```
 1: //
 2: //  AppDelegate.h
 3: //  Quartz2DPaint
 4: //
 5: //  Created by cj on 5/7/13.
 6: //  Copyright (c) 2013 cjdesch. All rights reserved.
 7: //
 8:
 9: #import <Cocoa/Cocoa.h>
10:
11: @interface AppDelegate : NSObject <NSApplicationDelegate>
12:
13: @property (assign) IBOutlet NSWindow *window;
14:
15: @end
```

```
 1: //
 2: //  AppDelegate.m
 3: //  Quartz2DPaint
 4: //
 5: //  Created by cj on 5/7/13.
 6: //  Copyright (c) 2013 cjdesch. All rights reserved.
 7: //
 8:
 9: #import "AppDelegate.h"
10:
11: @implementation AppDelegate
12:
13: - (void)applicationDidFinishLaunching:(NSNotification *)aNotification
14: {
15:     // Insert code here to initialize your application
16: }
17:
18: @end
```

```
 1: //
 2: //  Brush.h
 3: //  Paint
 4: //
 5:
 6:
 7:
 8:
 9: @class Canvas;
10:
11: @interface Brush : NSObject {
12:         // Information about the brush that's always used
13:         float                           mRadius;
14:         CGMutablePathRef        mShape;
15:         CGColorRef                      mColor;
16:         float                           mSoftness; // 0.0 - 1.0, 0.0 being hard, 1.0 be all soft
17:         BOOL                            mHard; // should have a hard edge?
18:
19:         // Cached information that's only used when actually tracking/drawing
20:         CGImageRef                      mMask;
21:         NSPoint                         mLastPoint;
22:         float                           mLeftOverDistance;
23: }
24:
25: - (void) mouseDown:(NSEvent *)theEvent inView:(NSView *)view onCanvas:(Canvas *)canvas;
26: - (void) mouseDragged:(NSEvent *)theEvent inView:(NSView *)view onCanvas:(Canvas *)canvas;
27: - (void) mouseUp:(NSEvent *)theEvent inView:(NSView *)view onCanvas:(Canvas *)canvas;
28:
29: @end
```

```
    1: //
    2: //  Brush.m
    3: //  Paint
    4: //
    5:
    6:
    7: #import "Brush.h"
    8: #import "Canvas.h"
    9: #import <QuartzCore/QuartzCore.h>
   10:
   11: @interface Brush (Private)
   12:
   13: - (NSPoint) canvasLocation:(NSEvent *)theEvent view:(NSView *)view;
   14: - (void) stampStart:(NSPoint)startPoint end:(NSPoint)endPoint inView:(NSView *)view onCanvas:(Canvas *
)canvas;
   15:
   16: - (CGContextRef) createBitmapContext;
   17: - (void) disposeBitmapContext:(CGContextRef)bitmapContext;
   18: - (CGImageRef) createShapeImage;
   19:
   20: @end
   21:
   22: @implementation Brush
   23:
   24: - (id) init
   25: {
   26:         self = [super init];
   27:
   28:         if ( self ) {
   29:                 mRadius = 10.0;
   30:
   31:                 // Create the shape of the tip of the brush. Code currently assumes the bounding
   32:                 //      box of the shape is square (height == width)
   33:                 mShape = CGPathCreateMutable();
   34:                 CGPathAddEllipseInRect(mShape, nil, CGRectMake(0, 0, 2 * mRadius, 2 * mRadius));
   35:                 //CGPathAddRect(mShape, nil, CGRectMake(0, 0, 2 * mRadius, 2 * mRadius));
   36:
   37:                 // Create the color for the brush
   38:                 CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(kCGColorSpaceGenericRGB);
   39:                 float components[] = { 0.0, 0.0, 1.0, 1.0 }; // I like blue
   40:                 mColor = CGColorCreate(colorspace, components);
   41:                 CGColorSpaceRelease(colorspace);
   42:
   43:                 // The "softness" of the brush edges
   44:                 mSoftness = 0.5;
   45:                 mHard = NO;
   46:
   47:                 // Initialize variables that will be used during tracking
   48:                 mMask = nil;
   49:                 mLastPoint = NSZeroPoint;
   50:                 mLeftOverDistance = 0.0;
   51:         }
   52:
   53:         return self;
   54: }
   55:
   56: - (void) dealloc
   57: {
   58:         // Clean up our shape and color
   59:         CGPathRelease(mShape);
   60:         CGColorRelease(mColor);
   61:
   62:
   63: }
   64:
   65: - (void) mouseDown:(NSEvent *)theEvent inView:(NSView *)view onCanvas:(Canvas *)canvas
   66: {
   67:     NSLog(@"BrushDown");
   68:         // Translate the event point location into a canvas point
   69:         NSPoint currentPoint = [self canvasLocation:theEvent view:view];
   70:
   71:         // Initialize all the tracking information. This includes creating an image
   72:         //      of the brush tip
   73:         mMask = [self createShapeImage];
   74:         mLastPoint = currentPoint;
   75:         mLeftOverDistance = 0.0;
   76:
   77:         // Since this is a mouse down, we want to stamp the brush's image not matter
   78:         //      what.
   79:         [canvas stampMask:mMask at:currentPoint];
```

```
   80:
   81:         // This isn't very efficient, but we need to tell the view to redraw. A better
   82:         //      version would have the canvas itself to generate an invalidate for the view
   83:         //      (since it knows exactly where the bits changed).
   84:         [view setNeedsDisplay:YES];
   85: }
   86:
   87: - (void) mouseDragged:(NSEvent *)theEvent inView:(NSView *)view onCanvas:(Canvas *)canvas
   88: {
   89:
   90:         NSLog(@"Brushdragg");
   91:         // Translate the event point location into a canvas point
   92:         NSPoint currentPoint = [self canvasLocation:theEvent view:view];
   93:
   94:         // Stamp the brush in a line, from the last mouse location to the current one
   95:         [self stampStart:mLastPoint end:currentPoint inView:view onCanvas:canvas];
   96:
   97:         // Remember the current point, so that next time we know where to start
   98:         //      the line
   99:         mLastPoint = currentPoint;
  100: }
  101:
  102: - (void) mouseUp:(NSEvent *)theEvent inView:(NSView *)view onCanvas:(Canvas *)canvas
  103: {
  104:         // Translate the event point location into a canvas point
  105:         NSPoint currentPoint = [self canvasLocation:theEvent view:view];
  106:
  107:         // Stamp the brush in a line, from the last mouse location to the current one
  108:         [self stampStart:mLastPoint end:currentPoint inView:view onCanvas:canvas];
  109:
  110:         // This is a mouse up, so we are done tracking. Use this opportunity to clean
  111:         //      up all the tracking information, including the brush tip image.
  112:         CGImageRelease(mMask);
  113:         mMask = nil;
  114:         mLastPoint = NSZeroPoint;
  115:         mLeftOverDistance = 0.0;
  116: }
  117:
  118: @end
  119:
  120: @implementation Brush (Private)
  121:
  122: - (NSPoint) canvasLocation:(NSEvent *)theEvent view:(NSView *)view
  123: {
  124:         // Currently we assume that the NSView here is a CanvasView, which means
  125:         //      that the view is not scaled or offset. i.e. There is a one to one
  126:         //      correlation between the view coordinates and the canvas coordinates.
  127:         NSPoint eventLocation = [theEvent locationInWindow];
  128:         return [view convertPoint:eventLocation fromView:nil];
  129: }
  130:
  131: - (void) stampStart:(NSPoint)startPoint end:(NSPoint)endPoint inView:(NSView *)view onCanvas:(Canvas *
)canvas
  132: {
  133:         // We need to ask the canvas to draw a line using the brush. Keep track
  134:         //      of the distance left over that we didn't draw this time (so we draw
  135:         //      it next time).
  136:         mLeftOverDistance = [canvas stampMask:mMask from:startPoint to:endPoint leftOverDistance:mLeft
OverDistance];
  137:
  138:         // This isn't very efficient, but we need to tell the view to redraw. A better
  139:         //      version would have the canvas itself to generate an invalidate for the view
  140:         //      (since it knows exactly where the bits changed).
  141:         [view setNeedsDisplay:YES];
  142: }
  143:
  144: - (CGContextRef) createBitmapContext
  145: {
  146:         // Create the offscreen bitmap context that we can draw the brush tip into.
  147:         //      The context should be the size of the shape bounding box.
  148:         CGRect boundingBox = CGPathGetBoundingBox(mShape);
  149:
  150:         size_t width = CGRectGetWidth(boundingBox);
  151:         size_t height = CGRectGetHeight(boundingBox);
  152:         size_t bitsPerComponent = 8;
  153:         size_t bytesPerRow = ((width * 4) + 0x0000000F) & ~0x0000000F; // 16 byte aligned is good
  154:         size_t dataSize = bytesPerRow * height;
  155:         void* data = calloc(1, dataSize);
  156:         CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(kCGColorSpaceGenericRGB);
  157:
```

```
158:         CGContextRef bitmapContext = CGBitmapContextCreate(data, width, height, bitsPerComponent,
159:
 bytesPerRow, colorspace,
160:
 kCGImageAlphaPremultipliedFirst);
161:
162:         CGColorSpaceRelease(colorspace);
163:
164:         // Clear the context to transparent, 'cause we'll be using transparency
165:         CGContextClearRect(bitmapContext, CGRectMake(0, 0, width, height));
166:
167:         return bitmapContext;
168: }
169:
170: - (void) disposeBitmapContext:(CGContextRef)bitmapContext
171: {
172:         // Free up the offscreen bitmap
173:         void * data = CGBitmapContextGetData(bitmapContext);
174:         CGContextRelease(bitmapContext);
175:         free(data);
176: }
177:
178: - (CGImageRef) createShapeImage
179: {
180:         // Create a bitmap context to hold our brush image
181:         CGContextRef bitmapContext = [self createBitmapContext];
182:
183:         // If we're not going to have a hard edge, set the alpha to 50% (using a
184:         //      transparency layer) so the brush strokes fade in and out more.
185:         if ( !mHard )
186:                 CGContextSetAlpha(bitmapContext, 0.5);
187:         CGContextBeginTransparencyLayer(bitmapContext, nil);
188:
189:         // I like a little color in my brushes
190:         CGContextSetFillColorWithColor(bitmapContext, mColor);
191:
192:         // The way we acheive "softness" on the edges of the brush is to draw
193:         //      the shape full size with some transparency, then keep drawing the shape
194:         //      at smaller sizes with the same transparency level. Thus, the center
195:         //      builds up and is darker, while edges remain partially transparent.
196:
197:         // First, based on the softness setting, determine the radius of the fully
198:         //      opaque pixels.
199:         int innerRadius = (int)ceil(mSoftness * (0.5 - mRadius) + mRadius);
200:         int outerRadius = (int)ceil(mRadius);
201:         int i = 0;
202:
203:         // The alpha level is always proportial to the difference between the inner, opaque
204:         //      radius and the outer, transparent radius.
205:         float alphaStep = 1.0 / (outerRadius - innerRadius + 1);
206:
207:         // Since we're drawing shape on top of shape, we only need to set the alpha once
208:         CGContextSetAlpha(bitmapContext, alphaStep);
209:
210:         for (i = outerRadius; i >= innerRadius; --i) {
211:                 CGContextSaveGState(bitmapContext);
212:
213:                 // First, center the shape onto the context.
214:                 CGContextTranslateCTM(bitmapContext, outerRadius - i, outerRadius - i);
215:
216:                 // Second, scale the the brush shape, such that each successive iteration
217:                 //      is two pixels smaller in width and height than the previous iteration.
218:                 float scale = (2.0 * (float)i) / (2.0 * (float)outerRadius);
219:                 CGContextScaleCTM(bitmapContext, scale, scale);
220:
221:                 // Finally, actually add the path and fill it
222:                 CGContextAddPath(bitmapContext, mShape);
223:                 CGContextEOFillPath(bitmapContext);
224:
225:                 CGContextRestoreGState(bitmapContext);
226:         }
227:
228:         // We're done drawing, composite the tip onto the context using whatever
229:         //      alpha we had set up before BeginTransparencyLayer.
230:         CGContextEndTransparencyLayer(bitmapContext);
231:
232:         // Create the brush tip image from our bitmap context
233:         CGImageRef image = CGBitmapContextCreateImage(bitmapContext);
234:
235:         // Free up the offscreen bitmap
```

```
236:         [self disposeBitmapContext:bitmapContext];
237:
238:         return image;
239: }
240:
241:
242: @end
```

```
    1: //
    2: //  Canvas.h
    3: //  Paint
    4: //
    5:
    6: @interface Canvas : NSObject {
    7:         // The canvas is simply backed by a bitmap context. A CGLayerRef would be
    8:         //      better, but you have to know your destination context before you create
    9:         //      it (which we don't).
   10:         CGContextRef    mBitmapContext;
   11: }
   12:
   13: // Constructor that creates a canvas at the specified size. Canvas cannot be resized.
   14: - (id) initWithSize:(NSSize)size;
   15:
   16: // Draws the contents of the canvas into the specified context. Handy for views
   17: //      that host a canvas.
   18: - (void)drawRect:(NSRect)rect inContext:(NSGraphicsContext*)context;
   19:
   20: // Graphics privimites for the canvas. The first draws a line given the brush
   21: //      image, and the second, draws a point given the brush image.
   22: - (float)stampMask:(CGImageRef)mask from:(NSPoint)startPoint to:(NSPoint)endPoint leftOverDistance:(fl
oat)leftOverDistance;
   23: - (void)stampMask:(CGImageRef)mask at:(NSPoint)point;
   24:
   25: @end
```

```
    1: //
    2: //  Canvas.m
    3: //  Paint
    4: //
    5:
    6:
    7: #import "Canvas.h"
    8:
    9: @implementation Canvas
   10:
   11: - (id) initWithSize:(NSSize)size
   12: {
   13:         self = [super init];
   14:
   15:         if ( self ) {
   16:                 // Create a bitmap context for the canvas. To keep things simple
   17:                 //      we're going to use a 32-bit ARGB format.
   18:                 size_t width = size.width;
   19:                 size_t height = size.height;
   20:                 size_t bitsPerComponent = 8;
   21:                 size_t bytesPerRow = ((width * 4) + 0x0000000F) & ~0x0000000F; // 16-byte aligned is g
ood
   22:                 size_t dataSize = bytesPerRow * height;
   23:                 void* data = calloc(1, dataSize);
   24:                 CGColorSpaceRef colorspace = CGColorSpaceCreateWithName(kCGColorSpaceGenericRGB);
   25:
   26:                 mBitmapContext = CGBitmapContextCreate(data, width, height, bitsPerComponent,
   27:                                                                                 bytesPerRow,
 colorspace,
   28:                                                                                 kCGImageAlph
aPremultipliedFirst);
   29:
   30:                 CGColorSpaceRelease(colorspace);
   31:
   32:                 // Paint on a white background so the user has something to start with.
   33:                 CGContextSaveGState(mBitmapContext);
   34:
   35:                 CGRect fillRect = CGRectMake(0, 0, CGBitmapContextGetWidth(mBitmapContext),
   36:                                                                         CGBitmapContextGetHeight(mBi
tmapContext));
   37:
   38:                 CGContextSetRGBFillColor(mBitmapContext, 1.0, 1.0, 1.0, 1.0);
   39:                 CGContextFillRect(mBitmapContext, fillRect);
   40:
   41:                 CGContextRestoreGState(mBitmapContext);
   42:         }
   43:
   44:         return self;
   45: }
   46:
   47: - (void) dealloc
   48: {
   49:         // Free up our bitmap context
   50:         void* data = CGBitmapContextGetData(mBitmapContext);
   51:         CGContextRelease(mBitmapContext);
   52:         free(data);
   53:
   54: }
   55:
   56: - (void)drawRect:(NSRect)rect inContext:(NSGraphicsContext*)context
   57: {
   58:         // Here we simply want to render our bitmap context into the view's
   59:         //      context. It's going to be a straight forward bit blit. First,
   60:         //      create an image from our bitmap context.
   61:         CGImageRef imageRef = CGBitmapContextCreateImage(mBitmapContext);
   62:
   63:         // Grab the destination context
   64:         CGContextRef contextRef = [context graphicsPort];
   65:         CGContextSaveGState(contextRef);
   66:
   67:         // Composite on the image at the bottom left of the context
   68:         CGRect imageRect = CGRectMake(0, 0, CGBitmapContextGetWidth(mBitmapContext),
   69:                                                                 CGBitmapContextGetHeight(mBitmapCont
ext));
   70:         CGContextDrawImage(contextRef, imageRect, imageRef);
   71:
   72:         CGImageRelease(imageRef);
   73:
   74:         CGContextRestoreGState(contextRef);
   75: }
```

```
   76:
   77: - (float)stampMask:(CGImageRef)mask from:(NSPoint)startPoint to:(NSPoint)endPoint leftOverDistance:(fl
oat)leftOverDistance
   78: {
   79:         // Set the spacing between the stamps. By trail and error, I've
   80:         //      determined that 1/10 of the brush width (currently hard coded to 20)
   81:         //      is a good interval.
   82:         float spacing = CGImageGetWidth(mask) * 0.1;
   83:
   84:         // Anything less that half a pixel is overkill and could hurt performance.
   85:         if ( spacing < 0.5 )
   86:                 spacing = 0.5;
   87:
   88:         // Determine the delta of the x and y. This will determine the slope
   89:         //      of the line we want to draw.
   90:         float deltaX = endPoint.x - startPoint.x;
   91:         float deltaY = endPoint.y - startPoint.y;
   92:
   93:         // Normalize the delta vector we just computed, and that becomes our step increment
   94:         //      for drawing our line, since the distance of a normalized vector is always 1
   95:         float distance = sqrt( deltaX * deltaX + deltaY * deltaY );
   96:         float stepX = 0.0;
   97:         float stepY = 0.0;
   98:         if ( distance > 0.0 ) {
   99:                 float invertDistance = 1.0 / distance;
  100:                 stepX = deltaX * invertDistance;
  101:                 stepY = deltaY * invertDistance;
  102:         }
  103:
  104:         float offsetX = 0.0;
  105:         float offsetY = 0.0;
  106:
  107:         // We're careful to only stamp at the specified interval, so its possible
  108:         //      that we have the last part of the previous line left to draw. Be sure
  109:         //      to add that into the total distance we have to draw.
  110:         float totalDistance = leftOverDistance + distance;
  111:
  112:         // While we still have distance to cover, stamp
  113:         while ( totalDistance >= spacing ) {
  114:                 // Increment where we put the stamp
  115:                 if ( leftOverDistance > 0 ) {
  116:                         // If we're making up distance we didn't cover the last
  117:                         //      time we drew a line, take that into account when calculating
  118:                         //      the offset. leftOverDistance is always < spacing.
  119:                         offsetX += stepX * (spacing - leftOverDistance);
  120:                         offsetY += stepY * (spacing - leftOverDistance);
  121:
  122:                         leftOverDistance -= spacing;
  123:                 } else {
  124:                         // The normal case. The offset increment is the normalized vector
  125:                         //      times the spacing
  126:                         offsetX += stepX * spacing;
  127:                         offsetY += stepY * spacing;
  128:                 }
  129:
  130:                 // Calculate where to put the current stamp at.
  131:                 NSPoint stampAt = NSMakePoint(startPoint.x + offsetX, startPoint.y + offsetY);
  132:
  133:                 // Ka-chunk! Draw the image at the current location
  134:                 [self stampMask:mask at: stampAt];
  135:
  136:                 // Remove the distance we just covered
  137:                 totalDistance -= spacing;
  138:         }
  139:
  140:         // Return the distance that we didn't get to cover when drawing the line.
  141:         //      It is going to be less than spacing.
  142:         return totalDistance;
  143: }
  144:
  145: - (void)stampMask:(CGImageRef)mask at:(NSPoint)point
  146: {
  147:         // When we stamp the image, we want the center of the image to be
  148:         //      at the point specified.
  149:         CGContextSaveGState(mBitmapContext);
  150:
  151:         // So we can position the image correct, compute where the bottom left
  152:         //      of the image should go, and modify the CTM so that 0, 0 is there.
  153:         CGPoint bottomLeft = CGPointMake( point.x - CGImageGetWidth(mask) * 0.5,
  154:                                                                         point.y - CGImageGetHeight(m
```

```
ask) * 0.5 );
  155:         CGContextTranslateCTM(mBitmapContext, bottomLeft.x, bottomLeft.y);
  156:
  157:         // Now that it's properly lined up, draw the image
  158:         CGRect maskRect = CGRectMake(0, 0, CGImageGetWidth(mask), CGImageGetHeight(mask));
  159:         CGContextDrawImage(mBitmapContext, maskRect, mask);
  160:
  161:         CGContextRestoreGState(mBitmapContext);
  162: }
  163:
  164: @end
```

```
 1: //
 2: //  CanvasView.h
 3: //  Paint
 4: //
 5:
 6:
 7: @class Canvas;
 8: @class Brush;
 9:
10: @interface CanvasView : NSView {
11:         // The canvas we will render to the screen
12:         Canvas          *canvas;
13:         // The brush that we will pass events to
14:         Brush           *brush;
15: }
16:
17: @end
```

```
 1: //
 2: //  CanvasView.m
 3: //  Paint
 4: //
 5:
 6:
 7: #import "CanvasView.h"
 8: #import "Canvas.h"
 9: #import "Brush.h"
10:
11: @implementation CanvasView
12:
13: - (id)initWithFrame:(NSRect)frame {
14:     self = [super initWithFrame:frame];
15:     if (self) {
16:         // Create both the canvas and the brush. Create the canvas
17:                 //      the same size as our initial size. Note that the canvas will not
18:                 //      resize along with us.
19:                 canvas = [[Canvas alloc] initWithSize:frame.size];
20:                 brush = [[Brush alloc] init];
21:     }
22:     return self;
23: }
24:
25:
26:
27: - (void)drawRect:(NSRect)rect {
28:     NSLog(@"Canvas DrawRect");
29:         // Simply ask the canvas to draw into the current context, given the
30:         //      rectangle specified. A more sophisticated view might draw a border
31:         //      around the canvas, or a pasteboard in the case that the view was
32:         //      bigger than the canvas.
33:         NSGraphicsContext* context = [NSGraphicsContext currentContext];
34:
35:         [canvas drawRect:rect inContext:context];
36: }
37:
38: - (void)mouseDown:(NSEvent *)theEvent
39: {
40:     NSLog(@"Canvas Mouse Down");
41:         // Simply pass the mouse event to the brush. Also give it the canvas to
42:         //      work on, and a reference to ourselves, so it can translate the mouse
43:         //      locations.
44:         [brush mouseDown:theEvent inView:self onCanvas:canvas];
45:
46:     [self setNeedsDisplay:YES];
47:
48:
49: }
50:
51: - (void)mouseDragged:(NSEvent *)theEvent
52: {
53:     NSLog(@"Canvas Mouse Dragged");
54:         // Simply pass the mouse event to the brush. Also give it the canvas to
55:         //      work on, and a reference to ourselves, so it can translate the mouse
56:         //      locations.
57:         [brush mouseDragged:theEvent inView:self onCanvas:canvas];
58: }
59:
60: - (void)mouseUp:(NSEvent *)theEvent
61: {
62:         // Simply pass the mouse event to the brush. Also give it the canvas to
63:         //      work on, and a reference to ourselves, so it can translate the mouse
64:         //      locations.
65:         [brush mouseUp:theEvent inView:self onCanvas:canvas];
66:
67:
68: }
69:
70: @end
```

```
    1: //
    2: //  DrawingTool.h
    3: //  LeapPaint
    4: //
    5: //  Created by cj on 3/17/13.
    6: //  Copyright (c) 2013 cjdesch. All rights reserved.
    7: //
    8:
    9: #import <Foundation/Foundation.h>
   10: #import "ToolSettings.h"
   11:
   12: @interface DrawingTool : NSObject
   13:
   14:
   15: @property (strong) NSImage *drawingSnapshot;
   16: @property (strong) NSImageView *drawingImageView;
   17: @property (weak) ToolSettings *settings;
   18: @property (assign) CGPoint firstPoint;
   19: @property (copy) NSString *imageName;
   20: @property (copy) NSString *toolName;
   21:
   22: - (void)inputBegan:(NSEvent *)theEvent locationInView:(CGPoint)currentPoint  withSettings:(ToolSetting
s *)settings;
   23: - (void)inputMoved:(NSEvent *)theEvent locationInView:(CGPoint)currentPoint;
   24: - (void)inputEnded:(NSEvent *)theEvent locationInView:(CGPoint)currentPoint;
   25:
   26:
   27: @property (copy) dispatch_block_t completion;
   28:
   29: - (DrawingTool*)initWithCompletion:(dispatch_block_t)completion;
   30:
   31: @end
```

```
//
//  DrawingTool.m
//  LeapPaint
//
//  Created by cj on 3/17/13.
//  Copyright (c) 2013 cjdesch. All rights reserved.
//

#import "DrawingTool.h"

@implementation DrawingTool


- (DrawingTool*)initWithCompletion:(dispatch_block_t)completion {

    if (self = [super init])
    {
        self.completion = completion;
    }
    return self;

}

//override to disable double-buffered drawing (e.g. for eraser)
- (BOOL)doubleBufferDrawing
{
    return YES;
}


- (void)inputBegan:(NSEvent *)theEvent locationInView:(CGPoint)currentPoint withSettings:(ToolSettings
*)settings{
    self.settings = settings;
    self.firstPoint = currentPoint;
}

- (void)inputMoved:(NSEvent *)theEvent locationInView:(CGPoint)currentPoint{

}

- (void)inputEnded:(NSEvent *)theEvent locationInView:(CGPoint)currentPoint{

    if (self.completion) {
        self.completion();
    }
}


@end
```

```
 1: //
 2: //  LPBrushTool.h
 3: //  LeapPaint
 4: //
 5: //  Created by cj on 3/19/13.
 6: //  Copyright (c) 2013 cjdesch. All rights reserved.
 7: //
 8:
 9: #import <Foundation/Foundation.h>
10: #import <QuartzCore/QuartzCore.h>
11: #import "LPDrawingTool.h"
12:
13: @interface LPBrushTool : LPDrawingTool {
14:
15:     NSString* templateImageName;
16:     NSImage* templateImage;
17: }
18:
19:
20:
21: @end
```

```
    1: //
    2: //  LPBrushTool.m
    3: //  LeapPaint
    4: //
    5: //  Created by cj on 3/19/13.
    6: //  Copyright (c) 2013 cjdesch. All rights reserved.
    7: //
    8:
    9: #import "LPBrushTool.h"
   10:
   11: @implementation LPBrushTool
   12:
   13: - (void)inputBegan:(NSEvent *)theEvent  withSettings:(ToolSettings *)settings{
   14:     self.settings = settings;
   15:     templateImage = [NSImage imageNamed:templateImageName];
   16:     CGSize size = CGSizeMake(self.lineWidth, self.lineWidth);
   17:     templateImage = [self imageResize:templateImage newSize:NSSizeFromCGSize(size)];
   18:     //templateImage = self adjustImage:templateImage withHue:<#(float)#>
   19:
   20: }
   21:
   22: - (void)inputMoved:(NSEvent *)theEvent {
   23:
   24: }
   25:
   26: - (void)inputEnded:(NSEvent *)theEvent {
   27:
   28: }
   29:
   30:
   31: - (void)toolDidLoad{
   32:
   33:     //NSLog(@"Tool Did Load Subclass");
   34:    // templateImage = [NSImage imageNamed:templateImageName];
   35: //     self.template = [[NSImage imageNamed:templateImageName withTint:self.primaryColor] resizedImage
:CGSizeMake(self.lineWidth), self.lineWidth) interpolationQuality:kCGInterpolationDefault];
   36: }
   37:
   38: - (NSImage *)imageResize:(NSImage*)anImage
   39:                  newSize:(NSSize)newSize
   40: {
   41:     NSImage *sourceImage = anImage;
   42:     [sourceImage setScalesWhenResized:YES];
   43:
   44:     // Report an error if the source isn't a valid image
   45:     if (![sourceImage isValid])
   46:     {
   47:         NSLog(@"Invalid Image");
   48:     } else
   49:     {
   50:         NSImage *smallImage = [[NSImage alloc] initWithSize: newSize] ;
   51:         [smallImage lockFocus];
   52:         [sourceImage setSize: newSize];
   53:         [[NSGraphicsContext currentContext] setImageInterpolation:NSImageInterpolationHigh];
   54:         [sourceImage compositeToPoint:NSZeroPoint operation:NSCompositeCopy];
   55:         [smallImage unlockFocus];
   56:         return smallImage;
   57:     }
   58:     return nil;
   59: }
   60:
   61: - (NSImage*) adjustImage:(NSImage*)img withHue:(float)hue {
   62:
   63:     CIImage *inputImage = [[CIImage alloc] initWithData:[img TIFFRepresentation]];
   64:
   65:     CIFilter *hueAdjust = [CIFilter filterWithName:@"CIHueAdjust"];
   66:     [hueAdjust setValue: inputImage forKey: @"inputImage"];
   67:     [hueAdjust setValue: [NSNumber numberWithFloat: hue]
   68:                  forKey: @"inputAngle"];
   69:     CIImage *outputImage = [hueAdjust valueForKey: @"outputImage"];
   70:
   71:     NSImage *resultImage = [[NSImage alloc] initWithSize:[outputImage extent].size];
   72:     NSCIImageRep *rep = [NSCIImageRep imageRepWithCIImage:outputImage];
   73:     [resultImage addRepresentation:rep];
   74:
   75:     return resultImage;
   76:
   77: }
   78:
   79: @end
```

```
    1: //
    2: //  LPDrawingTool.h
    3: //  LeapPaint
    4: //
    5: //  Created by cj on 3/18/13.
    6: //  Copyright (c) 2013 cjdesch. All rights reserved.
    7: //
    8:
    9: #import <Foundation/Foundation.h>
   10: #import "ToolSettings.h"
   11:
   12:
   13: @interface LPDrawingTool : NSObject{
   14:
   15:
   16: }
   17:
   18:
   19: @property (copy) NSString *drawingTool;
   20: @property (assign) int lineWidth;
   21: @property (assign) int transparency;
   22: @property (assign) int fontSize;
   23: @property (strong) NSColor *primaryColor;
   24: @property (strong) NSColor *secondaryColor;
   25: @property (strong) NSMutableArray* points;
   26: @property (strong) NSString* brushTextureName;
   27: @property (weak) ToolSettings *settings;
   28:
   29:
   30: - (id)initWithWidth:(int)w withTransparency:(int)t withPrimaryColor:(NSColor*)p withSecondaryColor:(NS
Color*)s;
   31:
   32: - (void)toolDidLoad;
   33: @end
```

```
    1: //
    2: //  LPDrawingTool.m
    3: //  LeapPaint
    4: //
    5: //  Created by cj on 3/18/13.
    6: //  Copyright (c) 2013 cjdesch. All rights reserved.
    7: //
    8:
    9: #import "LPDrawingTool.h"
   10:
   11: @implementation LPDrawingTool
   12: @synthesize lineWidth;
   13: @synthesize transparency;
   14:
   15: @synthesize primaryColor;
   16: @synthesize secondaryColor;
   17: @synthesize points;
   18: @synthesize drawingTool;
   19:
   20:
   21: - (id)initWithWidth:(int)w withTransparency:(int)t withPrimaryColor:(NSColor*)p withSecondaryColor:(NS
Color*)s{
   22:     if (self = [super init]){
   23:
   24:         self.points = [[NSMutableArray alloc] init];
   25:         self.primaryColor = p;
   26:         self.secondaryColor = s;
   27:
   28:         self.lineWidth = w;
   29:         self.transparency = t;
   30:
   31:         [self toolDidLoad];
   32:
   33:     }
   34:
   35:     return self;
   36: }
   37:
   38: - (void)inputBegan:(NSEvent *)theEvent  withSettings:(ToolSettings *)settings{
   39:     self.settings = settings;
   40:
   41: }
   42:
   43: - (void)inputMoved:(NSEvent *)theEvent {
   44:
   45: }
   46:
   47: - (void)inputEnded:(NSEvent *)theEvent {
   48:
   49: }
   50:
   51:
   52:
   53: - (void)toolDidLoad{
   54:     //NSLog(@"ToolDidLoad");
   55:
   56: }
   57:
   58: @end
```

```
 1: //
 2: //  LPPenTool.h
 3: //  LeapPaint
 4: //
 5: //  Created by cj on 3/19/13.
 6: //  Copyright (c) 2013 cjdesch. All rights reserved.
 7: //
 8:
 9: #import <Foundation/Foundation.h>
10: #import "LPDrawingTool.h"
11: @interface LPPenTool : LPDrawingTool
12:
13: @end
```

```
 1: //
 2: //  LPPenTool.m
 3: //  LeapPaint
 4: //
 5: //  Created by cj on 3/19/13.
 6: //  Copyright (c) 2013 cjdesch. All rights reserved.
 7: //
 8:
 9: #import "LPPenTool.h"
10:
11: @implementation LPPenTool
12:
13: @end
```

```
 1: //
 2: //  main.m
 3: //  Quartz2DPaint
 4: //
 5: //  Created by cj on 5/7/13.
 6: //  Copyright (c) 2013 cjdesch. All rights reserved.
 7: //
 8:
 9: #import <Cocoa/Cocoa.h>
10:
11: int main(int argc, char *argv[])
12: {
13:     return NSApplicationMain(argc, (const char **)argv);
14: }
```

```
 1: //
 2: //  MyPoint.h
 3: //  010-NSView
 4: //
 5: // wrapper for NSPoint
 6: #import <Cocoa/Cocoa.h>
 7:
 8: @interface MyPoint : NSObject {
 9:         NSPoint myNSPoint;
10: }
11: - (id) initWithNSPoint:(NSPoint)pNSPoint;
12: - (NSPoint) myNSPoint;
13: - (float)x;
14: - (float)y;
15:
16: @end
```

```
 1: //
 2: //  MyPoint.m
 3: //  010-NSView
 4: //
 5: #import "MyPoint.h"
 6:
 7: @implementation MyPoint
 8:
 9: - (id) initWithNSPoint:(NSPoint)pNSPoint;
10: {
11:         if ((self = [super init]) == nil) {
12:                 return self;
13:         } // end if
14:
15:         myNSPoint.x     = pNSPoint.x;
16:         myNSPoint.y     = pNSPoint.y;
17:
18:     return self;
19:
20: } // end initWithNSPoint
21:
22: - (NSPoint) myNSPoint;
23: {
24:         return myNSPoint;
25: } // end myNSPoint
26:
27: - (float)x;
28: {
29:         return myNSPoint.x;
30: } // end x
31:
32: - (float)y;
33: {
34:         return myNSPoint.y;
35: } // end y
36:
37:
38:
39:
40:
41: @end
```

```
 1: //
 2: //  PaintCanvasView.h
 3: //  LeapPaint
 4: //
 5: //  Created by cj on 3/17/13.
 6: //  Copyright (c) 2013 cjdesch. All rights reserved.
 7: //
 8:
 9: #import <Cocoa/Cocoa.h>
10: #import "MyPoint.h"
11: #import "DrawingTool.h"
12: #import "ToolSettings.h"
13: #import "PenTool.h"
14: #import "LPDrawingTool.h"
15: #import "LeapObjectiveC.h"
16: #import "TrackedFinger.h"
17: #import "SimplePoint.h"
18:
19: @interface PaintCanvasView : NSView <LeapListener>{
20:
21:     LeapController *controller;
22:
23:     NSMutableArray  * myDrawingObjects;
24:     NSMutableArray      * myMutaryOfBrushStrokes;
25:         NSMutableArray  * myMutaryOfPoints;
26:     LPDrawingTool* drawingTool;
27:
28:
29:
30:     NSString* selectedTool;
31:
32:     NSColor* currentColor;
33:     float currentLineWidth;
34:     NSString* brushType;
35:     NSMutableDictionary* trackableList;
36:
37:
38:
39: }
40:
41: @property (nonatomic, strong)     NSMutableArray* tools;
42:
43: - (void)clearScreen;
44: - (void)setColor:(NSColor*)color;
45: - (void)setLineWidth:(float)w;
46: - (void)setTool:(NSString*)tool;
47:
48: @end
```

```
//
//  PaintCanvasView.m
//  LeapPaint
//
//  Created by cj on 3/17/13.
//  Copyright (c) 2013 cjdesch. All rights reserved.
//

#import "PaintCanvasView.h"


@interface PaintCanvasView()

@property (nonatomic, strong) ToolSettings *toolSettings;
@property (nonatomic, strong) NSMutableArray *drawingTools;

#pragma mark - Undo stack

@property (assign) int undoStackLocation;
@property (assign) int undoStackCount;

@end

@implementation PaintCanvasView

@synthesize drawingTools;
@synthesize toolSettings;

@synthesize tools;

- (id)initWithFrame:(NSRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        // Initialization code here.

        NSLog(@"Init");

        myMutaryOfBrushStrokes  = [[NSMutableArray alloc] init];
        myDrawingObjects = [[NSMutableArray alloc] init];

        drawingTools = [[NSMutableArray alloc] init];



        self.toolSettings = [[ToolSettings alloc] init];
         [self.toolSettings loadFromUserDefaults];
        [self initializeTools];

        //NSColorPanel* colorPanel = [NSColorPanel sharedColorPanel];

        currentColor = [NSColor blueColor];
        currentLineWidth = 1.0;

        trackableList = [[NSMutableDictionary alloc] init];

        [self run];

        self.tools = [[NSMutableArray alloc] initWithObjects:@"Pen",@"Brush", nil];
        selectedTool = @"Pen";


        // initialise random numebr generator used in drawRect for creating colours etc.
        srand(time(NULL));
    }

    return self;
}

#pragma mark - SampleDelegate Callbacks
- (void)run
{
    controller = [[LeapController alloc] init];
    [controller addListener:self];
    NSLog(@"running");
}

- (void)onInit:(NSNotification *)notification{
    NSLog(@"Leap: Initialized");
}
```

```
 81:
 82: - (void)onConnect:(NSNotification *)notification;
 83: {
 84:     NSLog(@"Leap: Connected");
 85:     LeapController *aController = (LeapController *)[notification object];
 86:     //[aController enableGesture:LEAP_GESTURE_TYPE_CIRCLE enable:YES];
 87:     //[aController enableGesture:LEAP_GESTURE_TYPE_KEY_TAP enable:YES];
 88:     //[aController enableGesture:LEAP_GESTURE_TYPE_SCREEN_TAP enable:YES];
 89:     //[aController enableGesture:LEAP_GESTURE_TYPE_SWIPE enable:YES];
 90: }
 91:
 92: - (void)onDisconnect:(NSNotification *)notification{
 93:     NSLog(@"Leap: Disconnected");
 94: }
 95:
 96: - (void)onExit:(NSNotification *)notification{
 97:     NSLog(@"Leap: Exited");
 98: }
 99:
100:
101: - (void)onFrame:(NSNotification *)notification{
102:
103:     //NSLog(@"OnFrame");
104:     LeapController *aController = (LeapController *)[notification object];
105:     // Get the most recent frame and report some basic information
106:     LeapFrame *frame = [aController frame:0];
107:
108:
109:     if ([[frame hands] count] != 0) {
110:         // Get the first hand
111:         LeapHand *hand = [[frame hands] objectAtIndex:0];
112:
113:         // Check if the hand has any fingers
114:         NSArray *fingers = [hand fingers];
115:
116:         if ([fingers count] != 0) {
117:
118:             // Calculate the hand's average finger tip position
119:             LeapVector *avgPos = [[LeapVector alloc] init];
120:             for (int i = 0; i < [fingers count]; i++) {
121:                 LeapFinger *finger = [fingers objectAtIndex:i];
122:                 avgPos = [avgPos plus:[finger tipPosition]];
123:
124:                 if (avgPos.z < 0){
125:                     NSString* fingerID = [NSString stringWithFormat:@"%d", finger.id];
126:
127:                     //Check if the Finger ID exists in the list already
128:                     if ([trackableList objectForKey:fingerID]) {
129:
130:                         //If it does exist update the position on the screen
131:                         TrackedFinger* sprite = [trackableList objectForKey:fingerID];
132:                         sprite.position = [self covertLeapCoordinates:CGPointMake(finger.tipPosition.x
, finger.tipPosition.y)];
133:                         sprite.updated = TRUE;
134:
135:                         [self updateFingerDraw:sprite];
136:
137:                         //SimplePoint* simplePoint = [[SimplePoint alloc] initWithPosition:sprite.posi
tion];
138:
139:
140:                     }else{
141:
142:                         //NSLog(@"x %0.0f y %0.0f z %0.0f", finger.tipPosition.x, finger.tipPosition.y
, finger.tipPosition.z);
143:                         // CGPoint point = [[CCDirector sharedDirector] convertEventToGL:event];
144:                         //CGPoint mouseLocation = [self convertToNodeSpace:point];
145:
146:                         //Add it to the dictionary
147:                         //TrackedFinger* redDot = [self addRedDot:[self covertLeapCoordinates:CGPointM
ake(finger.tipPosition.x, finger.tipPosition.y)] finger:fingerID];
148:                         TrackedFinger* redDot = [[TrackedFinger alloc] initWithID:fingerID withPositio
n:[self covertLeapCoordinates:CGPointMake(finger.tipPosition.x, finger.tipPosition.y)]];
149:                         [trackableList setObject:redDot forKey:fingerID];
150:
151:                         [self beginFingerDraw:redDot];
152:
153:                     }
154:                 }
155:
```

```
  156:             }
  157:
  158:             avgPos = [avgPos divide:[fingers count]];
  159:
  160:             //NSLog(@"Hand has %ld fingers, average finger tip position %@", [fingers count], avgPos);
  161:             for (LeapFinger* finger in fingers){
  162:
  163:                 //NSLog(@"Finger ID %d %ld", finger.id, (unsigned long)[finger hash]);
  164:             }
  165:
  166:         }
  167:
  168:         //
  169:         [self checkFingerExists];
  170:     }
  171:
  172: }
  173:
  174:
  175: /*
  176: - (void)drawRect:(NSRect)dirtyRect
  177: {
  178:     NSLog(@"drawRect");
  179:     // Drawing code here.
  180:     // colour the background white
  181:
  182:         [[NSColor whiteColor] set];             // this is Cocoa
  183:         NSRectFill( dirtyRect );
  184:
  185:         if ([myMutaryOfBrushStrokes count] == 0) {
  186:                 return;
  187:         } // end if
  188:
  189:         // This is Quartz
  190:         NSGraphicsContext       *       tvarNSGraphicsContext   = [NSGraphicsContext currentContext];
  191:         CGContextRef                    tvarCGContextRef                = (CGContextRef) [tvarNSGraphi
csContext graphicsPort];
  192:
  193:         NSUInteger tvarIntNumberOfStrokes       = [myMutaryOfBrushStrokes count];
  194:
  195:         NSUInteger i;
  196:         for (i = 0; i < tvarIntNumberOfStrokes; i++) {
  197:
  198:                 CGContextSetRGBStrokeColor(tvarCGContextRef,[self randVar],[self randVar],[self randVa
r],[self randVar]);
  199:                 CGContextSetLineWidth(tvarCGContextRef, (1.0 + ([self randVar] * 10.0)) );
  200:
  201:                 myMutaryOfPoints        = [myMutaryOfBrushStrokes objectAtIndex:i];
  202:
  203:                 NSUInteger tvarIntNumberOfPoints        = [myMutaryOfPoints count];
       // always >= 2
  204:                 MyPoint * tvarLastPointObj                      = [myMutaryOfPoints objectAtIndex:0];
  205:                 CGContextBeginPath(tvarCGContextRef);
  206:                 CGContextMoveToPoint(tvarCGContextRef,[tvarLastPointObj x],[tvarLastPointObj y]);
  207:
  208:                 NSUInteger j;
  209:                 for (j = 1; j < tvarIntNumberOfPoints; j++) {  // note the index starts at 1
  210:                         MyPoint * tvarCurPointObj                       = [myMutaryOfPoints objectAtIn
dex:j];
  211:                         CGContextAddLineToPoint(tvarCGContextRef,[tvarCurPointObj x],[tvarCurPointObj
y]);
  212:                 } // end for
  213:
  214:                 CGContextDrawPath(tvarCGContextRef,kCGPathStroke);
  215:
  216:         } // end for
  217:
  218:
  219: }
  220:
  221:  */
  222:
  223: - (void)drawRect:(NSRect)dirtyRect
  224: {
  225:     //NSLog(@"drawRect");
  226:     // Drawing code here.
  227:     // colour the background white
  228:
  229:         [[NSColor whiteColor] set];             // this is Cocoa
  230:         NSRectFill( dirtyRect );
```

```
  231:
  232:         if ([myMutaryOfBrushStrokes count] == 0) {
  233:                 return;
  234:         } // end if
  235:
  236:         // This is Quartz
  237:         NSGraphicsContext       *       tvarNSGraphicsContext   = [NSGraphicsContext currentContext];
  238:         CGContextRef                    tvarCGContextRef                = (CGContextRef) [tvarNSGraphi
csContext graphicsPort];
  239:
  240:         NSUInteger tvarIntNumberOfStrokes       = [myMutaryOfBrushStrokes count];
  241:
  242:         NSUInteger i;
  243:         for (i = 0; i < tvarIntNumberOfStrokes; i++) {
  244:
  245:
  246:                 drawingTool     = [myMutaryOfBrushStrokes objectAtIndex:i];
  247:         CGFloat red, green, blue, a;
  248:         [drawingTool.primaryColor getRed:&red green:&green blue:&blue alpha:&a];
  249:
  250:                 CGContextSetRGBStrokeColor(tvarCGContextRef, red,green,blue,a);
  251:                 CGContextSetLineWidth(tvarCGContextRef, drawingTool.lineWidth );
  252:
  253:
  254:                 NSUInteger tvarIntNumberOfPoints        = [drawingTool.points count];//[myMutaryOfPoin
ts count];                              // always >= 2
  255:                 MyPoint * tvarLastPointObj                      = [drawingTool.points objectAtIndex:0]
;
  256:                 CGContextBeginPath(tvarCGContextRef);
  257:                 CGContextMoveToPoint(tvarCGContextRef,[tvarLastPointObj x],[tvarLastPointObj y]);
  258:
  259:                 NSUInteger j;
  260:                 for (j = 1; j < tvarIntNumberOfPoints; j++) {  // note the index starts at 1
  261:                         MyPoint * tvarCurPointObj                       = [drawingTool.points objectAt
Index:j];
  262:                         CGContextAddLineToPoint(tvarCGContextRef,[tvarCurPointObj x],[tvarCurPointObj
y]);
  263:                 } // end for
  264:
  265:                 CGContextDrawPath(tvarCGContextRef,kCGPathStroke);
  266:
  267:         } // end for
  268:
  269:
  270: }
  271:
  272: /*
  273: -(void)mouseDown:(NSEvent *)pTheEvent {
  274:
  275:     //Selected Tool
  276:     //NSLog(@"Mouse Down");
  277:
  278:     DrawingTool *drawingTool = [self activeTool];
  279:     if (drawingTool) {
  280:
  281:         NSPoint tvarMousePointInWindow  = [pTheEvent locationInWindow];
  282:         NSPoint tvarMousePointInView    = [self convertPoint:tvarMousePointInWindow fromView:self];
  283:         [drawingTool inputBegan:pTheEvent locationInView:tvarMousePointInView withSettings:self.toolSe
ttings];
  284:     }
  285:
  286:
  287:
  288:
  289: } // end mouseDown
  290:
  291: -(void)mouseDragged:(NSEvent *)pTheEvent {
  292:
  293:     //NSLog(@"Mouse Dragg");
  294:
  295:     DrawingTool *drawingTool = [self activeTool];
  296:     if (drawingTool) {
  297:
  298:         NSPoint tvarMousePointInWindow  = [pTheEvent locationInWindow];
  299:         NSPoint tvarMousePointInView    = [self convertPoint:tvarMousePointInWindow fromView:self];
  300:         [drawingTool inputMoved:pTheEvent locationInView:tvarMousePointInView];
  301:
  302:     }
  303:
  304:
```

```
305: } // end mouseDragged
306:
307: -(void)mouseUp:(NSEvent *)pTheEvent {
308:
309:     //NSLog(@"Mouse up");
310:
311:     DrawingTool *drawingTool = [self activeTool];
312:     if (drawingTool) {
313:
314:
315:         NSPoint tvarMousePointInWindow  = [pTheEvent locationInWindow];
316:         NSPoint tvarMousePointInView    = [self convertPoint:tvarMousePointInWindow fromView:self];
317:         [drawingTool inputEnded:pTheEvent locationInView:tvarMousePointInView];
318:
319:     }
320:
321:
322: } // end mouseUp
323:
324:  */
325:
326:
327:
328: /*
329: -(void)mouseDown:(NSEvent *)pTheEvent {
330:
331:         myMutaryOfPoints        = [[NSMutableArray alloc]init];
332:         [myMutaryOfBrushStrokes addObject:myMutaryOfPoints];
333:
334:         NSPoint tvarMousePointInWindow  = [pTheEvent locationInWindow];
335:         NSPoint tvarMousePointInView    = [self convertPoint:tvarMousePointInWindow fromView:nil];
336:         MyPoint * tvarMyPointObj                = [[MyPoint alloc]initWithNSPoint:tvarMousePointInView
];
337:
338:         [myMutaryOfPoints addObject:tvarMyPointObj];
339:
340: } // end mouseDown
341:
342: -(void)mouseDragged:(NSEvent *)pTheEvent {
343:
344:         NSPoint tvarMousePointInWindow  = [pTheEvent locationInWindow];
345:         NSPoint tvarMousePointInView    = [self convertPoint:tvarMousePointInWindow fromView:nil];
346:         MyPoint * tvarMyPointObj                = [[MyPoint alloc]initWithNSPoint:tvarMousePointInView
];
347:
348:         [myMutaryOfPoints addObject:tvarMyPointObj];
349:
350:         [self setNeedsDisplay:YES];
351:
352: } // end mouseDragged
353:
354: -(void)mouseUp:(NSEvent *)pTheEvent {
355:
356:         NSPoint tvarMousePointInWindow  = [pTheEvent locationInWindow];
357:         NSPoint tvarMousePointInView    = [self convertPoint:tvarMousePointInWindow fromView:nil];
358:         MyPoint * tvarMyPointObj                = [[MyPoint alloc]initWithNSPoint:tvarMousePointInView
];
359:
360:         [myMutaryOfPoints addObject:tvarMyPointObj];
361:
362:         [self setNeedsDisplay:YES];
363:
364: } // end mouseUp
365:
366: */
367:
368:
369: - (void)beginFingerDraw:(id)sender{
370:
371:     TrackedFinger* trackedFinger = (TrackedFinger*)sender;
372:     [self beginDraw:trackedFinger.position withFinger:trackedFinger];
373:
374: }
375:
376: - (void)updateFingerDraw:(id)sender{
377:     TrackedFinger* trackedFinger = (TrackedFinger*)sender;
378:     [self updateDraw:trackedFinger.position withFinger:trackedFinger];
379:
380: }
381:
```

```
382: - (void)endFingerDraw:(id)sender{
383:     TrackedFinger* trackedFinger = (TrackedFinger*)sender;
384:     [self endDraw:trackedFinger.position withFinger:trackedFinger];
385: }
386:
387:
388: //The further negative, the thicker the line.
389: - (void)beginDraw:(CGPoint)point withFinger:(TrackedFinger*)finger{
390:
391:     //if there is no finger, we are drawing with the mouse
392:     if(finger == nil){
393:         drawingTool     = [[LPDrawingTool alloc] initWithWidth:currentLineWidth withTransparency:1 wit
hPrimaryColor:currentColor withSecondaryColor:[NSColor redColor]];
394:
395:         [myMutaryOfBrushStrokes addObject:drawingTool];
396:         MyPoint * tvarMyPointObj                = [[MyPoint alloc]initWithNSPoint:point];
397:         [drawingTool.points addObject:tvarMyPointObj];
398:     }else{
399:         drawingTool     = [[LPDrawingTool alloc] initWithWidth:currentLineWidth withTransparency:1 wit
hPrimaryColor:currentColor withSecondaryColor:[NSColor redColor]];
400:         finger.tool = drawingTool;
401:         [myMutaryOfBrushStrokes addObject:drawingTool];
402:         MyPoint * tvarMyPointObj                = [[MyPoint alloc]initWithNSPoint:point];
403:         [drawingTool.points addObject:tvarMyPointObj];
404:     }
405:
406: }
407:
408: - (void)updateDraw:(CGPoint)point withFinger:(TrackedFinger*)finger{
409:
410:     if(finger == nil){
411:         MyPoint * tvarMyPointObj                = [[MyPoint alloc]initWithNSPoint:point];
412:         [drawingTool.points addObject:tvarMyPointObj];
413:         [self setNeedsDisplay:YES];
414:     }else{
415:         MyPoint * tvarMyPointObj                = [[MyPoint alloc]initWithNSPoint:point];
416:         drawingTool     = finger.tool;
417:         [drawingTool.points addObject:tvarMyPointObj];
418:         [self setNeedsDisplay:YES];
419:     }
420: }
421:
422: - (void)endDraw:(CGPoint)point withFinger:(TrackedFinger*)finger {
423:
424:    if(finger == nil){
425:        MyPoint * tvarMyPointObj         = [[MyPoint alloc]initWithNSPoint:point];
426:        [drawingTool.points addObject:tvarMyPointObj];
427:        [self setNeedsDisplay:YES];
428:
429:    }else{
430:        MyPoint * tvarMyPointObj         = [[MyPoint alloc]initWithNSPoint:point];
431:        drawingTool      = finger.tool;
432:        [drawingTool.points addObject:tvarMyPointObj];
433:        [self setNeedsDisplay:YES];
434:    }
435:
436: }
437:
438:
439: -(void)mouseDown:(NSEvent *)pTheEvent {
440:     NSPoint tvarMousePointInWindow      = [pTheEvent locationInWindow];
441:         NSPoint tvarMousePointInView    = [self convertPoint:tvarMousePointInWindow fromView:nil];
442:
443:     [self beginDraw:tvarMousePointInView withFinger:nil];
444:
445: } // end mouseDown
446:
447: -(void)mouseDragged:(NSEvent *)pTheEvent {
448:
449:         NSPoint tvarMousePointInWindow  = [pTheEvent locationInWindow];
450:         NSPoint tvarMousePointInView    = [self convertPoint:tvarMousePointInWindow fromView:nil];
451:
452:     [self updateDraw:tvarMousePointInView withFinger:nil];
453:
454:
455: } // end mouseDragged
456:
457: -(void)mouseUp:(NSEvent *)pTheEvent {
458:
459:         NSPoint tvarMousePointInWindow  = [pTheEvent locationInWindow];
```

```
460:         NSPoint tvarMousePointInView    = [self convertPoint:tvarMousePointInWindow fromView:nil];
461:
462:     [self endDraw:tvarMousePointInView withFinger:nil];
463:
464:
465: } // end mouseUp
466:
467:
468: - (float)randVar;
469: {
470:         return ( (float)(rand() % 10000 ) / 10000.0);
471: } // end randVar
472:
473:
474: - (DrawingTool*)activeTool {
475:
476:     for (DrawingTool *tool in self.drawingTools) {
477:         if ([tool.toolName isEqualToString:self.toolSettings.drawingTool]) {
478:             return tool;
479:         }
480:     }
481:     return nil;
482: }
483:
484:
485: - (LPDrawingTool*)getActiveTool{
486:
487:     if (selectedTool )
488:
489:
490:     return nil;
491: }
492:
493: - (void)initializeTools {
494:
495:     self.drawingTools = [[NSMutableArray alloc] init];
496:
497:     //pen tool
498:     DrawingTool *tool = [[PenTool alloc] initWithCompletion:^{
499:
500:         //[self addDrawingToUndoStack];
501:         NSLog(@"Tool Completion");
502:
503:     }];
504:     tool.toolName = @"Pen";
505:     tool.imageName = @"pen-ink-mini.png";
506:     [self.drawingTools addObject:tool];
507:
508:
509:
510:
511: }
512:
513:
514: - (void)clearScreen{
515:
516:     [myMutaryOfBrushStrokes removeAllObjects];
517:     [self setNeedsDisplay:YES];
518: }
519:
520: - (void)setColor:(NSColor*)color{
521:
522:     currentColor = color;
523: }
524:
525: - (void)setLineWidth:(float)w{
526:
527:     currentLineWidth = w;
528:
529: }
530:
531: - (void)setTool:(NSString*)tool{
532:     selectedTool =  tool;
533: }
534:
535: - (CGPoint)covertLeapCoordinates:(CGPoint)p{
536:
537:     CGRect rect = NSRectToCGRect([self bounds]);
538:     CGSize s = rect.size;
539:     float screenCenter = 0.0f;
```

```
540:     float xScale = 1.75f;
541:     float yScale = 1.25f;
542:     return CGPointMake((s.width/2)+ (( p.x - screenCenter) * xScale), p.y * yScale);
543: }
544:
545: - (void)checkFingerExists{
546:
547:     for (id key in [trackableList allKeys]) {
548:         TrackedFinger* sprite = [trackableList objectForKey:key];
549:         if (sprite.updated) {
550:             sprite.updated = FALSE;
551:             //return;
552:         }else{
553:             [self endFingerDraw:sprite];
554:             [trackableList removeObjectForKey:key];
555:
556:         }
557:     }
558: }
559:
560: @end
```

```
 1: //
 2: //  PaintView.h
 3: //  LeapPaint
 4: //
 5: //  Created by cj on 3/17/13.
 6: //  Copyright (c) 2013 cjdesch. All rights reserved.
 7: //
 8:
 9: #import <Cocoa/Cocoa.h>
10: #import <OpenGL/OpenGL.h>
11: #include <OpenGL/gl.h>
12: #import "Brush.h"
13: #import "Canvas.h"
14: //CONSTANTS:
15:
16: #define kBrushOpacity           (1.0 / 3.0)
17: #define kBrushPixelStep         3
18: #define kBrushScale                     2
19: #define kLuminosity                     0.75
20: #define kSaturation                     1.0
21:
22:
23: @interface PaintView : NSOpenGLView {
24:     NSTrackingArea* trackingArea;
25: @private
26:
27:     // The pixel dimensions of the backbuffer
28:         GLint backingWidth;
29:         GLint backingHeight;
30:         CGImageRef              brushImage;
31:
32:     // OpenGL names for the renderbuffer and framebuffers used to render to this view
33:         GLuint viewRenderbuffer, viewFramebuffer;
34:
35:         GLuint  brushTexture;
36:         CGPoint location;
37:         CGPoint previousLocation;
38:         Boolean firstTouch;
39:         Boolean needsErase;
40:
41:     Brush* brush;
42:     Canvas* canvas;
43:
44: }
45:
46: @property(nonatomic, readwrite) CGPoint location;
47: @property(nonatomic, readwrite) CGPoint previousLocation;
48:
49: - (void)erase;
50: - (void)setBrushColorWithRed:(CGFloat)red green:(CGFloat)green blue:(CGFloat)blue;
51:
52:
53:
54: @end
```

```
    1: //
    2: //  PaintView.m
    3: //  LeapPaint
    4: //
    5: //  Created by cj on 3/17/13.
    6: //  Copyright (c) 2013 cjdesch. All rights reserved.
    7: //
    8:
    9: #import "PaintView.h"
   10:
   11: @implementation PaintView
   12:
   13: @synthesize  location;
   14: @synthesize  previousLocation;
   15:
   16:
   17: - (id)initWithFrame:(NSRect)frame
   18: {
   19:     NSMutableArray*     recordedPaths;
   20:         CGImageRef              brushImage;
   21:         CGContextRef    brushContext;
   22:         GLubyte                 *brushData;
   23:         size_t                  width, height;
   24:
   25:     self = [super initWithFrame:frame];
   26:     if (self) {
   27:         // Initialization code here.
   28:         canvas = [[Canvas alloc] initWithSize:frame.size];
   29:         brush = [[Brush alloc] init];
   30:
   31:         [[NSNotificationCenter defaultCenter] addObserver:self
   32:                                                  selector:@selector(_surfaceNeedsUpdate:)
   33:                                                      name:NSViewGlobalFrameDidChangeNotification
   34:                                                    object:self];
   35:
   36:         // Create a texture from an image
   37:                 // First create a UIImage object from the data in a image file, and then extract the C
ore Graphics image
   38:                 brushImage = [self nsImageToCGImageRef:[NSImage imageNamed:@"Particle.png"]];
   39:
   40:
   41:                 // Get the width and height of the image
   42:                 width = CGImageGetWidth(brushImage);
   43:                 height = CGImageGetHeight(brushImage);
   44:
   45:                 // Texture dimensions must be a power of 2. If you write an application that allows us
ers to supply an image,
   46:                 // you'll want to add code that checks the dimensions and takes appropriate action if
they are not a power of 2.
   47:
   48:                 // Make sure the image exists
   49:                 if(brushImage) {
   50:                         // Allocate  memory needed for the bitmap context
   51:                         brushData = (GLubyte *) calloc(width * height * 4, sizeof(GLubyte));
   52:                         // Use  the bitmatp creation function provided by the Core Graphics framework.
   53:                         brushContext = CGBitmapContextCreate(brushData, width, height, 8, width * 4, C
GImageGetColorSpace(brushImage), kCGImageAlphaPremultipliedLast);
   54:                         // After you create the context, you can draw the  image to the context.
   55:                         CGContextDrawImage(brushContext, CGRectMake(0.0, 0.0, (CGFloat)width, (CGFloat
)height), brushImage);
   56:                         // You don't need the context at this point, so you need to release it to avoi
d memory leaks.
   57:                         CGContextRelease(brushContext);
   58:                         // Use OpenGL ES to generate a name for the texture.
   59:                         glGenTextures(1, &brushTexture);
   60:                         // Bind the texture name.
   61:                         glBindTexture(GL_TEXTURE_2D, brushTexture);
   62:                         // Set the texture parameters to use a minifying filter and a linear filer (we
ighted average)
   63:                         glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
   64:                         // Specify a 2D texture image, providing the a pointer to the image data in me
mory
   65:                         glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED
_BYTE, brushData);
   66:                         // Release  the image data; it's no longer needed
   67:             free(brushData);
   68:
   69:
   70:         }
   71:
```

```
   72:         // Set the view's scale factor
   73:
   74:                 //self.contentScaleFactor = 1.0;
   75:         GLfloat scale = 1.0;
   76:
   77:                 // Setup OpenGL states
   78:                 glMatrixMode(GL_PROJECTION);
   79:                 CGRect frame = self.bounds;
   80:                 //CGFloat scale = self.contentScaleFactor;
   81:                 // Setup the view port in Pixels
   82:
   83:                 //glOrthof(0, frame.size.width * scale, 0, frame.size.height * scale, -1, 1);
   84:                 glViewport(0, 0, frame.size.width * scale, frame.size.height * scale);
   85:                 glMatrixMode(GL_MODELVIEW);
   86:
   87:                 glDisable(GL_DITHER);
   88:                 glEnable(GL_TEXTURE_2D);
   89:                 glEnableClientState(GL_VERTEX_ARRAY);
   90:
   91:             glEnable(GL_BLEND);
   92:                 // Set a blending function appropriate for premultiplied alpha pixel data
   93:                 glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA);
   94:
   95:                 //glEnable(GL_POINT_SPRITE_OES);
   96:                 //glTexEnvf(GL_POINT_SPRITE_OES, GL_COORD_REPLACE_OES, GL_TRUE);
   97:         glEnable(GL_POINT_SPRITE);
   98:         glTexEnvf(GL_POINT_SPRITE, GL_COORD_REPLACE, GL_TRUE);
   99:                 glPointSize(width / kBrushScale);
  100:
  101:                 // Make sure to start with a cleared buffer
  102:                 needsErase = YES;
  103:
  104:                 // Playback recorded path, which is "Shake Me"
  105:                 recordedPaths = [NSMutableArray arrayWithContentsOfFile:[[NSBundle mainBundle] pathFor
Resource:@"Recording" ofType:@"data"]];
  106:                 if([recordedPaths count])
  107:                         [self performSelector:@selector(playback:) withObject:recordedPaths afterDelay
:0.2];
  108:
  109:
  110:         trackingArea = [[NSTrackingArea alloc] initWithRect:self.bounds
  111:                                                     options: (NSTrackingMouseEnteredAndExited | NSTrac
kingMouseMoved | NSTrackingActiveInKeyWindow )
  112:                                                       owner:self userInfo:nil];
  113:         [self addTrackingArea:trackingArea];
  114:
  115:     }
  116:
  117:     return self;
  118: }
  119:
  120:
  121: - (void)updateTrackingAreas {
  122:     NSRect eyeBox;
  123:     [self removeTrackingArea:trackingArea];
  124:
  125:     trackingArea = [[NSTrackingArea alloc] initWithRect:self.bounds
  126:                                                 options: (NSTrackingMouseEnteredAndExited | NSTracking
MouseMoved | NSTrackingActiveInKeyWindow)
  127:                                                   owner:self userInfo:nil];
  128:     [self addTrackingArea:trackingArea];
  129:
  130: }
  131:
  132: -(void) drawRect: (NSRect) dirtyRect
  133: {
  134:
  135:
  136:
  137:
  138:     NSLog(@"Draw dirty rect");
  139:     glClearColor(0, 0, 0, 0);
  140:     glClear(GL_COLOR_BUFFER_BIT);
  141:     drawAnObject();
  142:     glFlush();
  143:
  144:
  145:
  146:     NSGraphicsContext* context = [NSGraphicsContext currentContext];
  147:
```

```
  148:         [canvas drawRect:dirtyRect inContext:context];
  149:
  150: }
  151:
  152:
  153: static void drawAnObject ()
  154: {
  155:     glColor3f(1.0f, 0.85f, 0.35f);
  156:     glBegin(GL_TRIANGLES);
  157:     {
  158:         glVertex3f(  0.0,  0.6, 0.0);
  159:         glVertex3f( -0.2, -0.3, 0.0);
  160:         glVertex3f(  0.2, -0.3 ,0.0);
  161:     }
  162:     glEnd();
  163: }
  164:
  165:
  166: - (void) _surfaceNeedsUpdate:(NSNotification*)notification
  167: {
  168:     [self update];
  169: }
  170: /*
  171: - (BOOL)createFramebuffer
  172: {
  173:         // Generate IDs for a framebuffer object and a color renderbuffer
  174:         glGenFramebuffersOES(1, &viewFramebuffer);
  175:         glGenRenderbuffersOES(1, &viewRenderbuffer);
  176:
  177:
  178:         //glBindFramebufferOES(GL_FRAMEBUFFER_OES, viewFramebuffer);
  179:     glBindFramebuffer(GL_FRAMEBUFFER, viewFramebuffer);
  180:
  181: //      glBindRenderbufferOES(GL_RENDERBUFFER_OES, viewRenderbuffer);
  182:     glBindRenderbuffer(GL_RENDERBUFFER, viewRenderbuffer);
  183:         // This call associates the storage for the current render buffer with the EAGLDrawable (our C
AEAGLLayer)
  184:         // allowing us to draw into a buffer that will later be rendered to screen wherever the layer
is (which corresponds with our view).
  185:
  186:
  187:
  188:         //[context renderbufferStorage:GL_RENDERBUFFER_OES fromDrawable:(id<EAGLDrawable>)self.layer];
  189:     self
  190:     [self.openGLContext renderbufferStorage:GL_RENDERBUFFER fromDrawable:(id<EAGLDrawable>)self.layer]
;
  191:
  192:         glFramebufferRenderbufferOES(GL_FRAMEBUFFER_OES, GL_COLOR_ATTACHMENT0_OES, GL_RENDERBUFFER_OES
, viewRenderbuffer);
  193:
  194:         glGetRenderbufferParameterivOES(GL_RENDERBUFFER_OES, GL_RENDERBUFFER_WIDTH_OES, &backingWidth)
;
  195:         glGetRenderbufferParameterivOES(GL_RENDERBUFFER_OES, GL_RENDERBUFFER_HEIGHT_OES, &backingHeigh
t);
  196:
  197:         // For this sample, we also need a depth buffer, so we'll create and attach one via another re
nderbuffer.
  198:         glGenRenderbuffersOES(1, &depthRenderbuffer);
  199:         glBindRenderbufferOES(GL_RENDERBUFFER_OES, depthRenderbuffer);
  200:         glRenderbufferStorageOES(GL_RENDERBUFFER_OES, GL_DEPTH_COMPONENT16_OES, backingWidth, backingH
eight);
  201:         glFramebufferRenderbufferOES(GL_FRAMEBUFFER_OES, GL_DEPTH_ATTACHMENT_OES, GL_RENDERBUFFER_OES,
 depthRenderbuffer);
  202:
  203:         if(glCheckFramebufferStatusOES(GL_FRAMEBUFFER_OES) != GL_FRAMEBUFFER_COMPLETE_OES)
  204:         {
  205:                 NSLog(@"failed to make complete framebuffer object %x", glCheckFramebufferStatusOES(GL
_FRAMEBUFFER_OES));
  206:                 return NO;
  207:         }
  208:
  209:         return YES;
  210: }
  211:
  212:
  213: // Clean up any buffers we have allocated.
  214: - (void)destroyFramebuffer
  215: {
  216:         glDeleteFramebuffersOES(1, &viewFramebuffer);
  217:         viewFramebuffer = 0;
```

```
  218:         glDeleteRenderbuffersOES(1, &viewRenderbuffer);
  219:         viewRenderbuffer = 0;
  220:
  221:         if(depthRenderbuffer)
  222:         {
  223:                 glDeleteRenderbuffersOES(1, &depthRenderbuffer);
  224:                 depthRenderbuffer = 0;
  225:         }
  226: }
  227:
  228: */
  229: // Erases the screen
  230: - (void) erase
  231: {
  232:     NSLog(@"Earse");
  233:
  234:         //[EAGLContext setCurrentContext:context];
  235:
  236:         // Clear the buffer
  237:         //glBindFramebufferOES(GL_FRAMEBUFFER_OES, viewFramebuffer);
  238:     glBindFramebuffer(GL_FRAMEBUFFER, viewFramebuffer);
  239:         glClearColor(0.0, 0.0, 0.0, 0.0);
  240:         glClear(GL_COLOR_BUFFER_BIT);
  241:
  242:         // Display the buffer
  243:         //glBindRenderbufferOES(GL_RENDERBUFFER_OES, viewRenderbuffer);
  244:     glBindRenderbuffer(GL_RENDERBUFFER, viewRenderbuffer);
  245:         //[context presentRenderbuffer:GL_RENDERBUFFER_OES];
  246:     //
  247: }
  248:
  249: // Drawings a line onscreen based on where the user touches
  250: - (void) renderLineFromPoint:(CGPoint)start toPoint:(CGPoint)end
  251: {
  252:         static GLfloat*         vertexBuffer = NULL;
  253:         static NSUInteger       vertexMax = 64;
  254:         NSUInteger                      vertexCount = 0,
  255:     count,
  256:     i;
  257:
  258:         //[EAGLContext setCurrentContext:context];
  259:         //glBindFramebufferOES(GL_FRAMEBUFFER_OES, viewFramebuffer);
  260:     glBindFramebuffer(GL_FRAMEBUFFER, viewFramebuffer);
  261:
  262:
  263:         // Convert locations from Points to Pixels
  264:         //CGFloat scale = self.contentScaleFactor;
  265:     CGFloat scale = 1.0;
  266:         start.x *= scale;
  267:         start.y *= scale;
  268:         end.x *= scale;
  269:         end.y *= scale;
  270:
  271:         // Allocate vertex array buffer
  272:         if(vertexBuffer == NULL)
  273:                 vertexBuffer = malloc(vertexMax * 2 * sizeof(GLfloat));
  274:
  275:         // Add points to the buffer so there are drawing points every X pixels
  276:         count = MAX(ceilf(sqrtf((end.x - start.x) * (end.x - start.x) + (end.y - start.y) * (end.y - s
tart.y)) / kBrushPixelStep), 1);
  277:         for(i = 0; i < count; ++i) {
  278:                 if(vertexCount == vertexMax) {
  279:                         vertexMax = 2 * vertexMax;
  280:                         vertexBuffer = realloc(vertexBuffer, vertexMax * 2 * sizeof(GLfloat));
  281:                 }
  282:
  283:                 vertexBuffer[2 * vertexCount + 0] = start.x + (end.x - start.x) * ((GLfloat)i / (GLflo
at)count);
  284:                 vertexBuffer[2 * vertexCount + 1] = start.y + (end.y - start.y) * ((GLfloat)i / (GLflo
at)count);
  285:                 vertexCount += 1;
  286:         }
  287:
  288:         // Render the vertex array
  289:         glVertexPointer(2, GL_FLOAT, 0, vertexBuffer);
  290:         glDrawArrays(GL_POINTS, 0, vertexCount);
  291:
  292:         // Display the buffer
  293:         //glBindRenderbufferOES(GL_RENDERBUFFER_OES, viewRenderbuffer);
  294:     glBindRenderbuffer(GL_RENDERBUFFER, viewRenderbuffer);
```

```
295:         //[context presentRenderbuffer:GL_RENDERBUFFER_OES];
296:     [self setNeedsDisplay:YES];
297: }
298:
299: // Reads previously recorded points and draws them onscreen. This is the Shake Me message that appears
when the application launches.
300: - (void) playback:(NSMutableArray*)recordedPaths
301: {
302:         NSData*                         data = [recordedPaths objectAtIndex:0];
303:         CGPoint*                        point = (CGPoint*)[data bytes];
304:         NSUInteger                      count = [data length] / sizeof(CGPoint),
305:     i;
306:
307:         // Render the current path
308:         for(i = 0; i < count - 1; ++i, ++point)
309:                 [self renderLineFromPoint:*point toPoint:*(point + 1)];
310:
311:         // Render the next path after a short delay
312:         [recordedPaths removeObjectAtIndex:0];
313:         if([recordedPaths count])
314:                 [self performSelector:@selector(playback:) withObject:recordedPaths afterDelay:0.01];
315: }
316:
317: /*
318: // Handles the start of a touch
319: - (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
320: {
321:         CGRect                          bounds = [self bounds];
322:     UITouch*    touch = [[event touchesForView:self] anyObject];
323:         firstTouch = YES;
324:         // Convert touch point from UIView referential to OpenGL one (upside-down flip)
325:         location = [touch locationInView:self];
326:         location.y = bounds.size.height - location.y;
327: }
328: */
329:
330: // Handles the start of a touch
331: - (void)mouseDown:(NSEvent *)theEvent
332: {
333:         NSLog(@"Mouse up");
334:         CGRect                          bounds = [self bounds];
335:
336:     //UITouch*  touch = [[event touchesForView:self] anyObject];
337:
338:         firstTouch = YES;
339:     NSPoint mouseLocation = [self convertPoint:[theEvent locationInWindow] fromView:nil];
340:         // Convert touch point from UIView referential to OpenGL one (upside-down flip)
341:         //location = [touch locationInView:self];
342:         location.y = bounds.size.height - mouseLocation.y;
343:
344:         [brush mouseDown:theEvent inView:self onCanvas:canvas];
345: }
346:
347:
348:
349: /*
350: // Handles the continuation of a touch.
351: - (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
352: {
353:
354:         CGRect                          bounds = [self bounds];
355:         UITouch*                        touch = [[event touchesForView:self] anyObject];
356:
357:         // Convert touch point from UIView referential to OpenGL one (upside-down flip)
358:         if (firstTouch) {
359:                 firstTouch = NO;
360:                 previousLocation = [touch previousLocationInView:self];
361:                 previousLocation.y = bounds.size.height - previousLocation.y;
362:         } else {
363:                 location = [touch locationInView:self];
364:             location.y = bounds.size.height - location.y;
365:                 previousLocation = [touch previousLocationInView:self];
366:                 previousLocation.y = bounds.size.height - previousLocation.y;
367:         }
368:
369:         // Render the stroke
370:         [self renderLineFromPoint:previousLocation toPoint:location];
371: }*/
372:
373: // Handles the continuation of a touch.
```

```
374: - (void)mouseDragged:(NSEvent *)theEvent{
375:         NSLog(@"Mouse Dragged");
376:     CGRect                          bounds = [self bounds];
377:         //UITouch*                  touch = [[event touchesForView:self] anyObject];
378:     NSPoint mouseLocation = [self convertPoint:[theEvent locationInWindow] fromView:nil];
379:
380:
381:         // Convert touch point from UIView referential to OpenGL one (upside-down flip)
382:         if (firstTouch) {
383:                 firstTouch = NO;
384:
385:                 //previousLocation = [touch previousLocationInView:self];
386:         previousLocation = location;
387:                 previousLocation.y = bounds.size.height - previousLocation.y;
388:         } else {
389:                 //location = [touch locationInView:self];
390:         previousLocation = location;
391:                 previousLocation.y = bounds.size.height - previousLocation.y;
392:         location = mouseLocation;
393:             location.y = bounds.size.height - location.y;
394:
395:         }
396:
397:         // Render the stroke
398:         //[self renderLineFromPoint:previousLocation toPoint:location];
399:     //[self drawSomething];
400:
401:         [brush mouseDragged:theEvent inView:self onCanvas:canvas];
402: }
403:
404: - (void)drawSomething{
405:
406:
407: }
408:
409: /*
410: // Handles the end of a touch event when the touch is a tap.
411: - (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
412: {
413:         CGRect                          bounds = [self bounds];
414:     UITouch*    touch = [[event touchesForView:self] anyObject];
415:         if (firstTouch) {
416:                 firstTouch = NO;
417:                 previousLocation = [touch previousLocationInView:self];
418:                 previousLocation.y = bounds.size.height - previousLocation.y;
419:                 [self renderLineFromPoint:previousLocation toPoint:location];
420:         }
421: }
422: */
423: - (void)mouseUp:(NSEvent *)theEvent{
424:
425:     NSLog(@"Mouse up");
426:     CGRect                          bounds = [self bounds];
427:     //UITouch*  touch = [[event touchesForView:self] anyObject];
428:
429:     NSPoint mouseLocation = [self convertPoint:[theEvent locationInWindow] fromView:nil];
430:
431:         if (firstTouch) {
432:                 firstTouch = NO;
433:                 //previousLocation = [touch previousLocationInView:self];
434:                 //previousLocation.y = bounds.size.height - previousLocation.y;
435:         location = mouseLocation;
436:                 //[self renderLineFromPoint:previousLocation toPoint:location];
437:         }
438:
439:         [brush mouseUp:theEvent inView:self onCanvas:canvas];}
440: /*
441: // Handles the end of a touch event.
442: - (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event
443: {
444:         // If appropriate, add code necessary to save the state of the application.
445:         // This application is not saving state.
446: }*/
447:
448: // Handles the end of a touch event.
449: - (void)mouseExited:(NSEvent *)theEvent
450: {
451:         // If appropriate, add code necessary to save the state of the application.
452:         // This application is not saving state.
453: }
```

```
454:
455:
456: - (void)setBrushColorWithRed:(CGFloat)red green:(CGFloat)green blue:(CGFloat)blue
457: {
458:         // Set the brush color using premultiplied alpha values
459:         glColor4f(red   * kBrushOpacity,
460:                           green * kBrushOpacity,
461:                           blue  * kBrushOpacity,
462:                           kBrushOpacity);
463: }
464:
465:
466: //Move to utilities
467: - (CGImageRef)nsImageToCGImageRef:(NSImage*)image;
468: {
469:     NSData * imageData = [image TIFFRepresentation];
470:     CGImageRef imageRef;
471:     if(!imageData) return nil;
472:     //Bridge
473:     CGImageSourceRef imageSource = CGImageSourceCreateWithData((__bridge CFDataRef)imageData, NULL);
474:     imageRef = CGImageSourceCreateImageAtIndex(imageSource, 0, NULL);
475:     return imageRef;
476: }
477:
478:
479: void ManipulateImagePixelData(CGImageRef inImage)
480: {
481:     // Create the bitmap context
482:     CGContextRef cgctx = CreateARGBBitmapContext(inImage);
483:     if (cgctx == NULL)
484:     {
485:         // error creating context
486:         return;
487:     }
488:
489:     // Get image width, height. We'll use the entire image.
490:     size_t w = CGImageGetWidth(inImage);
491:     size_t h = CGImageGetHeight(inImage);
492:     CGRect rect = {{0,0},{w,h}};
493:
494:     // Draw the image to the bitmap context. Once we draw, the memory
495:     // allocated for the context for rendering will then contain the
496:     // raw image data in the specified color space.
497:     CGContextDrawImage(cgctx, rect, inImage);
498:
499:     // Now we can get a pointer to the image data associated with the bitmap
500:     // context.
501:     void *data = CGBitmapContextGetData (cgctx);
502:     if (data != NULL)
503:     {
504:
505:         // **** You have a pointer to the image data ****
506:
507:         // **** Do stuff with the data here ****
508:
509:     }
510:
511:     // When finished, release the context
512:     CGContextRelease(cgctx);
513:     // Free image data memory for the context
514:     if (data)
515:     {
516:         free(data);
517:     }
518:
519: }
520:
521: CGContextRef CreateARGBBitmapContext (CGImageRef inImage)
522: {
523:     CGContextRef    context = NULL;
524:     CGColorSpaceRef colorSpace;
525:     void *          bitmapData;
526:     int             bitmapByteCount;
527:     int             bitmapBytesPerRow;
528:
529:     // Get image width, height. We'll use the entire image.
530:     size_t pixelsWide = CGImageGetWidth(inImage);
531:     size_t pixelsHigh = CGImageGetHeight(inImage);
532:
533:     // Declare the number of bytes per row. Each pixel in the bitmap in this
```

```
534:     // example is represented by 4 bytes; 8 bits each of red, green, blue, and
535:     // alpha.
536:     bitmapBytesPerRow   = (pixelsWide * 4);
537:     bitmapByteCount     = (bitmapBytesPerRow * pixelsHigh);
538: 
539:     // Use the generic RGB color space.
540:     colorSpace = CGColorSpaceCreateWithName(kCGColorSpaceGenericRGB);
541:     if (colorSpace == NULL)
542:     {
543:         fprintf(stderr, "Error allocating color space\n");
544:         return NULL;
545:     }
546: 
547:     // Allocate memory for image data. This is the destination in memory
548:     // where any drawing to the bitmap context will be rendered.
549:     bitmapData = malloc( bitmapByteCount );
550:     if (bitmapData == NULL)
551:     {
552:         fprintf (stderr, "Memory not allocated!");
553:         CGColorSpaceRelease( colorSpace );
554:         return NULL;
555:     }
556: 
557:     // Create the bitmap context. We want pre-multiplied ARGB, 8-bits
558:     // per component. Regardless of what the source image format is
559:     // (CMYK, Grayscale, and so on) it will be converted over to the format
560:     // specified here by CGBitmapContextCreate.
561:     context = CGBitmapContextCreate (bitmapData,
562:                                      pixelsWide,
563:                                      pixelsHigh,
564:                                      8,      // bits per component
565:                                      bitmapBytesPerRow,
566:                                      colorSpace,
567:                                      kCGImageAlphaPremultipliedFirst);
568:     if (context == NULL)
569:     {
570:         free (bitmapData);
571:         fprintf (stderr, "Context not created!");
572:     }
573: 
574:     // Make sure and release colorspace before returning
575:     CGColorSpaceRelease( colorSpace );
576: 
577:     return context;
578: }
579: 
580: @end
```

```
 1: //
 2: //  PenTool.h
 3: //  LeapPaint
 4: //
 5: //  Created by cj on 3/17/13.
 6: //  Copyright (c) 2013 cjdesch. All rights reserved.
 7: //
 8:
 9: #import <Foundation/Foundation.h>
10: #import "DrawingTool.h"
11:
12: @interface PenTool : DrawingTool
13:
14: @property (assign) float pathDistance;
15: @property (assign) CGPoint previousPoint1;
16: @property (assign) CGPoint previousPoint2;
17: @property (assign) CGMutablePathRef path;
18: @property (assign) CGPoint lastPoint;
19:
20:
21: - (void)createPath;
22:
23:
24: @end
```

```
//
//  PenTool.m
//  LeapPaint
//
//  Created by cj on 3/17/13.
//  Copyright (c) 2013 cjdesch. All rights reserved.
//

#import "PenTool.h"

@implementation PenTool

static const CGFloat kPointMinDistance = 5;
static const CGFloat kPointMinDistanceSquared = kPointMinDistance * kPointMinDistance;

- (void)inputBegan:(NSEvent *)theEvent locationInView:(CGPoint)currentPoint withSettings:(ToolSettings
*)settings{

    //[super touchBegan:touch inImageView:imageView withSettings:settings];
    [super inputBegan:theEvent locationInView:currentPoint withSettings:settings];

    [self createPath];

    //self.previousPoint1 = [touch previousLocationInView:self.drawingImageView];
    //self.previousPoint2 = [touch previousLocationInView:self.drawingImageView];

    self.previousPoint1 = currentPoint;
    self.previousPoint2 = currentPoint;


    [self inputMoved:theEvent locationInView:currentPoint];

}

- (void)inputMoved:(NSEvent *)theEvent locationInView:(CGPoint)currentPoint{

    //CGPoint currentPoint = [touch locationInView:self.drawingImageView];
    //NSPoint tvarMousePointInWindow    = [theEvent locationInWindow];


    /* check if the point is farther than min dist from previous */
    CGFloat dx = currentPoint.x - self.lastPoint.x;
    CGFloat dy = currentPoint.y - self.lastPoint.y;

    if (dx * dx + dy * dy < kPointMinDistanceSquared) {
        return;
    }

    //call touch moved after the above check - image will be restored
    [super inputMoved:theEvent locationInView:currentPoint];

    [self drawCurveEndingAtTouch:theEvent locationInView:currentPoint];

    if (self.pathDistance > 100) {
        //commit changes and recreate path, faster
        self.drawingSnapshot = self.drawingImageView.image;
        [self createPath];
    }

    self.lastPoint = currentPoint;

}

- (void)inputEnded:(NSEvent *)theEvent locationInView:(CGPoint)currentPoint{



    //first reset the image, we're still adjusting/drawing the curve
    //self.drawingImageView.image = self.drawingSnapshot;

    //finish the curve along to the final touch
    [self drawCurveEndingAtTouch:theEvent locationInView:currentPoint];

    [self releasePath];

    [super inputEnded:theEvent locationInView:currentPoint];

}

// drawing curves using quadratic beziers based on https://github.com/levinunnink/Smooth-Line-View
```

```
 80: - (void)drawCurveEndingAtTouch:(NSEvent *)theEvent locationInView:(CGPoint)currentPoint{
 81:
 82:     [self addTouchToPath:theEvent locationInView:currentPoint];
 83:
 84:     [self drawCurveForPath];
 85:
 86: }
 87:
 88:
 89: - (void)addTouchToPath:(NSEvent *)theEvent  locationInView:(CGPoint)currentPoint{
 90:
 91:
 92:     self.previousPoint2 = self.previousPoint1;
 93:     self.previousPoint1 = currentPoint;
 94:
 95:     CGPoint mid1 = midPoint(self.previousPoint1, self.previousPoint2);
 96:     CGPoint mid2 = midPoint(currentPoint, self.previousPoint1);
 97:     CGMutablePathRef subpath = CGPathCreateMutable();
 98:
 99:     CGPathMoveToPoint(subpath, NULL, mid1.x, mid1.y);
100:     CGPathAddQuadCurveToPoint(subpath, NULL, self.previousPoint1.x, self.previousPoint1.y, mid2.x, mid
2.y);
101:
102:     CGPathAddPath(self.path, NULL, subpath);
103:
104:     CGFloat dx = currentPoint.x - self.lastPoint.x;
105:     CGFloat dy = currentPoint.y - self.lastPoint.y;
106:     float lastDistance = sqrt(dx * dx + dy * dy);
107:     self.pathDistance += lastDistance;
108:
109:     CGPathRelease(subpath);
110:
111: }
112:
113: - (void)drawCurveForPath {
114:
115:     //[self setupImageContextForDrawing];
116:     //NSGraphicsContext* theContext = [NSGraphicsContext currentContext];
117:     //CGContextRef context = UIGraphicsGetCurrentContext();
118:     CGContextRef context = (CGContextRef)[[NSGraphicsContext currentContext] graphicsPort];
119:
120:     CGContextAddPath(context, self.path);
121:     CGContextStrokePath(context);
122:     //self.drawingImageView.image = UIGraphicsGetImageFromCurrentImageContext();
123:
124:
125:     CGContextFlush(context);
126:
127:
128: }
129:
130: CGPoint midPoint(CGPoint p1, CGPoint p2) {
131:     return CGPointMake((p1.x + p2.x) * 0.5, (p1.y + p2.y) * 0.5);
132: }
133:
134: - (void)releasePath {
135:
136:     if (_path != nil) {
137:         CGPathRelease(_path);
138:         _path = nil;
139:     }
140:     self.pathDistance = 0.0;
141:
142: }
143:
144: - (void)createPath {
145:
146:     [self releasePath];
147:
148:     //analyzer will report this leaks - this is released in the above method
149:     self.path = CGPathCreateMutable();
150:     self.pathDistance = 0.0;
151:
152: }
153:
154:
155:
156: @end
```

```
 1: //
 2: //  SimplePoint.h
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 2/19/13.
 6: //
 7: //
 8:
 9: #import <Foundation/Foundation.h>
10:
11:
12: @interface SimplePoint : NSObject {
13:
14: }
15:
16: @property (nonatomic, readwrite) float x;
17: @property (nonatomic, readwrite) float y;
18: //@property (nonatomic, readwrite) float z;
19: - (id)initWithPosition:(CGPoint)p;
20: - (id)initWithX:(float)x withY:(float)y;
21:
22:
23: @end
```

```
 1: //
 2: //  SimplePoint.m
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 2/19/13.
 6: //
 7: //
 8:
 9: #import "SimplePoint.h"
10:
11: @implementation SimplePoint
12:
13: - (id)initWithPosition:(CGPoint)p{
14:     if (self = [super init]) {
15:
16:         self.x = p.x;
17:         self.y = p.y;
18:
19:     }
20:     return self;
21: }
22:
23: - (id)initWithX:(float)x withY:(float)y{
24:
25:     if (self = [super init]) {
26:
27:         self.x = x;
28:         self.y = y;
29:
30:     }
31:     return self;
32: }
33:
34:
35: @end
```

```
 1: //
 2: //  ToolSettings.h
 3: //  LeapPaint
 4: //
 5: //  Created by cj on 3/18/13.
 6: //  Copyright (c) 2013 cjdesch. All rights reserved.
 7: //
 8:
 9: #import <Foundation/Foundation.h>
10:
11: @interface ToolSettings : NSObject
12:
13:
14: @property (copy) NSString *drawingTool;
15: @property (assign) int lineWidth;
16: @property (assign) int transparency;
17: @property (assign) int fontSize;
18: @property (strong) NSColor *primaryColor;
19: @property (strong) NSColor *secondaryColor;
20:
21: - (void)loadFromUserDefaults;
22: - (void)saveToUserDefaults;
23:
24: @end
```

```
    1: //
    2: //  ToolSettings.m
    3: //  LeapPaint
    4: //
    5: //  Created by cj on 3/18/13.
    6: //  Copyright (c) 2013 cjdesch. All rights reserved.
    7: //
    8:
    9: #import "ToolSettings.h"
   10:
   11: @implementation ToolSettings
   12:
   13: NSString* const kSDSettingsColor1RedKey      =  @"DRAWING_COLOR1_RED";
   14: NSString* const kSDSettingsColor1BlueKey     =  @"DRAWING_COLOR1_BLUE";
   15: NSString* const kSDSettingsColor1GreenKey    =  @"DRAWING_COLOR1_GREEN";
   16: NSString* const kSDSettingsColor1AlphaKey    =  @"DRAWING_COLOR1_ALPHA";
   17: NSString* const kSDSettingsColor2RedKey      =  @"DRAWING_COLOR2_RED";
   18: NSString* const kSDSettingsColor2BlueKey     =  @"DRAWING_COLOR2_BLUE";
   19: NSString* const kSDSettingsColor2GreenKey    =  @"DRAWING_COLOR2_GREEN";
   20: NSString* const kSDSettingsColor2AlphaKey    =  @"DRAWING_COLOR2_ALPHA";
   21: NSString* const kSDSettingsLineWidth         =  @"DRAWING_LINE_WIDTH";
   22: NSString* const kSDSettingsTransparency      =  @"DRAWING_TRANSPARENCY";
   23: NSString* const kSDSettingsDrawingTool       =  @"DRAWING_TOOL";
   24: NSString* const kSDSettingsFontSize          =  @"DRAWING_FONT_SIZE";
   25:
   26:
   27: - (void)loadFromUserDefaults {
   28:
   29:     NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
   30:
   31:     if ([defaults objectForKey:kSDSettingsColor1AlphaKey]) {
   32:         self.primaryColor = [NSColor colorWithCalibratedRed:[defaults floatForKey:kSDSettingsColor1Red
Key] green:[defaults floatForKey:kSDSettingsColor1GreenKey] blue:[defaults floatForKey:kSDSettingsColor1BlueK
ey] alpha:[defaults floatForKey:kSDSettingsColor1AlphaKey]];
   33:     } else {
   34:         self.primaryColor = [NSColor blueColor];
   35:     }
   36:
   37:     if ([defaults objectForKey:kSDSettingsColor2AlphaKey]) {
   38:
   39:
   40:         self.secondaryColor = [NSColor colorWithCalibratedRed:[defaults floatForKey:kSDSettingsColor2R
edKey] green:[defaults floatForKey:kSDSettingsColor2GreenKey] blue:[defaults floatForKey:kSDSettingsColor2Blu
eKey] alpha:[defaults floatForKey:kSDSettingsColor2AlphaKey]];
   41:     } else {
   42:         self.secondaryColor = [NSColor redColor];
   43:     }
   44:
   45:     if ([defaults objectForKey:kSDSettingsLineWidth]) {
   46:         self.lineWidth = [defaults integerForKey:kSDSettingsLineWidth];
   47:     } else {
   48:         self.lineWidth = 10;
   49:     }
   50:
   51:     if ([defaults objectForKey:kSDSettingsTransparency]) {
   52:         self.transparency = [defaults integerForKey:kSDSettingsTransparency];
   53:     } else {
   54:         self.transparency = 0;
   55:     }
   56:
   57:     if ([defaults objectForKey:kSDSettingsDrawingTool]) {
   58:         self.drawingTool = [defaults stringForKey:kSDSettingsDrawingTool];
   59:         //fix old stored setting (Brush #1, Brush #2, etc)
   60:         if ([self.drawingTool rangeOfString:@"Brush #"].location != NSNotFound) {
   61:             self.drawingTool = @"Brush";
   62:         }
   63:     } else {
   64:         self.drawingTool = @"Pen";
   65:     }
   66:
   67:     if ([defaults objectForKey:kSDSettingsFontSize]) {
   68:         self.fontSize = [defaults integerForKey:kSDSettingsFontSize];
   69:     } else {
   70:         self.fontSize = 50;
   71:     }
   72:
   73: }
   74:
   75: - (void)saveToUserDefaults {
   76:
```

```
   77:         NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
   78:
   79:     CGFloat red = 0.0, green = 0.0, blue = 0.0, alpha =0.0;
   80:     [self.primaryColor getRed:&red green:&green blue:&blue alpha:&alpha];
   81:
   82:     [defaults setFloat:red forKey:kSDSettingsColor1RedKey];
   83:     [defaults setFloat:green forKey:kSDSettingsColor1GreenKey];
   84:     [defaults setFloat:blue forKey:kSDSettingsColor1BlueKey];
   85:     [defaults setFloat:alpha forKey:kSDSettingsColor1AlphaKey];
   86:
   87:     [self.secondaryColor getRed:&red green:&green blue:&blue alpha:&alpha];
   88:
   89:     [defaults setFloat:red forKey:kSDSettingsColor2RedKey];
   90:     [defaults setFloat:green forKey:kSDSettingsColor2GreenKey];
   91:     [defaults setFloat:blue forKey:kSDSettingsColor2BlueKey];
   92:     [defaults setFloat:alpha forKey:kSDSettingsColor2AlphaKey];
   93:
   94:     [defaults setInteger:self.lineWidth forKey:kSDSettingsLineWidth];
   95:     [defaults setInteger:self.transparency forKey:kSDSettingsTransparency];
   96:     [defaults setObject:self.drawingTool forKey:kSDSettingsDrawingTool];
   97:     [defaults setInteger:self.fontSize forKey:kSDSettingsFontSize];
   98:
   99:     [defaults synchronize];
  100:
  101: }
  102:
  103: @end
```

```
 1: //
 2: //  TrackedFinger.h
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 2/8/13.
 6: //
 7: //
 8:
 9: #import <Foundation/Foundation.h>
10: #import "LPDrawingTool.h"
11: @interface TrackedFinger : NSObject
12:
13: @property (nonatomic, strong) NSString* fingerID;
14: @property (nonatomic, readwrite) BOOL updated;
15: @property (nonatomic, readwrite) CGPoint position;
16: @property (weak) LPDrawingTool* tool;
17:
18: - (id)initWithID:(NSString*)finger withPosition:(CGPoint)p;
19: @end
```

```
 1: //
 2: //  TrackedFinger.m
 3: //  LeapPuzz
 4: //
 5: //  Created by cj on 2/8/13.
 6: //
 7: //
 8:
 9: #import "TrackedFinger.h"
10:
11: @implementation TrackedFinger
12: @synthesize tool;
13: - (id)initWithID:(NSString*)finger withPosition:(CGPoint)p{
14:     if (self = [super init]) {
15:         self.fingerID = finger;
16:         self.updated = TRUE;
17:         self.position = p;
18:     }
19:     return self;
20: }
21: @end
```