7.1 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
```

```objc
//
//  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
```

```objc
//
//  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/CocoaDraw
ingGuide/Images/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/CocoaDraw
ingGuide/Images/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
```