

Compression of 2D Curves

Prashant Thakare
thakare@cc.gatech.edu
College of Computing
Georgia Institute of Technology

Justin Jang
jang@cc.gatech.edu
College of Computing
Georgia Institute of Technology

Abstract

Transmitting graphical data over the network for real-time rendering normally requires compressing it to meet bandwidth requirements of the system. Here, we address a subset of the problem by experimenting with various techniques for compression of 2D curves. Particularly, variants of two major techniques for compression - prediction (lossless) and simplification (lossy) - are tried. A Huffman encoding based codec and Hausdorff error are used to evaluate performance of the schemes. Results obtained are analyzed to evaluate their impact based upon the compression parameters and the curve chosen.

1. Introduction

Compression of graphical data becomes necessary mainly due to constraints imposed by computing resources such as network bandwidth, disk storage and sometimes, processing time. One of the simple compression mechanisms could be to leave the structure of the data unaltered while compressing the data itself.

Data compression can be achieved by mechanisms such as Simplification and Prediction. A Huffman encoding based codec can be used to store the data instead of raw integers to enhance compression.

For comparing performance of various schemes, metrics for compression achieved and error introduced are required. Compression can be measured by the bits taken by Huffman encoding per vertex while error can be measured by Hausdorff error between original and compressed data.

Various schemes implementing these techniques are tried below on data representing 2D curves. With such information about the data, specific schemes can be implemented to give better results.

We define a 2D curve as a list of ordered 2D coordinates where consecutive coordinates are connected with an edge. Compressing a 2D curve involves the following stages.

Normalization translates and scales an arbitrary curve into the range $[0, 1]$ (for x and y).

Quantization samples the normalized curve with a regular discrete grid in the range $[0, 2^B-1]$ (for x and y), where B is the quantization bits and is an external parameter.

Simplification is a lossy mechanism which calls for the removal of vertices that are not necessary for maintaining the shape of the curve with respect to a particular error tolerance.

Prediction is a lossless mechanism in which data is predicted using already available information storing corrections in prediction instead of the actual data.

Encoding a list of symbols is necessary for compression. In our case, these symbols will be the list of residual values left over from the prediction process.

2. Prediction

Prediction is used to guess the value of a vertex given one or more previous vertices in the curve. The error in prediction (residuals, i.e. symbols) is stored instead of data. The same prediction is done during decompression and residuals are used to correct the prediction.

By definition, a predictor that performs well will reduce the entropy (a measure of unpredictability) and thus facilitate better compression. Obviously, no single predictor is perfect for all curves, but we seek to find a predictor that performs better than others for curves of interest.

The following are schemes for predicting the next vertex (D) in the curve given one or more previous vertices (C, B, and A) in the curve:

Linear Predictor In this naïve predictor, next vertex is predicted to be same as current vertex. Thus vertex D is predicted as: $D = C$, assuming $V_{i<0}$ as 0.

Such prediction is also equivalent to storing displacements relative to the current vertex. Compression is achieved as the range of displacements is generally smaller than the range of coordinates in a curve. The predictor works better for curves with little variation in the edges.

First-Order Predictor Here, the next vertex is predicted to take same displacement as taken by current vertex. Thus, $D = C + \underline{BC}$.

This predictor takes into account one-level history of current vertex during prediction. Vertices are assumed to travel in straight lines storing the deviation and offset. The Predictor works better for curves with large linear portions in the curve failing at the discontinuities.

Second-Order Predictor This predictor increases the displacement taken by difference between current and previous velocities. Thus, $D = C + \underline{BC} + (\underline{BC} - \underline{AB})$.

Two-level history of the current vertex is used for prediction. Thus the predictor takes time to predict but starts predicting more accurately for curves with few variations. Certain time is again required to recover after a failure as it is propagated to one more level.

Taylor from B A parabolic trajectory is assumed to be taken by the vertex. Using Taylor series expansions from vertex B, $D = C + \underline{BC} + (\underline{BC} - \underline{AB})/2$.

First, shape of curves commonly occurred are approximated to a parabola. Second, a two-level history is used for the series expansion. Again this predictor is good for curves having small parts close to parabolic trajectory but need time to recover from failures.

Taylor from C Similar to above, this predictor takes into account the effect of only first order series expansion, $D = A + 3*\underline{BC}$

The recovery time is reduced here by taking little history into account at the cost of increased errors for regular vertices. While the parabolic patches of the curve have bigger residuals, the predictor can quickly transition between discontinuities.

Replica Displacement vector for D is obtained by rotating \underline{BC} by angle between \underline{BC} and \underline{AB} : $CD_x = BC_x \cos(\emptyset) + BC_y \sin(\emptyset)$, $CD_y = BC_y \cos(\emptyset) - BC_x \sin(\emptyset)$, and scaling by the ratio $s = |\underline{BC}|/|\underline{AB}|$, $D = C + s*CD$; where \emptyset is angle between AB and BC.

This predictor has a bias towards circular curves or symmetry. Effects of high leaps are nullified by bounding back by proportionate amounts. Instead of parabolic patches as above, this does well for circular patches but still suffers from the recovery issue.

3. Simplification

Simplification techniques are used to decrease the amount of data itself. A simple technique for simplifications could be removing duplicate neighboring vertices. This technique is generally lossless as it rids of only the extraneous data. Such redundancy may be created due to the quantization when a coarser quantization reduces neighboring vertices to same cell on the integer grid.

Simplification can be used with quantization for reducing the amount of data to a great extent. Finely sampled vertices on the curve can be replaced by edges without noticeable jaggedness to an extent, sometimes also controlled by the resolution required/used at the decompressor. Entropy may be reduced as a side effect of having a smaller data set. Another side effect of quantization+simplification is increased average edge length which is important factor in determining performance of predictors.

4. Implementation

Huffman Encoding

Our codec uses the Huffman encoding which maps a list of symbols to a list of binary encodings, one encoding per unique symbol. This mapping is called the *codebook*. Our algorithm for computing a Huffman encoding takes a list of integers as input and produces a codebook for each unique integer (symbol).

The first step counts occurrences for each symbol and orders a list of unique symbols using a priority queue with the lowest counts having priority. A Huffman tree is built from bottom up by iteratively grabbing two symbols (nodes) from the priority queue, creating a new parent node for those two nodes, and reinserting this new node into the priority queue. The count for the new parent node is simply the sum of the counts of the two child nodes. This continues until there is one node in the priority queue, which is the root of the tree. This algorithm produces a binary tree where all the leaf nodes represent a unique symbol. A traversal from the root to any leaf yields the encoding for the respective symbol, where traveling left appends a "0" bit and traveling right appends a "1" bit. Huffman decoding simply uses the input bit stream to drive a traversal of the Huffman tree. When a leaf node is reached, the symbol at that node has been detected and the traversal restarts at the root. Thus, a list of bits can be translated or *decoded* into a list of symbols.

Compressing the Tree

Because a Huffman tree is a binary tree, its structure can be encoded with a simple string of 1's and 0's. If we let "0" signify a non-leaf node and "1" signify a leaf node, a depth-first traversal results in a sequence of 1's and 0's that uniquely represents a binary tree structure. Furthermore, the end of the sequence is signified implicitly by the status of the traversal, i.e. if there are still any non-leaf nodes without two children (in particular, the right child is missing), the traversal continues to grab the next bit from the input sequence. The symbols at the leaf nodes can simply be stored in the order reached by the depth-first traversal.

5. Results

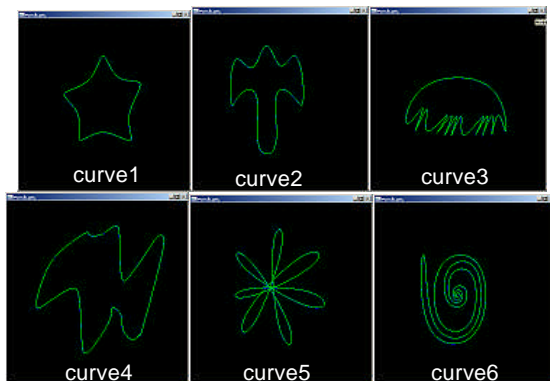


Figure 1: Various curves.

First order and Taylor C predictors, both giving less weights to predecessors, turn out to be winners due to their simplicity. Other predictors, while doing good job on patches of curve (or special curves), fail to be consistent in presence of discontinuities.

Circle seems to be an ideal curve for prediction as it has a constant angular velocity. Thus correct predictions would be followed by series of correct predictions, in absence of discontinuities. Replica, surprisingly fails to impress for this curve. The reason can be attributed to the preciseness required in the formulation, minor variations in \underline{AB} or \underline{BC} could largely change predicted \underline{CD} as it is scaled by their ratio. As circles used are quantized, \underline{AB} and \underline{BC} may not be exactly same around the arc.

For curves with various changes in the curve behavior but involving smoother transitions, a second order predictor fails at discontinuities and but gets enough time to recover. Expectedly, the entropy is much more than the predictable circle but all predictors

are equally good at predicting. While for curves with lots of sharp and short discontinuities, the liberty of allowing time to recover is taken off. Second order predictors suffer from this while simpler ones maintain their consistency.

A few curves were sampled densely in a part and coarsely in other, while still retaining relative smoothness. Surprisingly, higher order predictors fumble over this extra data, while the extra data should have added to the learning.

For entirely random curves, performance of all the predictors is almost the same. Linear predictor works a little better as there are no extrapolations over the large range. The prior history is rendered useless in such curves.

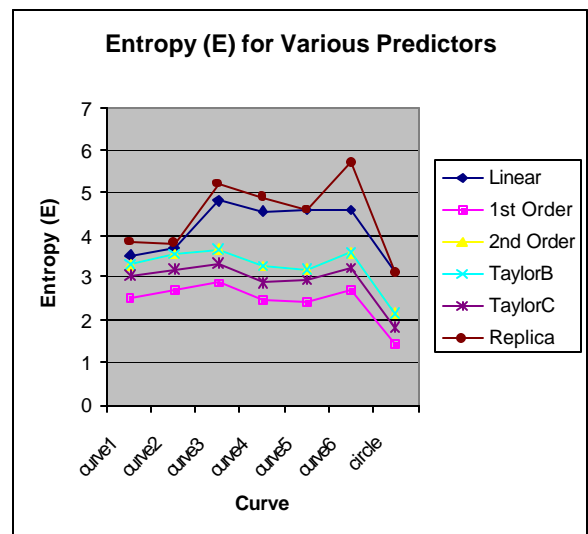


Figure 2: Entropy resulting from different predictors on the various curves.

The encoding used achieves entropy close to twice the theoretical best. It can be seen that the entropy is generally directly proportional to $\log(L)$ (where L is the average edge length), irrespective of the predictor used. Longer edges leave a possibility of wider choices of predicted vertex thereby increasing the range of residues - thus the entropy. For the 1st order predictor, the entropy of the simplified curve tended towards a common minimum for all the curves at around 6 bits. (Figure 3.) This shows that some simplification can aid compression but only to a limit.

The results show that using subdivision to recover a smooth curve from a drastically quantized curve results in only modest reductions, sometimes increases, in the Hausdorff error. Visual examination of the resulting curves confirms that the curves generally maintain the

stair-steps left over from the quantization. (Figure 4.) The B-spline subdivision did the best job of smoothing out these jaggies while 4-point subdivision seemed to exacerbate them. This is also reflected in the quantitative data. (Figure 5.) The exceptions occurred in curves with more varying sampling densities, where the 4-point performed best.

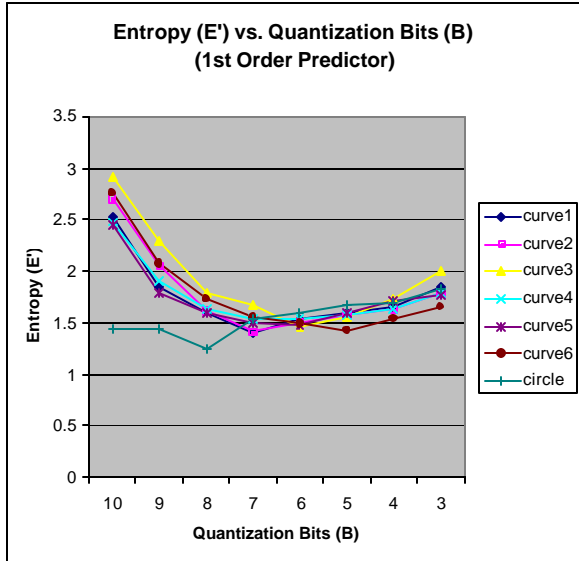


Figure 3: Entropy in the simplified curve vs. quantization bits for the 1st order predictor.

6. Conclusion and Future Work

We have presented results on compressing 2D curves using both lossy and lossless techniques. Simplification, while lossy, can in fact lower the entropy and increase the compression, but only to certain quantization value. Prediction is a lossless way of increasing compression. A significant gain in compression is attained even with a linear predictor. Among predictors, those that rely on greater-than-1 level history fair less favorably.

Using subdivision schemes to recover information from drastically quantized curves fail to recover the smoothness of the original curves.

For future work, the best quantization value can be obtained using heuristics and can be used in conjunction with simplification. In addition, better simplification schemes such as iterative vertex removal guided by actual curve shape can be used.

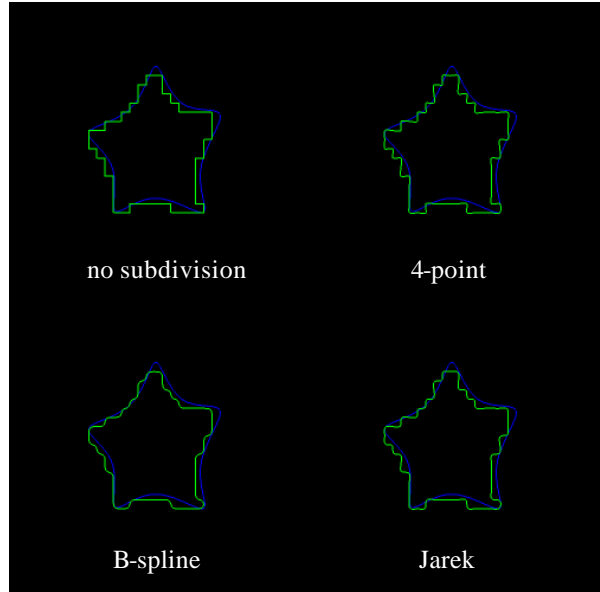


Figure 4: Effect of subdivision on appearance of coarsely quantized curves ($B = 4$).

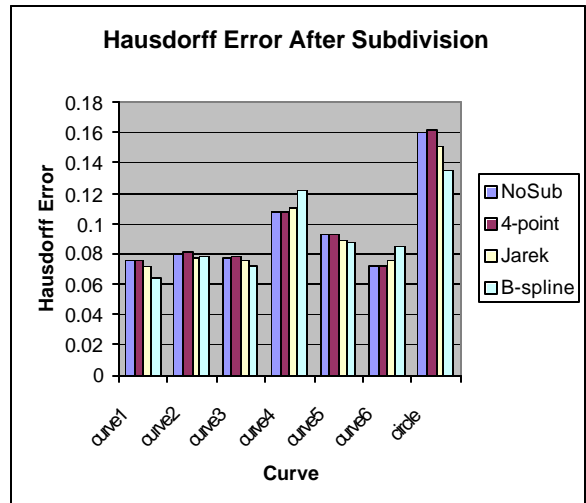


Figure 5: Effect of subdivision on Hausdorff error on coarsely quantized curves ($B = 4$).

7. References