Spatial and Temporal Databases, Fall '15
October 5, 2015

Topic 3: Spatial Operations for intersection, union, and difference based on plane sweep method

Amritesh Randhi, Sarath Francis, Arvindh Mani,
Mickey Vellukunnel

(1) Motivation.

Spatial operations are computational manipulations of spatial objects that deepen our understanding of spatial phenomena. Spatial operations have been a focus of research in a number of disciplines like cognitive science, robotics, linguistics, artificial intelligence, and spatial reasoning. In particular as predicates, they support the design of suitable query languages for spatial data retrieval and analysis in spatial database systems and Geographic Information Systems. The most frequently used operations for spatial queries are the *set operations.* These operations apply to objects granularly represented as sets of points: *intersection, union and difference*. They are used in queries when computing a new spatial object. An example of this would be the real-life spatial database query, "Part of highways inside the State of California". Intersection is also the basis for *map overlays,* a central query for applications handling map layers. New layers are created by *overlaying* two or more layers; for example, land use can be overlaid with altitude. In this example, map overlay is the intersection between two sets of regions.

(2) Problems to be solved.

(1) Computation of spatial intersection between two spatial objects.

(2) Computation of spatial union between two spatial objects.

(3) Computation of spatial difference between two spatial objects.

(3) Data types (classes) and operations (methods) that are provided in our interfaces.

(1) Data types/Classes

(1) Generic **PlaneSweep** class

(2) Operations/methods

(1) SpatialObject* *function_name*(SpatialObject*, SpatialObject*)

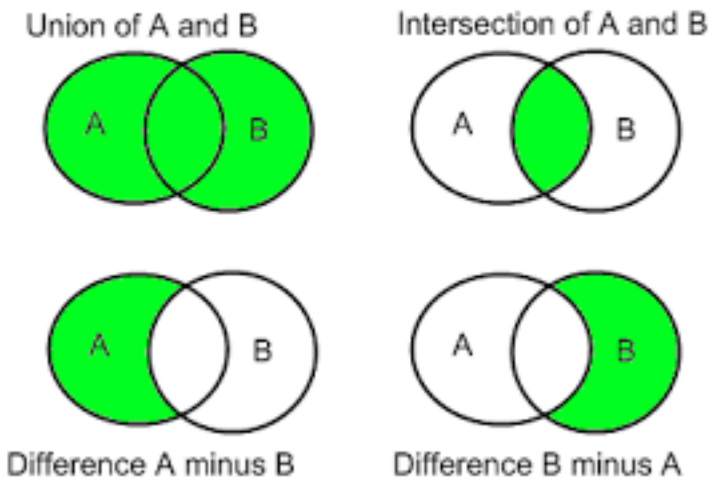
(4) Operations semantics descriptions, the input data types, and the output data types.

(1) The *function_name* operation defined above could be either of the three spatial operations of **intersection**, **union**, **difference**.

(1) **Intersection**: Given two spatial objects, give the Intersection between them, which is another spatial object containing the spatial region common to them both.

(2) **Union**: Given two spatial objects, give the Union between them, which is another spatial object containing all of the spatial region from both the spatial objects.

(3) **Difference**: Given two spatial objects, give the Difference between them, which is another spatial object, containing the spatial region from the first object minus the spatial region from the second object common in the first.



(2) Input data types

(1) SpatialObject: This is a union object in C++ of the form,

(1) union ***SpatialObject*** {

point2D *point;

line2D *line;

region2D *region;

};

1. **point2D** is a data type which is a collection of (x,y) coordinates from the geometric point set Ω.
2. **line2D** is a data type which is an ordered collection of **halfSegment2D** objects.
3. **region2D** is a data type which is an ordered collection of **halfSegment2D** objects.
    1. ***halfSegment2D*** is itself a datatype which is a collection of objects of the form {**segment2D**, **dominatingPoint**}.
        1. ***segment2D*** is a set (p,q) where p, q are of type point2D where p<q.
        2. ***dominatingPoint*** is a boolean variable which captures whether the half segment is the starting or ending portion of the segment.

(3) Output data types

(1) **SpatialObject**: Same as above. Here we can specify that the output data type could be internally different from the internal data types of the input *spatialObjects*. For example, if the internal data types of the intersection function is two region objects, then the output *spatialObject* could internally be either a *point, line or region* data type based on the behavior of the two objects.

(5) Interface requirements from other groups.

(1) We would require definitions and implementations of all the data types mentioned above in a header file, i.e. point2D, line2D, point2D, segment2D, halfSegment2D.