Quil: An experimental system for online recognition of handwritten music notation

Geoff Chirgwin

December 31, 2010

Thesis submitted to the Faculty of the School of Informatics and Computing, Indiana University, Bloomington, in partial fulfillment for the requirements of the degree

Master of Science

In

Human Computer Interaction / Music Informatics
Accepted by the faculty of the School of Informatics and Computing, Indiana University, in partial fulfillment of the requirements for the degree Master of Science.

Master of Science Candidate: Geoff Chirgwin
Capstone Instructor: Erik Stolterman, Martin Siegel
Advisor(s): Christopher Raphael, Larry Yaeger

Final Project Approval

<table>
<thead>
<tr>
<th>Person</th>
<th>Signature</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Advisor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secondary Advisor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capstone Instructor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program Director</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graduate Records Admin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Associate Dean</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Month and Year ________________
1 Abstract

This paper describes a system for the online recognition of handwritten music notation. Quil is an experimental implementation of this system. The present software is an attempt to develop a transparent user interface not unlike pencil and paper. It is best suited for direct manipulation or gestural hardware interfaces using either a stylus or a finger for input, but can be used with a mouse or trackpad. Though still inchoate, the current implementation should work well enough on any
late-model web browser with HTML5 support. No additional software installation is required and no browser plugins are required.

2 Introduction

In the process of notating music, there are two persistent antique relics: 1) the use of pencil and paper and 2) the system of conventional western music notation (CWMN). These two things are here to stay, and at their core, they are changing at a very slow pace.

While the problem is not intractable, it may well be impossible – or close to impossible – to solve in a generalizable way. And even if a general solution is found, the implementation must be good enough to be accepted by users of the system. The true adoption of a system of music notation handwriting recognition will only be the result of solving these peripheral problems in addition to the core problems central to recognition.

We have attempted to make the problem more tractable by defining clear and modest limits from the outset. Admittedly, we took an already small subset of music representation and made it even smaller by limiting it to only the most basic discrete symbols representing pitch and rhythm elements.

Of course, there are other means by which to convey musical ideas and intent– such as audio recordings made by the creator, where the performer receives an audio recording and learns the material by rote– but those are entirely outside the scope of this paper. There are also numerous other systems for capturing musical ideas using visual symbols. These are also outside the scope of the work described here.

Here we are only concerned with capturing musical ideas and conveying intent using conventional Western music notation (CWMN). CWMN is a system that can– despite its idiosyncrasies– succinctly capture a large breadth and depth of musical information, as well as convey richly prescriptive musical ideas. While CWMN can be used as a descriptive notation– that is, to describe a previously existing performance, such as in the case of a human-produced transcription of an audio recording– we are primarily focused on using CWMN to prescribe a future musical performance.

We will ideally generalize the user group of this system to include any person wishing to create a CWMN sketch. That is, a user wishing to capture a simple prescriptive CWMN score from their imagination. In practice, because of the limitations of the present software, the most common practical use case here is a composer roughly sketching a new idea.

3 Background

3.1 Notating music by computer

The basic landscape of music notation software has changed surprisingly little since Don Byrd’s landmark dissertation on the subject in 1984. [Byr84] Fully automated high quality music notation was beyond the state of the art then, and it remains so today. The user input techniques have remained little changed; they are 1) piano keyboard, 2) alphanumeric keyboard, 3) mouse, or some combination of the three. Current systems generally use a palette of symbols that can be selected and placed on a staff by manipulating them with a mouse pointer. Keyboard shortcuts and interactive keyboard input in combination with graphical user interfaces are also quite common.
3.2 Drawing music

Even in short informal observations of musicians hand-copying a given printed musical, the anecdotal evidence is surprising. Notating music is most often an activity that is private, and therefore not often observed by others. Because of this, musicians generally have little knowledge of the details of how others notate music by hand. Take, for example the treble (or “G”) clef. Even though two different users may draw the symbol with a single pen stroke, they may use one of two directions (either outside to inside or inside to outside). We can look to the research – not necessarily music-related – in human motor control to learn something about drawing music.

Drawing and writing almost exclusively provide the expression of thoughts to others. [Ros91] Notating music lies somewhere between drawing and writing. Rosenbaum describes the model of writing in three stages:

1. The abstract message is selected,
2. The abstract message is translated into appropriate linguistic and expressive segments,
3. The segments are realized as a series of efferent commands.

3.3 Music is visual, not just aural

It is important to distinguish the aural (both real in the live acoustic world and imagined inside the mind of the creator) from the visual facets of music. Much of the aural work is done by the mind of the creator, possibly with the approximate feedback – perhaps musical pitch only – from an acoustic instrument or software external to the presently described system. What remains is a purely visual and symbolic world. This may seem strange, but it can be preferable to have a simple and crude software pitch feedback (or no acoustic feedback at all) rather than that which is close but misses the mark. Crude pitch-only or no in-software feedback is often sufficient for a trained musician and can prevent a sort of uncanny valley, caused when the audio recreation is fairly true to the acoustic instrument it is emulating (but perhaps still low fidelity, non-audio sense). [Mor70]

3.4 Prior work in optical music recognition

In order to describe related work, it is important to first make several distinctions. Firstly, the distinction between online and offline recognition should be made. By online recognition, we mean systems with which the user interacts more or less in real time. By offline recognition, we mean systems that recognize CWMN from a pre-generated score, most likely a printed page (or a scanned bitmap thereof). Secondly, the distinction between typeset and handwritten CWMN should be made.

Several research systems for gestural or online input have been developed, notably by Susan E. George at the University of South Australia [Geo05], and at work at IBM’s T.J. Watson Research Center. As described, the Pen-based Music Editor developed at T.J. Watson seems to be the effort closest yet to the ideal user interface. [Sil96] Purely gestural interfaces should be differentiated by their requirement of learning a specialized set of gestures that are not necessarily related to CWMN. There has been much more work done in offline optical music recognition (OMR) for both handwritten and typeset CWMN, with significant contributions from Ichiro Fujinaga at McGill
University, beginning with his doctoral dissertation. [Fuj97]. Offline recognition is fundamentally different in that it deals with pre-existing printed documents. There are several commercially available offline OMR programs that work on scanned images of typeset scores. There has also been some recent work in offline recognition of handwritten scores [PF09], [RCC10]

4 Ideal

The system presented here is an attempt to develop a transparent user interface not unlike pencil and paper. It is best suited for gestural interfaces using either a stylus or a finger for input, but can be used with a mouse or trackpad. The user can enter music notation with their existing knowledge of CWMN and directly manipulate the recognized symbols.

Ideally, a music notation user interface should allow the user to sketch music notation quickly enough before they forget it. Any tools used should not get in the way and should not force the user to think in a particular way, especially one that is unnatural or uncomfortable for the user. And, most importantly, the tools shouldn’t force any decisions to be made prematurely for the tools’ sake (or more accurately the convenience of implementation on the part of the person programming the system).

We aim to implement this system with no extensions or modifications to CWMN and no specialized gestures for the user to learn. Previous gestural notation implementations have required the user to learn a set of specialized gestures. One such system is the The Music Notepad [For98], developed by a group at Brown University. While the hardware interface is close to ideal, the software interface is far from ideal.

The general quality of the handwritten sketch is unacceptable for expression of thoughts to others, whereas the final typeset version is suitable for this purpose. Ideally, the path— in terms of both human time and effort— between these two versions will be a short as possible.

Figure 1: An example CWMN sketch.
The Bride

Duration: c. 2:10 without repeats

\( \frac{\text{if you’re not crying, someone hasn’t done their job}}{	ext{sempre tenuto, sadder than the saddest thing you know}} \)

\( \frac{\text{if you’re not crying, someone hasn’t done their job}}{	ext{sempre tenuto, sadder than the saddest thing you know}} \)

\( \frac{\text{if you’re not crying, someone hasn’t done their job}}{	ext{sempre tenuto, sadder than the saddest thing you know}} \)

\( \frac{\text{if you’re not crying, someone hasn’t done their job}}{	ext{sempre tenuto, sadder than the saddest thing you know}} \)
5 Prototypes

The initial implementation was naive and crude, but served a purpose nonetheless. It was intended to do nothing more than demonstrate the concept and start the research conversation. This implementation was based heavily on the Noel Billing’s [Bil10] Adobe Flash implementation for recognizing discrete roman alphabetic characters. Using ActionScript version 3 (AS3) [GH06] and Macromedia eXtensible Markup Language (MXML) [Inc07], this implementation runs in the Adobe Runtime Environment (AIR) [Inc10], a virtual machine for deploying Adobe Flash standalone desktop applications. Since these tools are either open source and freely available, it is possible to build and run this application without purchasing any software. A working version of this implementation is available on the World Wide Web (WWW or simply Web). [Chi10c]

There are two qualities of this initial demonstration of concept worth noting. Firstly, the staff lines provide nothing more than a vertical scale reference. Secondly, an attempt was made to add staff lines to the drawing surface of the pen tablet itself. This was mocked up by placing a piece of paper with hand-drawn staff lines over the surface of the tablet. An attempt was made to match the scale and spacing of the lines on the tablet to those on the screen. The main limitation of this approach is the physical separation and seeming disconnectedness between visual feedback on the screen and the drawing hand. This mockup would be better with a touchscreen, where the drawing surface and display were one and the same.

To remove some of the cumbersome installation requirements of the AIR version, this implementation was ported to pure JavaScript (ECMAScript) [Int10] and Hypertext Markup Language version 5 (HTML5) [Hic10]. A working version of this implementation is also available on the WWW. [Chi10e] A very large majority of the initial prototype code was in the ActionScript language. Since ActionScript is effectively an extended variation of ECMAScript, this bulk of this port was relatively trivial. Since ActionScript is a stricter language, this port largely consisted of deletions such as omitting the static typing ActionScript introduces to an otherwise dynamically typed language.

These early implementations have one major limitation; they only support a single character which can only consist of a single stroke. Another limitation is that they only support a very limited set of symbols: 1) whole note, 2) stem down quarter note, 3) stem up quarter note, 4) eighth rest, 5) quarter rest, 6) stem up eighth note, 7) stem down eighth note, 8) stem down half note, 9) stem up half note, 10) flat, 11) treble clef. These eleven symbols were cherry picked based on the criterion of single-stroke drawability.

Prior to this work, but relevant enough to warrant a mention is an early prototype in purely gestural CWMN input. [Chi08]

6 Implementation

The final implementation can be described in terms of its three major component stages:

1. **segmentation** is defined as which combination of user-input strokes form a symbol. A stroke is defined as a vector of Cartesian points captured from pen down to pen up. The user-drawn ink is segmented to determine which combination of strokes are most likely to combine to form a symbol.
2. **classification** is defined as what is the most likely symbol for a given combination of strokes. Classification always works on user-drawn templates of ink. At present, all templates are based on the author’s hand. It is important to note that drawn symbols are nearly always very different from typeset or printed CWMN symbols. Only a small subset of all the symbols that constitute CWMN are recognized. This subset of symbols currently only includes discrete symbols, and was chosen to capture the meat of CWMN: all core symbols directly relating directly to pitch and rhythm. Recognition of continuous symbols, such as beams or slurs, is not currently possible. After both template strokes and input strokes are normalized by scale and to an equidistant fixed number of points template matching is performed by calculating the mean of the distances from each drawn point to the nearest hand-trained template point and vice-versa. In other words, this comparison is bi-directional between drawn ink and the template. This could also be described as nearest neighbor Euclidean distance.

3. **language model** is defined as how the discrete symbols fit together in a musically meaningful way. A hand-coded collection of bigrams define how likely one symbol is to follow another. It seems unlikely that the language model is of that much importance, as these bigrams seem to have little meaning or significance. Because of this suspected insignificance, we chose to simply hand-code, rather than train the language model on a large corpus of symbolic music.
Once each symbol is recognized, it is typeset more or less at the same position of the user-drawn ink. It is possible to proportionately scale up and down everything visible on screen, and to change the color and opacity of the drawn ink in order to increase contrast against the typeset symbols. A
testing framework can record and replay serialized ink, as well as take bitmap captures for iterative test comparisons and identification of regressions. The user can also interactively correct misclassified symbols by explicitly selecting the intended symbol from a contextual menu. This menu contains a set of the first n symbols (where n is roughly 9) ordered in decreasing order of likeliness. Because of this ordering, it is hoped that the intended symbol will appear in this list, but that may not always be the case. We made some attempts to allow interactive session-based model feedback incorporating user correction, but it would more beneficial to capture the user correction data as part of a larger scheme for training an improved classifier.

In this implementation, we attempt to recognize a large subset of CWMN symbols. This subset has been expanded to include 1) bass ("F") clef, 2) treble ("G") clef, 3) flat, 4&5) sixteenth note (stem upwards or downwards), 6&7) thirty-second note (stem upwards or downwards), 8&9) eighth note (stem upwards or downwards), 10&11) half note (stem upwards or downwards), 12&13) quarter note (stem upwards or downwards), 14) whole note, 15) sixteenth rest, 16) eighth rest, 17) half rest, 18) quarter rest, 19) whole rest, 20) sharp, 21) natural, 22) barline, 23) dot (augmentation), 24-32) Arabic numerals nine through one. Even though we have nearly tripled the number of recognized symbols from the initial prototype, this is still a very small subset of all possible CWMN symbols. It might prove impossible to define such a complete set of CWMN symbols upon which a consensus among experts can be made.

Figure 9: Two symbols drawn with five strokes. The first stroke forms a treble clef, while strokes 2 through four form a sharp

Figure 10: All recognized symbols.

6.1 Demonstrations and live application

A collection of demonstration videos and a video of a talk on the system can be found on the Web. [Chi10b] A live version of the application is available for use on the Web. [Chi10d]
7 Conclusion

In this paper, we have described an experimental and inchoate system for the online recognition of handwritten CWMN. Because we imposed limitations to what the system could handle— a single
staff containing a single voice and a very limited number of recognized symbols— we were able to develop a crudely working system.

8 Future Work

The weakest component of the current system is the symbol classifier, which should be replaced with a more sophisticated classifier. As of this writing, the most promising group of classifiers appears to be neural networks, possibly either convolutional neural networks or some variation using the spatial domain. [WLJJ93], [LBH⁺93] There is not yet enough training data to make such a classifier feasible. In the final implementation described in the paper, the system was only trained on the author’s hand.

Nearly all classifiers would require a wider sampling and much larger corpus of training data. One possible method for collecting large amounts of training data would be to crowd-source using a simple Web browser-based application, and allowing large numbers of users to input their individual interpretations of each symbol used in classification. A tricky problem here is the ten-
tendency to take the shown symbol literally. Briefly showing a typeset symbol and then hiding it is an attempt to eliminate the problem of the user literally tracing the typeset symbol. While not yet used or even tested much, we have already implemented a simple application to serve this purpose. [Chi10a] In this application, the user is presented with a series of symbols. Each symbol is displayed for a short time, after which the user draws that symbol. Each symbol is captured by the system as a series of points, which can then later be used to train the recognizer. It would also be possible to capture some kind of meaningful training data using the inline user feedback correction interface described previously.

Non-linear writing systems such as mathematical notation or Chinese are similar to music notation, and applying recognition techniques previously successful in those domains would likely benefit research in the CWMN domain. Finally, it might be interesting to attempt to distill a meaningful language model from a large corpus of visual symbolic music.

9 References

References


10 Source Code Listing

10.1 Gesture Recognition and Classification

```javascript
function Gesture ()
{
    /* static */
    DEFAULT_NUM_POINTS = 80;
    MIN_NUM_POINTS = 4;
    MAX_NUM_POINTS = 4096;

    /* public */
    this.symbol = "";
    this.numNormalPoints = DEFAULT_NUM_POINTS;
    this.strokes = [];
    this.matches = [];
    this.originalStrokeIndices = [];
    this.startingPoint = new Point();
    this.topLeft = null;
    this.bottomRight = null;
    this.height = null;
    this.width = null;
    this.strokeIdx = 0;
    this.bagOfPointsSorted = false;
    this.staffSize = 128;

    /* private */
    var _offsetX = 0;
    var _offsetY = 0;
    var _shapeNeedsUpdating = false;
    var _lineThickness = 7;
    var _lineColor = 0x000000;
    var _shapeMarginTop = 10;
    var _shapeMarginLeft = 10;
    var _length = -1;
    var _averageCenter;

    var MAX_SCALE_X = 1.15;
    var MIN_SCALE_X = 1 / MAX_SCALE_X;

    var MAX_SCALE_Y = 1.15;
    var MIN_SCALE_Y = 1 / MAX_SCALE_Y;

    var MAX_ASPECT_RATIO_DELTA = 1.0;

    this.getTopLeft = function ()
    {
        if (!this.topLeft)
            return null;
```
```javascript
{  
    this.getBoundingBox();
}

return this.topLeft;
}

this.getBottomRight = function ()
{
    if (!this.bottomRight)
    {
        this.getBoundingBox();
    }
    return this.bottomRight;
}

this.getWidth = function ()
{
    if (!this.width)
    {
        this.getBoundingBox();
    }
    return this.width;
}

this getHeight = function ()
{
    if (!this.height)
    {
        this.getBoundingBox();
    }
    return this.height;
}

this.addPoint = function (x, y)
{
    this.strokes[this.strokeIdx].push(new Point(x, y));
    _shapeNeedsUpdating = true;
}

this.strokeLength = function ()
{
    var length = 0;
    for (var i = 0; i < this.strokes.length; i++)
    {
        for (var j = 1; j < this.strokes[i].length; j++)
        {
            length += Point.distance(this.strokes[i][j - 1], this.strokes[i][j]);
        }
    }
    return length;
}
```
```javascript
this.normalizeShape = function() {
  var normalStrokeLength = this.strokeLength() / (this.numNormalPoints - 1);
  for (var i = 0; i < this.strokes.length; i++) {
    var newPoints = [];
    if (this.strokes[i].length <= 0)
      continue;
    newPoints.push(this.strokes[i][0]);
    var startPoint = this.strokes[i][0];
    var endPoint = this.strokes[i][0];
    var previousDistanceEnd = 0;
    var previousDistanceStart = 0;
    var distance = 0;
    var strokeLength = 0;
    var pointIdx = 1;
    var j = 0;
    while (j <= MAX_NUM_POINTS) {
      j ++;
      var excess = previousDistanceEnd - distance;
      if (excess >= normalStrokeLength)
        {
          distance += normalStrokeLength;
          var ratio = (distance - previousDistanceStart) / strokeLength;
          var newPoint = new Point((endPoint.x - startPoint.x) * ratio + startPoint.x,
                                    (endPoint.y - startPoint.y) * ratio + startPoint.y);
          newPoints.push(newPoint);
        }
      else
        {
          if (pointIdx == this.strokes[i].length)
            break;
          startPoint = endPoint;
          endPoint = this.strokes[i][pointIdx];
          previousDistanceStart = previousDistanceEnd;
          strokeLength = endPoint.subtract(startPoint).getLength();
          previousDistanceEnd += strokeLength;
          pointIdx ++;
        }
    }
```

// add final point
if (newPoints.length < this.numNormalPoints)
{
    newPoints.push(endPoint);
}
this.strokes[i] = newPoints;

// center on average center point
this.normalizeToAverageCenter = function()
{
    for (var i = 0; i < this.strokes.length; i++)
    {
        for (var j = 0; j < this.strokes[i].length; j++)
        {
            var point = this.strokes[i][j];
            point.x -= this.getAverageCenter().x;
            point.y -= this.getAverageCenter().y;
        }
    }
}

this.normalizeToFixedPoint = function(centerPoint, recalculateCentroid, offsetPoint)
{
    if (offsetPoint == undefined)
    {
        offsetPoint = new Point(0, 0);
    }
    if (recalculateCentroid == undefined)
    {
        recalculateCentroid = false;
    }
    var averageCenter = this.getAverageCenter(recalculateCentroid);
    for (var i = 0; i < this.strokes.length; i++)
    {
        for (var j = 0; j < this.strokes[i].length; j++)
        {
            var point = this.strokes[i][j];
            // only use vertical offset for staff margin?
            point.x = offsetPoint.x + point.x + (centerPoint.x - averageCenter.x);
            point.y = offsetPoint.y + point.y + (centerPoint.y - averageCenter.y);
        }
    }
    // Something is buggy about this
    this.setTopLeftToZero = function()
{ var offsetPoint = this.topLeft;
    var averageCenter = this.getAverageCenter();
    for (var i = 0; i < this.strokes.length; i++)
    {
        for (var j = 0; j < this.strokes[i].length; j++)
        {
            var point = this.strokes[i][j];
            point.x -= offsetPoint.x
            point.y -= offsetPoint.y;
        }
    }

    this.getAverageCenter = function (recalculate)
    {
        if (!recalculate && _averageCenter != null)
        {
            return _averageCenter;
        }

        var centerX = 0;
        var centerY = 0;
        if (this.strokes == undefined || this.strokes.length <= 0 || this.strokes[0] == undefined)
        {
            return new Point(0, 0);
        }

        for (var i = 0; i < this.strokes[0].length; i++)
        {
            var point = this.strokes[0][i];
            centerX += point.x;
            centerY += point.y;
        }

        centerX /= this.strokes[0].length;
        centerY /= this.strokes[0].length;

        _averageCenter = new Point(centerX, centerY);
        //global for sort function
        window._averageCenter = _averageCenter;
        return (_averageCenter);
    }

    this.getBoundingBox = function ()
    {
        if (this.strokes[0].length < 1)
        {
            return false;
        }
        var firstPoint = this.strokes[0][0];
        var minX = firstPoint.x;
```javascript
var minY = firstPoint.y;
var maxX = firstPoint.x;
var maxY = firstPoint.y;
for (var i = 0; i < this.strokes.length; i++)
{
    for (var j = 0; j < this.strokes[i].length; j++)
    {
        var point = this.strokes[i][j];
        if (point.x < minX)
        {
            minX = point.x;
        }
        if (point.y < minY)
        {
            minY = point.y;
        }
        if (point.x > maxX)
        {
            maxX = point.x;
        }
        if (point.y > maxY)
        {
            maxY = point.y;
        }
    }
}

// account for fuzz
var margin = 10;
minX -= margin;
minY -= margin;
maxX += margin;
maxY += margin;

var width = maxX - minX;
var height = maxY - minY;
this.width = width;
this.height = height;
this.topLeft = new Point(minX, minY);
this.bottomRight = new Point(maxX, maxY);
}

// match this gesture's scale to another gesture
this.matchScale = function (matchGesture)
{
    // width
    var scaleX = matchGesture.getWidth() / this.getWidth();
    // height
    var scaleY = matchGesture.getHeight() / this.getHeight();

    // always leave these ordered x:y
    var oldAspectRatio = this.getWidth() / this.getHeight();
    var newAspectRatio = matchGesture.getWidth() / matchGesture.getHeight();
```
308 var aspectRatioDelta = Math.abs(newAspectRatio - oldAspectRatio);
309
310 if (scaleX > MAX_SCALE_X)
311 {
312 scaleX = MAX_SCALE_X;
313 }
314 if (scaleX < MIN_SCALE_X)
315 {
316 scaleX = MIN_SCALE_X;
317 }
318
319 if (scaleY > MAX_SCALE_Y)
320 {
321 scaleY = MAX_SCALE_Y;
322 }
323 if (scaleY < MIN_SCALE_Y)
324 {
325 scaleY = MIN_SCALE_Y;
326 }
327
328 for (var i = 0; i < this.strokes.length; i++)
329 {
330 Point.scalePoints2D(this.strokes[i], scaleX, scaleY);
331 }
332
333 }
334
335 // privileged
336 this.normalizeScale = function ()
337 {
338 for (var i = 0; i < this.strokes.length; i++)
339 {
340 if (this.strokes[i].length <= 0)
341 {
342 continue;
343 }
344 var firstPoint = this.strokes[i][0];
345 var minX = firstPoint.x;
346 var minY = firstPoint.y;
347 var maxX = firstPoint.x;
348 var maxY = firstPoint.y;
349 for (var j = 0; j < this.strokes[i].length; j++)
350 {
351 var point = this.strokes[i][j];
352 if (point.x < minX)
353 {
354 minX = point.x;
355 }
356 if (point.y < minY)
357 {
358 minY = point.y;
359 }
360 if (point.x > maxX)
361 }
```javascript
{  
  maxX = point.x;
}

if (point.y > maxY)
{
  minY = point.y;
}

var width = maxX - minX;
var height = maxY - minY;
this.width = width;
this.height = height;
var scale = (width > height) ? width : height;

if (scale <= 0)
{
  return;
}
scale /= 1;
Point.scalePoints(this.strokes[i], scale);
}

this.scale = function(scaleFactor)
{

  for (var i = 0; i < this.strokes.length; i++)
  {
    for (var j = 0; j < this.strokes[i].length; j++)
    {
      var point = this.strokes[i][j];
      point.x *= scaleFactor;
      point.y *= scaleFactor;
    }
  }

  this.getNumPoints = function()
  {
    return this.strokes[0].length;
  }

  this.startGesture = function(x, y)
  {
    this.addPoint(x, y);
  }

  this.getLength = function()
  {
    if (_length == -1 || _shapeNeedsUpdating)
    {
    
    
```
    _length = 0;
    var lastPoint;
    for (var i = 0; i < this.strokes[0].length; i++)
    {
        var point = this.strokes[0][i];
        if (lastPoint !== null)
        {
            _length += Point.prototype.distance(lastPoint, point);
        }
        lastPoint = point;
    }
    return _length;
}

this.createFromJson = function(obj)
{
    this.symbol = obj.symbol;
    if ("staffSize" in obj)
    {
        this.staffSize = obj.staffSize;
    }
    this.strokes = [];
    for (var i = 0; i < obj.strokes.length; i++)
    {
        this.strokes[i] = [];
        for (var j = 0; j < obj.strokes[i].length; j++)
        {
            var point = obj.strokes[i][j];
            this.strokes[i].push(new Point(point.x, point.y));
        }
    }
    this.continueGesture = function(x, y)
    {
        this.addPoint(x, y);
    }
    this.newStroke = function() {
        if (this.strokes.length > Gesture.MAX_STROKES)
        {
            return false;
        }
        this.strokes.push(new Array());
        this.strokeIdx ++;
    }
    this.popStroke = function() {
        if (this.strokeIdx <= 0)
        {
    return null;
    }
    this.strokeIdx --;
    return this.strokes.pop();
    }
    }
    this.normalize = function ()
    {
        this.normalizeShape();
    }
    this.getPoints = function ()
    {
        return this.strokes[0];
    }
    this.getAllPoints = function ()
    {
        return this.strokes;
    }
    this.toString = function ()
    {
        var string = "gesture with " + this.strokes.length + "points 
:";
        for ( var i = 0; i < this.strokes.length; i ++)
        {
            string += this.strokes[i] + ",";
        }
        return string;
    }
    } // unimplemented
    this.getJson = function ()
    {
        var json = new String();
        var pointsAry = [];
        for ( var i = 0; i < this.strokes.length; i ++)
        {
            var point = this.strokes[i];
            pointsAry.push({ "x": point.x, "y": point.y});
        }
        return Json.encode(pointsAry);
    }
    // need this, does average center make a difference over (0,0)?
    this.pointAngleDifference = function (point1, point2)
    {
        var angle1 = Math.atan2(this.getAverageCenter().y - point1.y, this.getAverageCenter().x - point1.x);
        var angle2 = Math.atan2(this.getAverageCenter().y - point2.y, this.getAverageCenter().x - point2.x);
        return Math.abs(angle2 - angle1);
```javascript
this.sortAllPointsByAngleFromCentroid = function(point1, point2)
{
    var angle1 = Math.atan2(window._averageCenter.y - point1.y, window._averageCenter.x - point1.x);
    var angle2 = Math.atan2(window._averageCenter.y - point2.y, window._averageCenter.x - point2.x);

    if (angle1 > angle2)
    {
        return 1;
    }
    else if (angle1 < angle2)
    {
        return -1;
    }
    return 0;
}

// create an array of all points in all gestures
this.bagOfPoints = function()
{
    var bagOfPoints = [];
    for (var i = 0; i < this.strokes.length; i++)
    {
        bagOfPoints = bagOfPoints.concat(Point.clonePoints(this.strokes[i]));
    }
    return bagOfPoints;
}

// useful for visual debugging
this.drawGestureOnNewCanvas = function(parentElementId, text)
{
    // create a new canvas
    var newCanvas = document.createElement("canvas");
    // set dimensions of new canvas
    this.getBoundingBox();

    newCanvas.setAttribute("width", parseInt(this.width) * 4 + 100);
    newCanvas.setAttribute("height", parseInt(this.height) * 4 + 100);

    // border around canvas
    newCanvas.setAttribute("style", "border: 1px solid #666; ");

    var context = newCanvas.getContext('2d');
    context.fillStyle = "black";
    context.fillText(text, 10, 10);
    context.lineWidth = 5;
    // draw the gesture to it
    var marginPoint = this.getBottomRight().clone();
```

27
Gesture.drawOnCanvas(newCanvas, this, null, marginPoint);

// append the new canvas to the page
document.getElementById(parentElementId).appendChild(newCanvas);
return newCanvas;

this.clone = function() {
    var clonedObj = new Gesture();
    clonedObj.symbol = this.symbol;
    clonedObj.numNormalPoints = this.numNormalPoints;
    // deep copy of points
    clonedObj.strokes = this.cloneStrokes();
    clonedObj.startingPoint = this.startingPoint;
    clonedObj.topLeft = this.topLeft;
    clonedObj.bottomRight = this.bottomRight;
    clonedObj.height = this.height;
    clonedObj.width = this.width;
    clonedObj.strokeIdx = this.strokeIdx;
    clonedObj.bagOfPointsSorted = this.bagOfPointsSorted;
    clonedObj.getAverageCenter();
    return clonedObj;
}

this.cloneStrokes = function(startIdx, endIdx) {
    if (startIdx == undefined || startIdx < 0) {
        startIdx = 0;
    }
    if (endIdx == undefined || endIdx > this.strokes.length) {
        endIdx = this.strokes.length;
    }
    var clonedStrokes = [];
    for (var i = startIdx; i < endIdx; i++) {
        if (this.strokes[i].length > 0) {
            clonedStrokes.push(Point.clonePoints(this.strokes[i]));
        }
    }
    return clonedStrokes;
}
Gesture.prototype.endGesture = function()
{
    // return false if we don't have enough points to make a gesture
    return this.checkGesture();
}

Gesture.prototype.checkGesture = function()
{
    if (this.getPoints().length < MIN_NUM_POINTS)
    {
        return false;
    }
    return true;
}

Gesture.drawOnCanvas = function(canvas, gesture, strokeStyle, offsetPoint)
{
    var points = gesture.getAllPoints();
    var context = canvas.getContext('2d');
    context.strokeStyle = strokeStyle;
    context.lineWidth = 5;

    // check to see that there are points
    if (points.length < 1 || points[0].length < 1)
    {
        return;
    }
    for (var i = 0; i < points.length; i++)
    {
        context.beginPath();
        for (var j = 0; j < points[i].length; j++)
        {
            var point = points[i][j];
            if (offsetPoint)
            {
                point.add(offsetPoint);
            }
            if (j == 0)
            {
                context.moveTo(point.x, point.y);
            }
            else
            {
                context.lineTo(point.x, point.y);
            }
        }
        context.stroke();
    }
    context.lineWidth = 1;
}

// bitmap comparison
Gesture.bitmapRank = function(gesture1, gesture2)
```javascript
var width = 300;
var height = 340;

var canvas1 = document.getElementById("canvas1");
Utils.clearCanvas(canvas1);
Gesture.drawAndScaleToCanvas(canvas1, gesture1);

var canvas2 = document.getElementById("canvas2");
Utils.clearCanvas(canvas2);
Gesture.drawAndScaleToCanvas(canvas2, gesture2);

// Get the CanvasPixelArray from the given coordinates and dimensions.
var imgData1 = canvas1.getContext('2d').getImageData(0, 0, width, height);
var pixels1 = imgData1.data;
var imgData2 = canvas2.getContext('2d').getImageData(0, 0, width, height);
var pixels2 = imgData2.data;

var score = 0;
// Loop over each pixel
for (var i = 3; i < pixels1.length; i += 4) {
    score += Math.abs(pixels1[i] - pixels2[i]);
}
return score;

// returns best (highest) score for a symbol
Gesture.pointDistanceRank = function (gesture1, gesture2, canvas1, canvas2) {
    var bagOfPoints1 = gesture1.bagOfPoints();
    var bagOfPoints2 = gesture2.bagOfPoints();

    if (DEBUG_LEVEL > 2) {
        console.log("gesture1, gesture2", gesture1, gesture2);
        Gesture.visualDistanceNearest(gesture1, gesture2, bagOfPoints1, bagOfPoints2);
    }

    var totalDistance = 0;
    // | \rightarrow 2
    for (var i = 0; i < bagOfPoints1.length; i++) {
        totalDistance += Gesture.getNearestDistance(bagOfPoints1[i], bagOfPoints2);
    }

    // | \rightarrow 1
    for (i = 0; i < bagOfPoints2.length; i++) {
```
totalDistance += Gesture.getNearestDistance(bagOfPoints2[i],
    bagOfPoints1);
}

var score = 1 / (totalDistance / (bagOfPoints1.length + bagOfPoints2.length));
return score;

Gestures.visualDistance = function (gesture1, gesture2, bagOfPoints1, bagOfPoints2)
{
    var canvas1 = gesture1.drawGestureOnNewCanvas(MAIN_CONTAINER_ID, gesture1.
symbol);
    var context1 = canvas1.getContext("2d");
    var color1 = Utils.randColor();
    context1.fillStyle = color1;
    context1.strokeStyle = color1;
    context1.lineWidth = 3;

    for (i = 0; i < bagOfPoints1.length; i++)
    {
        context1.fillText("x", bagOfPoints1[i].x, bagOfPoints1[i].y);
        context1.beginPath();
        context1.moveTo(bagOfPoints1[i].x, bagOfPoints1[i].y);
        if (bagOfPoints2[i])
        {
            context1.lineTo(bagOfPoints2[i].x, bagOfPoints2[i].y);
        }
        context1.closePath();
        context1.stroke();
    }

    var color2 = Utils.randColor();
    context1.fillStyle = color2;
    context1.strokeStyle = color2;
    Gesture.drawOnCanvas(canvas1, gesture2);

    // 2 -> 1
    for (i = 0; i < bagOfPoints2.length; i++)
    {
        context1.fillText("x", bagOfPoints2[i].x, bagOfPoints2[i].y);
        context1.beginPath();
        context1.moveTo(bagOfPoints2[i].x, bagOfPoints2[i].y);
        context1.closePath();
        context1.stroke();
    }

    Gesture.visualDistanceNearest = function (gesture1, gesture2, bagOfPoints1,
    bagOfPoints2)
```javascript
781 {
782     // 1 -> 2
783     var canvas1 = gesture1.drawGestureOnNewCanvas(MAIN_CONTAINER_ID, gesture1.
784     symbol);
785     var context1 = canvas1.getContext("2d");
786     var color1 = Utils.randColor();
787     context1.fillStyle = color1;
788     context1.strokeStyle = color1;
789     context1.lineWidth = 3;
790
791     // 1 -> 2
792     for (i = 0; i < bagOfPoints1.length; i++)
793     {
794         context1.fillText("x", bagOfPoints1[i].x, bagOfPoints1[i].y);
795         context1.beginPath();
796         context1.moveTo(bagOfPoints1[i].x, bagOfPoints1[i].y);
797         if (bagOfPoints2[i])
798             var toPoint = Point.findNearestPoint(bagOfPoints1[i], bagOfPoints2
799                     );
800                     context1.lineTo(toPoint.x, toPoint.y);
801                     context1.closePath();
802                     context1.stroke();
803     }
804
805     var color2 = Utils.randColor();
806     context1.fillStyle = color2;
807     context1.strokeStyle = color2;
808     Gesture.drawOnCanvas(canvas1, gesture2);
809
810     // 2 -> 1
811     for (i = 0; i < bagOfPoints2.length; i++)
812     {
813         context1.fillText("x", bagOfPoints2[i].x, bagOfPoints2[i].y);
814         context1.beginPath();
815         context1.moveTo(bagOfPoints2[i].x, bagOfPoints2[i].y);
816         var toPoint = Point.findNearestPoint(bagOfPoints2[i], bagOfPoints1);
817         context1.lineTo(toPoint.x, toPoint.y);
818         context1.closePath();
819         context1.stroke();
820     }
821
822 }
823
824
825
826 Gesture.dotProduct = function (points1, points2)
827 {
828     if (points1.length != points2.length)
829     {
830         throw new Error("length mismatch while trying to calculate dot product
831             " + points1.length + " " + points2.length);
832     }
32```
```javascript
var dotProduct = 0;

for (var i = 0; i < points1.length; i++)
{
    var point1 = points1[i];
    var point2 = points2[i];
    dotProduct += point1.x * point2.x + point1.y * point2.y;
}
return dotProduct;

Gesture.drawAndScaleToCanvas = function(canvas, gesture)
{
    gesture.getBoundingBox();
    // clear canvas
    Utils.clearCanvas(canvas);
    Gesture.drawOnCanvas(canvas, gesture);
    // crop and scale
    var context = canvas.getContext('2d');
    var border = 8;
    var imageData = context.getImageData(gesture.topLeft.x - border,
                                          gesture.topLeft.y - border,
                                          gesture.width + border,
                                          gesture.height + border);
    var tempCanvas = document.createElement('canvas');
    tempCanvas.setAttribute('id', new Date().getTime());
    tempCanvas.setAttribute('width', imageData.width);
    tempCanvas.setAttribute('height', imageData.height);
    tempCanvas.getContext('2d').putImageData(imageData, 0, 0);
    Utils.clearCanvas(canvas);
    context.drawImage(tempCanvas, 0, 0, canvas.width, canvas.height);
}

Gesture.sortPointsByX = function(obj1, obj2)
{
    if (obj1.x > obj2.x)
    {
        return 1;
    }
    else if (obj1.x < obj2.x)
    {
        return -1;
    }
    return 0;
}

Gesture.sortPointsByY = function(obj1, obj2)
{
    if (obj1.y > obj2.y)
    {
        return 1;
    }
```

else if (obj1.y < obj2.y)
{
    return -1;
}

return 0;
}

Gesture.getNearestDistance = function(point, points)
{
    return Point.distance(Point.findNearestPoint(point, points), point);
}

// Maximum strokes allowed in a single symbol
Gesture.MAX_STROKES_PER_SYMBOL = 4;

// Maximum strokes allowed
Gesture.MAX_STROKES = 128;

../Feature.js

/*
 * small class to represent a classification feature
 */
function Feature(type, score, weight)
{
    this.type = type;
    this.score = score;
    this.weight = weight;

    this.getWeightedScore = function()
    {
        return this.score * this.weight;
    }
}

Feature.POINT_DISTANCE_TYPE = "pointDistanceType";
Feature.POINT_DISTANCE_WEIGHT = 0.8;

Feature ANGLE_DIFFERENCE_TYPE = "angleDifferenceType";
Feature ANGLE_DIFFERENCE_WEIGHT = 0.1;

// unused
Feature LANGUAGE_BIGRAM_TYPE = "languageBigram";
Feature LANGUAGE_BIGRAM_WEIGHT = 0.05;

../Match.js

/*
 * a match acts as a simple classification broker
 */
function Match(gesture)
{
    this.gesture = gesture;
}
```javascript
this.features = [];

// returns a unified score
this.getScore = function() {
    // weight and sum up all scores
    var totalScore = 0;
    for (var i = 0; i < this.features.length; i++) {
        totalScore += this.features[i].getWeightedScore();
    }
    if (DEBUG_LEVEL > 2) {
        console.log("totalScore");
    }
    return totalScore;
}

var self = this;

this.clone = function() {
    return new Match(self.gesture.clone(), self.score);
}

this.getFeatureByType = function(featureType) {
    for (var i = 0; i < this.features.length; i++) {
        if (this.features[i].type == featureType) {
            return this.features[i];
        }
    }
}

Match.cloneMatches = function(matchesArray) {
    var ma = [];
    for (var i = 0; i < matchesArray.length; i++) {
        ma.push(matchesArray[i].clone());
    }
    return ma;
}

Match.sortDescending = function(a, b) {
    if (DEBUG_LEVEL > 3) {
        console.log("sortDescending");
    }
}
```

function GestureClassifier(gestureLoadedCallback, staffSize, staffOffsetPoint) {
    var self = this;
    if (DEBUG_LEVEL > 0) {
        var GESTURE_DATA_FILENAME = "gestures_debug.json";
    } else {
        var GESTURE_DATA_FILENAME = "gestures.json";
    }
    var NORMALIZED_STAFF_SIZE = 128;
    var fixedPoint = new Point(150, 170);
    var _trainedGestures = [];
    //read gestures from json file
    var json = new Json();
    json.fromJson(JSON.stringify(GESTURE_DATA_FILENAME));
    json.loadedCallback = handleLoadedCallback;
    //canvas for the actual drawing of notation
    var _gestureLoadedCallback = gestureLoadedCallback;
    _canvasDraw = null;
    this.staffOffsetPoint = staffOffsetPoint;
    //language model
    this.language = new Language();

    function handleLoadedCallback() {
        var rawGestures = json.data;
        for (var i = 0; i < rawGestures.length; i++) {
            rg = rawGestures[i];
            var g = new Gesture();
            g.createFromJson(rg);
            _trainedGestures.push(g);
        }
        //normalize trained gestures to fixed size
        for (var i = 0; i < _trainedGestures.length; i++) {
            if ("staffSize" in _trainedGestures[i] && _trainedGestures[i].staffSize != NORMALIZED_STAFF_SIZE)
            {
            }
        }
    }
}
```javascript
    _trained Gestures[i].scale(NORMALIZED_STAFF_SIZE / 
    _trained Gestures[i].staffSize);
    }
    _trained Gestures[i].normalizeToFixedPoint(fixedPoint);
    _trained Gestures[i].normalize();
    }
    }
    _canvasDraw = new CanvasDraw(false, staffSize, staffOffsetPoint);
    if (_gestureLoadedCallback)
    {
        _gestureLoadedCallback(_canvasDraw);
    }
    }
    this.getCanvasDraw = function()
    {
        return _canvasDraw;
    }

    //gestureStaffSize is the staff size, in points
    this.match = function (drawnGesture, gestureStaffSize, previousMatch)
    {
        var canvas1 = document.getElementById("canvas1");
        var canvas2 = document.getElementById("canvas2");
        //scale drawn gesture to match scale of training templates
        var matchList = [];
        for (var i = 0; i < _trained Gestures.length; i++)
        {
            if (_trained Gestures[i].strokes.length <= 0)
            {
                continue;
            }
            var trained Gesture = _trained Gestures[i];

            //match drawn gesture scale to trained gesture scale
            //deep copy, so we can rescale each time
            var drawnGestureScaled = drawnGesture.clone();
            drawnGestureScaled.normalize();
            if (gestureStaffSize != NORMALIZED_STAFF_SIZE)
            {
                var scaleFactor = NORMALIZED_STAFF_SIZE / gestureStaffSize;
                drawnGestureScaled.scale(scaleFactor);
                if (DEBUG_LEVEL > 1)
                {
                    console.log(gestureStaffSize, NORMALIZED_STAFF_SIZE, ", scaleFactor", scaleFactor);
                }
            }
            drawnGestureScaled.normalizeToFixedPoint(fixedPoint, true);
```
var match = new Match(trainedGesture);

var feature = new Feature(Feature.POINT_DISTANCE_TYPE,
  Gesture.pointDistanceRank(trainedGesture,
    drawnGestureScaled),
  Feature.POINT_DISTANCE_WEIGHT);
match.features.push(feature);

var bagOfPoints1 = trainedGesture.bagOfPoints();
var bagOfPoints2 = drawnGestureScaled.bagOfPoints();

// invert, so high number is a good score (closer angles)
// and fudge a little so number is smaller
var pointAngleScore = (1 / Point.angleDifferences(bagOfPoints1,
  bagOfPoints2)) / 1000;

var feature = new Feature(Feature.ANGLE_DIFFERENCE_TYPE,
  pointAngleScore,
  Feature.ANGLE_DIFFERENCE_WEIGHT);
match.features.push(feature);

// language model, bigram from previous symbol to this symbol
if (previousMatch)
  {
    var feature = new Feature(Feature.LANGUAGE_BIGRAM_TYPE,
      self.language.bigramVector.
        getBigramFromTableBySymbolNames(
          previousMatch.gesture.symbol,
          trainedGesture.symbol).
        getProbability(),
      Feature.LANGUAGE_BIGRAM_WEIGHT);
    match.features.push(feature);
  }
if (DEBUG_LEVEL > 1)
  {
    var newCanvas = drawnGestureScaled.drawGestureOnNewCanvas(
      MAIN_CONTAINER_ID);
    matchList.push(match);
  }

  // descending
  matchList.sort(Match.sortDescending);
  return matchList;

this.selfMatch = function ()
```javascript
{  
  var inputGESTURES = json.data;

  for (var i = 0; i < inputGESTURES.length; i++)
  {
    var g = new Gesture();
    g.createFromJson(inputGESTURES[i]);
    if (g.staffSize != NORMALIZED_STAFF_SIZE)
    {
      var scaleFactor = NORMALIZED_STAFF_SIZE / g.staffSize;
      g.scale(scaleFactor);
    }
    g.normalizeToFixedPoint(fixedPoint);
    g.normalize();

    if (_trainedGESTURES[i].strokes.length <= 0)
    {
      continue;
    }
    var trainedGesture = _trainedGESTURES[i];
    var newCanvas = trainedGesture.drawGestureOnNewCanvas(MAIN_CONTAINER_ID);
    Gesture.drawOnCanvas(newCanvas, g, "rgba(100, 0, 0, 1.0)");
  }

  this.dynamicMatch = function (drawnGesture, gestureStaffSize)
  {
    var numSegements = drawnGesture.strokes.length - 1;
    var numStrokes = drawnGesture.strokes.length;
    var scores = [];
    var p = [];
    var q = [];
    var names = [];
    var maxStrokes = Math.min(Gesture.MAX_STROKES, numStrokes);
    var bestSymbols = [];

    // scale drawn gesture to match scale of training templates
    if (gestureStaffSize != NORMALIZED_STAFF_SIZE)
    {
      var scaleFactor = NORMALIZED_STAFF_SIZE / gestureStaffSize;
      drawnGesture.scale(scaleFactor);
    }
    drawnGesture.normalizeToFixedPoint(fixedPoint, false);
    drawnGesture.normalize();

    var historicalScores = [];
    var counts = [];
    var bestSymbols = [];
    var bestGesture;
    var bestScore = 0;
    for (var i = 0; i < numStrokes; i++)
    {
      
    
```

```
    historicalScores[i] = 0;
    counts[i] = 0;
  }
  for (var i = 0; i < numStrokes; i++)
  {
    historicalScores[i] = 0;
    // number of strokes to look back
    var maxStrokes = Math.min(Gesture.MAX_STROKES_PER_SYMBOL, i);
    for (var j = 0; j < maxStrokes; j++)
    {
      debugString = "i: " + i + " j: " + j + " " + (i - j);
      var chunk = new Gesture();
      chunk.strokes = drawnGesture.cloneStrokes(i - j - 1, i);
      debugString += " drawnGesture.strokes.length "+drawnGesture.
        strokes.length + " " + (i - j - 1) + " " + i;
      chunk.normalizeToFixedPoint(fixedPoint);
      chunk.normalize();
      bestScore = 0;
      bestGesture = new Gesture();
      // all gesture templates
      for (var k = 0; k < _trainedGestures.length; k++)
      {
        var trainedGesture = _trainedGestures[k];
        var score = Gesture.pointDistanceRank(trainedGesture, chunk);
        if (DEBUG_LEVEL > 1)
        {
          trainedGesture.drawGestureOnNewCanvas(
            MAIN_CONTAINER_ID, debugString);
          chunk.drawGestureOnNewCanvas(MAIN_CONTAINER_ID, debugString);
        }
        if (score > bestScore)
        {
          bestScore = score;
          bestGesture = trainedGesture;
        }
        debugString += " " + bestGesture.symbol;
      }
      var historicalScore = Math.pow((Math.pow(historicalScores[i - j],
        counts[i - j]) * bestScore),
        1.0/(counts[i - j] + 1));
      if (historicalScore > historicalScores[i])
      {
        historicalScores[i] = historicalScore;
        counts[i] = counts[i - j] + 1.0;
        bestSymbols[i] = j;
      }
    }
  for (var zz = 0; zz < bestSymbols.length; zz++)
  {  
```

var symbolIdx = bestSymbols[zz];
}

return;

var b = [];
var c = [];
var bestM = [];
for (var n = 0; n < numStrokes - 1; n++)
{
    b[n] = 0;
    var bestScore = 0;
    var bestM;
    var bestL;
    var bestName;
    var maxStrokes = Math.min(Gesture.MAX_STROKES, numStrokes);
    for (var m = 1; m < maxStrokes; m++)
    {
        var chunk = new Gesture();

        if (n - m + 1 < 0)
        {
            continue;
        }

        if (drawnGesture.strokes <= 0)
        {
            continue;
        }

        chunk.strokes = drawnGesture.strokes.slice(n - m + 1, n);
        chunk.normalizeToFixedPoint(fixedPoint, true);
        chunk.normalize();

        if (chunk.strokes.length <= 0)
        {
            continue;
        }

        // all models
        for (var j = 0; j < _trainedGestures.length; j++)
        {
            if (_trainedGestures[j].strokes.length <= 0)
            {
                continue;
            }

            var trainedGesture = _trainedGestures[j];

            if (DEBUG_LEVEL > 0)
            {
                console.log("trainedGesture", trainedGesture, "chunk", chunk);
            }

            var x = Gesture.pointDistanceRank(trainedGesture, chunk);
            var l = trainedGesture;
            if (x > bestScore)
{  
    bestScore = x;
    bestL = 1;
    bestName = trainedGesture.symbol;
}

var historicalScore = Math.pow(Math.pow(b[n - m], c[n - m]) * bestScore, 1.0/(c[n - m] + 1.0));

if (historicalScore > b[n]) {
    b[n] = historicalScore;
    c[n] = c[n - m] + 1.0;
    bestM[n] = m;
}

var debugString = "n= " + n + " score= " + bestScore +" symbol= " + bestName;

} // segmentation step...

// determine if a collection of strokes form a symbol
this.dynamicMatchHull = function (drawnGesture, canvas1, canvas2, gestureStaffSize, staff, context, pencil) {
	numSegements = drawnGesture.strokes.length - 1;
	var strokePolygons = [];
	srokeGestures = [];
	sroke = [];

// 1. construct an array of polygons, one for each stroke
for (var i = 0; i < numSegements; i++) {

    // determine convex hull of each stroke. If hull overlaps with previous stroke, group it will that one.
    var strokePolygon = new Polygon();
    // deep copy
    var strokePoints = drawnGesture.strokes[i].slice(0);
    strokePolygon.points = strokePoints;
    var debugString = "";
    strokePolygon.sortPoints();

    // get simple convex polygon of stroke
    strokePolygon.chainHull();
    var strokeGestures = new Gesture();

    // deep copy of unadulterated points
    strokeGestures.strokes[0] = Point.clonePoints(drawnGesture.strokes[i]);

    strokePolygons.push(strokePolygon);
}
strokeGestures.push(strokeGesture);

if (DEBUG_LEVEL > 2)
{
    var newCanvas = strokeGesture.drawGestureOnNewCanvas(MAIN_CONTAINER_ID);
}

// 2. loop through all polygons and find overlapping polygons
// 3. group all overlapping polygons as strokes of a single gesture
for (var i = 0; i < strokePolygons.length; i++)
{
    var strokePolygon2 = strokePolygons[i];
    var strokeGesture2 = strokeGestures[i];

    // determine if this stroke polygon overlaps any previous stroke polygon
    var haveOverlap = false;
    for (var j = 0; j < i; j++)
    {
        var strokePolygon1 = strokePolygons[j];
        var strokeGesture1 = strokeGestures[j];
        var pi = new PolygonIntersect();
        // if there’s any intersection, group together

        // deep copy points to leave originals unscaled
        var points1 = strokePolygon1.points.slice(0);
        var points2 = strokePolygon2.points.slice(0);

        // scale up a little to decrease gap/overlap tolerance
        var scaleFactor = 1.005;
        Point.scalePoints(points1, scaleFactor);
        Point.scalePoints(points2, scaleFactor);

        if (pi.intersection(points1, points2))
        {
            // append points from strokePolygon2 as a new stroke of gesture1
            strokeGesture1.strokes.push(strokeGesture2.strokes[0]);
            // append points from strokePolygon2 to strokePolygon1
            strokePolygon1.points = strokePolygon1.points.concat(strokePolygon2.points);

            // append strokePolygon2 points to gesture
            // remove strokeGesture2 from the stack
            strokeGestures.splice(i, 1);
            // ...and remove strokePolygon2 from the stack
            strokePolygons.splice(i, 1);

            i -= 2;
        }
    }
}
var previousMatch = null;

for (var i = 0; i < strokeGestures.length; i++) {
    var strokeGesture = strokeGestures[i];
    // append segmented, but otherwise unmodified strokes to Ink object
    // mainly useful for UI stuff

    // clone gesture
    var inkStrokeGesture = strokeGesture.clone();

    if (DEBUG_LEVEL > 0) {
        console.log("gestureStaffSize", gestureStaffSize);
        console.log("strokeGesture", strokeGesture);
    }
    var centroid = strokeGesture.getAverageCenter();
    var topLeft = strokeGesture.getTopLeft();
    var textAlign = new Point(topLeft.x, staff.snapToLineY(centroid.y));

    // template match
    var matchList = this.match(strokeGesture, gestureStaffSize, previousMatch);

    inkStrokeGesture.matches = inkStrokeGesture.matches.concat(matchList);
    pencil.ink.gestures.push(inkStrokeGesture);

    var bestSymbol = matchList[0].gesture.symbol;
    var bestGesture = matchList[0].gesture;

    debugString = i + bestSymbol;
    var symbol = new Symbol(bestSymbol);
    symbol.position = textAlign;
    staff.symbols.push(symbol);
    pencil.redrawCanvas();

    previousMatch = matchList[0];
}

return;

this.dynamicMatchBruteForce = function (drawnGesture, canvas1, canvas2, gestureStaffSize, staff, context, pencil) {
    var numSegements = drawnGesture.strokes.length - 1;
    var strokePolygons = [];
    var strokeGestures = [];
    var strokes = [];

    var x = 0;
// brute force
// 1. construct an array of polygons, one for each and every combination of strokes
var numStrokes = drawnGesture.strokes.length;
var historicalScores = [];
var counts = [];
var bestSymbols = [];
var bestGesture;
var bestScore = 0;

// scale drawn gesture to normalized scale
if ("staffSize" in drawnGesture && drawnGesture.staffSize != NORMALIZED_STAFF_SIZE)
{
drawnGesture.scale(NORMALIZED_STAFF_SIZE / drawnGesture.staffSize);
}

for (var i = 0; i < numStrokes; i++)
{
historicalScores[i] = 0;
counts[i] = 0;
}

for (var i = 0; i < numStrokes; i++)
{
historicalScores[i] = 0;

// number of strokes to look back
var maxStrokes = Math.min(Gesture.MAX_STROKES_PER_SYMBOL, i);
for (var j = 0; j < maxStrokes; j++)
{

debugString = "i : " + i + " j : " + j + " i - j : " + (i - j);
var chunk = new Gesture();
chunk.staffSize
chunk.strokes = drawnGesture.cloneStrokes(i - j - 1, i);
var centroid = chunk.getAverageCenter(true);
debugString += " drawnGesture.strokes.length " + drawnGesture.strokes.length + " + drawnGesture.strokes.length + " + (i - j - 1) + " + i;
chunk.normalizeToFixedPoint(fixedPoint, true);
chunk.normalize();
bestScore = 0;
bestGesture = new Gesture();
// all gesture templates
for (var k = 0; k < _trainedGestures.length; k++)
{
var trainedGesture = _trainedGestures[k];
var score = Gesture.pointDistanceRank(trainedGesture, chunk);
if (DEBUG_LEVEL > 1)
{
var newCanvas = trainedGesture.drawGestureOnNewCanvas(MAIN_CONTAINER_ID, debugString);
Gesture.drawOnCanvas(newCanvas, chunk, "rgba(100, 0, 0, 1.0)");
}
```javascript
if (score > bestScore)
{
    bestScore = score;
    bestGesture = trainedGesture;
}

destring += " " + bestGesture.symbol;
}
var historicalScore = Math.pow((Math.pow(historicalScores[i - j],
counts[i - j]) * bestScore), 1.0/(counts[i - j] + 1));
if (historicalScore > historicalScores[i])
{
    historicalScores[i] = historicalScore;
    counts[i] = counts[i - j] + 1.0;
    bestSymbols[i] = _trainedGestures[j].symbol;
}
}
console.log(bestSymbols, counts, historicalScores);
return;
}

this.retrainGesture = function(replacementName, replacementGesture)
{
    // alert("retrainGesture", replacementName, replacementGesture);
    for (var i = 0; i < _trainedGestures.length; i++)
    {
        if (_trainedGestures[i].symbol == replacementName)
        {
            _trainedGestures[i].staffSize = replacementGesture.staffSize;
            _trainedGestures[i].strokes = replacementGesture.strokes;

            // normalize
            if ("staffSize" in _trainedGestures[i] && _trainedGestures[i].
                staffSize !== NORMALIZED_STAFF_SIZE)
            {
                _trainedGestures[i].scale(NORMALIZED_STAFF_SIZE /
                _trainedGestures[i].staffSize);
            }
            _trainedGestures[i].normalizeToFixedPoint(fixedPoint, false);
            _trainedGestures[i].normalize();
        }
    }
}

this.setScaleTest = function(drawnGesture, gestureStaffSize)
{
    var scaleTestGesture;
    for (var i = 0; i < _trainedGestures.length; i++)
    {
        if (_trainedGestures[i].symbol == "scale_test")
```
```javascript
549  }
550  
551  }
552  
553  
554  
555  
556  
557  var scaleFactor = NORMALIZED_STAFF_SIZE / gestureStaffSize;
558  drawnGesture.scale(scaleFactor);
559  console.log(gestureStaffSize, NORMALIZED_STAFF_SIZE, "scaleFactor",
560                    scaleFactor);
561  
562  // offset by staff margin
563  drawnGesture.normalizeToPoint(fixedPoint);
564  drawnGesture.normalize();
565  
566  var newCanvas = scaleTestGesture.drawGestureOnNewCanvas(
567                MAIN_CONTAINER_ID);
568  Gesture.drawOnCanvas(newCanvas, drawnGesture, "rgba(100, 0, 0, 1.0)" );
569  
570  }

../SymbolOptions.js

1 /*
2  * represents and renders a user-selectable ordered list of symbol
3  * classification corrections
4 */
5 function SymbolOptions(gesture, canvas, currentSymbolIndex, mainStaff, pencil)
6 {
7    const DEFAULT_LENGTH = 12;
8    const DEFAULT_ITEM_HEIGHT = 30;
9    const DEFAULT_SPACING_X = 10;
10   const DEFAULT_SPACING_Y = 20;
11   const HIGHLIGHTED_INK_COLOR = "rgba(100, 0, 0, 1.0)";
12   const DEFAULT_INK_COLOR = "rgba(0, 0, 0, 1.0)";
13   
14   this.length = DEFAULT_LENGTH;
15   this.gesture = gesture;
16   var self = this;
17   var _canvas = canvas;
18   this.disableAllEvents;
19   this.targetAreas = [];
20   this.selectedSymbolIndex = -1;
21   this.currentSymbolIndex = currentSymbolIndex;
22   this.mainStaff = mainStaff;
23   this.pencil = pencil;
24   
25   this.init = function() {
26     _canvas.addEventListener("mousemove", symbolOptionsEventCanvas, false)
47
```
```javascript
this.render = function () {
  var matchList = Match.cloneMatches(self.gesture.matches);
  var ctx = _canvas.getContext("2d");
  var offsetX = DEFAULT_SPACING_X;
  var offsetY = DEFAULT_SPACING_Y;
  Util.clearCanvas(_canvas);
  var staff = new Staff();
  // iterate through options
  var targetArea = null;
  for (var i = 0; i < self.length; i++) {
    var offsetPoint = new Point(offsetX, offsetY);
    highlightSymbolIndex = 2;
    targetArea = new Symbol(matchList[i].gesture.symbol)
      .getBoundingBox(ctx, staff);
    targetArea.moveTo(new Point(offsetX, 0));
    if (self.selectedSymbolIndex == i) {
      staff.drawSymbol(ctx, offsetPoint, matchList[i].gesture.symbol
        , HIGHLIGHTED_INK_COLOR);
      Polygon.drawOnCanvas(_canvas, targetArea.getPolygon(),
                                HIGHLIGHTED_INK_COLOR);
    } else {
      staff.drawSymbol(ctx, offsetPoint, matchList[i].gesture.symbol
        , DEFAULT_INK_COLOR);
    }
    offsetX += targetArea.getWidth() + DEFAULT_SPACING_X;
    self.targetAreas.push(targetArea);
  }

  function symbolOptionsEventCanvas(event) {
    if (self.disableAllEvents)
      return;
    UtilProcessEvent(event);
    // Call the event handler
    var func = self[event.type];
    if (func)
      func(event);
```
this.mousemove = function(event)
{
    if (DEBUG_LEVEL > 2)
    {
        console.log("mouseover");
    }
    var mouseMovePoint = new Point(event.x, event.y);
    self.highlightSymbol(mouseMovePoint);
}

this.mousedown = function(event)
{
    var mouseDownPoint = new Point(event.x, event.y);
    _canvas.style.visibility = "hidden";
    self.disableAllEvents = true;
    self.selectSymbol();
}

this.mouseup = function(event)
{
    _canvas.style.visibility = "hidden";
    self.disableAllEvents = true;
    self.selectSymbol();
}

this.mouseout = function(event)
{
    _canvas.style.visibility = "hidden";
    self.disableAllEvents = false;
}

this.highlightSymbol = function(point)
{
    for (var i = 0; i < self.targetAreas.length; i++)
    {
        var targetArea = self.targetAreas[i];
        if (targetArea.contains(point))
        {
            // highlight target symbol
            self.selectedSymbolIndex = i;
            self.render();
            return;
        }
    }
}

this.selectSymbol = function()
{
    if (self.selectedSymbolIndex > 0)
    {
        if (DEBUG_LEVEL > 2)
        {
            console.log("self.selectedSymbolIndex", self.selectedSymbolIndex);
        }
    }
// give this symbol a score of best score + 1
var pointDistanceFeature = self.gesture.matches[ self.
    selectedSymbolIndex ].getFeatureByType( Feature.
    POINT_DISTANCE_TYPE);
pointDistanceFeature.score += 1.0;

// ... and re-sort symbol list
self.gesture.matches.sort( Match.sortDescending);

// replace ...
var replacementSymbol = self.mainStaff.symbols[ self.
    currentSymbolIndex ].clone();
replacementSymbol.name = self.gesture.matches[0].gesture.symbol;
self.mainStaff.symbols.splice( self.currentSymbolIndex, 1,
    replacementSymbol);
self.pencil.redrawCanvas();

// crude implementation of auto-learning
if ( useAutoLearn() )
{
    if (DEBUG_LEVEL > 2)
    {
        console.log( self.mainStaff.symbols.length + " " + self.
            currentSymbolIndex + replacementSymbol.name);
    }
    gestureClassifier.retrainGesture( replacementSymbol.name, this.
        pencil.ink.gestures[ self.currentSymbolIndex ].clone());
}
this.init();

10.2 Language Model

../common/Language.js

/*
 * Language model
 */

function Language ()
{
    // hardwire a rough set of bigrams
    this.bigramVector = new BigramVector();
    bv = this.bigramVector;
    bv.push( new Bigram("clef", "clef", 0.0));
    bv.push( new Bigram("clef", "rest", 1.0));
    bv.push( new Bigram("clef", "note", 1.0));
    bv.push( new Bigram("clef", "accidental", 1.0));
    bv.push( new Bigram("clef", "line", 0.5));
bv.push(new Bigram("clef", "augmentation", 0.0));
bv.push(new Bigram("clef", "time_signature", 1.0));
bv.push(new Bigram("rest", "clef", 0.1));
bv.push(new Bigram("rest", "rest", 1.0));
bv.push(new Bigram("rest", "note", 1.0));
bv.push(new Bigram("rest", "accidental", 1.0));
bv.push(new Bigram("rest", "line", 1.0));
bv.push(new Bigram("rest", "augmentation", 1.0));
bv.push(new Bigram("rest", "time_signature", 0.0));
bv.push(new Bigram("note", "clef", 0.01));
bv.push(new Bigram("note", "rest", 1.0));
bv.push(new Bigram("note", "note", 1.0));
bv.push(new Bigram("note", "accidental", 1.0));
bv.push(new Bigram("note", "line", 1.0));
bv.push(new Bigram("note", "augmentation", 1.0));
bv.push(new Bigram("note", "time_signature", 0.0));
bv.push(new Bigram("accidental", "clef", 0.01));
bv.push(new Bigram("accidental", "rest", 1.0));
bv.push(new Bigram("accidental", "note", 1.0));
bv.push(new Bigram("accidental", "accidental", 1.0));
bv.push(new Bigram("accidental", "line", 1.0));
bv.push(new Bigram("accidental", "augmentation", 1.0));
bv.push(new Bigram("accidental", "time_signature", 0.0));
bv.push(new Bigram("time_signature", "clef", 1.0));
bv.push(new Bigram("time_signature", "rest", 1.0));
bv.push(new Bigram("time_signature", "note", 1.0));
bv.push(new Bigram("time_signature", "accidental", 1.0));
bv.push(new Bigram("time_signature", "line", 1.0));
bv.push(new Bigram("time_signature", "augmentation", 1.0));
bv.push(new Bigram("time_signature", "time_signature", 1.0));
bv.push(new Bigram("line", "clef", 1.0));
bv.push(new Bigram("line", "rest", 1.0));
bv.push(new Bigram("line", "note", 1.0));
bv.push(new Bigram("line", "accidental", 1.0));
bv.push(new Bigram("line", "line", 0.0));
bv.push(new Bigram("line", "augmentation", 0.0));
bv.push(new Bigram("line", "time_signature", 1.0));
bv.push(new Bigram("augmentation", "clef", 0.5));
bv.push(new Bigram("augmentation", "rest", 1.0));
bv.push(new Bigram("augmentation", "note", 1.0));
bv.push(new Bigram("augmentation", "accidental", 1.0));
bv.push(new Bigram("augmentation", "line", 1.0));
bv.push(new Bigram("augmentation", "augmentation", 0.6));
bv.push(new Bigram("augmentation", "time_signature", 0.0));
```javascript
//common/Ngram.js

function Ngram() {
    // regular expressions
    this.from = [];
    // regular expressions
    this.to = [];

    this.probabilities = [];
}

//common/Bigram.js

function Bigram(from, to, probability) {
    var ngram = new Ngram();
    ngram.from = [from];
    ngram.to = [to];
    ngram.probabilities = [probability];

    this.getFrom = function() {
        return ngram.from[0];
    }

    this.getTo = function() {
        return ngram.to[0];
    }

    this.getProbability = function() {
        return ngram.probabilities[0];
    }
}

function BigramVector() {
    this.bigrams = [];
    this.bigramsTable = {};
    this.froms = {};
    this.tos = {};
    this.push = function(bigram) {
        this.bigrams.push(bigram);

        // add second dimension object
        if (!(bigram.getFrom() in this.bigramsTable)) {
            this.bigramsTable[bigram.getFrom()] = {};
        }

        if (bigram.getFrom() in this.froms) {
            this.froms[bigram.getFrom()] ++;
        }
        else
```
```javascript
  { this.froms[bigram.getFrom()] = 1; }

  if (bigram.getTo() in this.froms)
  { this.tos[bigram.getTo()] ++;
  }
  else
  { this.tos[bigram.getTo()] = 1;
  }
  
  this.bigramsTable[bigram.getFrom()][bigram.getTo()] = bigram;
}

this.getBigramFromTable = function(from, to)
{
  return this.bigramsTable[from][to];
}

this.getBigramFromTableBySymbolNames = function(fromSymbolName, toSymbolName)
{
  var from = "";
  var to = "";

  if (DEBUG_LEVEL > 1)
  {
    console.log("fromSymbolName, toSymbolName", fromSymbolName, toSymbolName);
  }
  
  // check for stop symbols
  if (Bigram.stopSymbolNames.indexOf(fromSymbolName) >= 0 || Bigram.stopSymbolNames.indexOf(toSymbolName) >= 0)
  {
    return new Bigram("", ",", 0);
  }
  
  // lookup "from" symbol, fuzzy, kind of by notation class name
  for (var fromKey in this.froms)
  {
    if (fromSymbolName.match(fromKey))
    {
      from = fromKey;
    }
  }
  
  // lookup "to" symbol, fuzzy, kind of by notation class name
  for (var toKey in this.froms)
  {
    if (toSymbolName.match(toKey))
    {
      to = toKey;
    }
  }
```
10.3 Drawing

../Ink.js

/*
 * ink represents and serializes user-drawn strokes
 */

function Ink()
{
  // an array of strokes; each stroke is an array of points, pen down to pen
  // up
  this.strokes = [];

  // as strokes are recognized, they are grouped by the recognition engine
  // and
  // pushed onto the gestures stack; each Gesture object has an array of
  // strokes
  this.gestures = [];

  var self = this;

  this.staff;

  // remove gesture and associated original ink strokes
  this.removeGesture = function(gestureIndex)
  {
    self.gestures.splice(gestureIndex, 1);
  }

  this.serializeStrokes = function()
  {
    if (this.strokes.length > 0)
    {
      // splice to prevent dangling extra stroke in serialization
      return Point.serializePointsNew(this.strokes.slice(0, this.strokes
        .length - 1));
    }
  }

  this.serializeGestureStrokes = function()
  {
    var allStrokes = [];

    for (var i = 0; i < this.gestures.length; i++)
    {
      
      Bigram.stopSymbolNames = ["scale_test"];
```javascript
var gesture = this.gestures[i];
for (var j = 0; j < gesture.strokes.length; j++) {
    var stroke = gesture.strokes[j];
    if (stroke.length > 0)
        allStrokes.push(stroke);
}

if (allStrokes.length > 0)
    return Point.serializePointsNew(allStrokes);

this.sortStrokesByLeadingEdge = function() {
    if (DEBUG_LEVEL > 0)
        console.log("this.strokes", this.strokes);
    var leastXs = [];
    //find leading edge
    for (var i = 0; i < this.strokes.length; i++) {
        leastXs[i] = {index: i, x: Number.MAX_VALUE};
        for (var j = 0; j < this.strokes[i].length; j++)
            if (point.x < leastXs[i].x)
                leastXs[i].x = point.x;
    }
    if (DEBUG_LEVEL > 0)
        console.log("leastXs", leastXs);
    leastXs.sort(Ink.sortByXAcending);

    //create a new strokes array in order of leastXs (leading edges)
    var newStrokesAry = [];
    for (var i = 0; i < leastXs.length; i++) {
        newStrokesAry.push(this.strokes[leastXs[i].index]);
    }
    this.strokes = newStrokesAry;
    if (DEBUG_LEVEL > 0)
```
```javascript
this.serialized = function() {
    var gestureStrokes = this.serializeGestureStrokes();
    if (gestureStrokes) {
        return {
            type: "ink",
            staffSize: this.staff.getSize(),
            strokes: gestureStrokes
        };
    }
}
}

this.clearAll = function() {
    this.strokes = [];
    this.gestures = [];
}

Ink.sortByXAscending = function(obj1, obj2) {
    return (obj1.x - obj2.x);
}

./Pencil.js

/*
 * Pencil is the main drawing utility
 */

function Pencil(trainingCallback, context, staff) {
    const COUNTDOWN_INTERVAL = 1.7;       // seconds
    const TOUCH_AND_HOLD_INTERVAL = 1.0;
    const STAFF_EXTENSION_WIDTH = 120;
    const STAFF_EXTENSION_TOLERANCE = 60;
    const DEFAULT_BACKGROUND_LAYER_ALPHA = 0.5;
    const DRAG_THRESHOLD = 5;
    const TOUCH_AND_HOLD_TOLERANCE = 4;
    const DEFAULT_INK_COLOR_RGBA = [0, 0, 0, 0.5];
    this.inkColor = "";
    var self = this;
    var _staffInitialized = false;
    var _showInk = true;
    var _showSymbols = true;
    this.started = false;
    this.drawing = false;
    this.dragging = false;
    this.gesture = new Gesture();
```
this.activeGestureTimeout = null;
this.trainingCallback = trainingCallback;
this.staff = staff;
this.ink = new Ink();
this.ink.staff = staff;
this.autoExtend = true;
this.clickedSymbolIndex = -1;
this.clickPositionOffset = new Point();
this.touchAndHoldTimeout = null;
this.touchAndHoldActive = false;
this.symbolOptionsCanvas = null;

// finger/mouse down
this.touchStart = function (event)
{
  if (!_staffInitialized)
  {
    self.setInkColor(DEFAULT_INK_COLOR_RGBA[0], DEFAULT_INK_COLOR_RGBA [1],
                      DEFAULT_INK_COLOR_RGBA[2], DEFAULT_INK_COLOR_RGBA [3]);
                      staff.startLayout(context);
                      _staffInitialized = true;
  }
  context.beginPath();
  context.moveTo(event._x, event._y);
  self.gesture.startGesture(event._x - staff.marginLeft, event._y -
                           staff.marginTop);
  self.gesture.staffSize = Staff.STAFF_SIZES[self.staff.staffSizeIndex];
  self.started = true;

  // keep track of whether or not user is still drawing by resetting
  // timeout with
  // every new stroke
  if (self.activeGestureTimeout != null)
  {
    clearTimeout(self.activeGestureTimeout);
  }

  // clear touch and hold timer
  if (self.touchAndHoldTimeout != null)
  {
    clearTimeout(self.touchAndHoldTimeout);
  }

  self.mouseDownPoint = new Point(event._x, event._y);
  self.clickedSymbolIndex = self.getSymbolIndexFromPoint(self.
                                 mouseDownPoint.clone());

  if (DEBUG_LEVEL > 2)
  {
    console.log(self.clickedSymbolIndex);
  }
}
74   if (self.clickedSymbolIndex >= 0)
75     {
76       self.clickPositionOffset = staff.symbols[self.clickedSymbolIndex].position.clone().add(self.clickedSymbolIndex);
77       if (DEBUG_LEVEL > 2)
78         {
79           console.log(self.clickPositionOffset.toString);
80         }
81       self.touchAndHoldTimeout = setTimeout(self.
82           handleTouchAndHoldOnSymbol, TOUCH_AND_HOLD_INTERVAL * 1000);
83     }
84 }
85   if (DEBUG_LEVEL > 2)
86     {
87       console.log("self.mouseDownPoint", self.mouseDownPoint);
88     }
89   self.mouseIsDown = true;
90 }
91
92 // alias event handler reference for non-mobile Safari browsers
this.mousedown = this.touchstart;
93
94 // finger/mouse move
this.touchmove = function (event)
95 {
96   self.mouseMovePoint = new Point(event._x, event._y);
97   if (self.touchAndHoldActive)
98     {
99       return;
100     }
101   if (self.started)
102     {
103     var mouseDownToMouseMoveDistance = Point.distance(self.
104       mouseDownPoint, self.mouseMovePoint);
105     if (mouseDownToMouseMoveDistance > TOUCH_AND_HOLD_TOLERANCE)
106       {
107         clearTimeout(self.touchAndHoldTimeout);
108       }
109     // drag gesture
110     if (self.clickedSymbolIndex >= 0)
111       {
112         if (DEBUG_LEVEL > 2)
113           {
114             console.log(staff.symbols[self.clickedSymbolIndex].position.toString());
115           }
116     // change symbol position
117     var position = staff.symbols[self.clickedSymbolIndex].position ;
118
position.x = self.mouseMovePoint.x - self.clickPositionOffset.x;
position.y = staff.snapToLineY(self.mouseMovePoint.y - self.clickPositionOffset.y - staff.marginTop);

// change ink position, only move horizontally
var inkPosition = self.ink.gestures[self.clickedSymbolIndex].getAverageCenter(true);
inkPosition.x = self.mouseMovePoint.x;
selves.ink.gestures[self.clickedSymbolIndex].normalizeToFixedPoint(inkPosition, true);
self.dragging = true;
sself.redrawCanvas();

} // no dragging

if (mouseDownToMouseMoveDistance > DRAG_THRESHOLD) {
    self.drawing = true;
    // (0, 0) is top left corner of staff, so subtract margins
    self.gesture.continueGesture(event._x - staff.marginLeft, event._y - staff.marginTop);
}

} // pencil is not down

if (self.autoExtend && event._x > context.canvas.width - (STAFF_EXTENSION_TOLERANCE)) {
    self.extendCanvas(STAFF_EXTENSION_WIDTH);
}

// check for out of bounds, and end gesture
if (self.mouseMovePoint.x <= 10 || self.mouseMovePoint.y <= 10) {
    self.toucheud(event);
}

// alias event handler reference for non-mobileSafari browsers
this.mousemove = this.touchmove;

// finger/mouse up
this.toucheud = function (event) {
    self.mouseUpPoint = new Point(event._x, event._y);
    self.mousetIsDown = false;
    self.started = false;


// add erase gesture?
if (self.touchAndHoldActive)
{
    self.gesture = new Gesture();
    self.hideSymbolOptions();
    self.touchAndHoldActive = false;
    return;
}
clearTimeout(self.touchAndHoldTimeout);

// click, no drag, delete gesture
if (!self.drawing && !self.dragging)
{
    if (self.clickedSymbolIndex >= 0)
    {
        staff.symbols.splice(self.clickedSymbolIndex, 1);
        self.ink.removeGesture(self.clickedSymbolIndex);
        self.redrawCanvas();
        // reset
        self.clickedSymbolIndex = -1;
    }
    // clear out gesture...
    self.gesture = new Gesture();
}
else if (self.dragging) // dragging an existing symbol
{
    // clear out gesture...
    self.gesture = new Gesture();
    self.dragging = false;
    // reset
    self.clickedSymbolIndex = -1;
}
else // gesture recognition
{
    // kickoff timer for gesture recognition
    self.activeGestureTimeout = setTimeout(self.endGesture, COUNTDOWN_INTERVAL * 1000);
    // add a new stroke
    self.drawing = false;
    self.gesture.newStroke();
    self.dragging = false;
}

} // alias event handler reference for non-mobileSafari browsers
this.mouseup = this.toucheXend;
this.handleTouchAndHoldOnSymbol = function()
var soCanvas = document.getElementById("symbolOptions");
Utils.clearCanvas(soCanvas);
self.touchAndHoldActive = true;
if (DEBUG_LEVEL > 2)
{
    console.log(self.ink.gestures[self.clickedSymbolIndex].matches);
}
var currentSymbolIndex = 0;
var so = new SymbolOptions(self.ink.gestures[self.clickedSymbolIndex],
    soCanvas, self.clickedSymbolIndex, staff, self);
if (DEBUG_LEVEL > 2)
{
    console.log("rendering options " + currentSymbolIndex);
}
// currentSymbolIndex argument unused, unimplemented
so.render(currentSymbolIndex);
soCanvas.style.top = self.mouseDownPoint.y + "px";
soCanvas.style.left = self.mouseDownPoint.x + "px";
soCanvas.style.visibility = "visible";
self.symbolOptionsCanvas = soCanvas;
if (DEBUG_LEVEL > 2)
{
    console.log("touch and hold");
}
this.hideSymbolOptions = function()
{
    self.symbolOptionsCanvas.style.visibility = "hidden";
}
// regular mode, not training
if (!self.trainingCallback)
{
    this.endGesture = function (event)
    {
        // alert (self.activeGestureTimeout);
        if (self.activeGestureTimeout !== null)
        {
            if (self.gesture.endGesture())
            {
                // append gesture to running ink
                for (var i = 0; i < self.gesture.strokes.length; i++)
                {
                    self.ink.strokes.push(Point.clonePoints(self.gesture.
                        strokes[i].slice(0)));
                }
                // clone pencil gesture
            }
            // use centroid
            var drawnRegistrationPoint = self.gesture.getAverageCenter
var canvas1 = document.getElementById("canvas1");
var canvas2 = document.getElementById("canvas2");
if (useScaleTestMode()) // scale test mode
{
gestureClassifier.scaleTest(self.gesture, staff.getSize());
}
else // normal mode
{
switch (CLASSIFICATION_TYPE)
{
    case CLASSIFICATION_TYPE_HULL:
        gestureClassifier.dynamicMatchHull(self.gesture, canvas1, canvas2, staff.getSize(), staff, context, self);
        break;
    case CLASSIFICATION_TYPE_DYNAMIC_1:
        gestureClassifier.dynamicMatchBruteForce(self.gesture, canvas1, canvas2, staff.getSize(), staff, context, self);
        break;
    case CLASSIFICATION_TYPE_DYNAMIC_2:
        gestureClassifier.dynamicMatch(self.gesture, canvas1, canvas2, staff.getSize());
        break;
}
// clear out gesture...
self.gesture = new Gesture();
}

this.extendCanvas = function(width)
{
    context.canvas.width += width;
}
```javascript
self.redrawCanvas();

this.redrawCanvas = function(scaleIndexDelta)
{
    var scaleFactor = 1.0;
    if (scaleIndexDelta)
    {
        scaleFactor = staff.adjustScale(scaleIndexDelta);
    }
    var canvas = context.canvas;
    // clear canvas
    Utils.clearCanvas(canvas);
    // scale canvas
    canvas.width *= scaleFactor;
    canvas.height = staff.heightInPixels() + staff.marginTop + staff.marginBottom;
    staff.render(context, !_showSymbols);
    scaleInk(scaleFactor);
    if (_showInk)
    {
        // var alpha = DEFAULT_BACKGROUND_LAYER_ALPHA;
        staff.startLayout(context, self.getInkColor());
        drawInk();
        staff.endLayout(context);
    }
}

this.toggleShowInk = function()
{
    _showInk = !_showInk;
    self.redrawCanvas();
}

this.toggleShowSymbols = function()
{
    _showSymbols = !_showSymbols;
    self.redrawCanvas();
}

this.setInkColor = function(r, g, b, a)
{
    self.inkColor = "rgba(" + r + "," + g + "," + b + "," + a + ");";
    self.redrawCanvas();
}

this.getInkColor = function()
{
    return self.inkColor;
}

this.resizeToNearestStaffSize = function(staffSize)
{
    var nearestIndex = staff.getNearestSizeIndex(staffSize);
```
```javascript
var scaleIndexDelta = nearestIndex - staff.staffSizeIndex;
self.redrawCanvas(scaleIndexDelta);

this.getSymbolIndexFromPoint = function(point)
{
    // construct a convex polygon from gesture ink and symbol
    for (var i = 0; i < staff.symbols.length; i++)
    {
        var targetPolygon = new Polygon();

        // start with symbol bounding box
        targetPolygon.points = staff.symbols[i].getBoundingBox(context, staff).getPoints();

        // concatenate all ink points for this gesture
        targetPolygon.points = targetPolygon.points.concat(self.ink.gestures[i].bagOfPoints());

        // get convex hull of all points
        targetPolygon.sortPoints();
targetPolygon.chainHull();

        // determine if point is inside polygon
        if (targetPolygon.pointInPolygon(point) == Polygon.INSIDE_POLYGON)
        {
            return i;
        }
    }
    return -1;
}

this.clearAll = function()
{
    self.gesture = new Gesture();
    self.ink.clearAll();
    self.staff.clearAll();
selself.redrawCanvas();
}

function scaleInk(scaleFactor)
{
    if (!scaleFactor)
    {
        return;
    }
    for (var i = 0; i < self.ink.strokes.length; i++)
    {
        // staff.marginTop
        Point.scalePoints(self.ink.strokes[i], scaleFactor);
    }

    // scale gestures, and their strokes
```
for (var i = 0; i < self.ink.gestures.length; i++)
{
    self.ink.gestures[i].scale(scaleFactor);
}

// redraw existing ink
this.drawInkFromStrokes = function(strokes, staffSize, marginOffset)
{
    var strokePoints = [];
    for (var i = 0; i < strokes.length; i++)
    {
        strokePoints[i] = [];
        for (var j = 0; j < strokes[i].length; j++)
        {
            var point = new Point(strokes[i][j].x, strokes[i][j].y);
            point.x -= marginOffset.x;
            point.y -= marginOffset.y;
            strokePoints[i][j] = point;
            if (staffSize != this.staff.getSize())
            {
                point.scale(this.staff.getSize() / staffSize);
            }
            if (j == 0)
            {
                context.moveTo(point.x, point.y);
                //console.log("1st point ink from ink", point);
            }
            else
            {
                context.lineTo(point.x, point.y);
            }
        }
    }
    context.stroke();
    return strokePoints;
}

/* replay ink */
this.replayInk = function(inkObj)
{
    var strokes = inkObj.strokes;
    var staffSize = inkObj.staffSize;
    self.gesture = new Gesture();

    var marginPoint = new Point(0, 0);
    if (!("version" in inkObj) || inkObj.version < CURRENT_INK_VERSION)
    {
        // old versions of test ink contain margin, so offset all points by that
        marginPoint = new Point(10, 92);
    }
    var strokePoints = this.drawInkFromStrokes(strokes, staffSize, marginPoint);
for (var i = 0; i < strokePoints.length; i++)
{
    for (var j = 0; j < strokePoints[i].length; j++)
    {
        self.gesture.continueGesture(strokePoints[i][j].x,
                                   strokePoints[i][j].y);
    }
    self.gesture.newStroke();
    self.ink.strokes.push(Point.clonePoints(strokePoints[i].slice(0)))
;
    }
var canvas1 = document.getElementById("canvas1");
var canvas2 = document.getElementById("canvas2");
gestureClassifier.dynamicMatchHull(self.gesture, canvas1, canvas2,
                                staff.getSize(), staff, context, self);
}

// draw from segmented gestures instead of original ink
function drawInk()
{
    for (var i = 0; i < self.ink.gestures.length; i++)
    {
        var strokes = self.ink.gestures[i].strokes;
        for (var j = 0; j < strokes.length; j++)
        {
            var stroke = strokes[j];
            for (var k = 0; k < stroke.length; k++)
            {
                var point = new Point(stroke[k].x + staff.marginLeft,
                                       stroke[k].y + staff.marginTop);
                if (k == 0)
                {
                    context.moveTo(point.x, point.y);
                    // console.log("1stpoint ink", point);
                }
                else
                {
                    context.lineTo(point.x, point.y);
                }
            }
        }
    }
    context.stroke();
}

CanvasDraw.js

function CanvasDraw(trainingCallback, staffSize, staffOffsetPoint)
{
    var _canvas, _context, _pencil;
```
var _staff = null;
this.trainingCallback = trainingCallback;
this.disableAllEvents = false;
var canvasDraw = this;
var self = this;

this.getContext = function ()
{
    return _context;
}

this.getCanvas = function ()
{
    return _canvas;
}

this.getPencil = function ()
{
    return _pencil;
}

this.getStaff = function ()
{
    return _staff;
}

function initialize()
{
    // Find the canvas element.
    _canvas = document.getElementById("canvasView");

    if (USE_CANVAS_SVG)
    {
        var canvasSVG = new CanvasSVG.Deferred();
        _canvas = canvasSVGIInit();
        canvasSVG.wrapCanvas(_canvas);
    }

    _context = _canvas.getContext("2d");

    Rastral.getContextDefaults(_context);

    _staff = new Staff(staffOffsetPoint);
    if (!staffSize)
    {
        staffSize = Staff.DEFAULT_STAFF_SIZE;
    }
    _staff.setSize(staffSize);
    _staff.render(_context);
    this.trainingCallback = trainingCallback;

    _pencil = new Pencil(this.trainingCallback, _context, _staff);
```
if (isIphone) {
  document.body.addEventListener("touchmove", function(e) {
    e.preventDefault();
  });
  event.preventDefault();
  _canvas.addEventListener("touchstart", eventCanvas, false);
  _canvas.addEventListener("touchmove", eventCanvas, false);
  _canvas.addEventListener("touchend", eventCanvas, false);
}
else {
  _canvas.addEventListener("mousedown", eventCanvas, false);
  _canvas.addEventListener("mousemove", eventCanvas, false);
  _canvas.addEventListener("mouseup", eventCanvas, false);
}
}

this.drawNotationSymbol = function(symbolName, topLeftPoint) {
  _staff.drawSymbol(_context, topLeftPoint, symbolName);
}

function drawBoundingBox(topLeftPoint, bottomRightPoint) {
  _context.strokeRect(topLeftPoint.x, topLeftPoint.y, bottomRightPoint.x - topLeftPoint.x, bottomRightPoint.y - topLeftPoint.y);
}

// set scale, maintaining aspect ratio 1.0 is normal size, 100%
this.setScale = function(scale) {
  _context.scale(scale, scale);
}

function eventCanvas(event) {
  if (self.disableAllEvents) {
    return;
  }
  Utils.processEvent(event);
  // Call the event handler
  var func = _pencil[event.type];
  if (func) {
    func(event);
  }
}

function canvasSVGInit()
111 { 
112 var cel = document.createElementNS(ns, "canvas"); 
113 cel.setAttributeNS(null, 'width', _canvas.width); 
114 cel.setAttributeNS(null, 'height', _canvas.height); 
115 cel.setAttributeNS(null, 'style', _canvas.getAttributeNS(null, 'style')); 
116 _canvas.parentNode.replaceChild(cel, _canvas); 
117 cel.setAttributeNS(null, 'id', "canvas"); 
118 return cel; 
119 }
120 
121 initialize();
122
123
124 }

10.4 Geometry

../common/Point.js

1 /* a point in Euclidean space */
2 function Point(x, y)
3 {
4 this.length = 0;
5 this.x = x || 0;
6 this.y = y || 0;
7 var self = this;
8
9 this.subtract = function(pointB)
10 {
11 return (new Point(this.x - pointB.x, this.y - pointB.y));
12 }
13 this.add = function(pointB)
14 {
15 return (new Point(this.x + pointB.x, this.y + pointB.y));
16 }
17 this.toString = function()
18 {
19 return ("length: " + this.length + " x: " + this.x + " y: " + this.y);
20 }
21
22 this.toJson = function()
23 {
24 return ("{"x": " + this.x + ","y": " + this.y + "}");
25 }
26
27 this.toSimpleObject = function()
28 {
29 return {x: this.x, y: this.y};
30 }
31
32 this.getLength = function()
```javascript
33   {
34     return Math.sqrt(this.x * this.x + this.y * this.y);
35   }
36   this.scale = function(scaleFactor)
37   {
38     this.x *= scaleFactor;
39     this.y *= scaleFactor;
40   }
41   this.scale2D = function(scaleFactorX, scaleFactorY)
42   {
43     this.x *= scaleFactorX;
44     this.y *= scaleFactorY;
45   }
46   this.clone = function()
47   {
48     var p = new Point(self.x, self.y);
49     p.length = self.length;
50     return p;
51   }
52   this.serialize = function()
53   {
54     var obj = {};
55     obj.x = this.x;
56     obj.y = this.y;
57   }
58
61   // static, public
62   Point.distance = function(p1, p2)
63   {
64     var dx, dy;
65     var dx = p2.x - p1.x;
66     var dy = p2.y - p1.y;
67     return Math.sqrt(dx * dx + dy * dy);
68   }
69
70   // determine if point is left of a given line
71   Point.isLeft = function (p0, p1, p2)
72   {
73     return (p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y);
74   }
75
76   Point.crossProduct = function (p1, p2)
77   {
78     return point1.x * point2.y - point1.y * point2.x;
79   }
80
81   Point.dotProduct = function (p1, p2)
82   {
83     return point1.x * point2.x + point1.y * point2.y;
84   }
85
86   Point.scalePoints = function (points, scaleFactor, yOffset)
```
for (var i = 0; i < points.length; i++) {
    var point = points[i];
    if (yOffset) {
        point.y -= yOffset;
    }
    point.scale(scaleFactor);
    if (yOffset) {
        point.y += yOffset;
    }
}

Point.scalePoints2D = function (points, scaleFactorX, scaleFactorY) {
    for (var i = 0; i < points.length; i++) {
        var point = points[i];
        point.scale(scaleFactorX, scaleFactorY);
    }
}

Point.clonePoints = function (points) {
    var cPoints = [];
    for (var i = 0; i < points.length; i++) {
        cPoints.push(points[i].clone());
    }
    return cPoints;
}

Point.serializePoints = function (points) {
    var pointStr = '[';
    for (var i = 0; i < points.length; i++) {
        if (points[i].length > 0) {
            pointStr += '[';
            for (var j = 0; j < points[i].length; j++) {
                var point = points[i][j];
                pointStr += point.toJson();
                if (j < points[i].length - 1) {
                    pointStr += ",";
                }
            }
            pointStr += "]\n";
        }
    }
    pointStr += ']n';
if (i < points.length - 1)
{
    pointStr += ",";
}
pointStr += ']';
return pointStr;
}

Point.serializePointsNew = function(points)
{
    var pointsAry = []
    for (var i = 0; i < points.length; i++)
    {
        pointsAry[i] = [];
        for (var j = 0; j < points[i].length; j++)
        {
            pointsAry[i][j] = points[i][j].toSimpleObject();
        }
    }
    return pointsAry;
}

// unused
Point.clonePoints2D = function(points)
{
    var pointsAry = []
    for (var i = 0; i < points.length; i++)
    {
        pointsAry[i] = [];
        for (var j = 0; j < points[i].length; j++)
        {
            pointsAry[i][j].push(points[i][j].clone());
        }
    }
    return pointsAry;
}

Point.angleDifference = function(point1, point2)
{
    var angle1 = Math.atan2(point1.y, point1.x);
    var angle2 = Math.atan2(point2.y, point2.x);
    return Math.abs(angle2 - angle1);
}

// returns mean of all angle difference in 2 vectors of points
Point.angleDifferences = function(points1, points2)
{
    var numPoints = points1.length;
    if (points2.length != numPoints)
    {
        // code continues here
    }
}
numPoints = Math.min(points1.length, points2.length);

var totalDifferences = 0;
for (var i = 0; i < numPoints - 1; i++) {
    totalDifferences += Math.abs(Point.angleDifference(points1[i],
                                   points1[i + 1]) - Point.angleDifference(points2[i],
                                   points2[i + 1]));
}

return totalDifferences / numPoints;

// Point.findNearestPoint = function(point, points)
{
    var nearestIndex = 0;
    var nearestPointDistance = Number.MAX_VALUE;
    for (var i = 0; i < points.length; i++) {
        var distance = Point.distance(point, points[i]);
        if (distance < nearestPointDistance) {
            nearestPointDistance = distance;
            nearestIndex = i;
        }
    }
    return points[nearestIndex];
}
```javascript
var isBetween = (point.x >= minX && point.x <= maxX &&
    point.y >= minY && point.y <= maxY);
return isBetween &&
    slope1 + SLOPE_MATCH_TOLERANCE >= slope2 &&
    slope1 - SLOPE_MATCH_TOLERANCE <= slope2;
}
this.toString = function ()
{
    return "line from " + p1.toString() + " to " + p2.toString();
}

../common/Rectangle.js

function Rectangle (topLeft, topRight, bottomRight, bottomLeft)
{
    this.topLeft = topLeft;
    this.topRight = topRight;
    this.bottomRight = bottomRight;
    this.bottomLeft = bottomLeft;
    var self = this;

    this.getWidth = function ()
    {
        return this.topRight.x - this.topLeft.x;
    }

    this.getHeight = function ()
    {
        return this.bottomLeft.y - this.topLeft.y;
    }

    this.getPoints = function ()
    {
        return [this.topLeft, this.topRight, this.bottomRight, this.bottomLeft ];
    }

    this.moveTo = function (point)
    {
        this.topLeft = this.topLeft.add(point);
        this.topRight = this.topRight.add(point);
        this.bottomRight = this.bottomRight.add(point);
        this.bottomLeft = this.bottomLeft.add(point);
    }

    this.getPolygon = function ()
    {
        var p = new Polygon();
        p.points = self.getPoints();
        // close rectangle
```
```javascript
38     p.points.push(this.topLeft);
39     return p;
40     }
41
42     this.contains = function(point)
43     {
44         if (point.x >= this.topLeft.x && point.x <= this.topRight.x
45             && point.y >= this.topLeft.y && point.y <= this.bottomLeft.y)
46             {
47                 return true;
48             }
49             return false;
50     }
51 }

../common/Polygon.js

1 function Polygon()
2 {
3     /* static */
4         DEFAULT_NUM_POINTS = 52;
5         MIN_NUM_POINTS = 4;
6         MAX_NUM_POINTS = 4096;
7
8     /* public */
9         this.symbol = "";
10        this.numNormalPoints = DEFAULT_NUM_POINTS;
11         this.points = [];
12
13        this.startingPoint = new Point();
14        this.topLeft = new Point();
15        this.bottomRight = new Point();
16        this.height = 0;
17        this.width = 0;
18        this.closed = false;
19        this.segmentIdx = 0;
20
21     /* private */
22        var _offsetX = 0;
23        var _offsetY = 0;
24        var _shapeNeedsUpdating = false;
25        var _lineThickness = 7;
26        var _lineColor = 0x000000;
27        var _shapeMarginTop = 10;
28        var _shapeMarginLeft = 10;
29        var _length = -1;
30        var _averageCenter;
31
32        this.addPoint = function(x, y)
33        {
34            this.points.push(new Point(x, y));
35            _shapeNeedsUpdating = true;
36        }
37    }
```
```javascript
this.strokeLength = function()
{
    var length = 0;
    for (var i = 1; i < this.points.length; i++)
    {
        length += Point.distance(this.points[i-1], this.points[i]);
    }
    return length;
}

this.normalizeShape = function()
{
    var newPoints = [];
    var normalSegmentLength = this.strokeLength() / (this.numNormalPoints - 1);
    newPoints.push(this.points[0]);
    var startPoint = this.points[0];
    var endPoint = this.points[0];
    var previousDistanceEnd = 0;
    var previousDistanceStart = 0;
    var distance = 0;
    var segmentLength = 0;
    var pointIdx = 1;
    var i = 0;
    while (i <= MAX_NUM_POINTS)
    {
        i ++;
        var excess = previousDistanceEnd - distance;
        if (excess >= normalSegmentLength)
        {
            distance += normalSegmentLength;
            var ratio = (distance - previousDistanceStart) / segmentLength ;
            var newPoint = new Point((endPoint.x - startPoint.x) * ratio +
                                      startPoint.x,
                                      (endPoint.y - startPoint.y) * ratio +
                                      startPoint.y);
            newPoints.push(newPoint);
        }
        else
        {
            if (pointIdx == this.points.length)
            {
                break;
            }
            startPoint = endPoint;
            endPoint = this.points[pointIdx];
            previousDistanceStart = previousDistanceEnd;
            segmentLength = endPoint.subtract(startPoint).getLength();
            previousDistanceEnd += segmentLength;
        }
    }
```

    pointIdx ++;
    }
}

// add final point
if (newPoints.length < this.numNormalPoints)
{
    newPoints.push(endPoint);
}

this.points = newPoints;
}

// center on average center point
this.normalizeToAverageCenter = function ()
{
    for (var i = 0; i < this.points.length; i++)
    {
        var point = this.points[i];
        point.x -= this.getAverageCenter().x;
        point.y -= this.getAverageCenter().y;
    }
}

this.getAverageCenter = function ()
{
    if (_averageCenter !== null)
    {
        return _averageCenter;
    }
    var centerX = 0;
    var centerY = 0;
    for (var i = 0; i < this.points.length; i++)
    {
        var point = this.points[i];
        centerX += point.x;
        centerY += point.y;
    }
    centerX /= this.points.length;
    centerY /= this.points.length;
    return new Point(centerX, centerY);
}

this.getBoundingBox = function ()
{
    var firstPoint = this.points[0];
    var minX = firstPoint.x;
    var minY = firstPoint.y;

    for (var i = 1; i < this.points.length; i++)
    {
        if (this.points[i].x < minX)
            minX = this.points[i].x;
        else if (this.points[i].x > maxX)
            maxX = this.points[i].x;
        if (this.points[i].y < minY)
            minY = this.points[i].y;
        else if (this.points[i].y > maxY)
            maxY = this.points[i].y;
    }
}
var maxX = firstPoint.x;
var maxY = firstPoint.y;
for (var i = 0; i < this.points.length; i++)
{
    var point = this.points[i];
    if (point.x < minX)
    {
        minX = point.x;
    }
    if (point.y < minY)
    {
        minY = point.y;
    }
    if (point.x > maxX)
    {
        maxX = point.x;
    }
    if (point.y > maxY)
    {
        maxY = point.y;
    }
}

var width = maxX - minX;
var height = maxY - minY;
this.width = width;
this.height = height;
this.topLeft = new Point(minX, minY);
this.bottomRight = new Point(maxX, maxY);

// privileged
this.normalizeScale = function()
{
    var firstPoint = this.points[0];
    var minX = firstPoint.x;
    var minY = firstPoint.y;
    var maxX = firstPoint.x;
    var maxY = firstPoint.y;
    for (var i = 0; i < this.points.length; i++)
    {
        var point = this.points[i];
        if (point.x < minX)
        {
            minX = point.x;
        }
        if (point.y < minY)
        {
            minY = point.y;
        }
        if (point.x > maxX)
        {
            maxX = point.x;
        }
        if (point.y > maxY)
        {
            maxY = point.y;
        }
    }
}
if (point.y > maxY)
{
    minY = point.y;
}

var width = maxX - minX;
var height = maxY - minY;
this.width = width;
this.height = height;
var scale = (width > height) ? width : height;

if (scale <= 0)
{
    return;
}

scale /= 1;

for (var i = 0; i < this.points.length; i++)
{
    var point = this.points[i];
    point.x *= scale;
    point.y *= scale;
}

this.scale = function (scaleFactor)
{
    for (var i = 0; i < this.points.length; i++)
    {
        var point = this.points[i];
        point.scale(scaleFactor);
    }}

this.getNumPoints = function ()
{
    return this.points.length;
}

this.startPolygon = function (x, y)
{
    this.addPoint(x, y);
}

this.getLength = function ()
{
    if (_length == -1 || _shapeNeedsUpdating)
    {
        _length = 0;
        var lastPoint;
```javascript
for (var i = 0; i < this.points.length; i++) {
  var point = this.points[i];
  if (lastPoint !== null) {
    this.length += Point.distance(lastPoint, point);
  }
  lastPoint = point;
}
return this.length;
}

this.createFromJson = function (obj) {
  this.symbol = obj.symbol; // alert (obj.symbol);
  this.points = [];
  for (var i = 0; i < obj.points.length; i++) {
    this.points.push(new Point(point.x, point.y));
  }
}

this.newSegment = function () {
  this.segmentIdx ++;
}

this.normalize = function () {
  this.normalizeScale();
  this.normalizeShape();
  this.normalizeToAverageCenter();
}

this.getPoints = function () {
  return this.points;
}

this.toString = function () {
  var string = "polygon with " + this.points.length + " points \n:”;
  for (var i = 0; i < this.points.length; i++) {
    string += this.points[i] + ",";
  }
  return string;
}

// unimplemented
this.getJson = function () {
  var json = new String();
```
```javascript
var pointsAry = [];
for (var i = 0; i < this.points.length; i++)
{
    var point = this.points[i];
    pointsAry.push({ "x": point.x, "y": point.y });
}
return Json.encode(pointsAry);

// sort points
this.sortPoints = function()
{
    this.points.sort(Polygon.sortPointsByX);
    this.points.sort(Polygon.sortPointsByY);
    // alert(this.points);
}

this.sortPointsByDistance = function()
{
    var points = [];
    // start with first point
    // alert(this.points);
    var firstPoint = this.points.shift();
    points.push(firstPoint);
    var nearestPointIdx = -1;
    var nearestPointLength = Infinity;

    var prevPoint = firstPoint;
    // determine each next-nearest point, by distance
    while (this.points.length > 0)
    {
        // alert(this.points.length + " " + prevPoint + "\n" + points);
        for (var i = 0; i < this.points.length; i++)
        {
            var pointLength = Point.distance(prevPoint, this.points[i]);
            // find nearest point
            if (nearestPointIdx == -1 || pointLength < nearestPointLength)
            {
                nearestPointIdx = i;
            }
        }
        prevPoint = this.points.splice(nearestPointIdx, 1)[0];
        points.push(prevPoint);
        nearestPointIdx = -1;
        nearestPointLength = Infinity;
    }
    // close shape
    points.push(firstPoint);
    this.points = points;
}```
// order points to resemble a convex polygon

```javascript
this.chainHull = function () {
  var P = this.points;
  var H = [];
  var bot = 0, top = -1;
  var i;
  n = P.length;
  var minmin = 0, minmax;
  var xmin = P[0].x;
  for (i = 1; i < n; i++) {
    if (P[i].x != xmin)
      break;
  }
  minmax = i - 1;
  if (minmax == n - 1) {
    H[++top] = P[minmin];
    if (P[minmax].y != P[minmin].y) //segment
      { H[++top] = P[minmax];
      }
    H[++top] = P[minmin]; //endpoint
    //return top + 1;
    //return H;
    this.points = H;
  }

  var maxmin, maxmax = n - 1;
  var xmax = P[n-1].x;
  for (i = n - 2; i >= 0; i--)
    { if (P[i].x != xmax)
      break;
    }
  maxmin = i + 1;

  // lower hull
  H[++top] = P[minmin];
  i = minmax;
  while (++i <= maxmin)
    { if (Point.isLeft( P[minmin], P[maxmin], P[i]) >= 0 && i < maxmin)
      continue;
    }
  while (top > 0) // at least 2 points
```
if (Point.isLeft(H[top - 1], H[top], P[i]) > 0)
{
    break;
}
else
{
    top--;
}
H[++top] = P[i];

// upper hull
if (maxmax != maxmin)
{
    H[++top] = P[maxmax];
}
bot = top;
i = maxmin;
while (--i >= minmax)
{
    if (Point.isLeft(P[maxmax], P[minmax], P[i]) >= 0 && i > minmax)
    {
        continue;
    }

    while (top > bot)
    {
        if (Point.isLeft(H[top - 1], H[top], P[i]) > 0)
        {
            break;
        }
        else
        {
            top--;
        }
    }
    H[++top] = P[i];
}
if (minmax != minmin)
{
    H[++top] = P[minmin];
}
this.points = H;

this.monotoneHull = function()
{
    var n = this.points.length;
    var k = 0;
    var hull = [];
    // Sort points
    this.sortPoints();
}
466 // Lower hull
467 for (var i = 0; i < n; i++)
468 {
469     while (k >= 2 && Polygon.cross(hull[k-2], hull[k-1], this.points[i]) <= 0)
470     {
471         k--;
472     }
473     hull[k++] = this.points[i];
474 }
475
476 // Build upper hull
477 for (var i = n-2, t = k+1; i >= 0; i--)
478 {
479     while (k >= t && Polygon.cross(hull[k-2], hull[k-1], this.points[i]) <= 0)
480     {
481         k--;
482     }
483     hull[k++] = this.points[i];
484 }
485
486 this.points = hull;
487 }
488
489 this.close = function()
490 {
491     if (this.points.length <= 0)
492     {
493         this.closed = false;
494         return false;
495     }
496     this.addPoint(this.points[0].x, this.points[0].y);
497     this.closed = true;
498 }
499
500 this.open = function()
501 {
502     if (this.points.length <= 0)
503     {
504         return false;
505     }
506     this.points.pop();
507     this.closed = false;
508 }
509
510 // Find a simple (non self-intersecting) closed polygonal chain from a series of unordered points
511 this.findConcavePolygon = function()
512 {
513     if (this.closed)
514     {
515         this.open();
516     }
517 }
window._averageCenter = this.getAverageCenter();
this.points.sort(this.sortAllPoints);
this.close();

this.sortAllPoints = function(point1, point2)
{
    var angle1 = Math.atan2(window._averageCenter.y - point1.y, window._averageCenter.x - point1.x);
    var angle2 = Math.atan2(window._averageCenter.y - point2.y, window._averageCenter.x - point2.x);

    if (angle1 > angle2)
    {
        return 1;
    }
    else if (angle1 < angle2)
    {
        return -1;
    }

    return 0;
}

//useful for visual debugging
this.drawOnNewCanvas = function(parentElementId, paddingPoint, text)
{
    // create a new canvas
    var newCanvas = document.createElement("canvas");
    // set dimensions of new canvas
    this.getBoundingBox();
    newCanvas.setAttribute("width", 400);
    newCanvas.setAttribute("height", 400);
    // border around canvas
    newCanvas.setAttribute("style", "border: 1px solid #666;");

    var context = newCanvas.getContext('2d');
    context.fillStyle = "black";
    context.fillText(text, 10, 10);

    //draw the polygon to it
    Polygon.drawOnCanvas(newCanvas, this);
    //append the new canvas to the page
    document.getElementById(parentElementId).appendChild(newCanvas);
}

this.pointInPolygon = function(point)
{
    if (!this.checkPolygon())
    {
        return false;
    }

    intersections = 0;

    }
var points = this.points;

// Check if the point is inside the polygon or on the boundary
for (var i = 1; i < points.length; i++) {
    p1 = points[i-1];
    p2 = points[i];
    var line = new Line(p1, p2);
    if (line.pointIsOnLine(point)) {
        return Polygon.ON_POLYGON_BOUNDARY;
    }

    if (p1.y == p2.y && p1.y == point.y && point.x > Math.min(p1.x, p2.x) &&
        point.x < Math.max(p1.x, p2.x)) {
        return Polygon.ON_POLYGON_BOUNDARY;
    }

    if (point.y > Math.min(p1.y, p2.y) && point.y <= Math.max(p1.y, p2.y) &&
        point.x <= Math.max(p1.x, p2.x) && p1.y != p2.y) {
        var xinters = (point.y - p1.y) * (p2.x - p1.x) / (p2.y - p1.y) + p1.x;
        if (xinters == point.x) {
            // Check if point is on the polygon boundary (other than horizontal)
            return Polygon.ON_POLYGON_BOUNDARY;
        }

        if (p1.x == p2.x || point.x <= xinters) {
            intersections++;
        }
    }
}

// If the number of edges we passed through is even, then it's in the polygon.
if (intersections % 2 != 0) {
    return Polygon.INSIDE_POLYGON;
} else {
    return Polygon.OUTSIDE_POLYGON;
}

// determine if two polygons intersect
this.intersects = function(polygon) {
}

}
Polygon.prototype.endPolygon = function ()
{
    /\ return false if we don't have enough points to make a polygon
    return this.checkPolygon();
}

Polygon.prototype.checkPolygon = function ()
{
    if (this.getPoints().length < MIN_NUM_POINTS)
    {
        return false;
    }
    return true;
}

Polygon.drawOnCanvas = function (canvas, polygon, lineStyle)
{
    var points = polygon.getPoints();
    var context = canvas.getContext('2d');
    context.beginPath();
    if (lineStyle)
    {
        context.strokeStyle = lineStyle;
    }
    for (var i = 0; i < points.length; i++)
    {
        var point = points[i];
        if (i == 0)
        {
            context.moveTo(point.x, point.y);
        }
        else
        {
            context.lineTo(point.x, point.y);
        }
    }
    context.stroke();
}

Polygon.dotProduct = function (points1, points2)
{
    if (points1.length != points2.length)
    {
        throw new Error("length mismatch while trying to calculate dot product
            " + points1.length + " " + points2.length);
    }
    var dotProduct = 0;
    for (var i = 0; i < points1.length; i++)
    {
var point1 = points1[i];
var point2 = points2[i];
dotProduct += point1.x * point2.x + point1.y * point2.y;
}
return dotProduct;
}
}

Polygon.drawAndScaleToCanvas = function(canvas, polygon)
{
    polygon.getBBox();
    // clear canvas
    Utils.clearCanvas(canvas);
    Polygon.drawOnCanvas(canvas, polygon);
    // crop and scale
    var context = canvas.getContext('2d');
    var border = 8;
    var imageData = context.getImageData(polygon.topLeft.x - border, polygon.topLeft.y - border, polygon.width + border, polygon.height + border);
    var tempCanvas = document.createElement("canvas");
    // Utils.clearCanvas(tempCanvas);
    tempCanvas.setAttribute("id", new Date().getTime());
    tempCanvas.setAttribute("width", imageData.width);
    tempCanvas.setAttribute("height", imageData.height);
    tempCanvas.getContext("2d").putImageData(imageData, 0, 0);
    Utils.clearCanvas(canvas);
    context.drawImage(tempCanvas, 0, 0, canvas.width, canvas.height);
    // alert(polygon.topLeft.x-border);
    // remove temp canvas??
}

Polygon.cross = function(O, A, B)
{
    return (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x);
}

Polygon.sortPointsByX = function(obj1, obj2)
{
    if (obj1.x > obj2.x)
    {
        return 1;
    }
    else if (obj1.x < obj2.x)
    {
        return -1;
    }
    return 0;
}

Polygon.sortPointsByY = function(obj1, obj2)
{
    if (obj1.y > obj2.y)
```javascript
Polygon.sortPointsByXY = function (obj1, obj2) {
    if (obj1.x > obj2.x && obj1.y > obj2.y) {
        return 3;
    } else if (obj1.x > obj2.x && obj1.y === obj2.y) {
        return 2;
    } else if (obj1.x === obj2.x && obj1.y > obj2.y) {
        return 1;
    } else if (obj1.x === obj2.x && obj1.y < obj2.y) {
        return -1;
    } else if (obj1.x < obj2.x && obj1.y === obj2.y) {
        return -2;
    } else if (obj1.x < obj2.x && obj1.y < obj2.y) {
        return -3;
    }
    return 0;
}
```

```
../common/PolygonIntersect.js
1 /* determine whether two polygons intersect */
2 /* ported from here: */
4 .sci.utah.edu/gf/project/Manta/scmsvn/%3Faction%3Dbrowse%26path%3D%252F
5 Ftrunk%252FModel%252FGroups%252FBSP%252Faip.h%26revision%3D2329%26view%3
```
function PolygonIntersect ()
{
    function Box (min, max)
    {
        this.min = min;
        this.max = max;
    }
    function Rng (mn, mx)
    {
        this.mn = mn;
        this.mx = mx;
    }
    function IPoint()
    {
        this.x;
        this.y;
    }
    function Vertex()
    {
        this.ip = new IPoint();
        this.rx = new Rng();
        this.ry = new Rng();
        this.inn;
    }
    var gamut = 500000000.;
    var mid = gamut / 2.;
    //

    function range (points, c, bbox)
    {
        while (c > 0)
        {
            bbox.min.x = Math.min (bbox.min.x, points[c].x);
            bbox.min.y = Math.min (bbox.min.y, points[c].y);
            bbox.max.x = Math.max (bbox.max.x, points[c].x);
            bbox.max.y = Math.max (bbox.max.y, points[c].y);
        }
    }
    function area (a, p, q)
    {
        return p.x * q.y - p.y * q.x + a.x * (p.y - q.y) + a.y * (q.x - p.x);
    }
    function ovl (p, q)
    {
    }
return p.mn < q.mx && q.mn < p.mx;

//

var ssss = 0;
var sclx = 0;
var scly = 0;

function contrib(f_x, f_y, t_x, t_y, w)
{
  ssss += w * (t_x - f_x) * (t_y + f_y) / 2;
}

function fit(x, cx, ix, fudge, B)
{
  c = cx;
  while (c-- > 0)
  {
    ix[c] = new Vertex();
    ix[c].ip = new IPoint();
    ix[c].ip.x = (((x[c].x - B.min.x) * sclx - mid) & ~7) | fudge | (c & 1);
    ix[c].ip.y = (((x[c].y - B.min.y) * scly - mid) & ~7) | fudge;
  }
  ix[0].ip.y += cx & 1;
  ix[cx] = ix[0];

  c = cx;
  while (c-- > 0)
  {
    ix[c].rx = ix[c].ip.x < ix[c + 1].ip.x ?
      new Rng(ix[c].ip.x, ix[c + 1].ip.x) :
      new Rng(ix[c + 1].ip.x, ix[c].ip.x);
    ix[c].ry = ix[c].ip.y < ix[c + 1].ip.y ?
      new Rng(ix[c].ip.y, ix[c + 1].ip.y) :
      new Rng(ix[c + 1].ip.y, ix[c].ip.y);
    ix[c].inn = 0;
  }
}

function cross(a, b, c, d, a1, a2, a3, a4)
{
  var r1 = a1 / (a1 + a2);
  var r2 = a3 / (a3 + a4);
  contrib((a.ip.x + r1 * (b.ip.x - a.ip.x)),
             (a.ip.y + r1 * (b.ip.y - a.ip.y)),
             b.ip.x, b.ip.y, 1);
  contrib(d.ip.x, d.ip.y,
function inness(P, cP, Q, cQ)
{
    var s = 0;
    var c = cQ;
    var p = P[0].ip;
    while (c-- > 0)
    {
        if (Q[c].rx.min < p.x && p.x < Q[c].rx.max)
        {
            var sgn = 0;
            if (area(p, Q[c].ip, Q[c + 1].ip) > 0)
                s += (sgn != Q[c].ip.x < Q[c + 1].ip.x) ? 0 : (sgn ? -1 : 1);
        }
    }
    for (j = 0; j < cP; ++j)
    {
        if (s != 0)
            contrib(P[j].ip.x, P[j].ip.y, P[j + 1].ip.x, P[j + 1].ip.y, s);
        s += P[j].inn;
    }
}
//

this.intersection = function(a, b)
{
    var na = a.length;
    var nb = b.length;
    var ipa = new Array(na + 1);
    var ipb = new Array(nb + 1);
    var bbox = new Box(new Point(Number.MAX_VALUE, Number.MAX_VALUE),
        new Point(-Number.MAX_VALUE, -Number.MAX_VALUE));
    if (na < 3 || nb < 3)
    {
        return 0;
    }
    range(a, na, bbox);
    range(b, nb, bbox);
    var rngx = bbox.max.x - bbox.min.x;
    sclx = gamut / rngx;
    rngy = bbox.max.y - bbox.min.y;
scly = gamut / rngy;
ascale = sclx * scly;

fit(a, na, ipa, 0, bbox);
fit(b, nb, ipb, 2, bbox);

for (j = 0; j < na; ++j)
{
    for (k = 0; k < nb; ++k)
    {
        if (ovl(ipa[j].rx, ipb[k].rx) && ovl(ipa[j].ry, ipb[k].ry))
        {
            var a1 = -area(ipa[j].ip, ipb[k].ip, ipb[k + 1].ip);
            var a2 = area(ipa[j + 1].ip, ipb[k].ip, ipb[k + 1].ip);
            var o = a1 < 0;
            if (o == a2 < 0)
            {
                var a3 = area(ipb[k].ip, ipa[j].ip, ipa[j + 1].ip);
                var a4 = -area(ipb[k + 1].ip, ipa[j].ip, ipa[j + 1].ip);
                if (a3 < 0 == a4 < 0)
                {
                    if (o)
                    {
                        cross(ipa[j], ipa[j + 1], ipb[k], ipb[k + 1],
                              a1, a2, a3, a4);
                    }
                    else
                    {
                        cross(ipb[k], ipb[k + 1], ipa[j], ipa[j + 1],
                              a3, a4, a1, a2);
                    }
                }
            }
        }
    }
    inness(ipa, na, ipb, nb);
inness(ipb, nb, ipa, na);
return ssss / ascale;
}

10.5 Training

../Trainer.js

// Trainer

function Trainer(outputDocId, staffSize)
{

this.output = document.getElementById(outputDocId);
this.symbolNameIdx = 0;
var self = this;
var TRAINING_DEBUG = false;
var SYMBOL_DISPLAY_MS = 700;
// array of objects
var _trainingDataArray = [];
this.trainingCallback = function(result)
{
  _trainingDataArray.push(
    {"symbol": symbolNames[self.symbolNameIdx],
    "staffSize": staff.getSize(), "strokes": JSON.parse(result)});
  if (TRAINING_DEBUG)
  {
    console.log(result, "result");
    self.outputTrainingData();
  }
  if (symbolNames.length - 1 > self.symbolNameIdx)
  {
    self.next();
  }
  else // done
  {
    Utils.clearCanvas(cd.getCanvas());
    cd.getContext().fillText("fine", 10, 200);
    self.outputTrainingData();
  }
}
this.outputTrainingData = function()
{
  if (TRAINING_DEBUG)
  {
    console.log(JSON.stringify(_trainingDataArray));
  }
  self.output.innerHTML = "<pre>" + JSON.stringify(_trainingDataArray, null, true) + "</pre>";
}
var marginLeft = 10;
var marginTop = 92;
var staffOffsetPoint = new Point(marginLeft, marginTop);
var cd = new CanvasDraw(this.trainingCallback, staffSize, staffOffsetPoint);
var pencil = cd.getPencil();
pencil.autoExtend = false;
var staff = cd.getStaff();
this.train = function()
{
  pencil.redrawCanvas();
pencil.gesture = new Gesture();
// disable drawing
cd.disableAllEvents = true;
var context = cd.getContext();
staff.startLayout(context);

// display symbol
cd.drawNotationSymbol(symbolNames[self.symbolNameIdx], new Point(100, staff.lineY(2)));
staff.endLayout(context);

// set timeout to hide symbol
setTimeout(self.hideSymbol, SYMBOL_DISPLAY_MS);

this.previous = function () {
  _trainingDataArray.pop();
  self.symbolNameIdx --;
  if (self.symbolNameIdx < 0)
  {
    self.symbolNameIdx = 0;
  }

  this.train();
}

this.next = function () {
  self.symbolNameIdx ++;
  if (self.symbolNameIdx >= symbolNames.length)
  {
    self.symbolNameIdx = symbolNames.length - 1;
  }

  this.train();
}

this.hideSymbol = function () {
  // enable drawing
  cd.disableAllEvents = false;
pencil.redrawCanvas();
}

this.train();

10.6 Utilities

../common/Utils.js
/*
 * class of utility functions.
 */

function UtilS()
{

Utils.clearCanvas = function(canvas)
{
    var context = canvas.getContext('2d');
    context.clearRect(0, 0, canvas.width, canvas.height);
}

// Unused

Utils.geometricMean = function(numbers)
{
    var total = 1;
    for (var i = 0; i < numbers.length; i++)
    {
        total *= numbers[i];
    }
    return Math.pow(total, (1 / numbers.length));
}

Utils.findNearestNumber = function(array, targetNum)
{
    var nearestNum = -1;
    var closestDistance = Number.MAX_VALUE;
    for (var i = 0; i < array.length; i++)
    {
        // exact match
        if (array[i] == targetNum)
        {
            return array[i];
        }
        else // nearest match
        {
            var distance = Math.abs(targetNum - array[i]);
            if (distance < closestDistance)
            {
                nearestNum = array[i];
                closestDistance = distance;
            }
        }
    }
    return nearestNum;
}

Utils.randColor = function()
{
    return 'rgba(' + Math.floor(Math.random() * 255) + ', ' + Math.floor(Math.random() * 255) + ', ' + Math.floor(Math.random() * 255) + ', 0.7);';
}
UtilsProcessEvent = function(event, offsetPoint)
{
    //standardize positioning coordinate attributes
    //across multiple browser DOM implementations
    if (event.targetTouches)
    {
        //iPhone
        if (event.targetTouches.length > 0)
        {
            event.x = event.targetTouches[0].pageX;
            event.y = event.targetTouches[0].pageY;
        }
        //prevent scrolling and other default gestural behaviors on iPhone
        if (isIPhone)
        {
            event.preventDefault();
        }
    }
    else if (event.layerX || event.layerX == 0)
    {
        //Firefox
        event.x = event.layerX;
        event.y = event.layerY;
    }
    else if (event.offsetX || event.offsetX == 0)
    {
        //Opera
        event.x = event.offsetX;
        event.y = event.offsetY;
    }
    //currently unused
    if (offsetPoint)
    {
        event.x -= offsetPoint.x;
        event.y -= offsetPoint.y;
    }
}

Utils.isInt = function(number)
{
    return parseInt(number) == number;
}

function XmlHttpRequest()
{
    //static
    XMLHttpRequest.create = function()
}
9 {  
10     var request = false;  
11     try  
12     {  
13         request = new ActiveXObject('Msxml2.XMLHTTP');  
14     }  
15     catch (err2)  
16     {  
17         try  
18         {  
19             request = new ActiveXObject('Microsoft.XMLHTTP');  
20         }  
21         catch (err3)  
22         {  
23             try  
24             {  
25                 request = new XMLHttpRequest();  
26             }  
27             catch (err1)  
28             {  
29                 request = false;  
30             }  
31         }  
32     }  
33     return request;  
34 }

../common/Json.js

1 function Json()  
2 {  
3     this.string;  
4     this.data;  
5     this.xmlHttp = XMLHttpRequest.create();  
6     this.loadedCallback = null;  
7 }  
8
9 Json.prototype.loadJson = function(fileName)  
10 {  
11     var self = this;  
12     this.xmlHttp.overrideMimeType("text/json");  
13     this.xmlHttp.open("GET", fileName, true);  
14     this.xmlHttp.onreadystatechange = handleReadyStateChange;  
15     this.xmlHttp.send(null);  
16
17     function handleReadyStateChange()  
18     {  
19         if (self.xmlHttp.readyState == 4)  
20         {  
21             self.string = self.xmlHttp.responseText;  
22             self.data = eval("(" + self.xmlHttp.responseText + ")");  
23             if (self.loadedCallback != null)  
24             {  
25                 self.loadedCallback();  
26             }  
27         }  
28     }  
29 }  
30
98
.../common/CouchServer.js

```javascript
function CouchServer()
{
}

CouchServer.DEFAULT_PROTOCOL = "http://";
CouchServer.DEFAULT_SERVER = "localhost";
CouchServer.DEFAULT_PORT = 5984;

CouchServer.defaultUri = function()
{
    return (CouchServer.DEFAULT_PROTOCOL + CouchServer.DEFAULT_SERVER + ":" +
        CouchServer.DEFAULT_PORT);
}
```

10.7 Hypertext Markup Language (HTML)

Main

```html
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width, user-scalable=no"/>
    <meta name="apple-mobile-web-app-capable" content="yes"/>
    <link rel="apple-touch-icon" href="media/iquil_icon.png"/>
    <meta charset="utf-8"/>
    <title>quilt</title>
</head>
<body>
<!-- utilities -->
<script type="text/javascript" src="common/Utils.js"></script>
<script type="text/javascript" src="common/XmlHttpRequest.js"></script>
<script type="text/javascript" src="common/Json.js"></script>
<script type="text/javascript" src="common/CouchServer.js"></script>
<script type="text/javascript" src="common/couch.js"></script>
<!-- geometry -->
<script type="text/javascript" src="common/Point.js"></script>
<script type="text/javascript" src="common/Line.js"></script>
<script type="text/javascript" src="common/Rectangle.js"></script>
```

99
```html
<script type="text/javascript" src="common/Polygon.js"></script>
<script type="text/javascript" src="common/PolygonIntersect.js"></script>

<canvas>

</canvas>

<!— drawing —>
<script type="text/javascript" src="Ink.js"></script>
<script type="text/javascript" src="Pencil.js"></script>
<script type="text/javascript" src="CanvasDraw.js"></script>

<!— gesture recognition —>
<script type="text/javascript" src="Gesture.js"></script>
<script type="text/javascript" src="Feature.js"></script>
<script type="text/javascript" src="Match.js"></script>
<script type="text/javascript" src="GestureClassifier.js"></script>
<script type="text/javascript" src="SymbolOptions.js"></script>

<!— language model —>
<script type="text/javascript" src="common/Ngram.js"></script>
<script type="text/javascript" src="common/Bigram.js"></script>
<script type="text/javascript" src="common/Language.js"></script>

<!— rastral, the notation rendering engine —>
<!— font embeds —>
<link rel="stylesheet" href="rastral/fonts/stylesheet.css" type="text/css" charset="utf-8">

<script type="text/javascript">
// define global fetamap var before loading the lilypond feta font maps
// this is really a C++ file, but will interpret fine as javascript
// so long as the var _fetaMap is defined first as an object
var _fetaMap = new Object();
// load feta (emmentaler) character mappings
</script>
<script type="text/javascript" src="rastral/fetaList.js"></script>
<script type="text/javascript" src="rastral/Rastral.js"></script>
<script type="text/javascript" src="rastral/Staff.js"></script>
<script type="text/javascript" src="rastral/Clef.js"></script>

<!— for canvas-svg, used only for testing... currently broken —>
<!— script type="text/javascript" src="lib/canvas_svg/canvas-getsvg.js"></script>—>
<style type="text/css">
body
{
  font-family: Verdana, Helvetica, Arial, sans-serif;
}
*
{
  -webkit-user-select: none;
}
* 
```
webKit-touch-callout: none; 

} /*
 * 
 * -webkit-tap-highlight-color: rgba(0,0,0,0);
 */ *
* {

-webkit-text-size-adjust: none;
} 

#canvasView 

{

   // border: 1px dashed #bbc;
     float: top;
 }

#canvas1 

{

   margin-left: 8px;
         border: 1px dashed #bbc;

   // float: left;
        visibility: hidden;
                display: none;
 }

#canvas2 

{

   margin-left: 8px;
         border: 1px dashed #bbc;

   // float: right;
        visibility: hidden;
                display: none;
 }

#controls 

{

   position: absolute;
       top: 500px;
 }

#inkColorCanvas 

{

   border: 1px solid black;
 }

#advancedOptions 

{

   visibility: hidden;
         border: 1px black dashed;
 }

#showHideAdvancedOptionsLink 

{

   text-decoration: none;
       color: black;
 }

#symbolOptions
```javascript
const IMAGE_PATH = "media/";
const MAIN_CONTAINER_ID = "container";
const DEBUG_LEVEL = 0;
const INK_PLAYBACK_INTERVAL_MS = 3000;
const CURRENT_INK_VERSION = 0.1;
const CLASSIFICATION_TYPE_HULL = "hull";
const CLASSIFICATION_TYPE_DYNAMIC_1 = "dynamic_one";
const CLASSIFICATION_TYPE_DYNAMIC_2 = "dynamic_two";
const CLASSIFICATION_TYPE = CLASSIFICATION_TYPE_HULL;
const MATCH_TEMPLATE_SCALE = true;

var gestureClassifier = null;
var notationImages = new Object();
var isIphone = navigator.userAgent.indexOf('iPhone ') >= 0 || navigator.userAgent.indexOf('iPad ') >= 0;
```
var gestureClassifier = null;
var staffSize = 160;
var advancedOptionsShown = false;
var db = new CouchDB( 'quil' );
var inkRowIndex = 0;
var inkRows = [];
var testInkLoaded = false;

const USE_CANVAS_SVG = false;

// var ns = "http://www.w3.org/1999/xhtml";

function handleOnLoad()
{
    var handleCanvasLoad = function (canvasDraw)
    {
        //canvasDraw.setScale( globalScale );
        //drawInkColorPalette();
        }
    // alert( globalScale );

    staffMarginLeft = 10;
    staffMarginTop = 100;
    staffOffsetPoint = new Point( staffMarginLeft, staffMarginTop )
    gestureClassifier = new GestureClassifier( handleCanvasLoad, staffSize, staffOffsetPoint );

    var staffRange = document.getElementById( "staffSizeRange" );
    staffRange.min = Staff.STAFF_SIZES[0];
    staffRange.max = Staff.STAFF_SIZES[ Staff.STAFF_SIZES.length - 1 ];
    staffRange.value = staffSize;

    //show testing options
    if (DEBUG_LEVEL > 0)
    {
        //document.getElementById( "testingOptions" ).style.visibility = "visible";
    }
}

function changeInkColor()
{
    var r = document.getElementById( "inkColorRed" ).value;
    var g = document.getElementById( "inkColorGreen" ).value;
    var b = document.getElementById( "inkColorBlue" ).value;
    var a = document.getElementById( "inkAlpha" ).value / 100;

gestureClassifier.getCanvasDraw().getPencil().setInkColor( r, g, b, a );
drawInkColorPalette();
function drawInkColorPalette()
{
    var paletteCanvas = document.getElementById("inkColorCanvas");
    if (paletteCanvas)
    {
        U0ps.clearCanvas(paletteCanvas);
        var paletteContext = paletteCanvas.getContext("2d");
        paletteContext.fillStyle = gestureClassifier.getCanvasDraw().
            getPencil().getInkColor();
        paletteContext.fillRect(0, 0, paletteCanvas.width, paletteCanvas.
            height);
    }
}

function showHideAdvancedOptions()
{
    advancedOptionsShown = !advancedOptionsShown;
    if (advancedOptionsShown)
    {
        document.getElementById("advancedOptions").style.visibility = 
            "visible";
        document.getElementById("showHideAdvancedOptionsLink").innerHTML = 
            "less options";
    }
    else
    {
        document.getElementById("advancedOptions").style.visibility = 
            "hidden";
        document.getElementById("showHideAdvancedOptionsLink").innerHTML = 
            "more options &hellip;";
    }
}

function clearAll()
{
    // var confirmation = confirm("Are you sure you want clear the page?");
    if (true)
    {
        gestureClassifier.getCanvasDraw().getPencil().clearAll();
    }
    return false;
}

function loadInk()
{
    inkRows = db.view("ink/all").rows;
    inkRowIndex = 0;
    testInkLoaded = true;
    displayInk();
}
```javascript
function nextInk()
{
    if (!testInkLoaded)
    {
        loadInk();
        return;
    }

    inkRowIndex ++;
    if (inkRowIndex >= inkRows.length)
    {
        inkRowIndex = inkRows.length - 1;
    }
    displayInk();
}

function prevInk()
{
    if (!testInkLoaded)
    {
        loadInk();
        return;
    }

    inkRowIndex --;
    if (inkRowIndex <= 0)
    {
        inkRowIndex = 0;
    }
    displayInk();
}

function displayInk()
{
    // update user feedback
    var testStep = document.getElementById("testStep");
    testStep.innerHTML = (inkRowIndex + 1) + " of "+ inkRows.length + " " + inkRows[inkRowIndex].value._id;

    var inkRow = inkRows[inkRowIndex];
    gestureClassifier.getCanvasDraw().getPencil().clearAll();
    gestureClassifier.getCanvasDraw().getPencil().replayInk(inkRow.value);

    // display previous image
    var testImageDiv = document.getElementById("testImage");
    var testImage = new Image();
    if ("imageData" in inkRows[inkRowIndex].value)
    {
        testImage.src = inkRows[inkRowIndex].value.imageData;
    }

    testImageDiv.innerHTML = "";
    testImageDiv.appendChild(testImage);
    if (autoPlayTests() && inkRowIndex < inkRows.length - 1)
```

inkRowIndex ++;
        setTimeout(displayInk, INK_PLAYBACK_INTERVAL_MS);
    }
    }

    // test ink scaling and registration by cycling through all ink and attempt
to classify it against itself
    function selfInkTest()
    {
        // cycle through scales
        for (var i = Staff.STAFF_SIZES.length; i > 0; i --)
        {
            gestureClassifier.getCanvasDraw().getPencil().
            resizeToNearestStaffSize(Staff.STAFF_SIZES[i]);
            gestureClassifier.selfMatch(Staff.STAFF_SIZES[i]);
        }
    }

    function saveInk()
    {
        var ink = gestureClassifier.getCanvasDraw().getPencil().ink.serialize();
        var recognizedSymbols = gestureClassifier.getCanvasDraw().getStaff().
            serializeSymbols();
        // todo, add bitmap snapshot of recognition
        var c = gestureClassifier.getCanvasDraw().getCanvas();
        var imageData = c.toDataURL();
        // console.log(imageData);
        ink.imageData = imageData;
        ink.recognizedSymbols = recognizedSymbols;
        ink.version = CURRENT_INK_VERSION;
        if (ink)
        {
            db.save(ink);
        }
    }

    function autoPlayTests()
    {
        return document.getElementById("autoplayCheckbox").checked;
    }
    function useAutoLearn()
    {
        return document.getElementById("autolearnCheckbox").checked;
    }
    function useScaleTestMode()
    {
        return document.getElementById("scaleTestModeCheckbox").checked;
    }
    function outputCanvasSVG()
```javascript
{  console.log(gestureClassifier.getCanvasDraw().getContext().getSVG());
}

-->  
</script>
</head>
<body onload="handleOnLoad();">
  <div id="container">
    <canvas id="canvasView" width="1000" height="500"></canvas>
    <canvas id="canvas1" width="300" height="340"></canvas>
    <canvas id="canvas2" width="300" height="340"></canvas>
  </div>
  <div id="controls">
    <form id="options">
      <input type="checkbox" onclick="gestureClassifier.getCanvasDraw().getPencil().toggleShowInk();" value="showInk?" checked="true" />
      <div class="spacerHorizontal"></div>
      <input type="checkbox" onclick="gestureClassifier.getCanvasDraw().getPencil().toggleShowSymbols();" value="show symbols?" checked="true" />
      <br />
      <input type="button" onclick="gestureClassifier.getCanvasDraw().getPencil().redrawCanvas(-1);" value="-" />
      <input id="staffSizeRange" type="range" step="1" min="11" max="160" value="128" onchange="gestureClassifier.getCanvasDraw().getPencil().resizeToNearestStaffSize(this.value);" size="8" />
      <input type="button" onclick="gestureClassifier.getCanvasDraw().getPencil().redrawCanvas(1);" value="+" />
      <br />
      <a href="#" onclick="showHideAdvancedOptions();" id="showHideAdvancedOptionsLink">more options &hellip;</a>
      <div id="advancedOptions">
        <input type="button" onclick="clearAll();" value="clear all" />
        Ink Color<br />
        Red<input id="inkColorRed" type="range" step="1" min="0" max="255" value="0" onchange="changeInkColor();" size="8" />
        Green<input id="inkColorGreen" type="range" step="1" min="0" max="255" value="0" onchange="changeInkColor();" size="8" />
```
Training

../training.html
<meta name="viewport" content="width=device-width, user-scalable=no"/>
<meta name="apple-mobile-web-app-capable" content="yes"/>
<link rel="apple-touch-icon" href="/iquil_icon.png">
<title>iquil</title>

<!-- utilities -->
<script type="text/javascript" src="/common/Utils.js"></script>
<script type="text/javascript" src="/common/XMLHttpRequest.js"></script>
<script type="text/javascript" src="/common/Json.js"></script>
<script type="text/javascript" src="/common/json2.js"></script>

<!-- geometry -->
<script type="text/javascript" src="/common/Point.js"></script>
<script type="text/javascript" src="/common/Polygon.js"></script>
<script type="text/javascript" src="/common/PolygonIntersect.js"></script>

<!-- drawing -->
<script type="text/javascript" src="/CanvasDraw.js"></script>
<script type="text/javascript" src="/Pencil.js"></script>
<script type="text/javascript" src="/Ink.js"></script>

<!-- gesture recognition -->
<script type="text/javascript" src="/Gesture.js"></script>
<script type="text/javascript" src="/GestureClassifier.js"></script>

<!-- rastral, the notation rendering engine -->

<!-- font embeds -->
<link rel="stylesheet" href="/rastral/fonts/stylesheets.css" type="text/css" charset="utf-8">

<script type="text/javascript">
// define global fetamap var before loading the lilypond feta font maps
// this is really a C++ file, but will interpret fine as javascript
// so long as the var _fetaMap is defined first as an object
var _fetaMap = new Object();
// load feta (emmantaler) character mappings
</script>

<script type="text/javascript" src="/rastral/rastralList.js"></script>
<script type="text/javascript" src="/rastral/Rastral.js"></script>
<script type="text/javascript" src="/rastral/Staff.js"></script>
<script type="text/javascript" src="/rastral/Clef.js"></script>

<!-- training -->
<script type="text/javascript" src="/Trainer.js"></script>

<style type="text/css">
! * {
  -webkit-user-select: none;
}
* {

<style>
/*
 -webkit-touch-callout: none;
 */
*
*-webkit-tap-highlight-color: rgba(0,0,0,0);
*/
*
-webkit-text-size-adjust: none;
#
canvasView
{
 border: 1px solid #000;
}
</style>

<script type= text/javascript>
// global
const USE_CANVAS_SVG = false;
var trainer = null;
var isIPhone = navigator.userAgent.indexOf('iPhone') >= 0;
var symbolNames = Rastral.ALL_SYMBOLS;
var staffSize = 112;
function handleOnLoad()
{
 trainer = new Trainer("trainingOutput", staffSize);
}
</script>

<body onload="handleOnLoad();">
<div id="container">
  <canvas id="canvasView" width="300" height="340"></canvas>
</div>
<form>
  <input type="button" value="back" onclick="trainer.previous()"/>
  <input type="button" value="redisplay" onclick="trainer.train()"/>
</form>
Instructions: draw each symbol<br/>
<div id="trainingOutput">
</div>
</body>
</html>
11 Presentation Slides

These slides can also be found in HTML format on the Word Wide Web at http://slides.iQuil.com/.

Figure 15: Slide 1

Quil:

An experimental system for online music notation handwriting recognition

These slides can be found at slides.iQuil.com
Figure 16: Slide 2

Who?
A little about me:
I like beautiful things. I like making things. I like solving problems with my intuition.
I’ve been called engineer, I’ve been mistaken for a mathematician, and there’s a piece of paper I’ll soon be awarded that says something about mastering science.

Figure 17: Slide 3

Who?
A little about me:
But, really I’m a musician; I’m a composer. Technology is only important when solving human problems.
A musician friend of mine recently referred to me as “Mr. Luddite” and I took it as a compliment.
Figure 18: Slide 4

Why?

A music notation user interface should allow the user to sketch music notation quickly enough before they forget it.
The tools used shouldn’t get in the way and shouldn’t force you to think in a certain way.
And, most importantly, the tools shouldn’t force any decisions to be made prematurely for the tools’ sake.

Figure 19: Slide 5

Current state of the art for music notation handwriting recognition

Several research systems for gestural or online input have been developed, notably by Susan E. George at the University of South Australia, and at work at IBM’s T.J. Watson Research Center.

Purely gestural interfaces should be differentiated by their requirement of learning a specialized set of gestures that are not necessarily related to CWMN.

There has been much more work done in offline optical music recognition (OMR) for both handwritten and typeset CWMN, with significant contributions from Ichiro Fujinaga at McGill University. Offline recognition is fundamentally different in that it deals with pre-existing printed documents.

There are several commercially available offline OMR programs that work on scanned images of typeset scores.
Current state of the art for commonly-used music notation software

Presently, a user typically enters music notation with a combination of a mouse, alphanumeric keyboard, and/or a piano style keyboard. Current systems often use a palette of symbols that can be selected and placed on a staff. If I begin sketching, how do I get to this mess...

Finished product

... to this, a finished product to be published and read by others.

The Bride

score typeset with Nightingale music notation software
Demonstration

Keep in mind this is experimental software. It is an inchoate implementation, components of which are naively implemented and still include a hack or two as demonstrated.

In other words, things might break.

---

how does it work?

1. **segmentation**: which combination of ink segments form a symbol.
2. **classification**: what is the most likely symbol for a collection of ink.
3. **language model**: how the discrete symbols fit together in a meaningful way.
1. segmentation

The user-drawn ink is segmented to determine which combination of strokes are most likely to combine to form a symbol.

if these five strokes are drawn

and the remaining four strokes

the first stroke forms a treble clef

combine to form a sharp
2. classification (templates)

Classification always works on user-drawn templates of ink. Drawn symbols are often very different from typeset symbols.

Here is a demonstration of how the training works.

These are all the symbols currently recognized:

\[ \text{CWMN symbols} \]

This is only a small subset of all the symbols that constitute CWMN, and only includes discrete symbols.

2. classification (matching)

After both template strokes and input strokes are normalized by scale and to an equidistant fixed number of points, template matching is performed by calculating the mean of the distances from each drawn point to the nearest hand-trained template point and vice-versa. In other words, this comparison is bi-directional between drawn ink and the template. This could also be described as nearest neighbor Euclidean distance.
Figure 27: Slide 13

3. language model

A hand-coded collection of bigrams define how likely one symbol is to follow another.

![Music notation example](image)

Figure 28: Slide 14

Typesetting

The music notation is typeset on the HTML5 Canvas using glyphs constructed from the embedded Feta (from Lilypond).
**Figure 29: Slide 15**

User Correction of symbols

Symbols can be corrected by the user. This feedback informs the classifier.

"Humans are always going to win." --Douglas Eck

---

**Figure 30: Slide 16**

**Future work**

Improved classifier (either an Artificial Neural Network or a Support Vector Machine)

Train the system on thousands of handwritten examples.

Continuous symbols: beams, slurs, ties.

Combining handwriting with real-time audio input for pitch (singing, for example).

Notation improvements: Including chords, multiple voices, multiple staves.

"Conventional Western music notation does not have well-defined borders; it fades away indefinitely in all directions." --Don Byrd

"There's always one more thing." --Chris Raphael
Acknowledgements

Thanks to Professors Christopher Raphael and Larry Yaeger for their guidance on this project.
Thanks to my wife for reminding me on a daily basis of her intense love of music and her intense hatred of new technology.
Thanks to Donald Byrd, who has taught me a tremendous amount about music notation and music notation software.
Thanks to Samuel Daunt for playing the viola for us today.
Thanks to everyone who has given the time to provide insight and feedback on this project, especially the HCI/design students and faculty.
Thanks to my friends and family.
Thanks to my undergraduate capstone students.
Thanks to the folks at the Revolution Bike and Bean for feeding my caffeine addiction. Thanks to Kate Grigg there for putting me in touch with Sam this afternoon.
Thanks to anyone else I forgot to mention here (you know who you are).

Questions, Show and Tell

Any volunteers with examples for Sam to perform? Any volunteers want to give the whiteboard a try?

As demonstrated: iQuil.com

Bleeding edge (better in theory, but probably more broken): bleed.iQuil.com

Brief demonstration video: demo.iQuil.com

These slides: slides.iQuil.com

Please continue the dialog: chargin at indiana dot edu

"The belief in a certain idea gives to the researcher the support for his work. Without this, he would be lost in a sea of doubts and insufficiently verified proofs." – Konrad Zuse
12 Acknowledgements

I would like to extend my deepest thanks to Professors Christopher Raphael and Larry Yaeger for their guidance on this project. I would have been content to be a fly on the wall of a room in which these men conversed about pretty much anything. But, as it was, they were talking about something of the deepest interest and I was part of the conversation.

I am grateful to my wife for reminding me on a daily basis of her intense love of music and her intense hatred of new technology.

I would like to extend my thanks to Donald Byrd, who has taught me a tremendous amount about music notation and music notation software. I will remember fondly our frequent discussions in the bowels of Cook Music Library.

While working with Joe Berkovitz on his music notation editor Noteflight, I learned countless lessons about writing software for music notation. During this period, he was a great teacher to me, even though that was not in his job description. The depth and breadth of Joe’s knowledge and ability would make it next to impossible to write such a job description.

Thanks to everyone who has given the time to provide insight and feedback on this project, especially the HCI/design students and faculty.

My undergraduate capstone students provided me with constant inspiration through their efforts; students are often the best teachers. I am grateful for the tremendous wealth of code examples and tools available on the internet.

Finally, I would like to extend thanks and apologies to anyone else I may have inadvertently left off this list.