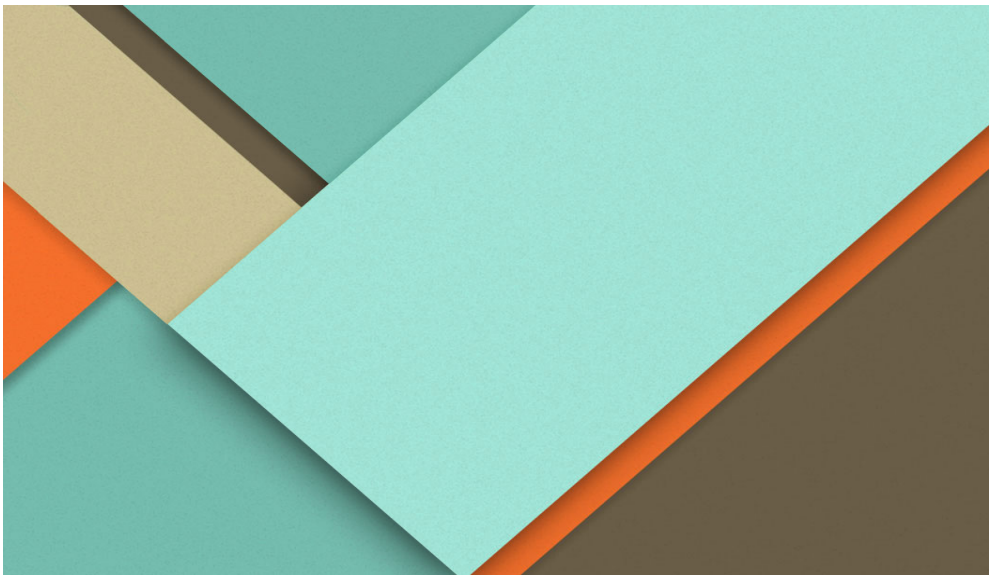

Spatial Data types
Point, Line and Region

10.04.2015

CIS 6930 - Fall 15, CISE UF

Ahmed Khaled, Namrata Choudhury, Deepa Narain, Revathi Kadari

Table of Contents

Motivation and Problems .....page 1
Data-types and Operations .....page 2
Semantic description of methods .....page 3
Formulation the Interface requirement for other groups .....page 5
References .....page 5

Overview

Motivation and Problems

Regular database systems and query engine cannot power up the applications of different requirements regarding managing geometric, geographical and spatial data objects. In order to facilitate efficient implementation strategies for topological relationships between complex spatial data types, we need to have a concrete idea of those data types. Applications analyzing topological relationships of spatial objects have some particular type of queries like spatial joins and spatial selections. To enable database capable of maintaining such requirements, spatial databases are required to include geometric data-types and structures.

Spatial types must include the basic and building space primitive blocks in-order to meet the complex applications and users queries. A spatial object is a complex object with associated value of spatial data type which might use relational, complex, object oriented data models. Though spatial query languages use the basic spatial data types, there is need to define the data types formally to consider the complex scenarios of Geographical Information Systems (GIS). Therefore, the definition and implementation of spatial data types is the most fundamental issue in developing a spatial database system.

Data-Types and Operations:

- "Point2D" Definition: Complex points - a finite collection of single points of type Poi2D.

  Operations:

  - Defining the point structure as an ordered set of Poi2D objects
  - Adding new points to the structure
  - Removing points from the structure
  - Check if a given point is in the structure or not
  - Destroying the point structure

- "Line2D" Definition: Geometric data-type in space as set of ordered 'n' segments (or 2n half segments); where two distinct half segments are either equal (for left and right half segment of segment), disjoint or meet.

  Operations:

  - Defining Line2D as an ordered set of Seg2D objects
  - Returning a vector of end points for each segment
  - Finding concatenated length of all the segments
  - Updating specific segments with the given end points, with a new vector of segments
  - Finding the number of blocks in the line
  - Returning a map of the block with the corresponding segment number
  - Destroying the line structure

- "Region2D" Definition: Complex/areal region data-type as set of ordered att-Seg2D structures that may have separations of the exterior (holes) and of the interior (multiple components)

Operations:

- Taking an ordered set of segments and creating the region object
- Returning a vector of end points of a segment
- Finding concatenated length of all the segments
- Updating specific segments with the given end points, with a new vector of segments
- Finding number of blocks in the region
- Computing the area of the region
- Computing the perimeter of the region
- Computing the appropriate minimum bounding rectangle of the region
- Returning the number of segments per block
- Destroying the region object

Semantic description of each method

	Method Name	Input	Output	What Does it Do?
	Point2D			
	Constructor	(n Poi2D)	(Point2D)	defines point2D as an ordered set of poi2D objects
	Constructor	file	(Point2D)	defines point2D as an ordered set of poi2D objects taken from a file
	Add	(Poi2D)	void	Adds a new point to the point2D object
	Delete	(Poi2D)	void	Removes a point from the point2D object
	Check	(Poi2D)	(bool)	Checks if a given point is in the point2D object or not
	Destructor	(Point2D)	void	Destroys the Point object

Line2D			
Constructor	(n Seg2D)	(Line2D)	defines Line2D as an ordered set of Seg2D objects
Constructor	(file)	(Line2D)	defines Line2D as an ordered set of Seg2D objects
EndPoints	(Line2D)	(n Poi2D)	returns the vector of the end-points of the line
ConcatenatedLength	(Line2D)	robust real	returns the concatenated length of the line
NumberOfSegments	(Line2D)	robust real	To find number of segments in given line2D
NumberOfBlocks	(Line2D)	robust real	returns the number of blocks of line structure
UpdateSeg	(index, Seg2D)	void	update the segment in the line 2D
Destructor	(Line2D)	void	Destroys the Line object
Region 2D			
Constructor	(n seg2D)	(Region 2D)	Takes ordered set of segments and creates a region object
Constructor	file	(Region2D)	Takes ordered set of segments from a file and creates a region object
EndPoints	(Region2D)	(n Poi2D)	returns the vector of all end-points
ConcatenatedLength	(Region2D)	robust real	returns the concatenated length of all segments
UpdateSegment	(poi2D, poi2D, n Seg2D)	void	update specific segment with the given endpoints, with the new vector of segments
NumberOfBlocks	(Region2D)	robust real	returns the number of blocks of region structure
Area	(Region2D)	Exp (robust real)	calculates the area of the region
Perimeter	(Region2D)	robust real	calculates the perimeter of the region

	MinBoundingRect	(Region2D)	(mbb2D)	calculates the minimum bounding rectangle for the region
	numberofSegs	(Region2D)	robust real	Returns the number of segments
	Destructor	(Region2D)	void	Destroys the region object

Formulation of interface requirements from other groups:

Our spatial complex data types highly depend on the accuracy of the big relational arithmetic operations and the primitive geometric types. The primitive geometric type for any complex spatial data type are simple point, simple minimum boundary rectangle besides the simple segment, half segment and attributed segment. All this primitive geometric types requires to be defined with either big integers or rational numbers, while other data-structures are built on top of this simple structure.

1. Simple 2D point, segment, half-segment, attributed segment and minimum boundary geometric types.
2. ability to access the internal structure and update the different values.
3. Big integer and rational number functions to assign and access values, through setting and getting are required.
4. Different mathematical operations and checks are required:
    - check the numerical relationship between two values (e.g. =, < and >).
    - performing basic arithmetic operations (e.g. addition, subtraction, division) besides the advanced operations (e.g. square root, exponentiation) for two values.

References:

1. Reasey Praing & Markus Schneider. Topological Feature Vectors for Exploring Topological Relationships. Int. Journal on Geographical Information Science(IJGIS), 23(3), 319-353, 2009.
2. R.H. Gueting. An Introduction to Spatial Databases. VLDB Journal (Special Issue on Spatial Databases), 3(4):357-399, 1994.