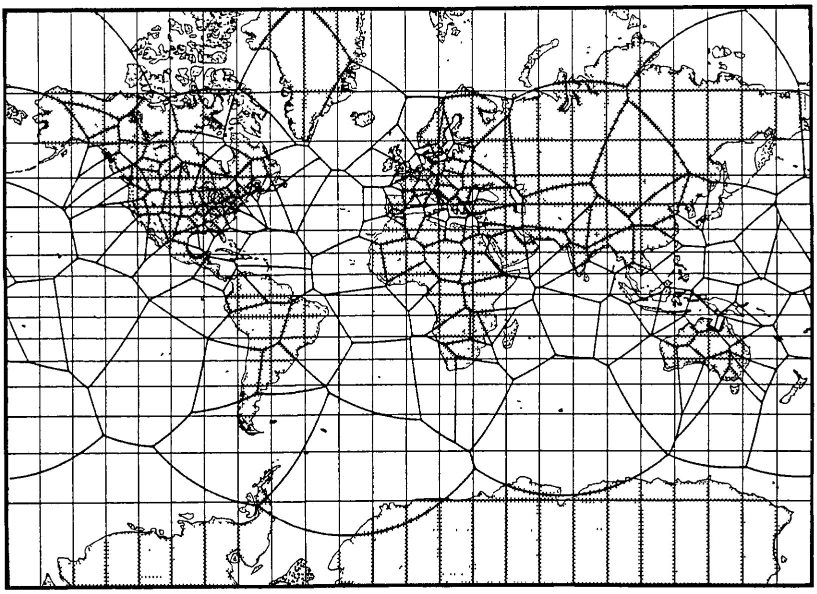
# HIPPARCHUS DATA STRUCTURES: POINTS, LINES AND REGIONS IN SPHERICAL VORONOI GRID

Hrvoje Lukatela
2320 Uxbridge Drive, Calgary, AB  T2N 3Z6 CANADA
(Envoy 100: lukatela)

## ABSTRACT

Hipparchus geo-spatial manager (cf. Auto-Carto/8 introduction paper) is an operational software package that provides geometronical and geo-relational functions to applications that manipulate spatial objects. It is capable of geodetic precision levels, and fully honors the isometric, spheroidal nature of the terrestrial surface and orbit data-space. A Voronoi tessellation is used as a base for its domain partitioning grid. This paper outlines data structures used to represent 0, 1 and 2 dimensional surface objects: sets consisting of discrete points and lines, and non-simply connected regions.

The paper also discusses general characteristics of a family of computational geometry algorithms which evaluate spatial unions and intersections by operating simultaneously on the digital model of the spheroidal Voronoi grid and on the object structures.



Hipparchus Geopositioning Model: An Example of Voronoi Cell Grid

## INTRODUCTION

Point, line and region sets represent the most common classes of spatial objects that a geographically-oriented application system must be capable of manipulating in a meaningful way. While the full nature of such manipulations depends on the domain and purpose of the application itself, spatial unions and intersections represent their reoccurring "building-blocks". It is therefore desirable to provide this functionality in a packaged form, in order to avoid re--programming of identical procedures in many different development projects. This is one of the functions of the Hipparchus software package. (cf. Auto-Carto/8: Hipparchus Geopositioning Model: an Overview, same author. Understanding of the model elements described therein is assumed.) This paper details basic data structures and computational methodology used in this particular functional segment of the package. The implementation follows true ellipsoidal frame of reference; to simplify this presentation, structures will be considered only in their spherical form.

Certain principles apply to all three object classes. Each class represents, conceptually, a pertinently dimensional set of surface points. Each set can consist of a finite number of simple component--sets of the same dimension. (Component-sets are "fragments" of the set in the strict sense of its spatial extent; all non-spatial characteristics describe the whole set or object, none can be specific to a particular component-set.) Since an object modeled by the system can loose completely its spatial extent, the system must not only recognize empty sets, but also be able to use them in union and intersection productions. One and two dimensional sets are numerically represented by finite, ordered sets of vertices, conceptually connected with great circle segments.

All three spatial object classes are commonly used and exchanged by various systems that build and use digital models of geographically distributed data. Assuming that point locations are defined in a coordinate system - appropriate to the particular reference surface - "neutral" form of numerical object representation is usually constructed as a set of ordered point coordinates, with component--sets delineated by a "not-a-coordinate" token. Two consecutive tokens signify absence of further component-sets, i.e. end of point data representing the object. A data-item following the last token identifies the object. Since component-sets of a point set are single point coordinates, its "neutral" representation can be (and commonly is) simplified by discarding the component-set delineation tokens, and terminating the whole set with a single token.

If different point coordinates are represented by Pta, Ptb, Ptc, etc., delineation tokens by *, and if a quote-string data-item is used to identify the object, examples of "neutral" representation of point, line and region objects could be:

Point set:
Pta Ptb Ptc Ptd Pte Ptf Ptg Pth Pti Ptj * "letter boxes"

Line set:
Pta Ptb Ptc * Ptd Pte * Ptf Ptg Pth Pti * Ptj Ptk * * "snow fences"

Non-simply connected region:
Pta Ptb Ptc Ptd Pte Ptf Ptg Pth * Ptl Ptk Ptj Pti * * "Crater Lake"

## SPHERICAL VORONOI GRID AND COMMON DATA ITEMS AND STRUCTURES

In addition to the spherical or spheroidal reference surface, Hipparchus spatial index - dual of the Voronoi polygon grid - forms another dominant spatial feature of the system. Ordered polygons represent a series of identifiable "data-cells". Each surface point belongs to one - and only one - cell. (Points on the edge belong, by convention, to a cell with a lower number.) No restrictions are placed on the relative position of objects and the cell grid: an object can be contained completely inside a single cell, or extend over any number of cells. Consecutive vertices on a line can belong to the same cell, to two neighbour cells, or to two cells which are not neighbors. Linear segment connecting two consecutive points can pass through any number of cells.

Only three abstract data types - in addition to the object identifier - are used in object representation: cell ordinal number, local point coordinate pair and cell-boundary intersection coordinate. (Their mapping into data types intrinsic to a particular computer environment can vary from implementation to implementation). In addition, Hipparchus structures include (unsigned) count of elements in various (ordered) lists and (aggregate) clusters, and locators of their subordinate lists or clusters. (Locators are the only elements which need conversion as the objects are moved from memory to external storage and back.) Implementation specific mechanism exists for manipulation of cell identifier lists, in order to save space and speed up list search algorithms.

Object structures are designed with two - often diverging - objectives in mind: efficiency of transformation between neutral, external and internal object representation and efficiency of evaluation of spatial unions and intersections.

Several structures are used as building-blocks in the representation of spatial objects:

**Occupied cell descriptor**: Structure describing a cell which contains one or more aggregate (non-ordered) points that belong to the object. The structure consists of:

**Cell identifier**
**Local coordinate cluster length**
**Local coordinate cluster locator**

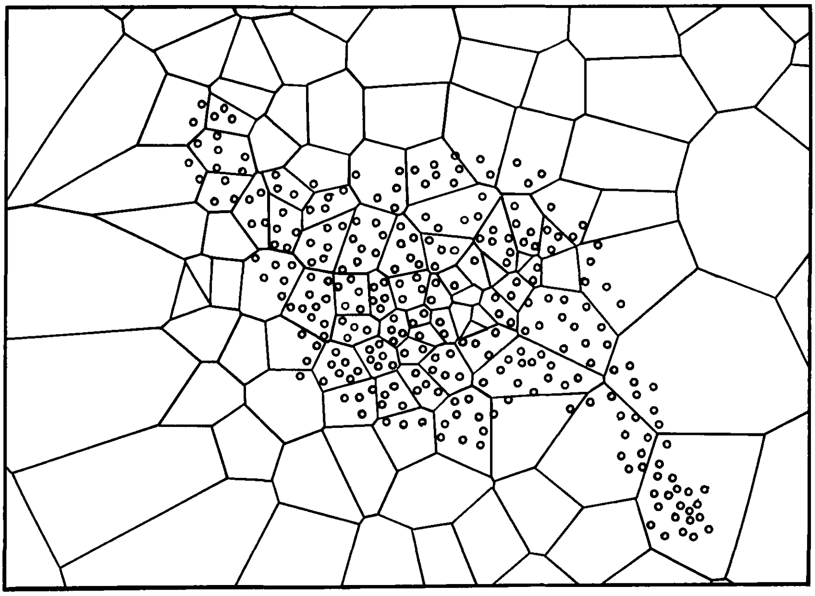
**Line descriptor**: Structure describing a line that forms either one among the components of a line set, or one among the boundary rings in a non-simply connected region. The structure consists of:

**Traversed cell descriptor list length**
**Traversed cell descriptor list locator**

**Traversed cell descriptor**: Structure describing either that section of a line or the boundary ring that is inside one cell, or the complete line/ring contained in a single cell. The structure consists of:

**Cell identifier**
**Local coordinate list length**
**Local coordinate list locator**
**Cell boundary intersection coordinate**

In the case where this structure portrays only a section of the line or ring, boundary intersection coordinate defines the point where the line leaves the cell. This item will be void if the whole line or boundary ring is inside a single cell, and, likewise, in the last cell of a line. In the case of single cell boundary ring, coordinate list attached to this structure is circular by definition, the start vertex point is not repeated at the end of the list.



**Point set object descriptor:** Structure defining a point set object and describing its spatial extent. The structure consists of:

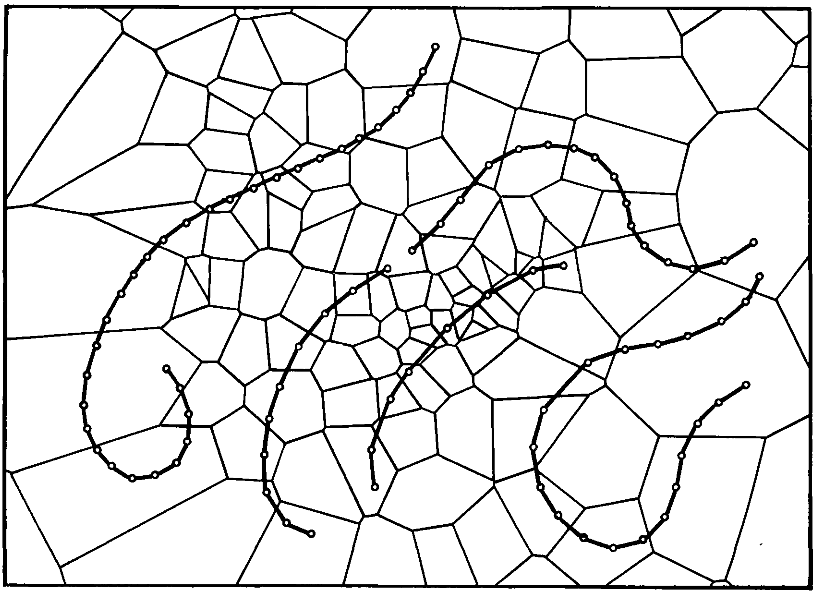
**Object identifier**
**Occupied cell descriptor list length**
**Occupied cell descriptor list locator**

Objects that by definition never exceed single point location (i.e. simple point objects) can be represented by a much simpler, self--contained structure:

**Object identifier**
**Cell identifier**
**Local point coordinate pair**

Since single point component-sets represent parts of the spatial extent of the same object, no two of them must be closer than the minimum spatial resolution that the system is capable of representing.

## LINE SET



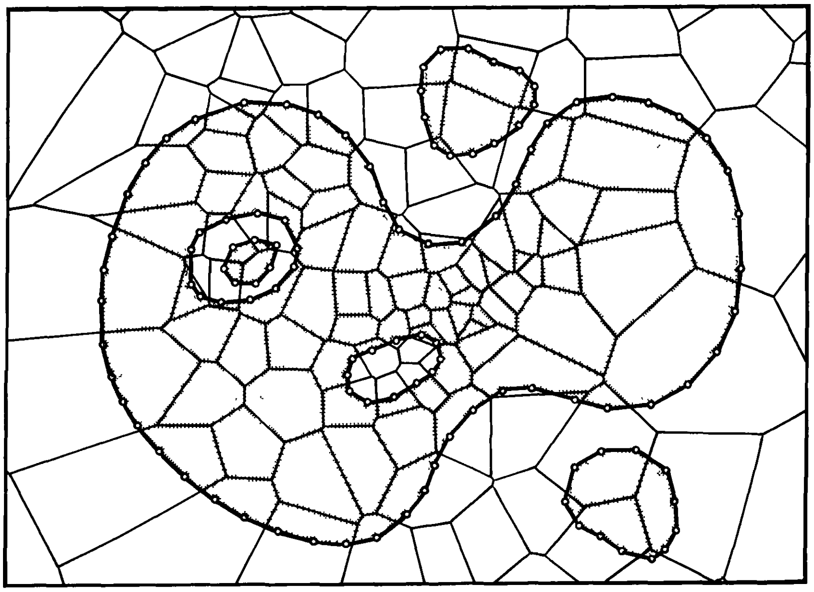
**Line set object descriptor:** Structure defining a line set object and describing its spatial extent. The structure consists of:

**Object identifier**
**Traversed cell identifier list**
**Line descriptor cluster length**
**Line descriptor cluster locator**

Consecutive vertices on the line must not coincide. The package provides the mechanism for the detection of coincident non-consecutive vertices, as well as intersection of linear segments. Such conditions are acceptable; any rules to the contrary must be defined within, and enforced by, the application. As mentioned above, a single segment can span many cells. Traversed cell may, in such case, contain no vertex; a single line fragment in it will be defined by the cell boundary intersection coordinate of two consecutive traversed cell descriptors.

Boundary intersection coordinate is a linear measure. Unlike vertices - which exist in "neutral" representation, and are hence intrinsic to the object - intersection is an artifact of the internal representation. The implementation therefore ensures that the intersection is numerically encoded with greater spatial resolution than that of the vertex.

## REGION



**Region object descriptor:** Structure defining a non-simply connected region object and describing its spatial extent. The structure consists of:

> **Object identifier**
> **Traversed cell identifier list**
> **Interior cell identifier list**
> **Line descriptor cluster length**
> **Line descriptor cluster locator**

Consecutive vertices must not coincide, the line must not cross itself, and the cell boundary intersection coordinate is treated as explained previously. Boundary ring direction is significant: by a common convention, interior is on the left-hand side.

The structure is capable of representing non-simply connected regions of any width (parallel "islands") or depth ("lake-island-lake..."). The package provides the width- and depth-unrestricted mechanism for the detection of inconsistencies in the ring direction.

While the manipulation of non-simply connected regions increases significantly the complexity of many algorithms employed by the package, this is an indispensable feature: either the union or the intersection of two simply-connected regions will, in general case, be a non-simply connected set. An application lacking this feature (i.e. an application restricted to simply-connected regions) would therefore be unable to treat a result of some of the two-dimensional unions or intersections in the same way as either of the production constituents. This might not be a major problem for applications that produce only graphical displays of such productions, but can weaken significantly those applications that create new spatial objects by applying a combination of spatial and non-spatial operators to the objects existing on their data bases.

## SPATIAL UNION AND INTERSECTION ALGORITHMS

The critical feature of all union and intersection algorithms in the Hipparchus package is their ability to avoid the spheroidal computational geometry in all instances where the spatial relationships can be resolved by simple comparison of cell identifier lists. As mentioned above, such lists are encoded in a form which exploits the sequences of cell numbers. (Strategic cell number clustering provides therefore an additional mechanism by which the application can improve the efficiency of the package.)

Once the problem requires that the computational geometry be performed, the algorithm will isolate cell or cells to which the solution is restricted. If no proximity criterion is involved - i.e. in case of unions and intersections between union-compatible sets - those cells will always represent the intersection of lists of traversed cells of two constituents. If the production involves a proximity criterion, the minimum and maximum cell-vertex distances in a pair of cells is used to reject from the geometry processing those sections of the object which either can not form part of the solution, or which are known to be part of the solution and can be mapped directly into the output set.

Since, in general case, a cell can contain fractions of more than one component-set, all geometry result elements must be stacked up, and the complete list must be traversed in order to properly connect new line or boundary ring segments. Such traverses are also restricted to the elements occurring within one cell.

Two-dimensional union and intersection algorithms must recognize coincident linear segments, in order to detect collapsing boundaries of two-dimensional objects. Where the complete boundary of both two-dimensional object collapses, the algorithm must be capable of determining whether the resulting object covers the whole domain, or whether is has no spatial coverage at all.

Line component-sets are obviously aggregate, but the boundary rings in a non-simply connected region could be ordered (width- or depth--first), according to their position in the component-set hierarchy. It is interesting that such ordering contributes nothing toward the simplification of union and intersection algorithms, and presents the problem of developing a reasonable convention which defines highest--order component-set on the spheroidal surface. Consequently, boundary rings in the Hipparchus structures are considered to be aggregate.

Finally, all computational geometry in Hipparchus package is performed using global coordinates. This avoids completely the problem of possible topology discrepancies between the spheroid and the projection plane; consequently, the design is free from any restrictions on the maximum length of the segment between consecutive vertices. Direction cosine form of the global coordinates provides for computational geometry algorithms based on vector algebra; such algorithms are both easier to program and simpler to test than the algorithms based on the conventional spherical latitude/longitude coordinates.