COMPRESSION OPTIMIZED TRACING OF DIGITAL CURVES USING GRAPH THEORY

András Hajdu, Ioannis Pitas

Department of Informatics, University of Thessaloniki, Box 451, 54124 Thessaloniki, Greece
Email: pitas@aiia.csd.auth.gr

ABSTRACT

The use of an alphabet of line segments to compose a curve is a possible approach for curve data compression. An existing state-of-the-art method considers a quadtree decomposition of the curve to perform the substitution of the curve parts from the alphabet of line segments. In this paper, we propose a graph theory based algorithm for tracing the curve directly to eliminate the quadtree decomposition needs. This approach obviously improves the compression efficiency, as longer line segments can be used. We tune our method further by selecting optimal turns at junctions during tracing the curve. We also discuss briefly how other application fields can take advantage of the presented approach.

Index Terms— Graph theory, image coding, interpolation, piecewise linear approximation

1. INTRODUCTION

Digital planar curves are used in several fields of computer graphics, discrete geometry and digital image analysis. Many results have been revealed regarding their geometric behavior since [1]. A special topic is digital curve compression. Besides simple techniques like chain coding, a usual way is to partition the curve into straight line segments [2] for compression. These techniques usually focus on simple (non-intersecting) curves and assume the knowledge of the sequential order of the curve points. A state-of-the-art approach considers an alphabet of short line segments (beamlets) to compose the whole curve [3]. This method divides the curve into smaller parts using quadtree decomposition till having a *single* linear curve segment in every quadtree cell that can be substituted by a beamlet. The advantage of this approach is that any curve can be handled by sufficiently fine quadtree decomposition. However, a drawback is the obligation of decomposing further, when a cell contains segments that already could be coded separately.

In this paper, we propose an approach to trace curves having arbitrary topology to improve compression ratio, when splitting the curve into straight line segments. It will become obvious how the method improves efficiency regarding [3].

The structure of the paper is as follows. In section 2 we recall the graph theoretical background that serves as a basis to trace curves. We also explain how the suitable graph representation of the digital curve can be obtained. Section 3 describes how our technique is adjusted to be optimal for coding the curve with straight line segments. The method selected for compression and the related results are presented in section 4. Finally, some conclusions on other possible applications are discussed in section 5.

2. TRACING CURVES USING GRAPH THEORY

In this section we recall some notions and results of graph theory that are considered to trace a curve. Moreover, we explain the techniques that were needed to obtain the suitable graph representation.

2.1. Graph theoretical background

The *graph* G is defined as a pair (V, E), where V is a set of *vertices*, and E is a set of *edges* between the vertices $E = \{(u, v) \mid u, v \in V\}$. As our aim is to give graph representations of curves, we focus only on *undirected* graphs, so $\forall u, v \in V : (u, v) = (v, u)$, and thus will use the notation $\{u, v\}$ (sets of vertices rather than ordered pairs). To cover a wide class of curves, we allow *loops* (edges of type $\{u, u\}$) and *multiple* edges. The *degree* of a vertex is the number of edges containing the vertex. A *path* is a list of edges $\{u_1, u_2\}, \{u_2, u_3\}, \ldots, \{u_{n-1}, u_n\}$ with $u_1 = u_n$ in the case of a *route*. G is *connected*, if its every two vertices have a path connecting them. An edge is called a *bridge*, if its removal makes a connected graph disconnected. A path through G which includes every edge exactly once is called *Euler path* and *Euler route* if the same vertex is the start and end of the path [4, 5]. Note that any Euler route is also an Euler path. G is an *Euler graph*, if it has an Euler path. An *Euler decomposition* of G has the form $G = \cup_{i=1}^{n} G_i$ such that all the G_i's are disjoint (no edges in common) Euler graphs. We recall some well-known facts on Euler graphs and their decomposition [6, 7]:

i) Every Euler graph is connected.

Research was partly supported by the project SHARE: Mobile Support for Rescue Forces, Integrating Multiple Modes of Interaction, EU FP6 Information Society Technologies, Contract Number FP6-004218.

ii) A connected graph contains an Euler route iff all of its vertices have even degree. The route can start from any vertex.
iii) A connected graph contains an Euler path iff at most two of its vertices have odd degree. The path starts from any of the vertices having odd degree and ends in the other one.
iv) Every connected graph can be decomposed into disjoint Euler graphs.

We are ready to summarize our approach in the following (CT) algorithm for the complete tracing of a curve:

CT algorithm

---

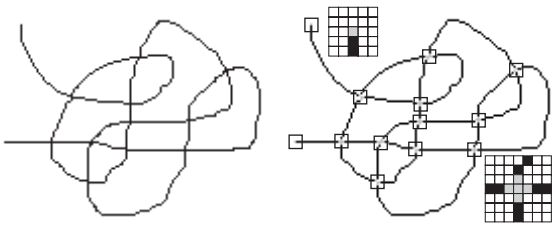
1. Transfer the curve C to a graph $G_C = (V_C, E_C)$, where V_C contains all the end and crossing points of C. The edges are the curve segments connecting them.

2. Create an Euler decomposition $\cup_{i=1}^{n} C_i$ of C based on G_C.

3. Trace all the C_i's separately through their Euler paths.

2.2. Extracting a graph from a digital curve

The first step of the CT algorithm can be realized easily using classical results of discrete geometry. Namely, to determine the vertices, the end points of C can be located as the ones having exactly one 8-connected neighbor. On the other hand, as we consider one pixel wide curves, we locate crossing points as the ones that have more than two 8-neighbors. If the input curve is not one pixel wide, we can apply a general preliminary thinning step to make it so. Figure 1 depicts the result of locating end- and crossing points (shown in gray). The crossing points belonging the same junction will define only one V_C vertice.



(a) (b)

Fig. 1. Locating vertices for the graph representation G_C. (a) Input planar curve C. (b) Extraction of the end and crossing points to act as vertices (examples zoomed for both types).

After extracting V_C from C, we determine the edge set E_C. For this task, first we locate edge end points as such non-vertex points that are 8-neighbors to vertices, and have exactly two 8-neighbors in C (if both of their 8-neighbors are vertices, the edge is degenerated having length of 1). Then, the edge end points are organized into pairs (edges) based on the condition that an 8-connected path can be found between them, whose elements are non-vertices. See Figure 2 for a close look on the selection of edge end points (shown by dots), and for the path (edge) points defined by them (shown by $\times$ marks).

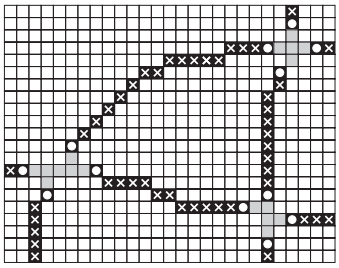


Fig. 2. Extracting edges by keeping curve geometry via locating edge end points and connecting them with paths.

Every two edge end points define exactly one edge. Note that loops and multiple edges are also handled by this approach without any difficulties. To find the 8-paths between edge end points we can use the recursive Floodfill8 algorithm starting from the edge end points. Thus, in the end, we will have the G_C representation of the curve C, shown as a simplified graph in Figure 3. The indices are assigned in the order of the vertex scanning procedure.

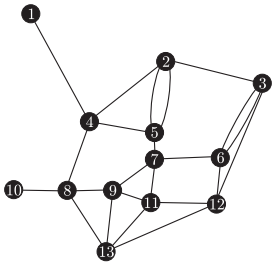


Fig. 3. The graph representation of the planar curve C.

The edges for E_C are stored as paths (curve segments) as a complete description of the original geometry. In the rest of the paper, we will restrict our investigation to such curves whose graph representation contains an Euler path.

3. OPTIMIZING TRACING FOR COMPRESSION

The first step to trace an Euler curve is to locate a starting vertex according to *iii)* in section 2.1. We check the vertices and select one having odd degree (e.g. end point of C). If all the degrees are even, we can choose an arbitrary vertex to start from. Then we take an edge from the starting vertex to initialize the tracer. For example, in the graph shown in

Figure 3 two vertices (#1, and #10) have odd degree. Thus, the Euler path should start from vertex #1 to finish at vertex #10, or vice versa.

As more Euler paths may exist, we have to decide which edge to take next, when reaching crossings. Our intention is to substitute the curve part with straight line segments for curve compression. The natural decision is to go in the most straight way through a junction, as we can expect the most optimal substitutation with a line segment in this case. Thus, let us assume that we arrive at such a crossing with an edge end point E_0, which has $E_1, \ldots, E_k$ more edge end points not visited yet. We calculate the centroid of the crossing as:

$$\overline{E} = \frac{1}{k} \sum_{j=0}^{k} E_j. \tag{1}$$

$\overline{E}$ can be rounded to have integer coordinates, or considered as a real valued vector, as well. Let α_i denote the angle $\angle E_0 \overline{E} E_i$. The direction defined by the E_l edge end point is said to be optimal, if:

$$|180^\circ - \alpha_l| = \min_{j=1}^{k} \{|180^\circ - \alpha_j|\}. \tag{2}$$

See Figure 4a for an example on how the decision is made in tracing continuation based on the above optimal selection. To extract a path between E_0 and E_l we can use the Floodfill8 algorithm again. Now, we have to start Floodfill8 from E_0 to connect the vertex points of the crossing with selecting the path with the minimal length. See Figure 4b for the final path through the crossing.

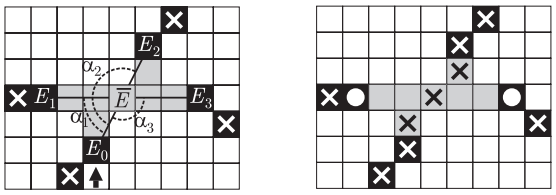


Fig. 4. Tracing through crossings optimally. (a) Finding the most straight direction. (b) Tracing through the crossing by connecting edge end points.

Using the above optimal decision, we are able to trace the whole curve along one of its Euler paths. The traced curve is composed by concatenating the edges with the short segment going through the vertices. For the full tracing of the curve see Figure 5, where, besides the start and end point of the path, arrows show the optimal traverse directions at the crossings. Using the vertex indexing in Figure 3, the Euler path is: {1,4,5,2,4,8,13,11,7,5,2,3,6,7,9,13,12,6,3,12,11,9,8,10}.

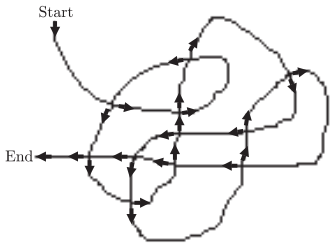


Fig. 5. Tracing the whole curve by choosing optimal directions at crossings.

During the extraction of an Euler path, note that we are not always free in choosing the most optimal direction at crossings. Fleury's algorithm guarantees to find an Euler path, and has higher priority in the combination with our optimal tracing method regarding its classic recommendations [8, 9]:

• Always leave one edge available to get back to the starting vertex or to the other odd vertex.

• Do not use an edge to go to a vertex unless there is another edge available to leave it.

4. COMPRESSING G_C WITH LINE SEGMENTS

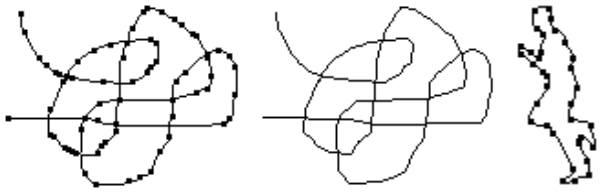
We can choose from a vast number of techniques to partition a curve into straight line segments [2]. These techniques can be further classified as *offline* (the curve is considered globally to find an optimal partitioning), or *online* (the curve is decomposed into line segments during traversing it). Our approach is equivalently suitable for both tasks, and we discuss an online coding possibility here. To partition the curve we use the robust method presented in [10].

To reach a coding schema from the straight line decomposition, we replace all the curve segments from an alphabet of line segments. To have a finite alphabet Λ, its letters are defined as digital line segments having length at most T pixels. Note that Λ also contains all the rotated versions of its letters. Moreover, to keep Λ compact, we consider unique straight line segments to connect two points. For this purpose, we consider the Bresenham line drawing algorithm [11] to create the letters in Λ. Note that, in this way, we allow some information loss, since the Bresenham segments may slightly differ from the ones extracted during the online curve segmentation process. On the other hand, these differences are really minor perceptually, since straightness is a common requirement. The cardinality of Λ, and the number of bits needed for coding a letter are given by:

$$|\Lambda| = 4T(T-1), \text{ and } \log_2 |\Lambda| \leq 2(\log_2 T + 1), \tag{3}$$

respectively.

Figure 6 depicts the results for our sample curve C of length 722 pixels shown in Figure 1a. We marked the end points of the line segments extracted from C in Figure 6a, and show the coded version in Figure 6b. We used $T = 32$ as a threshold for the maximum line segment length. C could be coded with 62 line segments, and thus 744 bits were needed to represent it by Λ. Approximately half of the curve points were affected by substituting with Bresenham line segments.



(a) (b) (c)

Fig. 6. Coding curves with Bresenham line segments. (a) Partitioning the curve. (b) The recovered curve. (c) Partitioning a silhouette template.

Based on more curves, we found an approximate 50% improvement in compression against JBEAM, see Table 1.

Curve	# of pixels	JBEAM (bits)	CT (bits)
General (C)	2127	1586	744
Lines	2745	1398	468
Spring	4113	2308	1224
Script	2511	1419	828

Table 1. Comparative quantitative results against JBEAM.

5. CONCLUSION AND DISCUSSION

Our approach can be improved and tuned in several ways. One point can be the selection of the most appropriate method for the Euler decomposition. Considering the parity of the vertices, it is easy to find Euler decompositions by starting from and finishing at odd vertices. According to preliminary results, the application of the linearity criteria looks feasible within this procedure, as well. Another graph theoretical approach can be to re-formulate our CT algorithm, and consider curve tracing as a Chinese Postman problem [12]. In this case some curve parts should be taken more than once during tracing which results in some redundancy of the final code.

The steps of our approach can have individual importance in other application fields. To "untie" curves can have impact in curve watermarking [13, 14], where the capability to provide the input data in terms of few large blocks is highly welcome. Tracing the curve according to its "natural" direction looks feasible in reconstructing hand-written text or figures, similarly to [15]. The coding technique can be applied to compress templates used e.g. for silhouette matching with an example shown in Figure 6c.

6. REFERENCES

[1] A. Rosenfeld, "Arcs and Curves in Digital Pictures," J. ACM, vol. 20(1), pp. 81-87, 1973.

[2] R. Klette and A. Rosenfeld, "Digital straightness - a review," *Disc. Appl. Math.*, vol. 139, pp. 197-230, 2004.

[3] X. Huo and J. Chen, "JBEAM: multiscale curve coding via beamlets," *IEEE Trans. IP*, vol. 14(11), pp. 1665-1677, 2005.

[4] N.L. Biggs, E.K. Lloyd and R.J. Wilson, "Graph Theory," *Calendon Press*, 1998.

[5] L. Euler, "Solutio problematis ad geometrian situs pertinentis," *Commentarii academiae scientarum Petropolitanae*, vol. 8, pp. 128-140, 1736.

[6] H. Fleischner, "Eulerian Graphs and Related Topics, Part 1. Vol. 1," volume 45 of Annals of Discrete Mathematics, *North-Holland Publishing Co.*, 1990.

[7] H. Fleischner, "Eulerian Graphs and Related Topics. Part 1. Vol. 2," volume 50 of Annals of Discrete Mathematics, *North-Holland Publishing Co.*, 1991.

[8] E. Lucas, "Recreations Mathematiques," *Gauthier-Villares*, 1891.

[9] S. Skiena, "Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica," *Addison-Wesley*, 1990.

[10] I. Debled-Rennesson and J. Reveilles, "A linear algorithm for segmentation of digital curves," *Int. J. Patt. Rec. and Artif. Intell.*, vol. 9, pp. 635-662, 1995.

[11] J.E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, pp. 25-30, 1965.

[12] M.G. Guan, "A survey on the Chinese postman problem," *J. Math. Res. Exp.*, vol. 4(1), pp. 113-119, 1984.

[13] H. Gou and M. Wu, "Fingerprinting Curves," *LNCS*, vol. 3304, pp. 13-28, 2004.

[14] V. Solachidis and I.Pitas, "Watermarking polygonal lines using Fourier descriptors," *IEEE CG& A*, vol. 24(3), pp. 44-51, 2004.

[15] E. Saund, "Finding perceptually closed paths in sketches and drawings," *IEEE Trans. PAMI*, vol. 25(4), pp. 475-491, 2003.