2012 10th IAPR International Workshop on Document Analysis Systems

Linear Compression of Digital Ink via Point Selection

Vadim Mazalov and Stephen M. Watt
Ontario Research Centre for Computer Algebra
Department of Computer Science
University of Western Ontario
London, Canada
vmazalov@uwo.ca, Stephen.Watt@uwo.ca

***Abstract*—We present a method to compress digital ink based on piecewise-linear approximation within a given error threshold. The objective is to achieve good compression ratio with very fast execution. The method is especially effective on types of handwriting that have large portions with nearly linear parts, e.g. hand drawn geometric objects. We compare this method with an enhanced version of our earlier functional approximation method, finding the new technique to give slightly worse compression while performing significantly faster. This suggests the presented method can be used in applications where speed of processing is of higher priority than the compression ratio.**

***Keywords*-digital ink; compression; sandwich algorithm; functional approximation**

I. Introduction

Handwriting is one of the most common forms of human expression and, today, in this information era, pen-based devices are able to capture handwriting in digital form. Ink is typically represented as a sequence of points sampled from a traced curve, often taken uniformly in time. Points are typically given as x and y values in a rectangular coordinate system, $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$, but other coordinates, such as pressure and angles, may be given as well. The sampling frequency and spatial resolution of hardware has been increasing over time, creating opportunities and challenges for ink processing applications. The opportunities are associated with the possibility of more detailed analysis, since a device can capture in high precision variations of pen movement. On the other hand, such high volumes of ink data require extra resources for processing and storage.

In this work we address the question of how to preserve the high precision of a curve, while decreasing the number of points representing it. In simple terms, this problem is solved by removing points that do not affect the shape of the curve significantly, while the error between the original and the approximating curves remains within a threshold. The method can be viewed as a dynamic adjustment of the density of points, depending on the shape of a stroke. More points are removed from straighter regions than regions with high curvature. Thus, we would expect geometric drawings with many lines to compress particularly well.

We have two subproblems that need to be solved:

1) decomposition of digital ink into pieces, suitable for compression, and
2) compression of the individual pieces.

We present fast, easy to implement solutions to both of these problems and show experimentally that the technique yields good compression for handwritten text and even better compression for hand drawn geometric objects. The discussed method is most useful for compression of linear pieces of a curve and can be implemented as a part of a multipurpose hybrid compression algorithm.

We also implement an enhanced version of the compression method [1], based on functional approximation, by representing coefficients in a more compact form. We measure the compression rate and time required to process the experimental datasets and compare with the performance of the linear method. While losing in compression, the linear method is found to perform more than $100\times$ faster.

The paper is organized as follows. Section II gives a few background comments on existing compression approaches and on linear approximation. An improvement to the functional approximation method is proposed in Section III. The linear compression algorithm is explained in Section IV. Section V presents details about the experimental setting and the results obtained. Section VI concludes the paper.

II. Related Work

Digital ink compression: A number of digital ink compression algorithms have been developed to date. One of the most popular lossless schemes is to use second differences followed by secondary compression [2]. The algorithm computes the second order differences of the data items in each information channel. For X the second difference is

$$\Delta_i^2 X_i = \Delta_i(X_{i+1} - X_i) = X_{i+2} - 2X_{i+1} + X_i.$$

As consecutive coordinate values tend to be close, the first differences will be small, and the second differences smaller and suitable for an entropy encoding algorithm.

An efficient lossy method was developed in [1]. This was based on piecewise functional approximation of curves by truncated orthogonal polynomial series and representation

978-0-7695-4661-2/12 $26.00 © 2012 IEEE
DOI 10.1109/DAS.2012.51

of the pieces by the approximating series coefficients. The desired approximation accuracy is achieved by dynamically changing the degree of approximation and the size of pieces.

Another lossy algorithm was presented in [3], based on stroke simplification. It suggests to eliminate excessive points, forming a skeleton of the original curve. The algorithm is based on iterative computation of chordal deviation – the distance between the original curve and its approximation. Points with the minimal distance are removed until the distance becomes larger than a threshold. A "substantially lossless" method was proposed in [4]. It allows the compression error's magnitude to be not greater than the sampling error's magnitude. In this approach, the original curve is split into segments and each segment is represented by some predefined shape, such as a polygon, ellipse, rectangle or Bezier curve. It is not mentioned how to obtain the shapes from a curve and what compression this approach gives.

Approximation of univariate convex functions: Several "sandwich" algorithms have been proposed for approximation of univariate convex functions. For example, see [5] for a method that requires derivative information along with the function values, and [6] for an iterative algorithm when only function values are available. The latter technique can be briefly described as follows. Consider a convex function defined on an interval I and some threshold of approximation error δ. Approximation on the interval is obtained by joining its boundary points. Let the approximation error of the interval I be δ_I and $\delta_I > \delta$. Then the interval I is split in subintervals, according to a partitioning rule. The procedure is repeated until the approximation error becomes less than δ for each subinterval. Several partitioning rules are considered, e.g. the maximal error rule that selects the point located on the maximal distance to the approximation curve. The algorithm converges quadratically if certain conditions on derivatives are satisfied, and linearly under other conditions.

Decomposition of digital curve in inflection-free parts: Several methods exist for decomposition of digital curves in segments without inflection, e.g. see [7], [8]. However, these algorithms are primarily designed for digital images to extract convex/concave pieces of an object to determine meaningful parts. In contrast, we are interested in the decomposition of digital ink. We note that the methods developed for binary images are in most cases not suitable for our purpose, since digital ink is represented as a sequence of points on a curve, rather than as a field of pixels in two dimensions.

III. Enhanced Compression via Functional Approximation

We propose a way to improve the functional approximation technique developed in [1]. As mentioned earlier, that method is based on piecewise approximation of curves by truncated series in an orthogonal polynomial basis. In [1] we experimented with Chebyshev, Legendre, Legendre-Sobolev polynomials and Fourier series and found Chebyshev polynomials to yield the best compression, as expected. In this work our goal is to improve performance of the method with Chebyshev polynomials as the orthogonal basis. The improvement is to be achieved by representing coefficients in a more compact form.

We consider the adaptive segmentation scheme of [1]. For each trace, the degree d of the approximation is selected dynamically. A higher degree provides a more accurate approximation of a curve, but increases the compressed size. In the adaptive scheme, the size of coefficients is also selected for each trace independently. Coefficients are recorded as floating-point numbers with base 2. The significand and the exponent are two's complement binary integers, encoded in a and p bits respectively. The value of p is fixed, and the value of a is dynamically adjusted for each stroke. The following representation of each information channel of a trace i is proposed:

- Encode the 0 order coefficient in $2a+p$ bits, since this coefficient regulates the initial position of the trace and is typically larger than the rest of the coefficients. This number of bits is device-dependent.
- Find the coefficient $c_M = \max|c_i|, i = 1..d$ and encode it in $a+p$ bits.
- Encode coefficients $c_j, j = 1..d$, as two's complement binary integers $r_j = \left\lfloor \frac{|c_M|}{c_j} \right\rceil$ in b_r bits, where $\lfloor x \rceil$ represents rounding of x to the integer.

Thus, a trace i is recorded as

$$a_i d_i \lambda_1 c_{10} c_{1M} r_{11} ... r_{1d_i} \lambda_2 c_{20} c_{2M} r_{21} ... r_{2d_i} ... \lambda_D$$

where a_i is the number of bits for encoding the significand; d_i is the degree of approximation; λ_j is the initial value of parameterization of a piece j; c_{j0} is the 0-order coefficient; $c_{jM} = \max|c_{jk}|, k = 1..d$; $r_{jk} = \left\lfloor \frac{|c_{jM}|}{c_{jk}} \right\rceil$, c_{jk} is the k-th coefficient of the j-th piece. This differs from the method of [1] by having the coefficients c_j represented as scalings rounded to integers rather than as significand-exponent pairs.

IV. The Linear Compression Algorithm

A. Decomposition into inflection-free parts

The method described in [6] is not suitable for digital ink as originally presented, since it requires parameterization and segmentation. We develop a method that does not require parameterization and can be used as the first step in processing.

Our compression method works with pieces locally curving in one direction or the other, but not changing back and forth. To be more precise, the curve should be decomposed into parts where the second derivative has constant sign, i.e the normal vector in the Frenet frame is pointing to the same side of the curve.

Algorithm 1 FormInflectionFreeSegments()

Input: *Points* – a stream of input points
Output: C – a list of inflection-free segments

```
C ← [ ] {list of inflection-free segments found}
S ← [ ] {current segment being collected}
i ← 0 {index of current point without duplication}
while Points.hasNext() do
  P ← Points.getNext()
  if i = 0 or P ≠ P_{i-1} then
    P_i ← P
    if |S| ≥ 2 then
      if P_i = P_0 then
        Append the list S to the end of the list C
        S ← [ ]
      else
        A_i   ← Angle(P_{i-2}, P_{i-1}, P_i) − π
        A_Beg ← Angle(P_i, P_0, P_1) − π
        A_End ← Angle(P_{i-1}, P_i, P_0) − π
        if A_i × A_{i-1} < 0
        or A_i × A_End < 0 or A_Beg × A_End < 0 then
          Append the list S to the end of the list C
          S ← [ ]
        end if
      end if
      Append P_i to the end of the list S
      i ← i + 1
    end if
  end if
end while
If S is non-empty, append it to the end of the list C
return C
```

Definition We say that a sequence of points $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ is an *inflection-free segment* if and only if the polygon formed by these points, after joining (x_1, y_1) and (x_n, y_n), is convex.

The property of a convex polygon that every internal angle is less than or equal to π is used in the *online* decomposition Algorithm 1. The algorithm, in the body of the **while** loop, lists operations performed on each incoming ink point to obtain a sequence of inflection-free segments. This takes into account that

- Two points are considered equal, if their coordinates are equal.
- $|P|$ denotes the number of points in the list P.
- $\mathrm{Angle}(P, Q, R)$ is the "oriented" angle between vectors $\overrightarrow{QP}$ and $\overrightarrow{QR}$. In other words, $\mathrm{Angle}(P, Q, R) = 2\pi - \mathrm{Angle}(R, Q, P)$. These angles can be found with the dot and cross products of given vectors.
- A_{Beg} is the complement of the oriented angle made by the beginning vector $\overrightarrow{P_0P_1}$ and the last point. A_{End} is the complement of the oriented angle made by the ending vector $\overrightarrow{P_{i-1}P_i}$ and the first point. A_i is the complement of the oriented angle made by the most current three points.
- We test for products less than zero to detect changes in direction of curvature. Two angles in the same direction will give a positive product (either as $+\times+$ or $-\times-$) and three collinear points will give a zero product.

B. Compression of inflection-free parts

Once the curve is decomposed as a collection of inflection-free segments, each piece is a subject to compression. Our compression technique is similar to the sandwich algorithm proposed in [6]. However, rather than looking at the lower and upper bounds of a function, we find the distance between a curve and its approximation. If either the maximal error $||\cdot||_{\max}$ or the root mean square error $||\cdot||_{\mathrm{rms}}$ on an interval is greater than the respective thresholds $\epsilon_{\max}$ or ϵ_{rms}, the curve is split into two parts. Other norms on the space of curves could be used if desired. The steps are presented in Algorithm 2, considering that $j.\mathrm{first}$ and $j.\mathrm{last}$ are respectively the first and the last points of the interval j.

Definition We write $\mathrm{pw}(L)$ for the piecewise linear curve defined by the list of points L. If two points a and b occur in a list L, with a preceding b, then we say that $[a, b]$ is an *interval* in L. We write $L|I$ for the sublist of L restricted to the interval I.

The point of division is found with one of the partitioning rules:
Rule 1: Based on the maximal *distance*: the decomposition point is selected based on the distance from the point to the line that goes through the boundary points of the interval.
Rule 2: Based on the *angle* formed at the point: if all of the oriented angles within the segment are less than π then the minimal angle is considered, otherwise (when all of the angles are greater than π) the maximal angle is found.

C. Complexity

The decomposition algorithm processes each incoming point in constant time $O(1)$. There are no additional operations at the last input. It is online, in that after each point a valid decomposition is maintained.

The best case time complexity of compression of a piece is $O(n)$. If the splits always divide a segment into two equal parts, and continue until there is a split at every point, the cost is $O(n \log n)$. If the splits are made unequally, always splitting n points as 1 and $n-1$, then the cost is $O(n^2)$.

D. Correctness

The termination condition of CompressCurve merits attention. If a function satisfies a maxnorm bound on each element of a partition, then it satisfies the maxnorm over the union of the parts. For RMS, note that if a domain D is partitioned as $D_1, ..., D_n$ and $\sqrt{\sum_{a\in D_i} f(a)/|D_i|} < \epsilon$, then $(\sum_{a\in D_1} + \cdots + \sum_{a\in D_n}) f(a) < (|D_1| + \cdots |D_n|)\epsilon^2$ so $\sqrt{\sum_{a\in D} f(a)/|D|} < \epsilon$, and take $f(a) = (S(a) - S^*(a))^2$.

Algorithm 2 CompressCurve(S, R)

Input: S – a list of points for an inflection-free segment
R – a partitioning rule (rule 1 or 2)
Output: L – a list of points such that
$||\mathrm{pw}(S) - \mathrm{pw}(L)||_{\max} < \epsilon_{\max}$ and
$||\mathrm{pw}(S) - \mathrm{pw}(L)||_{\mathrm{rms}} < \epsilon_{\mathrm{rms}}$

```
{J is a stack of intervals to be refined.}
J ← [ Interval with first and last point of S ]
L ← [ ]
while J ≠ [ ] do
    j ← Pop an interval from J
    a ← j.first; b ← j.last
    if ||pw(S|j) − pw(j)||_max > ε_max
    or ||pw(S|j) − pw(j)||_rms > ε_rms then
        {Split j according to rule R at some point c in S}
        j1 ← [a, c]
        j2 ← [c, b]
        Push j2 and then j1 onto the stack J
    else
        Append a and then b to the end of list L
    end if
    Remove element j from J
end while
return L
```

E. Discussion

Binary Encoding of Points: The sequence of points of a compressed trace can be encoded in binary for compact representation. Coordinates in our dataset have absolute value not greater than 2^{13} and can be recorded as two's complement integers in a sequence of groups of 14 bits.

Drifting of Approximation: The presented compression method is not suitable for repeated resampling. While the approximation to each inflection-free segment will lie within any required error bound, the approximation will lie completely on one side of the input curve. If the resulting piecewise linear function is then resampled and recompressed repeatedly, systematic drift may occur. To address the issue of drift under repetitive resampling and recompression, the line segments could be positioned to cross the original curve so that the error is equal on both sides of the original.

V. Experiments

A. Experimental Setting

The experimental dataset was collected in the Ontario Research Centre for Computer Algebra with a tablet device with the following specifications: 2540 dpi resolution, 133 pps data rate, and $\pm.02$ sampling error.

Two types of digital ink were collected for the experiments

- Handwriting. Different individuals have provided various parts of regular English text to ensure variations in length of strokes and writing styles. From the whole collection, we randomly selected 46 traces containing, on average, 51 points each.
- Geometric objects. We collected simple two-dimensional geometric objects, such as triangles, rectangles and lines. Then we randomly selected 33 traces containing, on average, 68 points each.

In the experiments, the root mean square error was taken as a portion of the maximal error $\epsilon_{\mathrm{rms}} = \frac{3}{4}\epsilon_{\max}$. Unlike the results reported in [1], we look at the absolute, not relative, approximation error and the binary stream of coefficients does not undergo further gzip compression. The compressed size is reported as S_c/S_o where S_c is the size of the compressed dataset and S_o is the size of the original dataset.

The compression algorithms were implemented and run on Maple 13 on an Intel Core 2 Duo 2.40 GHz CPU with 2GB RAM, running Ubuntu Linux version 2.6.24-19-generic.

B. Experimental Results

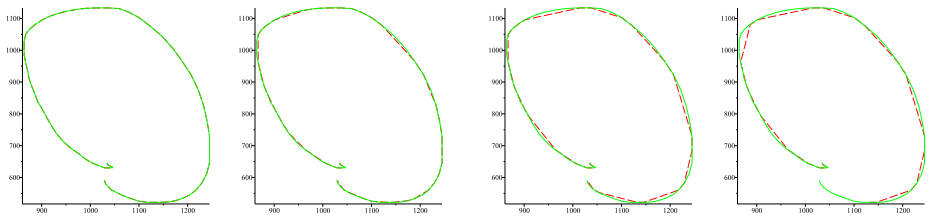
Optimal values of p and b_r: In the experiments we measure compressed size for different values of approximation error. Figure 1 shows an original curve and linear approximation for different maximal error thresholds. From the figure, one can observe that compressing the curve with the maximal error of up to 5 has almost no effect on representation of the curve and can be used in the applications that do not require high precision of ink, e.g. recognition.

In the first set of experiments, we look for the optimal values of p and b_r, see Section III. With fixed $b_r = 7$, the value of p was changed and the compressed size was measured for both datasets. Results for handwriting and geometric objects are shown in Tables I and II respectively. The value of $p = 4$ was found to be the most efficient.

With fixed $p = 4$, b_r was changed to find the optimal value. The compressed sizes for the datasets of handwriting and geometric objects are shown in Tables III and IV respectively. The value of $b_r = 5$ was selected.

Comparison of functional approximation with linear compression: The compression rate of the linear method was measured for the two segmentation rules explained in Section IV-B on both datasets. Figure 2 presents the results of the functional approximation and linear compression methods for different values of $\epsilon_{\max}$. The partitioning rules show similar performance on the handwriting dataset and almost identical on geometric objects. As expected, due to the nature of the linear algorithm, we obtained higher compression of geometric objects than handwritten text. The functional approximation method shows similar performance on both datasets.

The compression time is given in Table V for the dataset of handwriting and Table VI for geometric objects. The linear method performs almost instantly, compared to the compression with higher-order functional approximation. One can observe a trend of increase of the execution time of



$\epsilon_{\max} = 1$ $\epsilon_{\max} = 5$ $\epsilon_{\max} = 10$ $\epsilon_{\max} = 15$

Figure 1. Approximation of a sample with different error thresholds (dash line) and the original curve (solid line)

Table I
COMPRESSED SIZE (%) AS A FUNCTION OF THE MAXIMAL ERROR ($\epsilon_{\max}$) AND THE NUMBER OF EXPONENT BITS (p) FOR 7 COEFFICIENT BITS (b_r) FOR THE HANDWRITING DATASET

p \ $\epsilon_{\max}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	21.5	14.7	12.5	11.6	10.3	9.6	9.0	8.5	8.1	7.9	7.6	7.3	7.1	6.9	6.7
4	19.2	13.2	11.3	10.3	9.2	8.7	8.1	7.6	7.3	7.1	6.8	6.5	6.3	6.2	6.0
5	19.0	13.6	11.7	10.8	9.5	8.9	8.3	7.9	7.5	7.3	7.0	6.7	6.5	6.4	6.2

Table II
COMPRESSED SIZE (%) AS A FUNCTION OF THE MAXIMAL ERROR ($\epsilon_{\max}$) AND THE NUMBER OF EXPONENT BITS (p) FOR 7 COEFFICIENT BITS (b_r) FOR THE DATASET OF GEOMETRIC OBJECTS

p \ $\epsilon_{\max}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	22.3	17.2	15.0	13.7	12.5	11.7	10.9	10.5	9.8	9.5	8.9	8.7	8.4	8.1	7.8
4	20.6	15.0	13.0	11.9	10.8	10.0	9.3	9.0	8.5	8.2	7.7	7.6	7.3	7.1	6.8
5	21.8	16.2	13.9	12.8	11.6	10.8	9.9	9.6	9.1	8.6	8.2	8.0	7.7	7.4	7.1

Table III
COMPRESSED SIZE (%) AS A FUNCTION OF THE MAXIMAL ERROR ($\epsilon_{\max}$) AND THE NUMBER OF COEFFICIENT BITS (b_r) FOR 4 EXPONENT BITS (p) FOR THE HANDWRITING DATASET

b_r \ $\epsilon_{\max}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	20.3	14.6	12.2	11.0	9.7	8.7	8.2	7.5	7.2	7.0	6.6	6.3	6.1	5.9	5.8
5	18.6	13.5	11.4	10.0	9.0	8.4	7.6	7.2	7.0	6.7	6.5	6.2	6.0	5.8	5.6
6	18.5	13.0	11.1	10.0	8.9	8.3	7.8	7.4	7.0	6.8	6.6	6.3	6.1	5.9	5.8
7	19.2	13.2	11.3	10.3	9.2	8.7	8.1	7.6	7.3	7.1	6.8	6.5	6.3	6.2	6.0
8	19.1	13.6	11.7	10.7	9.6	9.0	8.4	7.9	7.6	7.4	7.1	6.8	6.6	6.5	6.3

Table IV
COMPRESSED SIZE (%) AS A FUNCTION OF THE MAXIMAL ERROR ($\epsilon_{\max}$) AND THE NUMBER OF COEFFICIENT BITS (b_r) FOR 4 EXPONENT BITS (p) FOR THE DATASET OF GEOMETRIC OBJECTS

b_r \ $\epsilon_{\max}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	19.5	14.6	12.8	11.6	10.6	9.8	9.1	8.8	8.2	7.9	7.4	7.2	6.9	6.7	6.3
5	19.5	14.4	12.4	11.4	10.4	9.7	9.0	8.7	8.0	7.7	7.3	7.1	6.8	6.6	6.3
6	20.2	14.7	12.6	11.6	10.5	9.9	9.1	8.8	8.3	7.9	7.5	7.3	7.1	6.8	6.6
7	20.6	15.0	13.0	11.9	10.8	10.0	9.3	9.0	8.5	8.2	7.7	7.6	7.3	7.1	6.8
8	20.2	14.7	12.6	11.6	10.5	9.9	9.1	8.8	8.3	7.9	7.5	7.3	7.1	6.8	6.6

Table V
TIME (IN SECONDS) FOR COMPRESSION OF THE HANDWRITING DATASET

Method \ $\epsilon_{\max}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L	25	20	21	21	17	17	17	19	18	15	16	15	15	20	16
F	879	1083	1287	1498	1700	1982	2188	2326	2479	2618	2727	2915	3019	3138	3327

Table VI
TIME (IN SECONDS) FOR COMPRESSION OF THE DATASET OF GEOMETRIC OBJECTS

Method \ $\epsilon_{\max}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L	12	10	8	9	8	7	8	8	9	9	9	8	8	9	8
F	1188	1355	1781	2034	2185	2346	2475	2593	2710	2830	2980	3086	3180	3281	3333

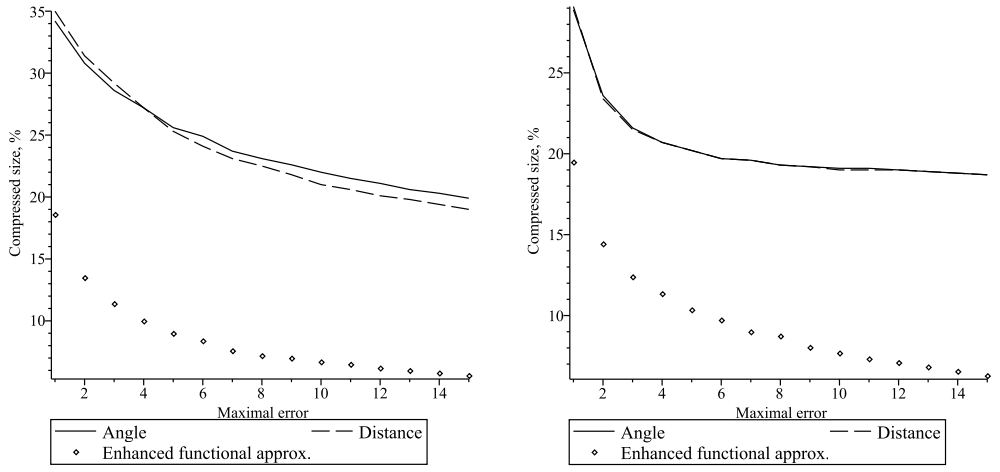


Figure 2. Compressed size depending on the maximal approximation error for handwriting and geometric objects: for Rule 1 (maximal distance) and Rule 2 (based on the angle), and for enhanced functional approximation

the functional approximation technique with the increase of the error threshold. In fact, the running time is around three times higher for $\epsilon_{\max} = 15$ compared to the execution time for $\epsilon_{\max} = 1$. This growth arises because more combinations of approximation degree and number of coefficient bits become suitable for approximation of pieces. Evaluation of those combinations is computationally intensive and can require significantly more time for high resolution devices.

VI. Conclusion

We have examined two methods for the compression of digital ink or, more generally, sampled curves in any dimension. One method selects a subset of the sample points to give a piecewise linear function that is within a given tolerance of the original. The second method adapts previous work based on orthogonal series approximation, representing the coefficients more efficiently. Our experiments show the piecewise linear approximation method to perform about $100\times$ faster than the functional approximation algorithm, but it yields a less compact representation. The proposed piecewise linear compression technique can be used when simplicity or speed are important, such as for hardware implementation and data transmission. On the other hand, the functional approximation method is suitable for applications that require compact storage of ink. Depending on the application and the choice of functional basis, in this representation certain recognition operations may be performed without decompression.

References

[1] V. Mazalov and S. M. Watt, "Digital ink compression via functional approximation." in *ICFHR'10*, 2010, pp. 688–694.

[2] *Ink serialized format specification*, Microsoft Inc., 2007.

[3] Z. Liu, H. S. Malvar, and Z. Zhang, "System and method for ink or handwriting compression," *United States Patent No US 7,302,106 B2*, November 2007.

[4] M. Chatterjee, "System and method for ink or handwriting compression," *United States Patent No US 6,549,675 B2*, April 2003.

[5] B. Fruhwirth, R. E. Burkard, and G. Rote, "Approximation of convex curves with application to the bicriterial minimum cost flow problem," *European Journal of Operational Research*, vol. 42, pp. 326–338, 1989.

[6] A. Y. D. Siem, D. d. Hertog, and A. L. Hoffmann, "A method for approximating univariate convex functions using only function value evaluations," *INFORMS J. on Computing*, vol. 23, pp. 591–604, Oct. 2011.

[7] I. Debled-Rennesson, J.-L. Remy, and J. Rouyer-Degli, "Detection of the discrete convexity of polyominoes," in *Proceedings of the 9th International Conference on Discrete Geometry for Computer Imagery*, ser. DGCI '00. London, UK: Springer-Verlag, 2000, pp. 491–504.

[8] H. Dorksen-Reiter and I. Debled-Rennesson, "Convex and concave parts of digital curves," in *Geometric Properties for Incomplete data*, R. Klette, R. Kozera, L. Noakes, and J. Weickert, Eds. Springer Netherlands, 2006, pp. 145–159.