Approaches for Curve Representation in Spatial Databases

Marius Dorian Zaharia
Computers Department
POLITEHNICA University of Bucharest
Splaiul Independentei, 313, Bucharest - 77206, Romania
zaharia@apolo.cs.pub.ro

Abstract

The paper describes some curve representation methods suitable to be used in spatial databases. A particular attention is given to hierarchically structured representation methods (which allow efficient search in curvilinear data sets) and to fractal methods (which allow to obtain very large compression rates when used to represent curvilinear data). Algorithms for building and manipulating such data structures are also specified.

Keywords: curvilinear data, spatial database, hierarchical data structure, fractal compression.

1. Introduction

Curvilinear data are used in applications in the domains of computer graphics, computer aided design, computer vision, robotics or cartography. Depending on the type of application the lines could be isolated or could form "meshes"; in this last case, the lines form the boundary of (usually) polygonal regions.

The representation of curvilinear data could be done mainly by the following methods:

- (1) Vector based representations: where the curve is approximated by a sequence of vectors whose extremities are placed in a list. An important case of those types of representations are the chain codes (simple chain codes or generalized chain codes) which are also vector lists where the vectors could have only a finite number of directions (due to the fact that the vector extremities are placed on a grid).
- (2) Representations based on control points. Some classes of curves have the shape completely determined by a small set of points (called control points). These curves (called approximation/interpolation curves) are used specially in computer aided design applications. There are a lot of categories of approximation/interpolation curves (Bézier curves, various types of spline curves) they differ by their degree of continuity and they could take various shapes (easy to visualize) depending on the shape of their control polygons.
- (3) In case of spatial data bases, manipulating curvilinear data (which could represent for example: the road network in a country, the water supply network in a town or level curves in a map) the representations should allow efficient implementations of search queries and of operations such as intersection or reunion of regions specified by their curvilinear boundaries. These types of representations are based on hierarchical structures.

- (4) Fractal based representation techniques try to recognise patterns contained in a collection of curves (usually modelled by sets of line segments). The principle is to find small simple data sets, that could generate by repeated transformations the original data set (or a close approximation of the original). This kind of representation could be considered as well a data compression technique. The compression is achieved by storing only the segments from the small data sets and the associated sets of transformations.

An interesting approach to index the information from large collections of line segments was described by Jagadish **[5]** which proposed an index structure (based on Hough transform) particularly well suited to solve efficiently queries as: finding the line segments that pass through a specified point, finding the line segments placed in the neighbourhood of a given point, finding the line segments that intersect a given line segment.

The methods most suitable to be used to represent curves in spatial databases are (1) (a very good survey of those methods is given in **[3]**), (3) and (4). The paper will describe some hierarchical approximation structures of curvilinear data and a fractal method to compress large collections of line segments forming closed polylines.

2. Hierarchical representation methods of curvilinear data

The **strip tree** is a hierarchical representation of a curve, by approximating it with a collection of minimal covering rectangles. A piece of curve bounded by two points P, Q will be covered by a rectangular strip having two sides parallel with PQ. These rectangle sides will pass through points (M_i) of the curve placed at a maximal distance from PQ, on both sides of PQ. A C^1 class curve will be tangent at the strip sides parallel with PQ.

One of the points (M_i) - let it be M - is considered splitting point of the curve segment PQ and the process of covering the curve by rectangular strips will be recursively repeated considering the curve segments bounded by points {P,M} and {M,Q} respectively. This subdivision process continues until the resulting strip widths will have values less than a predefined threshold value.

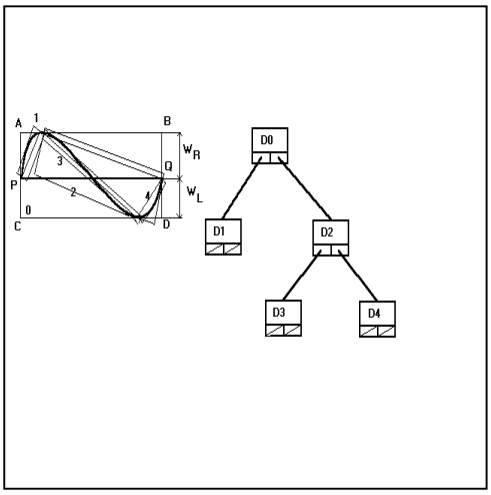

Figure 1: The strip tree representation of a curve

A strip tree node (N) is represented by an eight field record. The fields store: the coordinates of the boundary points P, Q, the widths W_L, W_R of the minimal covering rectangles placed on the left respectively right side of PQ, pointers to the two sons of (N) containing information about the strips resulted from the splitting process.
Given a curve, a method to build the associated strip tree starts from a set of splitting points $\{M_i \mid 0\le i\le n\}$ on the curve and finds the strips corresponding to curve segments M_i,M_{i+1}, $0\le i<n$. These strips (S_i will be associated to the leaf nodes of the tree. They will be grouped in pairs (S_i, S_{i+1}) and, through simple computations, the covering rectangle of each strip pair will be determined. In this way will be built the internal nodes placed on the same level in the tree. The process continues until a single strip will result.

Usually the algorithms to find the intersection or reunion of two curves represented by strip trees have logarithmic complexities. For example if someone wants to decide if two such curves intersect or not, he will try to reduce this problem at the simple case of only two strips. The following alternatives are possible:

- two curves have no intersection if the corresponding strips S_1, S_2 are disjoint.
- two continuous curves C_1, C_2 represented by strip trees intersect if there are two strips S_i (covering a C_1 curve segment bounded by points P_iQ_i), S_j (covering a C_2 curve segment bounded by points P_jQ_j) and all the strip sides parallel respectively with P_iQ_i and P_jQ_j are intersecting.
- two strips S_i, S_j as above could cover an intersection if $S_i \cap S_j \neq \phi$ and condition (b) is not satisfied.

The decision if two curves represented by strip trees intersect is implemented by:

```
function se_int(strip_tree T1,strip_tree T2) return boolean
  case * S_root(T1) ∩ S_root(t2)  of
    empty_intersection: return false
    possible_intersection:
      if area(S_root(T1))>area(S_root(T2)) then
        return se_int(left_son(T1),T2) or  se_int(right_son(T1),T2)
      else
        return se_int(left_son(T2),T1) or se_int(right_son(T2),T1)
      □
    obvious_intersection: return true
  □
end
```

The **arc tree** is a hierarchical curve representation structure in which the curve is decomposed in arcs of equal length. Usually the curve segment to be represented by an arc tree is given by its parametric (or explicit) equations. Let these (parametric) equations be denoted by: $C(t)=(C_x(t), C_y(t))$ where $0\le t\le 1$. In this case the total curve length is given by:

$$\int_0^1 \sqrt{C_x'(t)^2 + C_y'(t)^2}\,dt \qquad (1)$$

or by:

$$\int_0^1 \sqrt{1 + f'(x)^2}\,dx \qquad (2)$$

if the curve is specified by its explicit equation y=f(x).

To build an arc tree associated to a curve segment (given by $C:[a,b]\rightarrow R^2$ where $a,b\in[0,1]$) it is necessary first to compute the total length of this curve segment. The k-th order of

approximation of the curve is a polyline formed by 2^k line segments each of them connects two points with coordinates $(C(i/2^k), C((i+1)/2^k)$ with $0 \leq i < 2^k$.

In the previously mentioned case (a=0, b=1) the arc tree could be represented (using a C language specification) by:

```
typedef struct aa{
  p2d pct; /*splitting point associated to the current node */
  float t; /* value of t corresponding to split point */
  struct aa *fs, *fd;
} *arb;
typedef struct {
  p2d A, M, Z; /* tree's root contains 3 points */
  arb fius, fiud;
} *arb_arce;
```

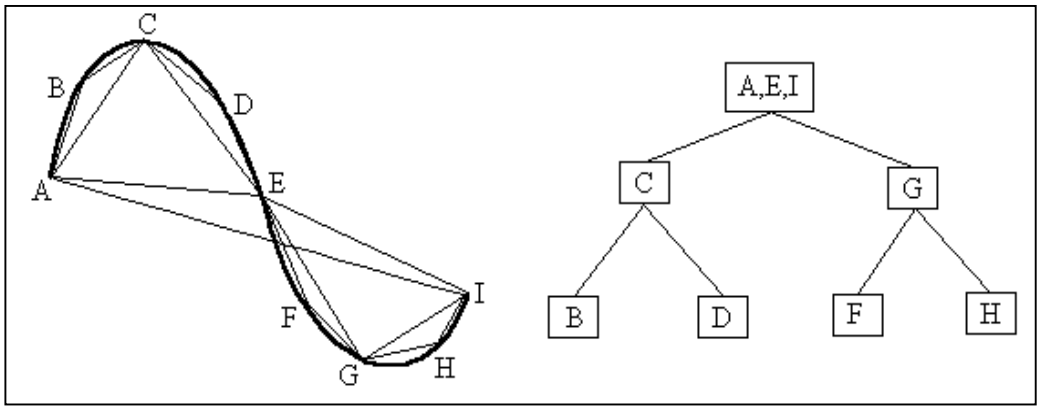

Figure 2: The arc tree representation of a curve

```
function creeaza_arb_arce(curb_ C) return arb_arce
/* creates the arc tree corresponding to the curve specified parametrically
through   C:[0,1] → R^2 */
        r ← creeaz__nod_arb_arce()
        r->A ← C(0)
        r->M ← C(0.5)
        r->Z ← C(1)
        divide(0, 0.5, r->fius)
        divide(0.5, 1, r->fiud)
        return r
end

procedure divide(real t1, real t2, arb a)
        a ← creeaz__nod_arb()
        a->t ← (t1+t2)/2
        a->pct ← C(a->t)
        *compute maximal error
        while  abatere_max>ε do
        /* build one more level in the arc_tree */
             a->fs ← divide2(a->fs, t1, (t1+t2)/2)
             a->fd ← divide2(a->fd, (t1+t2)/2, t2)
        □
end
```

```
function divide2(arb a, real t1, real t2) return arb
        if a=NULL then
             b ← creeaz__nod_arb()
             b->t ← (t1+t2)/2
             b->pct ← C(b->t)
             *update maximal error
             return b
        else
             a->fs ← divide2(a->fs, t1, a->t)
             a->fd ← divide2(a->fd, a->t, t2)
             return a
        □
end
```

It was considered (conforming to Günther specification **[4]**) that the arc tree is a balanced tree.

A method similar to the arc tree used to hierarchically represent a polyline is the binary searchable polygonal representation. This time the splitting points of the curve are the vertices of the polyline. Obviously the sections of the decomposition have no more equal lengths (as in case of arc trees).

The **Binary Searchable Polygonal Representation** (BSPR) approximates a polyline by decomposing it into a set of simple sections. A simple section corresponds to a rectangle (R) whose sides are parallel to the coordinate axes. This section is associated to a part of the polyline whose line segments are varying monotonicaly on both axes. The BSPR tree is built by a bottom-up technique starting from his leaves (which correspond to the simple sections) and joining every two adjacent simple sections to produce complex sections which will be associated to the internal tree nodes.

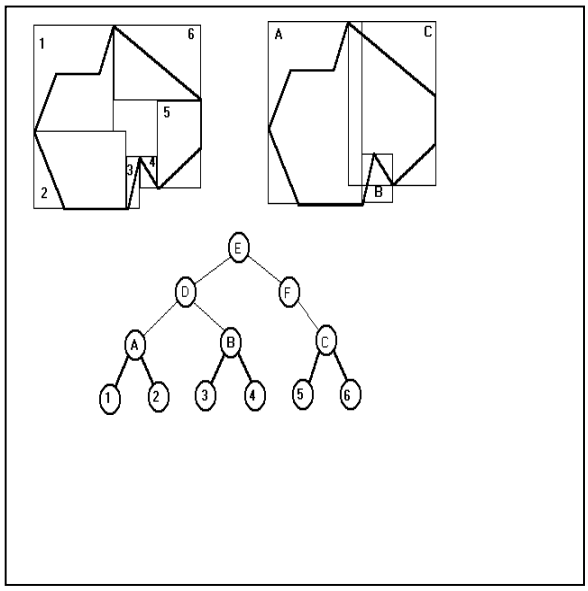

Figure 3: BSPR representation of a polyline

One section S is represented by: the minimal covering rectangle (R) (represented by two opposite corners (x_{min}, y_{min}), (x_{max}, y_{max})), his extremities (i.e. the first vertex of the first edge of the part of the polyline covered by the rectangle (R) and the second vertex of the last edge) and

pointers to other two sections (in which S is decomposed).

This kind of representation allows efficient implementations of operations as: finding reunion/intersection of two regions bounded by closed polylines or deciding if a point is or not interior to a polygon. The algorithms are similar to those used to process strip-trees representations. For example finding the intersection of two polygonal areas could be accomplished by:

```
function intersectie_BSPR(BSPR S1, BSPR S2) return punct
  if * sections S1, S2 are not overlapping then
    return NULL /*  intersection does not exist */
  □
  if *both sections are primitive
    (i.e. cover only one side of the polyline) then
    return * intersection is found through elementary computations
  □
  if S1∩S2≠Φ and at least one section (S2) is not primitive  then
    * S2 is splitted into two subsections S2', S2"
    p1 ← intersectie_BSPR(S1, S2')
    if  p1≠NULL then return p1
    else return intersectie_BSPR(S1, S2")
    □
  □
end
```

Another approach to represent curves (approximated by sets of line segments) is to divide the data space into cells. Each cell will have an associated list of line segments that intersect it. The information regarding each line segment has to be stored several times, once for each cell the line segment intersects.

The **PM-tree** is similar to the region quadtree. Each spatial region is repeatedly splitted in four equal parts until the moment when one region will include at most one edge of the polyline.

The spatial decomposition is driven by the following rules:

- The information about the vertices or edges of the polylines is stored only in leaf nodes
- A region (Reg) associated to a leaf node may include at most one vertex (V) of the polyline. In that case all the edges intersected by (Reg) must have (V) as extremity.
- If a region (Reg) associated to a leaf node contains no vertex of the polyline then (Reg) could intersect at most one edge of the polyline.

A PM-tree node (N) could be implemented as a record with the following fields of information: the size and the coordinates of the centre of the square region (Reg) associated to (N), the coordinates of the polyline vertex, the list of associated q-edges (a q-edge is the part of an edge included in (Reg)), the type of the node (leaf/internal).

As an example, the procedure which inserts a node in a PM-tree is:

```
procedure insert_PM(lista_muchii p, arbore_PM a)
  L ← line_clipping(p, a->cx, a->cy, a->lat)
  if L ≠ NULL then
    if *a is leaf node then
      L ← *add elements from L to a->le
      if compatibil(L, a->cx, a->cy, a->lat) then
        a->le ← L
        return
      else
        divide_nod_PM(a)
```

```
      □
    □
    for i ← 0,3 do
      insert_PM(p, a->fiu[i])
    □
  □
end
```

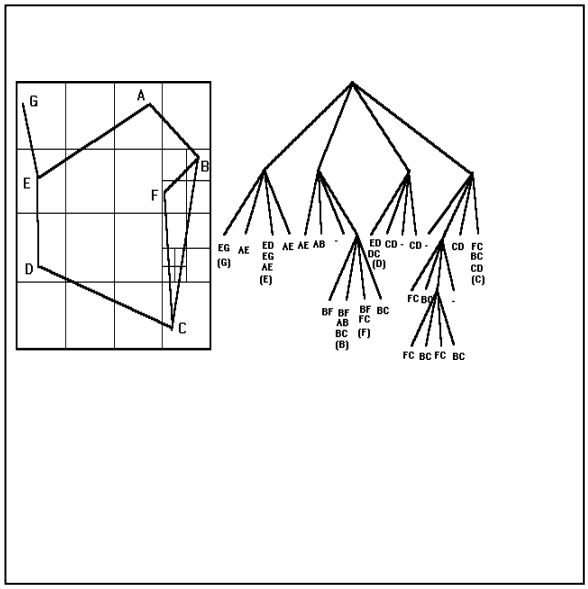

Figure 4: The PM-tree spatial decomposition

3. A fractal method for compression of curvilinear data

The algorithm presented below was used to compress a curvilinear data collection. The curves could be specified by their analytical descriptions or simply approximated by (not self intersecting) polylines. The polylines are supposed to be closed (otherwise they are automatically "closed" by the algorithm). The algorithm is suitable to be used in the domain of cartography, in that case the closed polylines could represent level curves in a map. It tries to identify similar curves. Two curves C_1, C_2 are considered to be similar if C_2 could be generated from C_1 by applying a set of transformations (translation, rotation, or scaling). The algorithm is asymmetric i.e. the computing effort to find the similar patterns in the polyline data collection is considerably larger than that necessary to restore the original data representation from the compressed one.

In order to decide if two polylines (or arbitrary closed curves) are similar, the algorithm will associate with each polyline a signature i.e. an array of n elements (the features of the polyline). A reference axis OS is attached with the polyline. Let O be the origin of that axis. A reference vertex (S) of the polygon is also selected. Starting from OS n equally spaced radius r_1, r_2, ..., r_n are drawn. Let P_1, P_2, ..., P_n the n intersection points of each radius with the curve. The signature array of curve C will contain the values: $v[i]=|OP_i|$. Using this method each curve will have a

unique set of features v[1], ..., v[n].

Two curves will be considered similar if:

$$\frac{v_1[i]}{v_2[i]} = \frac{v_1[i+1]}{v_2[i+1]}, \qquad \forall\ 1 \le i \le n-1 \qquad (3)$$

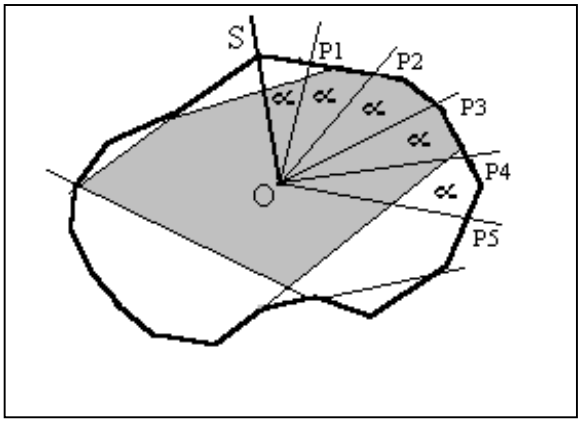

Figure 5: Finding the signature of a curve

To find a complete solution for the problem of identifying two similar curves (closed polylines) of the collection, a method to associate a reference axis OS to each of the two curves must be given. The reference point O is the centroid of the polygon kernel and the reference vertex S is the first vertex (in the counterclockwise order) of the longest edge of the polygon. If there are more such edges the start vertex of the longest decreasing edge sequence will be selected.

The method described allows obtaining compression rates of 100% and more when it is applied on very large polygonal collections.

4. Conclusions

The first three hierarchical data structures (strip tree, arc tree and BSPR) have the splitting points placed in positions independent from the co-ordinate system attached to the curve. Building or manipulating such data structures could necessitate complicate geometric computations (for example finding the point on a curve placed most distant from a given line). These representations are invariant with respect to translation and rotation transformations.

In the case of PM-tree the division points are placed at predefined positions. This kind of spatial decomposition could produce better results when solving proximity queries. It is appropriate to implement efficiently queries based on finding relations between curves and regions.

The idea to find a reference axis associated to a closed polygon and with orientation depending only on the shape of the polygon and not on its position in the data space was also described in section 3. This kind of algorithm is useful in finding self-similar polygonal patterns.

References:

[1] Brown, C., Shepherd, B., (1995) "Graphics File Formats", Manning Publications Co., 1995.
[2] Burton, W., (1977) "Representation of Many Sided Polygons and Polygonal Lines for Rapid Processing", Communications of the ACM, march 1977.
[3] Freeman, H., (1974) "Computer Processing of Line Drawing Data", ACM Computing Surveys, 1/1974.
[4] Günther, O., (1993) "Three Curve Representation Schemes", IEEE CG&A, May 1993.
[5] Jagadish, H., V., (1990) "On Indexing Line Segments", Proceedings of the 16-th VLDB Conference, Brisbane, 1990.
[6] Samet, H., (1988) "An Overview of Quadtrees, Octrees and Related Hierarchical Data Structures", Theoretical Foundations of Computer Graphics and CAD, NATO ASI Series vol. F40, Springer Verlag, 1988.
[7] Stoica, A., (1997) "Metode de compresie fractala a datelor curbilinii", Proiect de diploma Nr. 76/97, (coordonator s.l. ing. M. Zaharia) Universitatea POLITEHNICA Bucuresti, Catedra de Calculatoare, 1997.