filtered = nil;

        NSImageRep* rep = [representation.subject.representations objectAtIndex:0];
        [representation.subject removeRepresentation:rep];
        [representation.subject addRepresentation:[imageProcessing
         medianFilterOfSize:filterSize
onImage:representation.current]];
    } else {
        [representation setSubject:representation.current];
    }

    [[NSNotificationCenter defaultCenter]
postNotificationName:@"ImageUpdateReciever"
                                                    object:self];
```

```
}

- (IBAction) applyMaxFilter:(id)sender
{
    int filterSize = [sender intValue];

    if ( filterSize != 1 )
    {
        if ( !imageProcessing )
        {
            imageProcessing = [[ImageProcessing alloc] init];
        }

        // set other filters to 1
        [aveFilterSlider setIntValue:1];
        [medianFilterSlider setIntValue:1];
        [minFilterSlider setIntValue:1];

        // reset the filter.
        representation.filtered = nil;

        NSImageRep* rep = [representation.subject.representations objectAtIndex:0];
        [representation.subject removeRepresentation:rep];
        [representation.subject addRepresentation:[imageProcessing
        maxFilterOfSize:filterSize
onImage:representation.current]];
    } else {
        [representation setSubject:representation.current];
    }

    [[NSNotificationCenter defaultCenter]
postNotificationName:@"ImageUpdateReciever"
object:self];
}

- (IBAction) applyMinFilter:(id)sender
{
    int filterSize = [sender intValue];

    if ( filterSize != 1 )
    {
        if ( !imageProcessing )
        {
            imageProcessing = [[ImageProcessing alloc] init];
        }

        // set other filters to 1
```

```
        [aveFilterSlider setIntValue:1];
        [medianFilterSlider setIntValue:1];
        [maxFilterSlider setIntValue:1];

        // reset the filter.
        representation.filtered = nil;

        NSImageRep* rep = [representation.subject.representations objectAtIndex:0];
        [representation.subject removeRepresentation:rep];
        [representation.subject addRepresentation:[imageProcessing
minFilterOfSize:filterSize
onImage:representation.current]];
    } else {
        [representation setSubject:representation.current];
    }

    [[NSNotificationCenter defaultCenter]
postNotificationName:@"ImageUpdateReciever"
object:self];
}

- (IBAction) threshold:(id)sender
{
    int thresholdValue = [sender intValue];

    if ( !imageProcessing)
    {
        imageProcessing = [[ImageProcessing alloc] init];
    }

    if ( !representation.filtered )
    {
        NSImageRep* rep = [representation.subject.representations objectAtIndex:0];
        representation.filtered = [ImageRepresentation cacheImageFromRepresentation:
(NSBitmapImageRep*)rep];
    }

    NSBitmapImageRep* newRep = [imageProcessing threshold:representation.filtered
atValue:thresholdValue];
    NSImageRep* oldRep = [representation.subject.representations objectAtIndex:0];
    [representation.subject removeRepresentation:oldRep];
    [representation.subject addRepresentation:newRep];
    [representation setCurrent:representation.subject];

    [[NSNotificationCenter defaultCenter]
postNotificationName:@"ImageUpdateReciever" object:self];
}
```

```
- (IBAction) erode:(id)sender
{
    int size = [sender intValue];

    if ( size != 1 ) {

        if ( !morph )
        {
            morph = [[Morphology alloc] init];
        }

//        representation.filtered = nil;

        NSBitmapImageRep* newRep = [morph
simpleErosionOfImage:representation.current withNeighbourhoodSize:size];
        NSImageRep* rep = [representation.subject.representations objectAtIndex:0];
        [representation.subject removeRepresentation:rep];
        [representation.subject addRepresentation:newRep];
    } else {
        [representation setSubject:representation.current];
    }

    [[NSNotificationCenter defaultCenter]
postNotificationName:@"ImageUpdateReciever" object:self];
}

- (IBAction) dilate:(id)sender
{
    int size = [sender intValue];

    if ( size != 1 ) {

        if ( !morph )
        {
            morph = [[Morphology alloc] init];
        }

        NSBitmapImageRep* newRep = [morph
simpleDilationOfImage:representation.current withNeighbourhoodSize:size];
        NSImageRep* rep = [representation.subject.representations objectAtIndex:0];
        [representation.subject removeRepresentation:rep];
        [representation.subject addRepresentation:newRep];
    } else {
        [representation setSubject:representation.current];
    }
```

```
    [[NSNotificationCenter defaultCenter]
postNotificationName:@"ImageUpdateReciever" object:self];
}

- (IBAction) open:(id)sender
{
    int size = [sender intValue];

    if ( size != 1 ) {

        if ( !morph )
        {
            morph = [[Morphology alloc] init];
        }

        NSBitmapImageRep* newRep = [morph openingOnImage:representation.current
withNeighbourhoodSize:size];
        NSImageRep* rep = [representation.subject.representations objectAtIndex:0];
        [representation.subject removeRepresentation:rep];
        [representation.subject addRepresentation:newRep];
    } else {
        [representation setSubject:representation.current];
    }

    [[NSNotificationCenter defaultCenter]
postNotificationName:@"ImageUpdateReciever"
                                                    object:self];
}


- (IBAction) close:(id)sender
{
    int size = [sender intValue];

    if ( size != 1 ) {

        if ( !morph )
        {
            morph = [[Morphology alloc] init];
        }

        NSBitmapImageRep* newRep = [morph closingOnImage:representation.current
                                  withNeighbourhoodSize:size];
        NSImageRep* rep = [representation.subject.representations objectAtIndex:0];
        [representation.subject removeRepresentation:rep];
        [representation.subject addRepresentation:newRep];
    } else {
```

```
        [representation setSubject:representation.current];
    }

    NSLog(@"%d", size);

    [[NSNotificationCenter defaultCenter]
postNotificationName:@"ImageUpdateReciever"
                                                    object:self];
}

- (IBAction) switchPolarity:(id)sender
{

}

- (IBAction) thin:(id)sender
{
    ZhangSuenThin* zst = [[ZhangSuenThin alloc] init];

    NSBitmapImageRep* newRep = [zst thinImage:representation.subject];
    NSImageRep* rep = [representation.subject.representations objectAtIndex:0];
    [representation.subject removeRepresentation:rep];
    [representation.subject addRepresentation:newRep];

    [representation setCurrent:representation.subject];

    [[NSNotificationCenter defaultCenter]
postNotificationName:@"ImageUpdateReciever"
                                                    object:self];
}

- (IBAction) crop:(id)sender
{
    representation.filtered = nil;
    [[NSNotificationCenter defaultCenter]
postNotificationName:@"CropImageToolSelection"
                                                    object:self];
}

- (IBAction) resetImage:(id)sender
{
    [aveFilterSlider setIntValue:1];
    [medianFilterSlider setIntValue:1];
    [maxFilterSlider setIntValue:1];
    [minFilterSlider setIntValue:1];

    [thresholdSlider setIntValue:128];
```

```
    [erodeFilterSlider setIntValue:1];
    [dilateFilterSlider setIntValue:1];
    [openFilterSlider setIntValue:1];
    [closeFilterSlider setIntValue:1];

    // cropped bit.

    [self resetToOriginal];
}

- (IBAction) pinCurrent:(id)sender
{
    [representation setCurrent:representation.subject];
}

- (void) resetToOriginal
{
    [representation resetSubject];
    [[NSNotificationCenter defaultCenter] postNotificationName:@"ResetOriginalImage"
object:self];
}

- (IBAction) trace:(id)sender
{
    PixelTrace* tracer = [[PixelTrace alloc] init];
    NSArray* tracedPoints = [tracer
mooreNeighborContorTraceOfImage:representation.subject];

    // line simplification here

    dwc = [[DrawingWindowController alloc] initWithWindowNibName:@"DrawingWindow"];
    [dwc setDrawingData:tracedPoints];
    [dwc showWindow:nil];
}

- (IBAction) lineDensityHistorgram:(id)sender
{
    if ( !imageProcessing )
    {
        imageProcessing = [[ImageProcessing alloc] init];
    }

    if ( !imageAnalysis )
    {
        imageAnalysis = [[ImageAnalysis alloc] init];
    }
```

```
    int height = representation.subject.size.height;
    int* areaDensity = [imageAnalysis
pixelAreaDensityOfImage:representation.subject];
    int maxDensity = [IntArrayUtil maxFromArray:areaDensity ofSize:height];

    NSBitmapImageRep* areaDensityHistogramRep = [imageAnalysis
histogramRepresentationOfData:areaDensity
withWidth:maxDensity
andHeight:height];

    [ImageRepresentation saveImageFileFromRepresentation:areaDensityHistogramRep
                                                fileName:@"area"];
}

- (IBAction) greylevelHistogram:(id)sender
{

    if ( !imageProcessing )
    {
        imageProcessing = [[ImageProcessing alloc] init];
    }

    if ( !imageAnalysis )
    {
        imageAnalysis = [[ImageAnalysis alloc] init];
    }

    int* contrast = [imageProcessing
contrastHistogramOfImage:representation.subject];
    contrast = [imageProcessing normaliseConstrastHistogramData:contrast
ofSize:256];
    int maxValue = [IntArrayUtil maxFromArray:contrast ofSize:256];

    NSBitmapImageRep* contrastHistogram =
    [imageAnalysis
    histogramRepresentationOfData:contrast
withWidth:maxValue
andHeight:256];

    [ImageRepresentation
    saveImageFileFromRepresentation:contrastHistogram fileName:@"contrast"];
}


@end
```

```
//
//  JMWindowController.h
//  ImageCropUI
//
//  Created by James Mitchell on 22/01/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import <Cocoa/Cocoa.h>

@class DropZoneView;
@class ImageCropView;
@class ImageManipulationView;
@class ToolWindowController;
@class ImageRepresentation;

@interface MainWindowController : NSWindowController
{
    ImageRepresentation* representation;
    IBOutlet NSView* containerView;
    ToolWindowController* toolWindowController;
}

@property (nonatomic) ImageRepresentation* representation;
@property (nonatomic) DropZoneView* dropZoneView;
@property (nonatomic) ImageManipulationView* imgManipView;
@property (nonatomic) ImageCropView* imageCropView;
@property (nonatomic) NSScrollView* scrollView;

- (NSRect) determineViewBounds;
- (void) changeToDropZoneController;
- (void) handleDroppedImage;
- (void) imageFromDropZone;
- (void) displayToolWindow;
- (void) setImageManipulationView;
- (void) setCropView;

@end
```

```
//
//  JMWindowController.m
//  ImageCropUI
//
//  Created by James Mitchell on 22/01/2016.
```

```
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import "MainWindowController.h"
#import "DropZoneView.h"
#import "ImageCropView.h"
#import "ImageManipulationView.h"
#import "ToolWindowController.h"
#import "ImageRepresentation.h"
#import "PixelTrace.h"

@implementation MainWindowController

@synthesize representation;
@synthesize dropZoneView;
@synthesize imgManipView;
@synthesize imageCropView;
@synthesize scrollView;

- (void)awakeFromNib
{
    // REFERENCE: stackoverflow.com/questions/25250762/xcode-swift-window-without-
title-bar-but-with-close-minimize-and-resize-but
    self.window.titleVisibility = NSWindowTitleHidden;
    self.window.titlebarAppearsTransparent = YES;
    self.window.styleMask |= NSFullSizeContentViewWindowMask;

    [[NSNotificationCenter defaultCenter] addObserver:self

selector:@selector(changeToDropZoneController)
                                                 name:@"ViewChangeDropZoneReciever"
                                               object:nil];
}

- (void) windowDidLoad
{
    [super windowDidLoad];
    [self changeToDropZoneController];
}

- (void) changeToDropZoneController
{
    NSArray* subviews = [containerView subviews];
    dropZoneView = [[DropZoneView alloc] initWithFrame:[containerView bounds]];

    if ( [subviews count] != 0 )
    {
```

```
        [containerView replaceSubview:[subviews objectAtIndex:0] with:dropZoneView];
    } else {
        [containerView addSubview:dropZoneView];
    }

    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(handleDroppedImage)
                                                 name:@"ImageUploadReciever"
                                               object:nil];

    [self.window setContentView:dropZoneView];
    [dropZoneView setNeedsDisplay:YES];
}

- (void) handleDroppedImage
{
    [self imageFromDropZone];
    [dropZoneView removeFromSuperview];
    [self setImageManipulationView];
    [[NSNotificationCenter defaultCenter] removeObserver:self
name:@"ImageUploadReciever" object:nil];
}

- (void) updateImage
{
    [imgManipView setImage:representation.subject];
    [imgManipView setNeedsDisplay:YES];
}

- (void) imageFromDropZone
{
    representation = [[ImageRepresentation alloc] init];
    [representation setSubject:[dropZoneView image]];
    [representation setOriginal:[dropZoneView image]];
    [representation setCurrent:[dropZoneView image]];
}


- (void) displayToolWindow
{
    if ( !toolWindowController )
    {
        toolWindowController = [[ToolWindowController alloc]
initWithWindowNibName:@"ToolView"];
        [toolWindowController showWindow:nil];
    }
```

```
    [toolWindowController setRepresentation:representation];
}

- (void) setImageManipulationView
{
    NSRect viewBounds = [self determineViewBounds];
    imgManipView = [[ImageManipulationView alloc] initWithFrame:viewBounds];
    [imgManipView setImage:representation.subject];
    [imgManipView setNeedsDisplay:YES];

    // add the new view.
    scrollView = [[NSScrollView alloc] initWithFrame:viewBounds];
    [scrollView setHasVerticalScroller:YES];
    [scrollView setHasHorizontalScroller:YES];
    [scrollView setDocumentView:imgManipView];

    [containerView setBounds:viewBounds];
    [containerView addSubview:scrollView];
    [self.window setContentView:scrollView];

    NSRect frame = [self.window frame];
    frame.size = viewBounds.size;
    [self.window setFrame:frame display:YES animate:NO];

    [self displayToolWindow];
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(updateImage)
                                                 name:@"ImageUpdateReciever"
                                               object:nil];

    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(setCropView)
                                                 name:@"CropImageToolSelection"
                                               object:nil];

    [[NSNotificationCenter defaultCenter] addObserver:self

selector:@selector(setImageManipulationView)
                                                 name:@"ResetOriginalImage"
                                               object:nil];
}

- (void) setCropView
{
    NSRect viewBounds = [self determineViewBounds];

    [imgManipView removeFromSuperview];
```

```
    imageCropView = [[ImageCropView alloc] initWithFrame:viewBounds];
    [imageCropView setImage:representation.subject];

    scrollView = [[NSScrollView alloc] initWithFrame:viewBounds];
    [scrollView setHasVerticalScroller:YES];
    [scrollView setHasHorizontalScroller:YES];

    [scrollView setDocumentView:imageCropView];
    [containerView addSubview:scrollView];
    [self.window setContentView:scrollView];

    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(imageFromCrop)
                                                 name:@"ImageCropComplete"
                                               object:nil];

    [[NSApp mainWindow] makeKeyWindow];

    [[NSNotificationCenter defaultCenter] removeObserver:self
name:@"CropImageToolSelection" object:nil];
}

- (void) imageFromCrop
{
    [representation setSubject:[imageCropView croppedImage]];
    [representation setCurrent:[imageCropView croppedImage]];
    [imageCropView removeFromSuperview];

    [self setImageManipulationView];
}

/*
 * Where the image is larger then the container window
 * set the destination view to be the size of the image.
 */
- (NSRect) determineViewBounds
{
    NSRect viewBounds;
    int viewWidth;
    int viewHeight;

    int maxWidth = 1000;
    int maxHeight = 1000;

    if ( maxHeight < representation.subject.size.height )
    {
```

```
        viewHeight = maxHeight;
    } else {
        viewHeight = representation.subject.size.height;
    }

    if ( maxWidth < representation.subject.size.width )
    {
        viewWidth = maxWidth;
    } else {
        viewWidth = representation.subject.size.width;
    }

    viewBounds = NSMakeRect(0, 0, viewWidth, viewHeight);

    return viewBounds;
}

@end
```

```
//
//  PixelTrace.h
//  ImageCropUI
//
//  Created by James Mitchell on 13/04/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import <Cocoa/Cocoa.h>

@interface PixelTrace : NSObject

- (void) tracePixelsOfImage:(NSImage*)image;
- (NSArray*) mooreNeighborContorTraceOfImage:(NSImage*)image;

@end
```

```
//
//  PixelTrace.m
//  ImageCropUI
//
//  Created by James Mitchell on 13/04/2016.
```

```
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import "PixelTrace.h"
#import "ImageRepresentation.h"

@implementation PixelTrace

- (void) tracePixelsOfImage:(NSImage*)image
{
    NSBitmapImageRep *representation = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char *data = [representation bitmapData];

    int width = image.size.width;
    int height = image.size.height;
    int index = 0;
    int searchPixel = 0;
    BOOL match = NO;

    NSMutableArray* points = [[NSMutableArray alloc] init];

    // find the first black pixel.
    // or collect all the balck points.
    for ( int y = 0; y < height; y++ )
    {
        for ( int x = 0; x < width; x++ )
        {
            index = x + y * width;

            if ( data[index] == searchPixel )
            {
                match = YES;
                NSPoint point = NSMakePoint(x, y);
                NSValue *p = [NSValue valueWithPoint:point];
                [points addObject:p];
//                break;
            }
        }
//        if ( match ) break;
    }


//    struct PointNode *head = nil;

    [points sortUsingComparator: ^NSComparisonResult(id v1, id v2) {
        NSPoint point1;
```

```
        [v1 getValue:&point1];

        NSPoint point2;
        [v2 getValue:&point2];

        return point1.x > point2.x;
    }];

    for ( NSValue *val in points)
    {
        NSPoint current;
        [val getValue:&current];

        NSLog(@"%f, %f", current.x, current.y);
    }

}

- (NSArray*) mooreNeighborContorTraceOfImage:(NSImage*)image
{
    NSBitmapImageRep *representation = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char *data = [representation bitmapData];

    int width = image.size.width;
    int height = image.size.height;
    int index = 0;
    int searchPixel = 0;
    BOOL match = NO;
    int x = 0, y = 0;

    NSPoint start, last;

    NSMutableArray* points = [[NSMutableArray alloc] init];

    // find the first black pixel.
    for (y = 0; y < height; y++ )
    {
        for (x = 0; x < width; x++ )
        {
            index = x + y * width;

            if ( data[index] == searchPixel )
            {
                match = YES;
                start = NSMakePoint(x, y);
                NSValue *p = [NSValue valueWithPoint:start];
```

```
                [points addObject:p];
                break;
            } else {
                last = NSMakePoint(x, y);
            }

        }
        if ( match ) break;
    }

    int next = 0;
    NSPoint offsets[] = {NSMakePoint(-1, -1),
                         NSMakePoint(-1, 0),
                         NSMakePoint(-1, 1),
                         NSMakePoint(0, 1),
                         NSMakePoint(1, 1),
                         NSMakePoint(1, 0),
                         NSMakePoint(1, -1),
                         NSMakePoint(0, -1)};

    NSPoint current = start;
    NSPoint consider = last;
    NSPoint backtrackPosition = last;
    NSPoint backtrackOffset = NSMakePoint(last.x - start.x, last.y - start.y);

    for ( int i = 0; i < 8; i++ )
    {
        if ( CGPointEqualToPoint(backtrackOffset, offsets[i]) )
        {
            next = i;
        }
    }

    BOOL run = YES;
    while ( run )
    {
        index = consider.x + consider.y * width;

        if ( data[index] == searchPixel )
        {
            // stopping critria
            if ( CGPointEqualToPoint(start, consider) )
            {
                run = NO;
                break;
            }
```

```
            NSValue *val = [[NSValue alloc] init];
            val = [NSValue valueWithPoint:consider];
            [points addObject:val];

            current = consider;
            backtrackOffset = NSMakePoint(backtrackPosition.x - current.x,
backtrackPosition.y - current.y);

            for ( int i = 0; i < 8; i++ )
            {
                if ( CGPointEqualToPoint(backtrackOffset, offsets[i]) )
                {
                    next = i; //(i + 1) < 8 ? (i + 1) : 0;
                    break;
                }
            }
            consider = NSMakePoint(current.x + offsets[next].x, current.y +
offsets[next].y);
        } else {
            backtrackPosition = consider;
            next++;
            if ( next == 8 ) next = 0;
            consider = NSMakePoint(current.x + offsets[next].x, current.y +
offsets[next].y);
        }
    }

    NSOrderedSet* set = [NSOrderedSet orderedSetWithArray:points];
    NSArray* distinctPoints = [set array];

    return distinctPoints;
}



@end
```

```
//
//  IP.h
//  ImageCrop
//
//  Created by James Mitchell on 09/01/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import <Foundation/Foundation.h>
@import AppKit;
@import CoreImage;

@interface ImageProcessing: NSObject

- (NSBitmapImageRep*) medianFilterOfSize:(int)size onImage:(NSImage*)image;
- (NSBitmapImageRep*) maxFilterOfSize:(int)size onImage:(NSImage*)image;
- (NSBitmapImageRep*) minFilterOfSize:(int)size onImage:(NSImage*)image;
- (NSBitmapImageRep*) simpleAveragingFilterOfSize:(int)size onImage:(NSImage*)image;
- (NSBitmapImageRep*) weightedAveragingFilterOfSize:(int)size onImage:
(NSImage*)image;
- (NSBitmapImageRep*) threshold:(NSImage*)image atValue:(int)value;
- (NSBitmapImageRep*) imageDifferenceOf:(NSImage*)image1 and:(NSImage*)image2;
- (NSBitmapImageRep*) imageNegativeOf:(NSImage*)image;

- (NSBitmapImageRep*) automaticContrastAdjustmentOfImage:(NSImage*)image;
- (int*) cumulativeHistogramFromData:(int*)data ofSize:(int)size;
- (int*) contrastHistogramOfImage:(NSImage*)image;
- (int*) normaliseConstrastHistogramData:(int*)data ofSize:(int)size;



@end
```

```
//
//  IP.m
//  ImageProcessingCLI
//
//  Created by James Mitchell on 09/01/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import "ImageProcessing.h"
#import "IntArrayUtil.h"
#import "ImageRepresentation.h"
```

```
@implementation ImageProcessing

#pragma mark -
#pragma mark Filters

- (NSBitmapImageRep*) medianFilterOfSize:(int)size onImage:(NSImage*)image;
{
    // create a represenation of the origional image
    NSBitmapImageRep *representation = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char *original = [representation bitmapData];

    // create a representation that will store the smoothed image.
    NSBitmapImageRep *output = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char *smoothed = [output bitmapData];

    int width = image.size.width;
    int height = image.size.height;

    int padding = (size - 1) / 2.0;
    int filter[size * size];

    for ( int y = padding; y < height - padding; y++ )
    {
        for (int x = padding; x < width - padding; x++)
        {

            int centre = x + y * width;
            int i = 0;

            for (int s = -padding; s < (padding + 1); s++) {

                for (int t = -padding; t < (padding + 1); t++) {

                    int index = (x + s) + ((y + t) * width);
                    filter[i++] = original[index];

                }
            }

            smoothed[centre] = [IntArrayUtil getMedianFromArray:filter ofSize:size *
size];
        }
    }
```

```
    return output;
}

- (NSBitmapImageRep*) maxFilterOfSize:(int)size onImage:(NSImage*)image;
{
    // create a represenation of the origional image
    NSBitmapImageRep *representation = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char *original = [representation bitmapData];

    // create a representation that will store the smoothed image.
    NSBitmapImageRep *output = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char *smoothed = [output bitmapData];

    int width = image.size.width;
    int height = image.size.height;

    int padding = (size - 1) / 2.0;
    int filter[size * size];

    for ( int y = padding; y < height - padding; y++ )
    {
        for (int x = padding; x < width - padding; x++)
        {

            int centre = x + y * width;
            int i = 0;

            for (int s = -padding; s < (padding + 1); s++)
            {

                for (int t = -padding; t < (padding + 1); t++)
                {

                    int index = (x + s) + ((y + t) * width);
                    filter[i++] = original[index];

                }
            }

            smoothed[centre] = [IntArrayUtil maxFromArray:filter ofSize:size *
size];
        }
    }

    return output;
```

```
}

- (NSBitmapImageRep*) minFilterOfSize:(int)size onImage:(NSImage*)image;
{
    // create a represenation of the origional image
    NSBitmapImageRep *representation = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char *original = [representation bitmapData];

    // create a representation that will store the smoothed image.
    NSBitmapImageRep *output = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char *smoothed = [output bitmapData];

    int width = image.size.width;
    int height = image.size.height;

    int padding = (size - 1) / 2.0;
    int filter[size * size];

    for ( int y = padding; y < height - padding; y++ )
    {
        for (int x = padding; x < width - padding; x++)
        {

            int centre = x + y * width;
            int i = 0;

            for (int s = -padding; s < (padding + 1); s++)
            {

                for (int t = -padding; t < (padding + 1); t++)
                {
                    int index = (x + s) + ((y + t) * width);
                    filter[i++] = original[index];
                }
            }

            smoothed[centre] = [IntArrayUtil minFromArray:filter ofSize:size *
size];
        }
    }

    return output;
}
```

```
- (NSBitmapImageRep*) simpleAveragingFilterOfSize:(int)size onImage:(NSImage*)image;
{

    // create a represenation of the origional image
    NSBitmapImageRep *representation = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char *original = [representation bitmapData];

    // create a representation that will store the smoothed image.
    NSBitmapImageRep *output = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char *smoothed = [output bitmapData];

    int width = image.size.width;
    int height = image.size.height;

    float weight = 1.0 / (float)(size * size); // e.g. 1/(3 * 3) = 0.111
    int padding = (size - 1) / 2.0;  // pad the image

    // iterate over each pixel of the image
    for ( int y = padding; y < height - padding; y++ )
    {
        for (int x = padding; x < width - padding; x++)
        {

            // find the centre pixel.
            int centre = x + y * width;
            int val = 0;

            // iterate over the filter
            for (int s = -padding; s < (padding + 1); s++)
            {
                for (int t = -padding; t < (padding + 1); t++)
                {

                    // offset the current x, y
                    int index = (x + s) + ((y + t) * width);
                    // add the values
                    val += original[index] * weight;

                }
            }

            // reject values over 255 to prevent
            if ( val > 255 ) val = 255;
            // apply the new value to centre of the filter
            smoothed[centre] = val;
```

```
        }
    }

    return output;
}

- (NSBitmapImageRep*) weightedAveragingFilterOfSize:(int)size onImage:
(NSImage*)image;
{
    // create a represenation of the origional image
    NSBitmapImageRep *representation = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char *original = [representation bitmapData];

    // create a representation that will store the smoothed image.
    NSBitmapImageRep *output = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char *smoothed = [output bitmapData];

    int width = image.size.width;
    int height = image.size.height;

    int padding = (3 - 1) / 2.0;  // pad the image

    int weights[9] = {1, 2, 1, 2, 4, 2, 1, 2, 1};
    float filter[9];

    // make filter
    for (int i = 0; i < 9; i++)
    {
        filter[i] = (float)weights[i] / 16.0;
    }

    // iterate over each pixel of the image
    for ( int y = padding; y < height - padding; y++ )
    {
        for (int x = padding; x < width - padding; x++)
        {

            // find the centre pixel.
            int centre = x + y * width;
            int val = 0;
            int i = 0;

            // iterate over the filter
            for (int s = -padding; s < (padding + 1); s++)
            {
```

```
                for (int t = -padding; t < (padding + 1); t++)
                {
                    // offset the current x, y
                    int index = (x + s) + ((y + t) * width);
                    // add the values
                    val += original[index] * filter[i++];
                }
            }

            // reject values over 255 to prevent
            if ( val > 255 ) val = 255;
            // apply the new value to centre of the filter
            smoothed[centre] = val;
        }
    }

    return output;
}

#pragma mark -
#pragma mark Thresholding

- (NSBitmapImageRep*) threshold:(NSImage*)image atValue:(int)value
{
    NSBitmapImageRep *output = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char *threshold = [output bitmapData];

    int width = image.size.width;
    int height = image.size.height;

    for ( int y = 0; y < height; y++ )
    {
        for ( int x = 0; x < width; x++ )
        {
            int index = x + (y * width);
            if ( threshold[index] < value)
            {
                threshold[index] = 0;
            } else {
                threshold[index] = 255;
            }
        }
    }

    return output;
}
```

```
// as a percentage of the image pixels.
- (int*) contrastHistogramOfImage:(NSImage*)image
{
    int range = 256;
    int* output = [IntArrayUtil zeroArrayOfSize:range];

    int width = image.size.width;
    int height = image.size.height;

    NSBitmapImageRep* representation = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char* data = [representation bitmapData];

    for ( int y = 0; y < height; y++ )
    {
        for ( int x = 0; x < width; x++ )
        {
            int index = x + (y * width);
            int val = data[index];

            output[val] += 1;
        }
    }

    return output;
}

// rename!
- (int*) normaliseConstrastHistogramData:(int*)data ofSize:(int)size
{
    int* output = [IntArrayUtil zeroArrayOfSize:size];
    int count = 0;

    for ( int i = 0; i < size; i++ )
    {
        count += data[i];
    }

    for ( int j = 0; j < size; j++ )
    {
        output[j] = ((float)data[j] / 10.0f);

    }

    return output;
}
```

```
// constrast streching: http://homepages.inf.ed.ac.uk/rbf/HIPR2/stretch.htm
// linear - Page 60 Principles of Digital Image Processing.
- (NSBitmapImageRep*) automaticContrastAdjustmentOfImage:(NSImage*)image
{
    int range = 256;

    // create representation of image.
    NSBitmapImageRep* representation = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char* data = [representation bitmapData];

    // get the histogram values.
    int* histogram = [self contrastHistogramOfImage:image];

    // get the high and low of the histogram.
    int high = 255;
    int low = 0;

    int i = 0;
    while ( (histogram[i] == 0) && (i < range))
    {
        i++;
    }

    low = i;

    i = 255;
    while ( (histogram[i] == 0) && (i > 0) )
    {
        i--;
    }

    high = i;

    int width = image.size.width;
    int height = image.size.height;

//    f(a) = (a - a[low]) * 255 / a[high] - a[low]
    for ( int y = 0; y < height; y++ )
    {
        for ( int x = 0; x < width; x++ )
        {
            int index = x + (y * width);
            int val = (data[index] - low) * (255 / (high - low));

            data[index] = val;
```

```
        }
    }


    return representation;
}

// Principle of DIP Fundermentals chap.3 p.52, chap.4 p.66
- (int*) cumulativeHistogramFromData:(int*)data ofSize:(int)size
{
    int* output = [IntArrayUtil zeroArrayOfSize:size];

    for ( int i = 1; i < size; i++ )
    {
        output[i] = data[i - 1] + data[i];
    }

    return output;
}

#pragma mark -
#pragma mark Other

- (NSBitmapImageRep*) imageDifferenceOf:(NSImage*)image1
                                    and:(NSImage*)image2
{
    NSImage* outputImage = [[NSImage alloc] initWithSize:image1.size];

    NSBitmapImageRep* rep1 = [ImageRepresentation
grayScaleRepresentationOfImage:image1];
    NSBitmapImageRep* rep2 = [ImageRepresentation
grayScaleRepresentationOfImage:image2];
    NSBitmapImageRep* output = [ImageRepresentation
grayScaleRepresentationOfImage:outputImage];

    unsigned char *one = [rep1 bitmapData];
    unsigned char *two = [rep2 bitmapData];
    unsigned char *three = [output bitmapData];

    int width = image1.size.width;
    int height = image1.size.height;

    for ( int y = 0; y < height; y++ )
    {
        for (int x = 0; x < width; x++)
        {
            int index = x + (y * width);
```

```
            three[index] = one[index] - two[index];
        }
    }

    return output;
}

- (NSBitmapImageRep*) imageNegativeOf:(NSImage*)image
{
    NSBitmapImageRep* output = [ImageRepresentation
grayScaleRepresentationOfImage:image];

    unsigned char* rep = [output bitmapData];

    int width = image.size.width;
    int height = image.size.height;

    for ( int y = 0; y < height; y++ )
    {
        for ( int x = 0; x < width; x++ )
        {
            int index = x + (y * width);

            int val = rep[index] - 255;

            if ( val < 0 )
            {
                val = val * -1;
            }

            rep[index] = val;
        }
    }



    return output;
}


@end
```

```
//
//  ImageRepresentation.h
//  ImageProcessingCLI
//
//  Created by James Mitchell on 06/02/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import <Foundation/Foundation.h>
@import AppKit;

@interface ImageRepresentation : NSObject
{
    NSImage* original;
    NSImage* current;
    NSImage* subject;
    NSImage* filtered;
    NSImage* thresholded;
}

@property (nonatomic) NSImage* original;
@property (nonatomic) NSImage* current;
@property (nonatomic) NSImage* subject;
@property (nonatomic) NSImage* filtered;
@property (nonatomic) NSImage* thresholded;

- (void) resetSubject;

// representation.
+ (NSImage*) cacheImageFromRepresentation:(NSBitmapImageRep *)representation;
+ (NSBitmapImageRep*) grayScaleRepresentationOfImage:(NSImage *)image;
+ (NSBitmapImageRep*) grayScaleRepresentationOfImage:(NSImage *)image
                                         withPadding:(int)padding;
+ (NSBitmapImageRep*) grayScaleRepresentationOfImage:(NSImage *)image
                                              atSize:(NSSize)size;


+ (void) saveImageFileFromRepresentation:(NSBitmapImageRep *)representation
                                fileName:(NSString*)filename;

@end
```

```
//
//  ImageRepresentation.m
```

```
//  ImageProcessingCLI
//
//  Created by James Mitchell on 06/02/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import "ImageRepresentation.h"

@implementation ImageRepresentation

@synthesize subject;
@synthesize original;
@synthesize filtered;
@synthesize thresholded;
@synthesize current;

- (void) setOriginal:(NSImage *)image
{
    original = [[NSImage alloc] init];
    [original addRepresentation:[ImageRepresentation
grayScaleRepresentationOfImage:image]];
}

- (void) setCurrent:(NSImage *)image
{
    current = [[NSImage alloc] init];
    [current addRepresentation:[ImageRepresentation
grayScaleRepresentationOfImage:image]];
}

- (void) setSubject:(NSImage*)image
{
    subject = [[NSImage alloc] init];
    [subject addRepresentation:[ImageRepresentation
grayScaleRepresentationOfImage:image]];
}

- (void) setThresholded:(NSImage *)image
{
    thresholded = [[NSImage alloc] init];
    [thresholded addRepresentation:[ImageRepresentation
grayScaleRepresentationOfImage:image]];
}

- (void) resetSubject
{
    subject = [[NSImage alloc] init];
```

```
    [subject addRepresentation:[ImageRepresentation
grayScaleRepresentationOfImage:original]];
}

+ (NSBitmapImageRep *) grayScaleRepresentationOfImage:(NSImage *)image
{
    return [self grayScaleRepresentationOfImage:image withPadding:0];
}

+ (NSBitmapImageRep*) grayScaleRepresentationOfImage:(NSImage *)image
                                              atSize:(NSSize)size
{
    NSBitmapImageRep *representation = [[NSBitmapImageRep alloc]
                                        initWithBitmapDataPlanes: NULL
                                        pixelsWide: (int)size.width
                                        pixelsHigh: (int)size.height
                                        bitsPerSample: 8
                                        samplesPerPixel: 1
                                        hasAlpha: NO
                                        isPlanar: NO
                                        colorSpaceName: NSCalibratedWhiteColorSpace
                                        bytesPerRow: (int)size.width
                                        bitsPerPixel: 8];

    NSGraphicsContext *context = [NSGraphicsContext
graphicsContextWithBitmapImageRep:representation];
    [NSGraphicsContext saveGraphicsState];
    [NSGraphicsContext setCurrentContext:context];

    // REFERENCE
developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CocoaDrawingGuide/Ima
ges/Images.html
    [image drawInRect:NSMakeRect(0, 0, (int)size.width, (int)size.height)
             fromRect:NSZeroRect
            operation:NSCompositeCopy
             fraction:1.0];

    [context flushGraphics];
    [NSGraphicsContext restoreGraphicsState];

    return representation;
}

+ (NSBitmapImageRep*) grayScaleRepresentationOfImage:(NSImage *)image
                                         withPadding:(int)padding
{
    NSBitmapImageRep *representation = [[NSBitmapImageRep alloc]
```

```
                                        initWithBitmapDataPlanes: NULL
                                        pixelsWide: image.size.width
                                        pixelsHigh: image.size.height
                                        bitsPerSample: 8
                                        samplesPerPixel: 1
                                        hasAlpha: NO
                                        isPlanar: NO
                                        colorSpaceName: NSCalibratedWhiteColorSpace
                                        bytesPerRow: image.size.width //* 4
                                        bitsPerPixel: 8];

    NSGraphicsContext *context = [NSGraphicsContext
graphicsContextWithBitmapImageRep:representation];
    [NSGraphicsContext saveGraphicsState];
    [NSGraphicsContext setCurrentContext:context];

    // REFERENCE
developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CocoaDrawingGuide/Ima
ges/Images.html
        [image drawAtPoint:NSZeroPoint
                  fromRect:NSZeroRect
                 operation:NSCompositeCopy
                  fraction:1.0];

    [context flushGraphics];
    [NSGraphicsContext restoreGraphicsState];

    return representation;
}

/*
 * Saves image to disk for my inspection.
 *
 */
+ (void) saveImageFileFromRepresentation:(NSBitmapImageRep *)representation
                                fileName:(NSString*)filename
{
    NSMutableString *saveTo = [NSMutableString stringWithString:@"~/Desktop/"];
    [saveTo appendString:filename];
    [saveTo appendString:@".png"];

    NSDictionary *imageProps = [NSDictionary dictionaryWithObject:[NSNumber
numberWithFloat:1.0]

forKey:NSImageCompressionFactor];

    NSData *newFile = [representation representationUsingType:NSPNGFileType
```

```
                                                    properties:imageProps];

    [newFile writeToFile:[saveTo stringByExpandingTildeInPath]
              atomically:NO];
}



+ (NSImage*) cacheImageFromRepresentation:(NSBitmapImageRep *)representation
{
    NSDictionary *imageProps = [NSDictionary dictionaryWithObject:[NSNumber
numberWithFloat:1.0]

forKey:NSImageCompressionFactor];

    NSData *newData = [representation representationUsingType:NSPNGFileType
properties:imageProps];
    return [[NSImage alloc] initWithData:newData];
}


@end
```

```
//
//  ImageAnalysis.h
//  ImageProcessingCLI
//
//  Created by James Mitchell on 22/02/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import <Foundation/Foundation.h>
@import AppKit;

@interface ImageAnalysis : NSObject

- (int*) pixelAreaDensityOfImage:(NSImage*)image;
- (NSBitmapImageRep*) histogramRepresentationOfData:(int*)data withWidth:(int)width
andHeight:(int)height;

@end
```

```
//
//  ImageAnalysis.m
//  ImageProcessingCLI
//
//  Created by James Mitchell on 22/02/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import "ImageAnalysis.h"
#import "ImageRepresentation.h"

@implementation ImageAnalysis

// iterate over image.
// at each y
// count each x value of ...

// takes image.
// returns unsigned char.

// pixel area histogram.

// of Thresholded image.
- (int*) pixelAreaDensityOfImage:(NSImage*)image
{
    int width = image.size.width;
    int height = image.size.height;

    NSBitmapImageRep* representation = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char* input = [representation bitmapData];

    int* output = malloc(sizeof(int) * height);

    for ( int y = 0; y < height; y++ )
    {
        int count = 0;

        for ( int x = 0; x < width; x++ )
        {
            int index = x + (y * width);
            int t = input[index];

            if ( t == 0 )
            {
                count++;
            }
```

```
        }

        output[y] = count;
    }

    return output;
}

- (NSBitmapImageRep*) histogramRepresentationOfData:(int*)data
                                          withWidth:(int)width
                                          andHeight:(int)height
{
    NSImage* outputImage = [[NSImage alloc] initWithSize:NSMakeSize(width, height)];

    [outputImage lockFocus];

    [[NSColor whiteColor] setFill];
    [NSBezierPath fillRect:NSMakeRect(0, 0, width, height)];

    int index = 0;

    for ( int y = height - 1; y > 0; y-- )
    {
        int density = data[index++];

        NSPoint start = NSMakePoint(0, (float)y + 0.5);
        NSPoint end = NSMakePoint(density, (float)y + 0.5);

        NSBezierPath* path = [[NSBezierPath alloc] init];

        [path moveToPoint:start];
        [path lineToPoint:end];

        [path setLineWidth:1.0];
        [path stroke];
    }

    [outputImage unlockFocus];

    return [ImageRepresentation grayScaleRepresentationOfImage:outputImage];
}


@end
```

```
//
//  Thinning.h
//  ImageProcessingCLI
//
//  Created by James Mitchell on 08/02/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import <Foundation/Foundation.h>
@import AppKit;

@interface ZhangSuenThin : NSObject
{
    int width;
    int height;
    BOOL complete;
    unsigned char* output;
}

- (NSBitmapImageRep*) thinImage:(NSImage*)image;
- (void) subIteration1;
- (void) subIteration2;

@end
```

```
//
//  Thinning.m
//  ImageProcessingCLI
//  Implementation of ZhangSuen Thinning Algorithm.
//
//  Created by James Mitchell on 08/02/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//


#import "ZhangSuenThin.h"
#import "ImageRepresentation.h"

@implementation ZhangSuenThin


- (NSBitmapImageRep*) thinImage:(NSImage*)image
{
    width = image.size.width;
```

```
    height = image.size.height;

    NSBitmapImageRep* outputRepresentation = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    output = [outputRepresentation bitmapData];

    complete = NO;

    while ( !complete )
    {
        [self subIteration1];
        [self subIteration2];
    }


    return outputRepresentation;
}

- (void) subIteration1
{

    complete = YES;
    BOOL change = NO;

    int size = 3;
    int padding = (size - 1) / 2.0;

    for ( int y = padding; y < height - padding; y++ )
    {
        for (int x = padding; x < width - padding; x++)
        {
            int p1 = (x) + (y * width);

            if ( output[p1] != 0 ) continue;

            int a = 0;
            int b = 0;

            int p2 = (x - 1) + (y * width);
            int p3 = (x - 1) + ((y + 1) * width);
            int p4 = (x) + ((y + 1) * width);
            int p5 = (x + 1) + ((y + 1) * width);
            int p6 = (x + 1) + (y * width);
            int p7 = (x + 1) + ((y - 1) * width);
            int p8 = (x) + ((y - 1) * width);
            int p9 = (x - 1) + ((y - 1) * width);
```

```
            // a)
            if ( output[p2] == 0 ) b++;
            if ( output[p3] == 0 ) b++;
            if ( output[p4] == 0 ) b++;
            if ( output[p5] == 0 ) b++;
            if ( output[p6] == 0 ) b++;
            if ( output[p7] == 0 ) b++;
            if ( output[p8] == 0 ) b++;
            if ( output[p9] == 0 ) b++;
            BOOL deleteA = ( (b <= 6) && (b >= 3) );

            // b)
            if ( (output[p2] == 255) && (output[p3] == 0) ) a++;
            if ( (output[p3] == 255) && (output[p4] == 0) ) a++;
            if ( (output[p4] == 255) && (output[p5] == 0) ) a++;
            if ( (output[p5] == 255) && (output[p6] == 0) ) a++;
            if ( (output[p6] == 255) && (output[p7] == 0) ) a++;
            if ( (output[p7] == 255) && (output[p8] == 0) ) a++;
            if ( (output[p8] == 255) && (output[p9] == 0) ) a++;
            if ( (output[p9] == 255) && (output[p2] == 0) ) a++;
            BOOL deleteB = (a == 1);

            // c) and d) if neighbours are white.
            BOOL deleteC = ((output[p2] == 255) || (output[p4] == 255) ||
(output[p6] == 255));
            BOOL deleteD = ((output[p4] == 255) || (output[p6] == 255) ||
(output[p8] == 255));

            if ( deleteA && deleteB && deleteC && deleteD )
            {
                output[p1] = 255;
                change = YES;
            }

        }
    }

    if ( change ) complete = NO;

}


- (void) subIteration2
{
    complete = YES;

    BOOL change = NO;
```

```
int size = 3;
int padding = (size - 1) / 2.0;

for ( int y = padding; y < height - padding; y++ )
{
    for (int x = padding; x < width - padding; x++)
    {

        int p1 = (x) + (y * width);

        if ( output[p1] != 0 ) continue;

        int a = 0;
        int b = 0;

        int p2 = (x - 1) + (y * width);
        int p3 = (x - 1) + ((y + 1) * width);
        int p4 = (x) + ((y + 1) * width);
        int p5 = (x + 1) + ((y + 1) * width);
        int p6 = (x + 1) + (y * width);
        int p7 = (x + 1) + ((y - 1) * width);
        int p8 = (x) + ((y - 1) * width);
        int p9 = (x - 1) + ((y - 1) * width);

        // a) has 3, 4, 5 neighbours
        if ( output[p2] == 0 ) b++;
        if ( output[p3] == 0 ) b++;
        if ( output[p4] == 0 ) b++;
        if ( output[p5] == 0 ) b++;
        if ( output[p6] == 0 ) b++;
        if ( output[p7] == 0 ) b++;
        if ( output[p8] == 0 ) b++;
        if ( output[p9] == 0 ) b++;
        BOOL deleteA = ( (b <= 6) && (b >= 3) );

        // b) transitions between 0 -> 1 (white -> block)
        if ( (output[p2] == 255) && (output[p3] == 0) ) a++;
        if ( (output[p3] == 255) && (output[p4] == 0) ) a++;
        if ( (output[p4] == 255) && (output[p5] == 0) ) a++;
        if ( (output[p5] == 255) && (output[p6] == 0) ) a++;
        if ( (output[p6] == 255) && (output[p7] == 0) ) a++;
        if ( (output[p7] == 255) && (output[p8] == 0) ) a++;
        if ( (output[p8] == 255) && (output[p9] == 0) ) a++;
        if ( (output[p9] == 255) && (output[p2] == 0) ) a++;
        BOOL deleteB = (a == 1);
```

```
            // c)
            BOOL deleteC = ( (output[p2] == 255) || (output[p4] == 255) ||
(output[p6] == 255) );
            BOOL deleteD = ( (output[p2] == 255) || (output[p6] == 255) ||
(output[p8] == 255) );

            if ( deleteA && deleteB && deleteC && deleteD )
            {
                output[p1] = 255;
                change = YES;
            }

        }
    }

    if ( change ) complete = NO;

}

@end
```

```
//
//  Morphology.h
//  ImageProcessingCLI
//
//  Created by James Mitchell on 04/02/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import <Foundation/Foundation.h>
@import AppKit;

@interface Morphology : NSObject


- (NSBitmapImageRep*) openingOnImage:(NSImage*)image withNeighbourhoodSize:
(int)size;
- (NSBitmapImageRep*) closingOnImage:(NSImage*)image withNeighbourhoodSize:
(int)size;

- (NSBitmapImageRep*) simpleDilationOfImage:(NSImage*)image withNeighbourhoodSize:
(int)size;
- (NSBitmapImageRep*) simpleErosionOfImage:(NSImage*)image withNeighbourhoodSize:
(int)size;

- (NSBitmapImageRep*) processImage:(NSImage *)image
                    withBackground:(int)background
                     andForeground:(int)foreground
              andNeighbourhoodSize:(int)element;

@end
```

```
//
//  Morphology.m
//  ImageProcessingCLI
//
//  Created by James Mitchell on 04/02/2016.
//  Copyright © 2016 James Mitchell. All rights reserved.
//

#import "Morphology.h"
#import "ImageRepresentation.h"

@implementation Morphology
```

```
- (NSBitmapImageRep*) openingOnImage:(NSImage*)image
                withNeighbourhoodSize:(int)size
{
    NSImage* temp = [[NSImage alloc] initWithSize:image.size];
    NSBitmapImageRep* eroded = [self simpleErosionOfImage:image
                                       withNeighbourhoodSize:size];

    temp = [ImageRepresentation cacheImageFromRepresentation:eroded];
    NSBitmapImageRep* dilated = [self simpleDilationOfImage:temp
                                         withNeighbourhoodSize:size];
    return dilated;
}

- (NSBitmapImageRep*) closingOnImage:(NSImage*)image
                withNeighbourhoodSize:(int)size
{
    NSImage* temp = [[NSImage alloc] initWithSize:image.size];
    NSBitmapImageRep* dilated = [self simpleDilationOfImage:image
                                         withNeighbourhoodSize:size];

    temp = [ImageRepresentation cacheImageFromRepresentation:dilated];
    NSBitmapImageRep* eroded = [self simpleErosionOfImage:temp
                                       withNeighbourhoodSize:size];
    return eroded;
}


- (NSBitmapImageRep*) simpleDilationOfImage:(NSImage*)image withNeighbourhoodSize:
(int)size
{
    return [self processImage:image
                withBackground:255
                 andForeground:0
          andNeighbourhoodSize:size];
}


- (NSBitmapImageRep*) simpleErosionOfImage:(NSImage*)image withNeighbourhoodSize:
(int)size
{
    return [self processImage:image
                withBackground:0
                 andForeground:255
          andNeighbourhoodSize:size];
}

- (NSBitmapImageRep*) processImage:(NSImage *)image
```

```
                    withBackground:(int)background
                     andForeground:(int)foreground
                andNeighbourhoodSize:(int)size
{

    NSBitmapImageRep* representation = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char *original = [representation bitmapData];

    NSBitmapImageRep* output = [ImageRepresentation
grayScaleRepresentationOfImage:image];
    unsigned char* processed = [output bitmapData];

    int width = image.size.width;
    int height = image.size.height;

    int padding = (size - 1) / 2.0;
    //    int filter[size * size];

    for ( int y = padding; y < height - padding; y++ )
    {
        for (int x = padding; x < width - padding; x++)
        {
            int centre = x + y * width;
            BOOL hits = NO;

            for (int s = -padding; s < (padding + 1); s++) {
                for (int t = -padding; t < (padding + 1); t++) {

                    int index = (x + s) + ((y + t) * width);

                    if ( original[index] == foreground )
                    {
                        hits = YES;
                    }
                }
            }

            processed[centre] = background;
            if ( hits )
            {
                processed[centre] = foreground;
            }
        }
    }

    return output;
```

```
}

@end
```

TOWARDS THE DEVELOPMENT OF IMAGE PROCESSING AND ANALYSIS TECHNIQUES TO INFLUENCE CREATION OF STRUCTURED TYPEFACES FROM HANDWRITTEN TEXT IMAGES.

JAMES MITCHELL - 10425907 - Towards the development of Image processing and analysis techniques to influence creation of structured typefaces from handwritten text images.

INTRODUCTION.

Written letters can be used only during the process of writing its self: the moment of production and use are one and the same.

– Fred Smeijers

Smeijers, 1996

Smeijers, 1996 Counterpunch: Making type in the sixteenth century, designing typefaces now. United Kingdom: Chronicle Books.

THE PROCESS OF DESIGNING A TYPEFACE HAS A HIGH BAR FOR ENTRY

TAKING MANY HOURS TO CREATE AND MANY MORE HOURS TO MASTER.

TYPE DEVELOPMENT IS NOT A SKILL AVAILABLE TO ALL

DESPITE ARTISTIC ABILITY OR INTENT.

THIS FACT TAKES THE POTENTIAL FOR COMPLETE PERSONAL AND VISUAL EXPRESSION AWAY FROM EVEN THE MOST CREATIVE USERS.

THE PRIMARY OBJECTIVE

DESIGN AND IMPLEMENT AN APPLICATION THAT WILL LOWER THE BAR FOR THE AVERAGE USER TO CREATE TYPEFACE FONTS FROM IMAGES OF HANDWRITTEN TEXT.

DESIRED FUNCTION.

Original Image Grayscaled

Thresholded 30%

abcdefghij
klmnopqrst
uvwxyz

BACKGROUND

METHODOLOGY

BREAKING DOWN THE PROBLEM.
COLLECTION OF RESOURCES.
RESEARCH, DEVELOPMENT, IMPLEMENTATION.

BREAKING DOWN THE PROBLEM

IMAGE PROCESSING AND ANALYSIS

LETTERFORM COMPOSITION.

GENERATING OUTPUT.

USER INTERFACE DEVELOPMENT.

IMAGE PROCESSING

CREATING THE RIGHT ENVIRONMENT.

AND ANALYSIS

FINDING THE FEATURES

LETTERFORM COMPOSITION

CREATION OF A TEMPLATE. APPLYING ATTRIBUTES.

GENERATING OUTPUT

SVG.
TRUETYPE.
OPENTYPE.

USER INTERFACE DEVELOPMENT.

PROVIDE STATION FOR INPUT.
OPTIONS FOR OUTPUT.

COLLECTION OF RESOURCES

HANDWRITTEN SUBJECTS. RESEARCH MATERIALS.

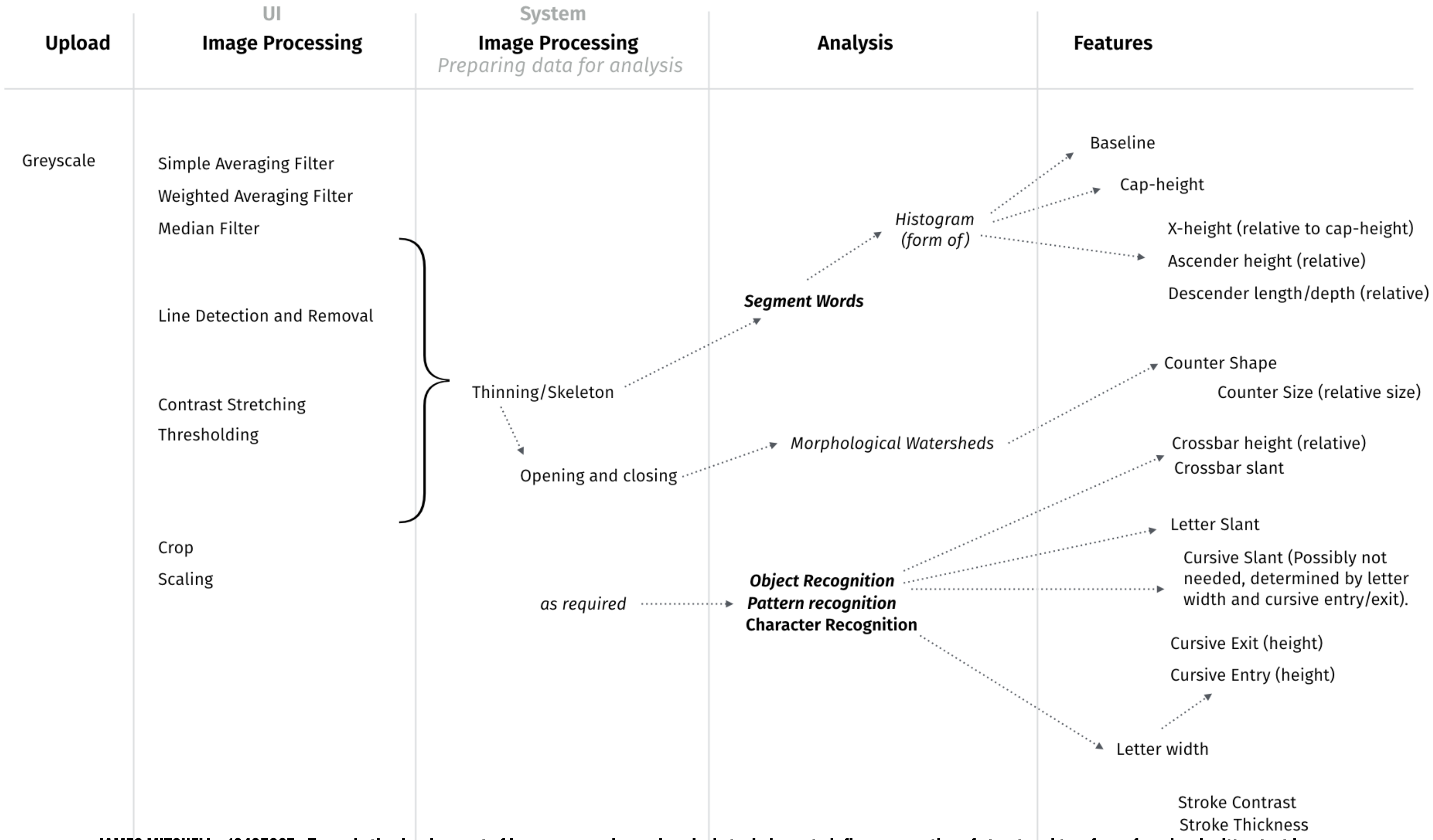
RESEARCH DEVELOPMENT AND IMPLEMENTATION.

BACKGROUND RESEARCH
PSEUDO CODE DEVELOPMENT
IMPLEMENTATION.

RESEARCH DEVELOPMENT AND IMPLEMENTATION.

IMAGE PROCESSING AND ANALYSIS.

JAMES MITCHELL - 10425907 - Towards the development of Image processing and analysis techniques to influence creation of structured typefaces from handwritten text images.


UI
System
Upload
Image Processing
Image Processing
Preparing data for analysis
Analysis
Features
Greyscale
Simple Averaging Filter
Weighted Averaging Filter
Median Filter
Line Detection and Removal
Contrast Stretching
Thresholding
Crop
Scaling
Thinning/Skeleton
Segment Words
Histogram
(form of)
Baseline
Cap-height
X-height (relative to cap-height)
Ascender height (relative)
Descender length/depth (relative)
Opening and closing
Morphological Watersheds
Counter Shape
Counter Size (relative size)
Crossbar height (relative)
Crossbar slant
as required
Object Recognition
Pattern recognition
Character Recognition
Letter Slant
Cursive Slant (Possibly not needed, determined by letter width and cursive entry/exit).
Cursive Exit (height)
Cursive Entry (height)
Letter width
Stroke Contrast
Stroke Thickness

RESEARCH DEVELOPMENT AND IMPLEMENTATION.

THRESHOLDING.

JAMES MITCHELL - 10425907 - Towards the development of Image processing and analysis techniques to influence creation of structured typefaces from handwritten text images.

THRESHOLDING.

FINDING THE EQUATION.

$$g(x,y) = T[f(x,y)]$$

$$g(x,y) = \begin{cases} 1 & if & f(x,y) & > & T \\ 0 & if & f(x,y) & \leq & T \end{cases}$$

Gonzalez and Woods, 2002

RESEARCH, DEVELOPMENT AND IMPLEMENTATION

DEVELOPING THE PSEUDO CODE.

```
T = threshold value

for each row y in representation
    for each column x in row

    if pixel(x, y) > T then
        pixel(x, y) greylevel = 1
    else
        pixel(x, y) greylevel = 0
```

RESEARCH, DEVELOPMENT AND IMPLEMENTATION

IMPLEMENTING.

```
- (NSBitmapImageRep *) thresholdWithValue:(int)value
{

    // get the image data
     NSBitmapImageRep *output = [self grayScaleRepresentationOfImage:self.image;
     unsigned char *data = [output bitmapData];

     // ..
}
```

```
    // Iterate over the image.

    for ( int y = 0; y < self.height; y++ )
    {
        for ( int x = 0; x < self.width; x++ )
        {
            int index = x + (y * self.width);

            // ..

        }
    }

    return output;
}
```

```
// apply the threshold
if ( data[index] < value) {
     data[index] = 0;
} else {
     data[index] = 255;
}
```

DEMONSTRATION.

CRITICAL APPRAISAL

JAMES MITCHELL - 10425907 - Towards the development of Image processing and analysis techniques to influence creation of structured typefaces from handwritten text images.

1. DELAY IN ESTABLISHING THE METHODOLOGY FOR RESEARCH AND DEVELOPMENT.

2. THE INCORRECT ORDER OF RESEARCH FOCUS.

2. THE INCORRECT ORDER OF RESEARCH FOCUS.

3. INABILITY TO CORRECTLY ESTABLISH THE PROBLEM PRIOR TO IMPLEMENTATION OF SOME FEATURES.