© Proceedings of the *7th Canadian Conference on Computational Geometry*
Québec (Québec, Canada), August 10-13, 1995, pp. 229-234.

Computer-Aided Road Network Design

P. Biscondi, J-M. Moreau
LISSE*

Abstract *This paper describes the process of computer-aided road design, for instance in the context of Geographic Information Systems, where* linear objects *(roads, rivers, ...) are defined by their axes and transverse profiles. Various techniques are applicable, but few have the sufficient flexibility to satisfy the strong constraints familiar to users in that field: Heterogeneous data, conflicting scales in neighboring objects, artefacts, etc. ...*

It is also desirable that once the whole graph has been expanded, the adjacency relations between roads and objects bordering them be preserved and optimized in a certain sense. The algorithms described in this paper combine all previous aspects, and were successfully implemented in a major flight-simulator application.

1 Introduction

The progress of Computational Geometry during the last two decades has made it possible to process larger and larger databases. Large-scale geographic applications represent actual landscapes and towns with even greater realism. Accordingly, the need for the automatic design of such bases is growing ([1], [4], [5], [6]).

The paper introduces a generic method for expanding road networks or river systems, using both the semantics of their definitions, and some automatic rules to adapt the geometry of other objects in the database, and thereby guarantee initial adjacency relations. Section 2 describes the various objects that will be considered. Section 3 presents the generic method suggested for consistent expansion, and Section 4 two variants to solve degenerate cases inherent to the problem. Section 5 details the impact of expansion on other objects, and Section 6 concludes on extensions and future work.

*Address: ENSM.SE, 158 cours Fauriel, F 42 023 St-Etienne, France. Email: `moreau@emse.fr`. This work was supported by a contract with SOGITEC (France), as part of the research on the SINDBAD flight-simulator application.

2 Objects

The overall process of creating real databases involves getting raw cartographic data, filtering redundant information or making it homogeneous, casting planimetric information on top of altimetric data (relief lines, contour lines, faults, markers,...), mapping objects (fields, houses, buildings, rivers, roads) onto the database, and then organizing its elements so that conflicts may be solved using a set of (geometric, semantic, ...) pre-defined and/or user-defined rules. In informal terms, fields should give way to roads or houses, roads should "yield" to railway tracks or water lines, altimetric data should be swallowed by rivers, etc.

When all these operations have been performed, the resulting information is fed to a series of geometric algorithms to make planar, interpolate, clip, triangulate and visualize, etc.

In a typical G.I.S. application, three types of objects may be considered:

Static objects: Such objects (houses, contour lines, altimetric extrema, etc.) are completely defined at design time, and are not supposed to be altered by the rest of the objects in the database, unless they become partially or entirely occulted by constructs with higher rank in the priority scale of objects.

Dynamic objects: Objects of this class (a generic term for roads, rivers and all sorts of *linear* objects) are defined by a "skeleton" (polyline) and a profile, both of which are used in the so-called *expansion* process; for instance, a road may be correctly approximated given a sequence of segments, and widths (orthogonal projections) describing the geometry of its sectional profile at each definition vertex. Although profiles may be complex, to represent ditches, verges, etc., we shall restrict ourselves to simple profiles – where only two *half-widths* are known –, as all the descriptions in this paper may be generalized to more complex situations.

Semi-dynamic objects: This last class is used to

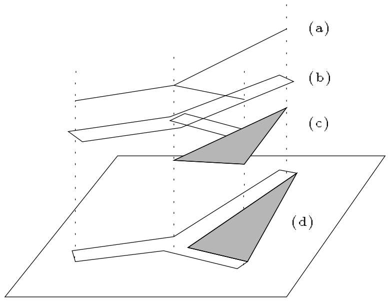

Figure 1: Some levels of description in a database.

describe all objects whose geometry is "influenced" by the expansion of dynamic objects, as explained below. Such objects share vertices with dynamic objects at design time, but their final geometry after the expansion of linear objects is impossible to guess by that time.

In this paper, we suggest several construction rules for the last two classes of objects, in order to satisfy various contraints present in databases. Consider Figure 1 representing a portion of, say, a G.I.S. database drawn on several levels.

(a) *Top level*: Only the axes defining dynamic linear objects are present here.

(b) *Road expansion level*: Linear objects should be "intelligently" expanded. Notice how a naive and independent expansion of the two axis systems on the figure would yield ill-shaped junctions.

(c) *Semi-dynamic objects*: Only the objects sharing vertices with definition axes are considered.

(d) *Lower level*: The resulting ideal graph. Notice how the junctions have been trimmed and the semi-dynamic objects have been "retracted" to the limits of their surrounding dynamic neighbors.

As may be inferred from the figure, the probability that any original semi-dynamic polygon crosses the expanded version of its adjacent dynamic linear objects at undesirable places is very high. Indeed if the "blending" of the two graphs is not performed in an "intelligent" way, the resulting output degenerates into many tiny edges, unpredictable at design time.

Supposing we are dealing with a two-dimensional database, the following problems must be solved:

1. Construct the *planar* graph of dynamic objects. At this stage, only the definition axes and their endpoints are considered.

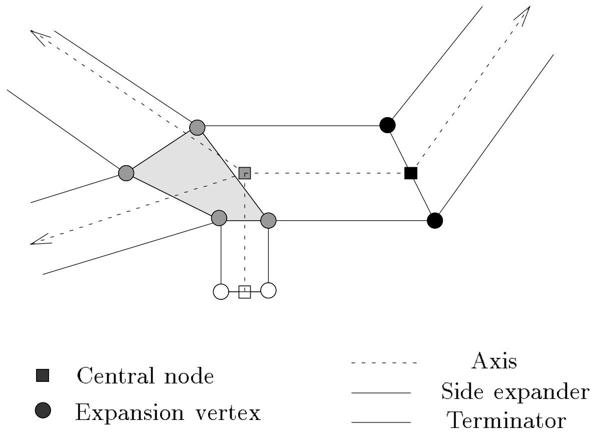

Figure 2: Crossroads terminology.

2. Expand all dynamic objects, and deduce the geometry of "crossroads" at vertices where different sections of dynamic objects are connected.
3. Adapt the graph of all semi-dynamic objects having at least one vertex in common with at least one dynamic object in the previous graph, so as to satisfy the (adjacency) constraints inferred.

3 The expansion process

3.1 Expansion basics and terminology

All graphs to be considered in the generic situation described here must be made planar before any other action is taken. To that effect, Bentley-Ottmann's line-intersection algorithm ([3]) was favoured, for practical reasons (among which the availability of local planar maps (section 3.3)).

Referring to Figure 2, let us call *(central) nodes* the vertices of the axis definitions graph – shown as squares. Each definition axis yields two (or in fact more, for complex profiles) *side expanders*, *i.e.* line segments parallel to the main definition axis segment, each at the distance specified by the associated half-width. Discs denote the *expansion vertices*, *i.e.* intersection points of successive pairs of expanders around the same central node.

If we call *sector* (Figures 2 & 3) the plane wedge defined by two consecutive definition axes around the same central node, then it is straightforward that there is exactly one expansion vertex in each sector of the same node and pertaining to it. Clearly, only the definition axes incident upon one given central node may have any influence on the topology of its associated crossroads, and hence on its *local* geometry. Another application of the line-segment algorithm on expanded linear objects (and the rest of the database) will be necessary to determine all "interactions" between final objects.

Nodes of degree 1, also called *dead ends*, deserve special care, as shown on the right-hand side of Figure 3. Finally, define as a *terminator* any edge linking two consecutive expansion vertices around the same node (Figure 2). As usual, terminators and expanders carry pointers to two faces, one of which pertains to the linear object they are naturally attached to, and the other to "the outside world" in general, except for certain terminators, as explained later.

From a global point of view, an expanded linear object consists of two chains of side expanders closed at each end by terminators. Either end may correspond to hitting another linear object with higher priority, or a dead end. From a local point of view, a crossroads is a series of terminators (between consecutive expansion vertices), organized according to the circular order of definition axes around the central node[1]. If desired, some of them may actually be omitted, to acknowledge the prevalence of the linear object with highest priority in the junction, for instance.

Hence, the storage space required for the expansion process remains linear in the number N of axes (each axis yields at most 4 expansion edges).

3.2 Formulæ

All the computations required in the construction of crossroads are based on the same principle: successive expanders around the same central node define (meet in) one unique expansion vertex.

For sector (e_0, e_1) with unit vectors $(\vec{u}_0, \vec{u}_1)$ (refer to Figure 3), the following system, correlating the widths w_0, w_1, abscissæ λ_0, λ_1 and sector angle θ, holds:

$$\vec{r}\sin\theta = w_0\vec{u}_1 + w_1\vec{u}_0$$

$$\lambda_i = \vec{r}\cdot\vec{u}_i = \frac{w_{1-i} + w_i\cos\theta}{\sin\theta}, \quad i = 0, 1$$

$$\vec{r} = \lambda_i\vec{u}_i + w_i\vec{u}_i^{\perp}$$

Note that the position of the associated expansion vertex is determined *if and only if*: $\vec{u}_0 + \vec{u}_1 \neq 0$ *or* $w_0 = w_1$. Otherwise, the difference of widths prevents expanders from being matched, and a compromise must be reached.

3.3 Creating expansion vertices

Suppose the graph of definition axes has size N after it has been made planar. If such structures do not come free with the line-segment algorithm (as is the case with Bentley-Ottmann's), it is possible, in overall time $O(N \log N)$, to sort, for each vertex v of the graph, all edges incident upon v in, say increasing polar angle, and to detect the linear object with highest priority incident upon it, H_v. We shall suppose that such ordered lists, referred to as *local planar maps* and noted $LPM(v)$, may be accessed in constant time from each vertex.

[1]Note on Figure 2 that, depending on the half-widths and the geometry of axes, the "central node" is not always as central as intuition has it...

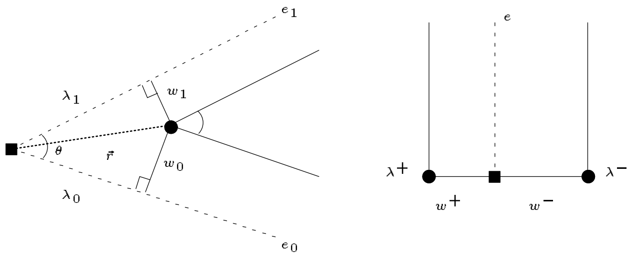

Figure 3: Sector and dead end ($\lambda^- = \lambda^+ = 0$).

Using these maps and the previous set of formulæ, all expansion vertices may then be constructed in additional $O(N)$ time.

3.4 Constructing expanders

Once expansion vertices have been computed it is possible to create the expansion edges, as follows:

```
    Procedure CONSTRUCT (NODE v)
    SET_VISIT(v)
    for each UNDIRECTED EDGE e = vw IN LPM(v) do
        if (MUSTYIELD(e, H_v)) then
            CREATE_GIVEWAY_TERMINATOR(e)
        if (!GET_VISIT(w)) then
            INITIATE_SIDE_EXPANDERS(e)
        else TERMINATE_SIDE_EXPANDERS(e)

for each v IN AXIS GRAPH do
    CLEAR_VISIT(v)
for each v IN AXIS GRAPH do
    CONSTRUCT(v)
```

Predicate MUSTYIELD returns true when the linear object associated with edge e has priority lower than that of H_v (as detected earlier), in which case the 'outward' face of the "give-way" terminator is made to point to H_v.

Clearly, side expanders are allocated/initialized, and finalized/output in synchronicity with the first and second visit of their corresponding definition axes, whereas terminators are generated "on the fly".

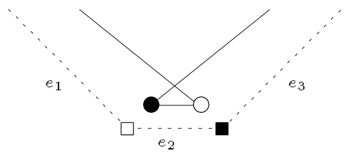

Figure 4: U-shaped section.

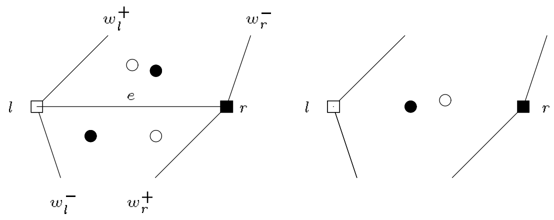

Figure 5: Graph reduction.

4 Handling degeneracies

Remark on data structures: The algorithms in this section make liberal use of the winged-edge structure of Baumgart ([2]) or its half-edge representation variant. In such a setting, any edge has from zero to two adjacent "wing" edges at either end, all accessible in constant time (refer to Figure 5 for a standard picture of the "4-winged-edge" structure). In the special cases where the degree of either endpoint, say l, is 1 or 2, the corresponding number of wings for e around l is 0 and 1, respectively. Edges also have, associated with them, exactly two couples of λ's: one "above", and one "below", corresponding to the abscissae of the four expansion vertices described earlier.

Consider the U-shaped graph of Figure 4: axis e_2 and its upper expander are mutually reversed. More generally, it may be impossible to satisfy both the geometry (axis lengths and angles) of a given graph, and the profile of its constraints (half-widths); in such a case, the graph will be said to be "*non-conformant*".

Furthermore, because of imprecision problems or heterogeneous data, it is frequent that sections of axis definitions (resp. vertices) – which should otherwise meet in one single point (resp. lie on definition axes) – do not! (See Figure 6 a, b, c for three typical cases.)

4.1 Solution 1: Preserving topology

To eradicate such degeneracies, one may wish to respect the graph topology at any cost, by pinning expansion vertices of degenerate expanders onto "safe" positions, regardless of the widths. By "safe", we mean in such a way that the abscissa (of the projection) of *any* expansion vertex on either sector axis involved never exceeds half its length, a strong guarantee that λ's will not be "reversed".

Such a rule may, for instance, be enforced using a convex linear combination of the three nodes forming the current sector, but, unfortunately, no weight choice will be suited to all cases: The drawback of this technique is bad aspect, as expanded objects will either widen or narrow exaggeratedly, as shown on Figure 6, cases a_1, b_1, c_1.

4.2 Solution 2: Preserving profiles

Let us now consider a new solution, called *Reduction*, which will preserve profiles instead of topology: "ill" edges will be discarded, and the graph will be refined so as to guarantee a clean expansion (Figure 5). Our goal is now to *derive a conformant graph from the original one.*

We shall need a stack with the standard `Push` and `Pop` primitives, respectively setting a specific `StackStatus` field to 1 and 0. Supposing all λ's precomputed, all `StackStatus` fields intially set to -1 ("never pushed"), and one canonical edge for each connected component of the axis graph initially pushed into the stack, the following scheme is applied:

```
while ((e = Pop()) != ∅) do
    Determine all wings of e
    if (λ-Reversed(e)) then
        Discard e
        Restore both local planar maps
        Recompute λ's for the updated sectors
        for each wing w_i do
            if (StackStatus(w_i) < 1) then Push(w_i)
    else
        for each wing w_i do
            if (StackStatus(w_i) < 0) then Push(w_i)
```

λ-Reversed is a predicate returning *true* if and only if either couple of λ's above and below e are reversed compared with the mutual order of the original definition vertices.

Typical effects of this technique are shown on cases 6 a_2, b_2, c_2, where the edges of the reduced graphs are shown dotted. Notice that, despite the potential discontinuities in the resulting road system, objects overlap, as in cases b_2 and c_2, according to intuition.

4.2.1 Running time

The reduction algorithm runs in time linear in the number of edges.

Proof: Obviously, time complexity is equivalent to the number of elements popped from the stack. Any edge may be popped more than once, but any pop

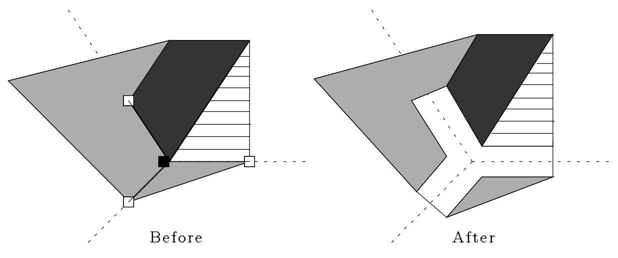

Figure 7: Automatic retraction.

may be imputed to one unique edge (itself or some previously discarded edge).

Therefore, the **while** loop is iterated at most $N_x + 4N_d$ times – where $N_x \in O(N)$ is the total number of definition axes, and $N_d \in O(N)$ the amount of those discarded – which is less than $5N$. Hence, the algorithm runs in $O(N)$ time $\diamond$

5 Automatic retraction

This section presents an automatic process to force semi-dynamic objects to "skirt" the expanded linear objects they have been semantically attached to at design time. Although a typical G.I.S. application will include standard clipping facilities (to solve superimposition problems for faces, according to well-established semantic rules), one should not leave such problems to them alone: The initial adjacency relationships between semi-dynamic objects and definition skeletons are lost after expansion, and should be correctly restored, without any artefact (micro-edges, in particular).

5.1 Principle

Basically, one has to synchronize the graphs for dynamic and semi-dynamic objects. If unicity of vertices is ensured in the database representation (through some basic scheme, for instance balanced trees), it is possible to locate vertices shared both by semi-dynamic and dynamic objects, in one single sweep.

Consider this sweeping process as it reaches node v. If v does not belong to both graphs, proceed to the next vertex. Else, each semi-dynamic edge incident upon v must simply be "matched" to one unique definition axis sector: Semi-dynamic edge e is matched to sector (e_i, e_j) if e strictly lies within the wedge, or else coincides with either e_i or e_j and the interior of the object it represents (partially) lies in the wedge. Both cases arise at the black square node on Figure 7 for the north-eastward axis separating the hatched and dark grey objects on the one hand, and for the three other incident axes on the other.

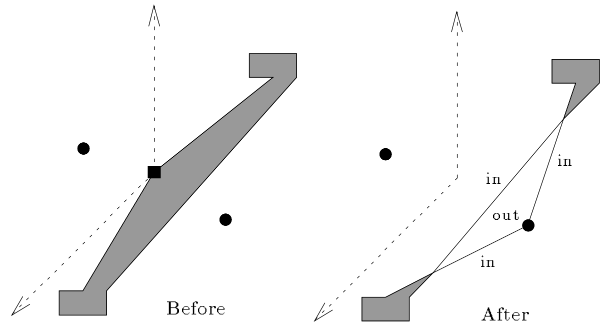

Figure 8: Turning polygons inside out.

Because they are both sorted by increasing polar angle, the local planar maps for v of both graphs may thus be synchronized in time linear in their total size. While doing so, each semi-dynamic edge e incident upon v is "retracted" using the unique expansion vertex in the sector matched to e.

The last action that remains to be taken before proceeding to the next sweep vertex consists in "closing" semi-dynamic objects around v in case they enter and leave through different sectors. This may be done in linear time still, by creating an edge between v and all relevant expansion/retraction vertices, with the help of local face information consistency.

5.2 Consistency considerations

The reconstructed planar maps for auto-retracted semi-dynamic objects may unfortunately be incorrect in situations where the original edges of such objects unexpectedly "double-cross" sector boundaries. Take as a very simple example Figure 8. The central node-vertex of the semi-dynamic object depicted is retracted to the appropriate expansion vertex, while all its others vertices remain unchanged. The resulting auto-retracted object is turned inside out, and has an incorrect topology.

Such cases – which frequently happen when the objects in the database come from very different sources – may only be detected by testing the resulting semi-dynamic objects for simplicity and face consistency (again using a specialized version of Bentley-Ottmann's algorithm, for instance): All retracted objects failing the test are simply reassigned their input definition for all subsequent processing.

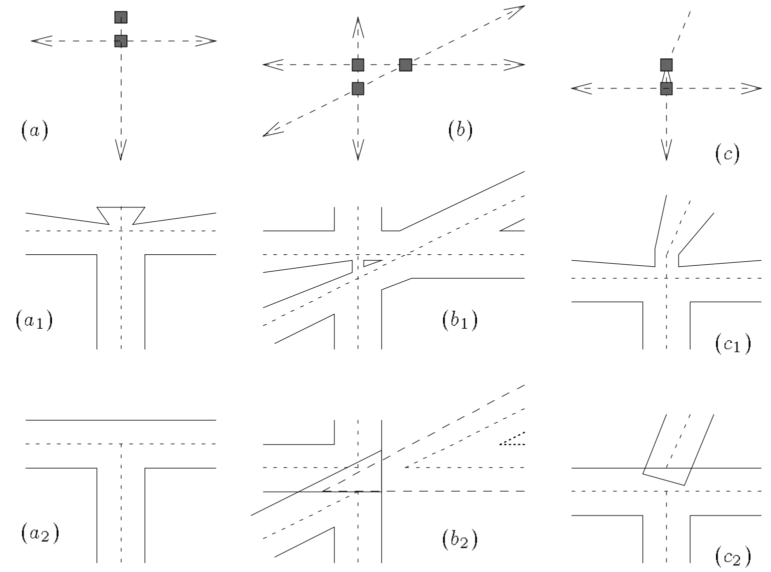

Figure 6: Handling typical degeneracies.

6 Extensions and Conclusion

The algorithms presented in this paper have been implemented in a real-scale application for flight simulation, where road design plays an important role.

There are obvious generalizations that should be brought to the solutions described here. For instance, one may wish for each section of linear objects to have its own four half-widths, which results in trapezoidal sections. It is also desirable to allow multiple profiles. Although such generalizations are by no means straightforward, their nature is not fundamentally different from the generic schemes presented above, and should be implemented in the future using similar strategies.

References

[1] Defense Mapping Agency. Product specifications for digital landmass system (dlms) data base. Technical report, DMA Aerospace Center, St. Louis, Missouri, July 1977.

[2] B.G. Baumgart. A polyhedron representation for Computer Vision. In *National Computer Conference*, pages 589–596, 1975.

[3] J.L. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comp.*, C-28(9):643–647, 09 1979.

[4] C. Gold. Spatial data structures – the extension from one to two dimensions. In *Mapping and Spatial Modelling in Navigation*, pages 11–39, NATO ASI Series F No. 65, Springer-Verlag, Berlin, 1990.

[5] O. Günther. *Efficient Structures for Geometric Data Management.* Number 337 in Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1988.

[6] P. van Oosterom. Spatial data structures in Geographic Infomation Systems. In *NCGA's Mapping and Geographic Information Systems*, pages 104–118, Orlando, Florida, 1988.