Processing Object-Orientation-based Direction Queries in SDB

Xuan Liu

IBM T.J. Watson Research Center

Shashi Shekhar

University of Minnesota

Sanjay Chawla

Vignett Corporation

Outline

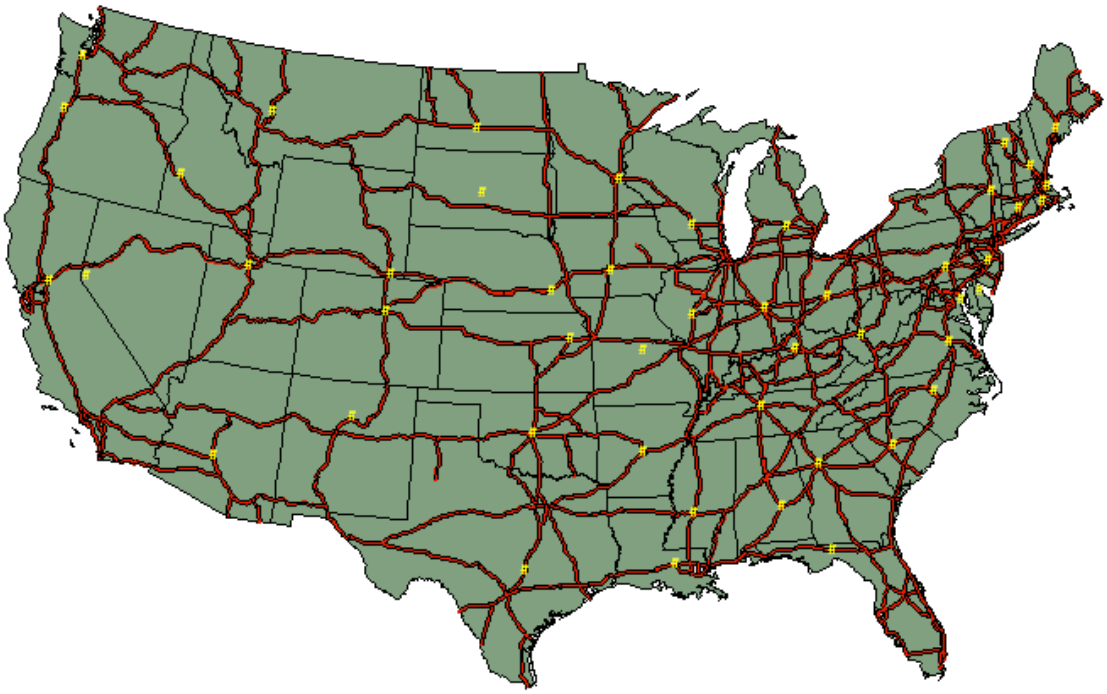
⇒ Introduction

- Problem Formulation and Related Works
- Our Approach
- Experimental Evaluation
- Conclusions and Future Work

Spatial Databases

- Effective and Efficient Management of Spatial Data
- Example Applications
  - Geographic Information Systems(GIS)
  - Computer Aided Design(CAD)
  - Multimedia Information System
- Commercial Examples
  - Informix's Spatial Datablades
  - Oracle's Spatial Cartridge
  - ESRI's Spatial Database Engine(SDE)

Examples of Spatial data



- Spatial data
  - Cities(points), Highways(Lines), States(Polygons)
- Non-spatial data
  - Cities(pop), Highways(Name), States(GDP)

Spatial Queries

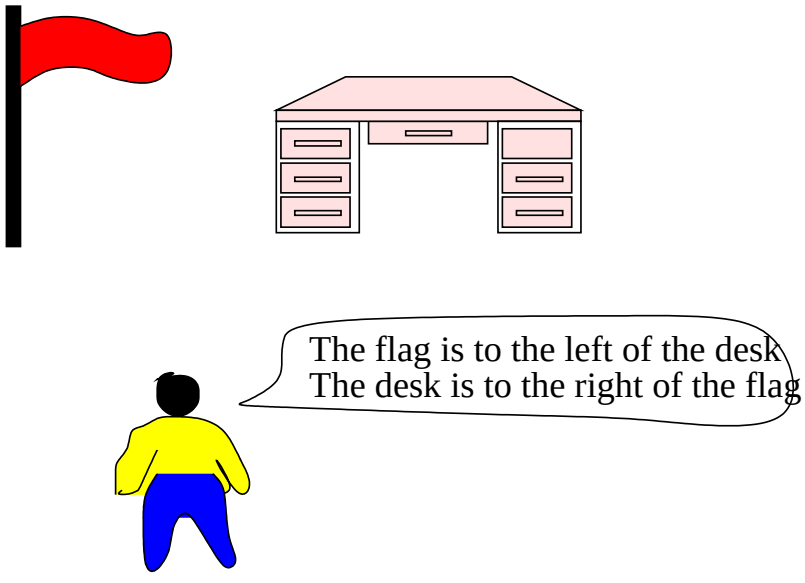
- Distance-based
  - List all cities within 200 miles of New York
- Topological(adjacent, overlap, etc.)
  - Which state has minimum number of neighbors?
- Direction-based(South, left, etc.)
  - List all cities North of Minneapolis
  - List all cities above Minneapolis

Table of GIS Operations

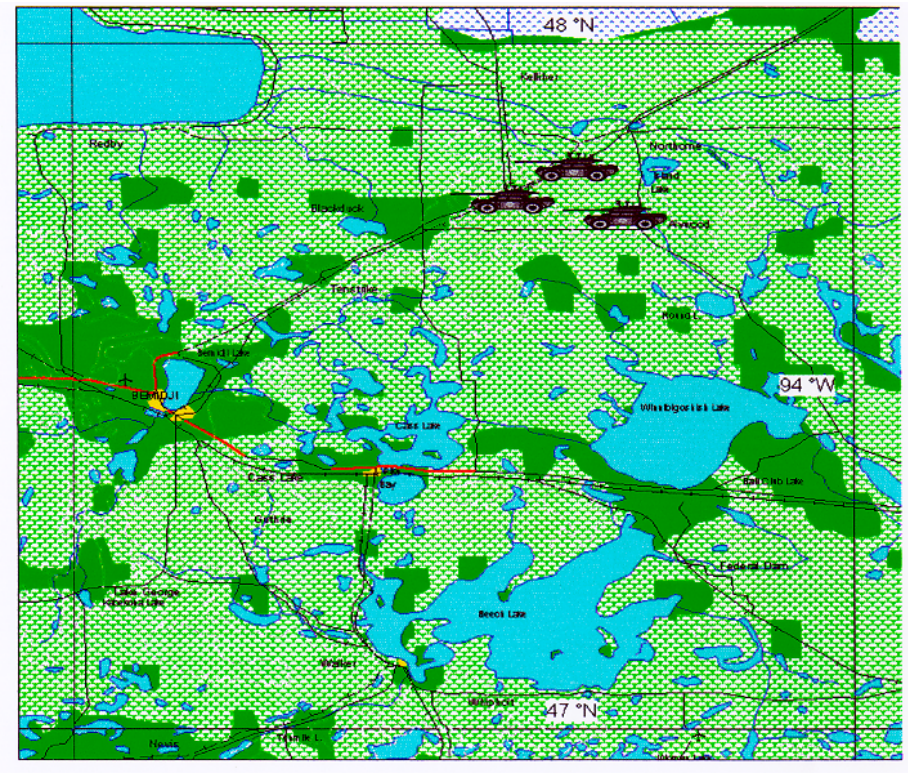
Data model	Operator Group	Operation
Vector Object	Set-Oriented	equals, is a member of, is empty, is a subset of, is disjoint from, intersection, union, difference, cardinality
	Topological	boundary, interior, closure, meets, overlaps, is inside, covers, connected, components, extremes, is within
	Metric	distance, bearing/angle, length, area, perimeter, centroid
	Directional	east, north, northwest, left, front, between
	Network	successors, ancestors, connected, shortest-path
Raster field	Local	Point-wise sums, differences, maximums, means, etc
	Focal	slope, aspect, weighted average of neighborhood
	Zonal	sum or mean or maximum of field values in each zone

Different Direction Reference Systems

- Absolute Direction
  - east, west, northwest,...
  - Defined w.r.t. the global coordinate system.
- Object-orientation-based Direction
  - front, right, right-front, ...
  - Defined w.r.t. the orientation of the reference object
- Viewer-orientation-based Direction
  - front, right, right-front, ...
  - Defined w.r.t. the orientation of the viewer



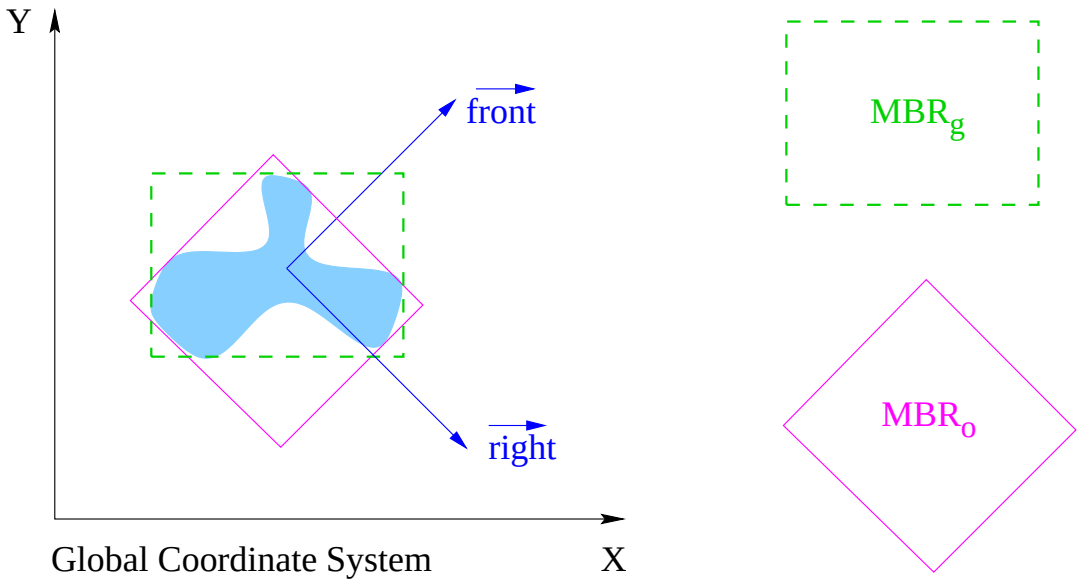
Object-orientation-based Direction Queries



- Query 1: List the farm fields which are **left-front** of the tank and are suitable for tank movement.
- Query 2: List the swamps **behind** the Tank.

Basic Concepts: MBR Approximation

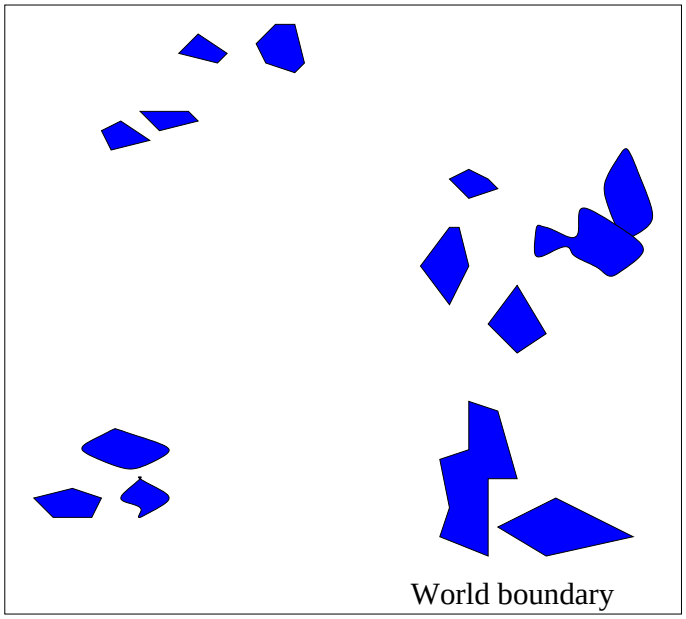
- Minimum Bounding Rectangle
- Different MBRs according to different coordinate systems



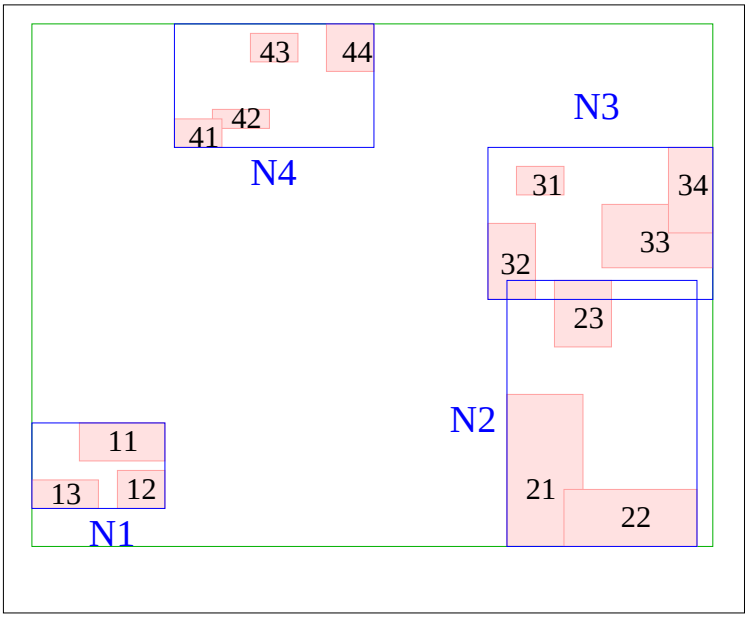
- Notation
  - Data objects are approximated by MBR_g
  - Reference objects are approximated by MBR_o
  - MBR refers to MBR_g in the rest of talk

Example data set

- Dataset



- Corresponding R-tree



Outline

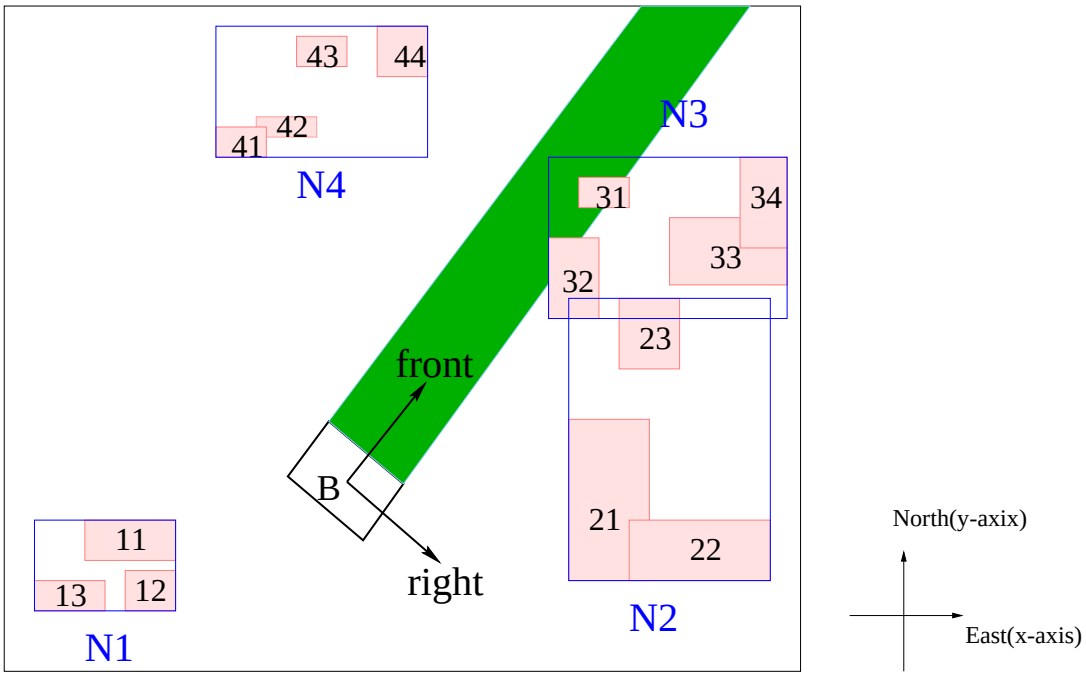
- Introduction
- ⇒ Problem Formulation and Related Works
- Our Approach
- Experimental Evaluation
- Conclusions and Future Work

Problem Statement: OODQ

- Object-orientation-based Direction Query (OODQ)
- **Given:**
  - **A query involving object-orientation-based directions**
  - **A R-tree index of the data set**
- **Find:**
  - **All objects that satisfy the selection conditions**
- **Objective:**
  - **Minimize the number of page access**
- **Constraint:**
  - **Objects are approximated by MBR**
  - **One R-tree index based on global coordinate system**

Example of OODQ Problem

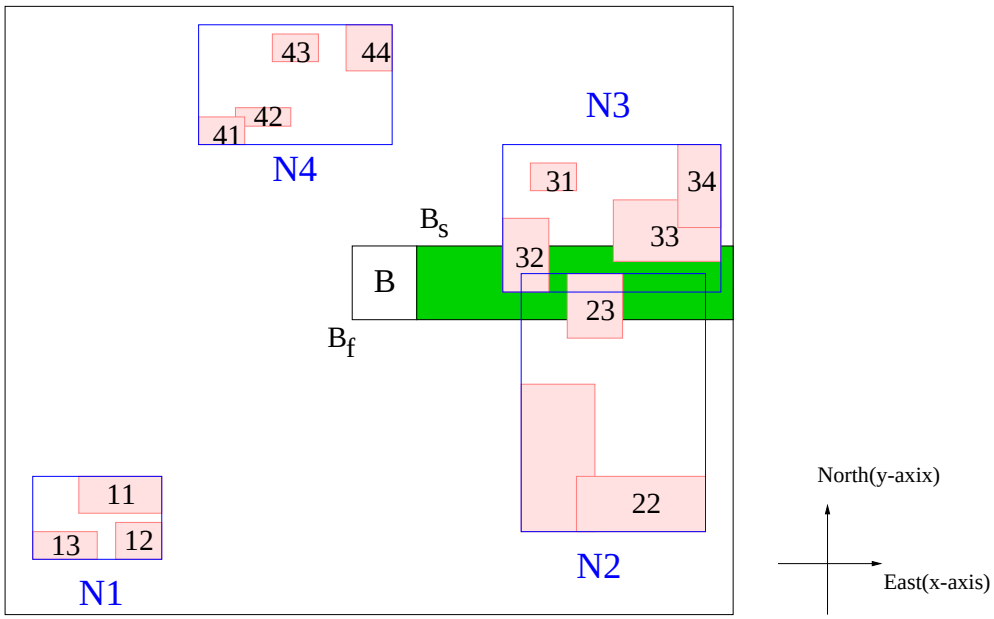
- Query: List all objects **in front** of B



- Results
  - N31, N32

Related Works on Processing Direction Queries

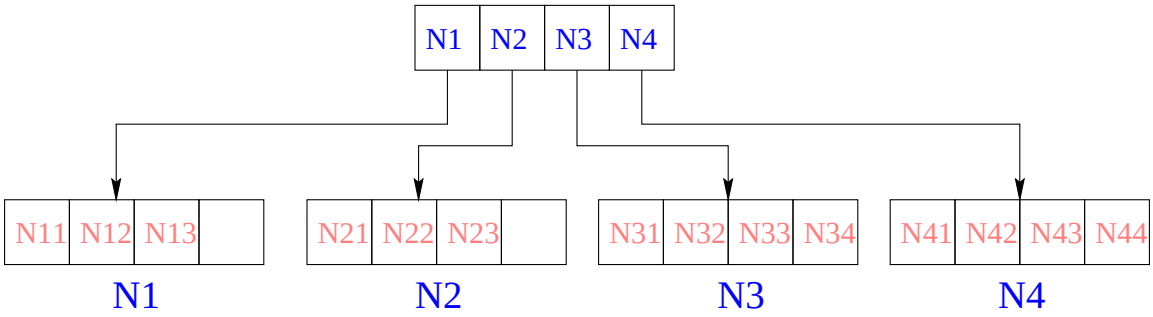
- Focusing on absolute directions[TP95, Te96, PTS94]
- Using range query strategy
- Example: List all objects east of B



- The direction region is an orthogonal rectangle

Range Query Strategy for Absolute Directions(RQS_AD)

- R-tree



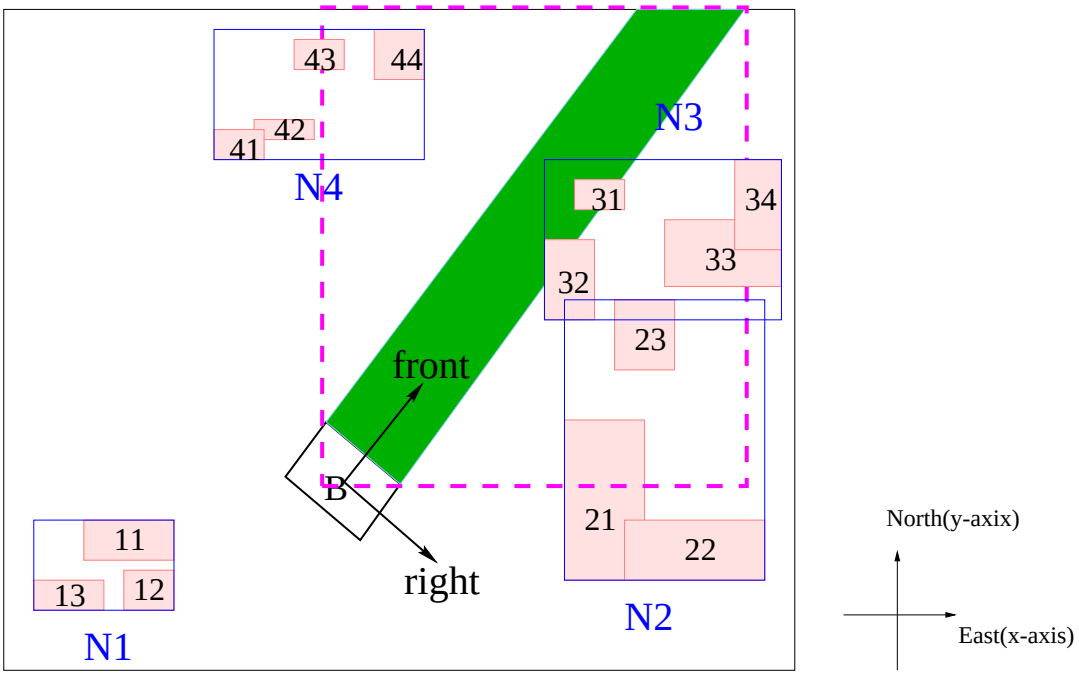
- Trace of query processing

Steps	MBRs excluded	MBRs remain
Step 1	N1, N4	N2, N3
Step 2	N21, N22, N31, N34	N23, N32, N33

- Range query strategy(RQS) is efficient[PTS94]

Processing Object-orientation-based Queries(OODQ)

- Query: List all objects in front of B



- Challenges
  - Direction region is not orthogonal
  - Orientation may change due to motion
    - Recompute R-tree is expensive

Extending RQS for OODQ

- Processing OODQ using RQS

```
RQS(R-tree *rtree, OODQ aQuery) {
    directionRegion = computeDirRegion(aQuery);
    directionMBR = getMBR(directionRegion);
    intermediateResult = RQS_AD(rtree, directionMBR);
    resultSet = postFilter(intermediateResult);
}
```

  - Applying RQS_AD using directionMBR

Steps	MBRs excluded	MBRs remain
level 1	N1	N2, N3, N4
level 2	N41, N42, N22	N21, N23, N31, N32, N33, N43, N44

  - postFilter step
    - Excluding N21, N23, N33, N43, N44
    - Result = N31, N32

Limitations and Our Contributions

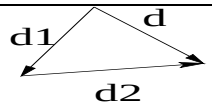
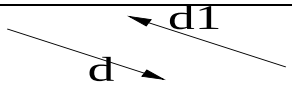
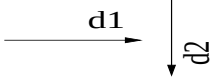
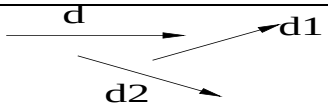
- Limitations
  - Need to know world boundary
  - Need to calculate the directionMBR
  - May incur large unnecessary I/O and CPU cost
  - PostFilter step is needed even for MBR objects
- Our Contributions
  - New ADT for open shapes
  - New strategy: Openshape-based strategy(OSS)
    - Do not need to know world boundary
    - Reduce unnecessary I/O and CPU cost
    - Eliminate PostFilter step for MBR objects
  - Experimental Evaluation:
    - OSS outperforms RQS in both I/O and CPU

Outline

- Introduction
- Problem Formulation and Related Works
- ⇒ Our Approach
  - Modeling Directions
  - OpenShape-based Strategy(OSS)
  - Defining OpenShape ADT
- Experimental Evaluation
- Conclusions and Future Work

Modeling Directions

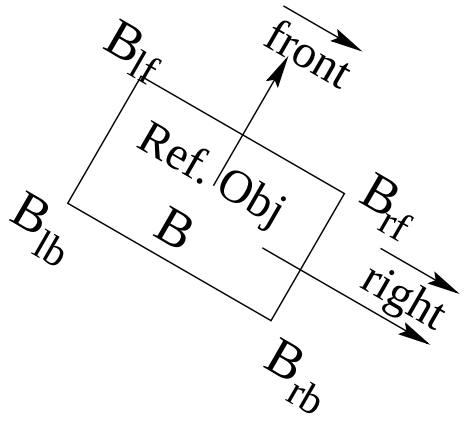
- Model direction as an **Object**
  - A unit vector
- Basic operations

composition	 d = composition(d1,d2)
reverse	 d = reverse(d1)
deviation	 dev(d1,d2)=dot-product(d1,d2)
isBetween	 d is between d1 and d2

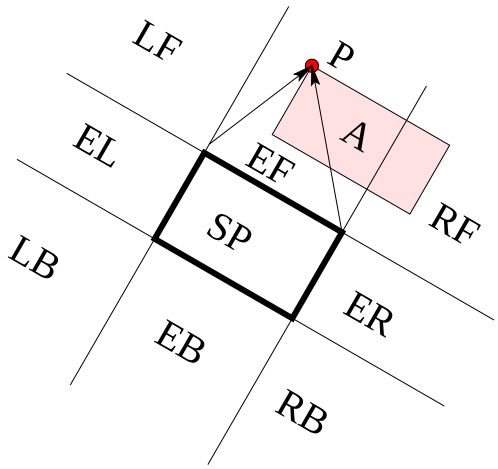
- Definition of Operations

Operations	Definition
composition	$\vec{d1} + \vec{d2} = \frac{\vec{d1}+\vec{d2}}{\lvert\vec{d1}+\vec{d2}\rvert}$
reverse	$(-1) \times \vec{d1}$
deviation	$cos\theta = \vec{d_1} \odot \vec{d_2}$
isBetween	$\vec{d}$ isBetween $\vec{d_1}$ and $\vec{d_2}$ if $\exists c_1 > 0, c_2 > 0$ s.t. $\vec{d} = c_1\vec{d_1} + c_2\vec{d_2}$

OODQ predicates for simple Regions



(a) Reference Object



(b) Direction Regions

- EF(A, B) is TRUE iff $\exists$ point P $\in$ A
  - $\vec{B_{lf}}P.isBetween(\vec{front}, \vec{right})\wedge$
  - $\vec{B_{rf}}P.isBetween(\vec{front}, \vec{right}.reverse())$

OODQ Predicates for Simple Regions

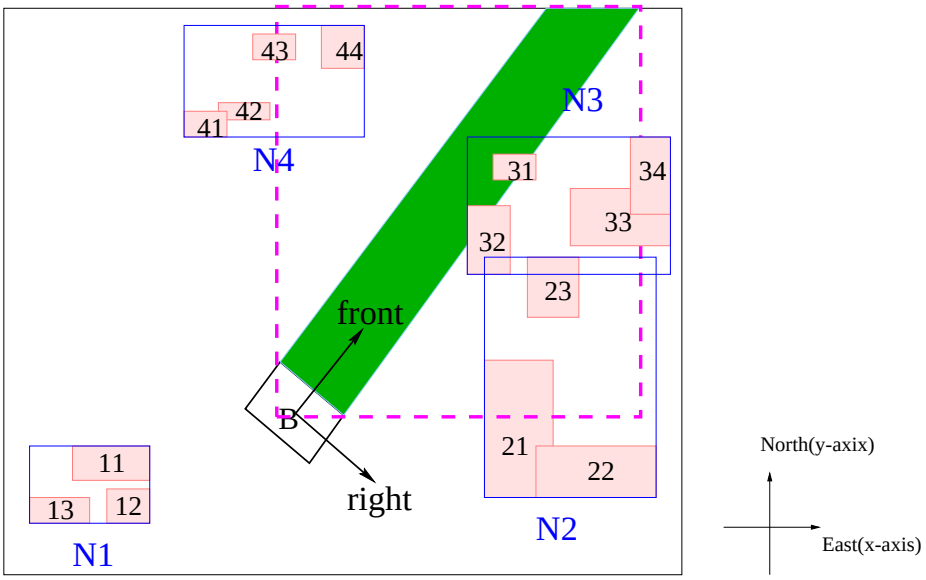
- Definition

Predicates	Definitions
$SP(A, B)$	$\exists$ point P $\in$ A, s.t. P $\in$ interior(B)
$EF(A, B)$	$\exists$ point P $\in$ A, s.t. $\vec{B_{lf}}P.isBetween(\vec{front}, \vec{right}) \wedge$ $\vec{B_{rf}}P.isBetween(\vec{front}, \vec{right}.reverse())$
$EB(A, B)$	$\exists$ point P $\in$ A, s.t. $\vec{B_{lb}}P.isBetween(\vec{front}.reverse(), \vec{right}) \wedge$ $\vec{B_{rb}}P.isBetween(\vec{front}.reverse(), \vec{right}.reverse())$
$ER(A, B)$	$\exists$ point P $\in$ A, s.t. $\vec{B_{rf}}P.isBetween(\vec{front}.reverse(), \vec{right}) \wedge$ $\vec{B_{rb}}P.isBetween(\vec{front}, \vec{right})$
$EL(A, B)$	$\exists$ point P $\in$ A, s.t. $\vec{B_{lb}}P.isBetween(\vec{front}, \vec{right}.reverse()) \wedge$ $\vec{B_{lf}}P.isBetween(\vec{front}, \vec{right})$
$RF(A, B)$	$\exists$ point P $\in$ A, s.t. $\vec{B_{rf}}P.isBetween(\vec{front}, \vec{right})$
$RB(A, B)$	$\exists$ point P $\in$ A, s.t. $\vec{B_{rb}}P.isBetween(\vec{front}.reverse(), \vec{right})$
$LF(A, B)$	$\exists$ point P $\in$ A, s.t. $\vec{B_{lf}}P.isBetween(\vec{front}, \vec{right}reverse())$
$LB(A, B)$	$\exists$ point P $\in$ A, s.t. $\vec{B_{lb}}P.isBetween(\vec{front}.reverse(), \vec{right}reverse())$

- **Lemma 1** ***Predicate set defined above is complete***

Open Shape-based Strategy(OSS)

- Basic idea
  - Use actual direction region as the filter
- Example revisit



- Trace of processing using OSS

Steps	MBRs excluded	MBRs remain
level 1	N1, N2, N4	N3
level 2	N33, N34	N31, N32

- R-tree page accessed: 1+1=2

OSS Algorithm

Input: $rtree$ points to the R-tree of data objects;
ref is the reference object with orientation;
dp is the direction predicate involved in the query;
Output: resultSet contains all object MBRs which satisfy the query

```
OSS(R-tree *rtree, OrientedObject ref, Predicate dp) {
    Object *resultSet = ∅ ;
    OpenShape dpRegion = ConstructOpenShape (ref, dp);
    osQuery(rtree, dpRegion, resultSet);
}

osQuery(R-tree *rtree, OpenShape dpRegion, Object *resultSet) {
    R-tree *currentNode = rtree;
    if dpRegion.interiorIntersects(currentNode.Mbr) {
        if (currentNode is a leaf node)
            Add currentNode to resultSet;
        else
            for each childNode ∈ child nodes of currentNode
                osQuery(childNode, dpRegion, resultSet);
    }
}
```

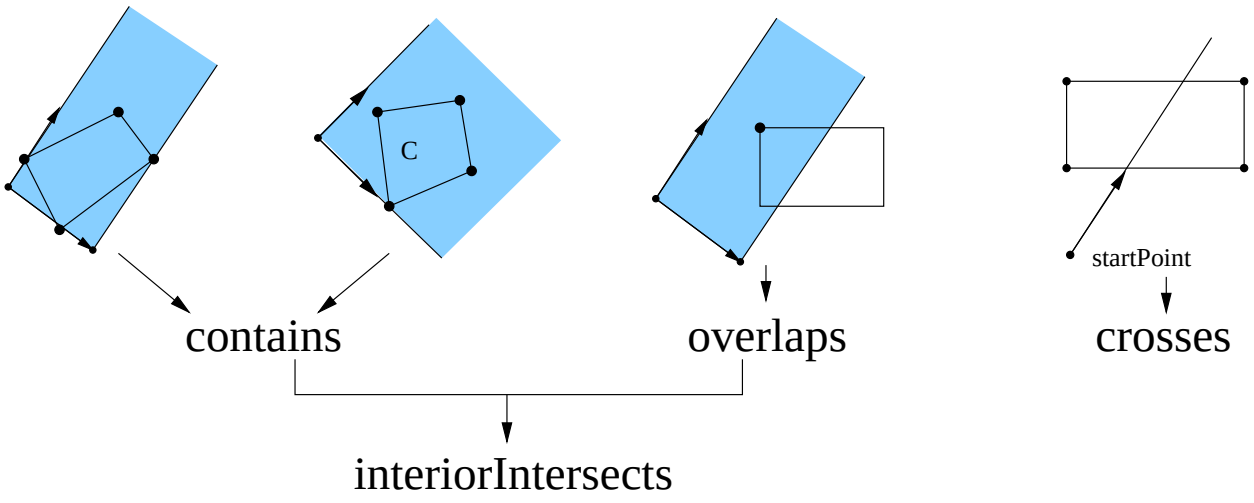
- **Lemma 2** ***Proposed strategy OSS is complete and correct.***

OpenShape ADT

- Open Shapes
  - Geometries which conceptually extends beyond the world boundary
- A base class hierarchy for open geometries
  - Function Interface (in C++ notation)

```
class   OpenShape {
public:
        virtual Boolean contains(Geometry) const;
        virtual Boolean crosses(Geometry) const;
        virtual Boolean interiorIntersects(Geometry) const;
        virtual Boolean overlaps(Geometry) const;
        ⋮
};
```

- Examples of Operations



Definitions of Operations

- O : OpenShape, C : close geometry
- crosses
  - O.crosses(C) = true iff

    $O.\vec{dir} \odot Direction(C - O.startPoint) = 1$; `(if C is a point)`

    $\exists$ `a point` P `s.t.` $P \in interior(O) \cap interior(C)$; `(if C is not a point)`
- contains
  - O.contains(C) = true iff

    $(O \cap C = C)$ `and` $(interior(O) \cap interior(C) \neq \emptyset)$
- overlaps
  - O.overlaps(C) = true iff

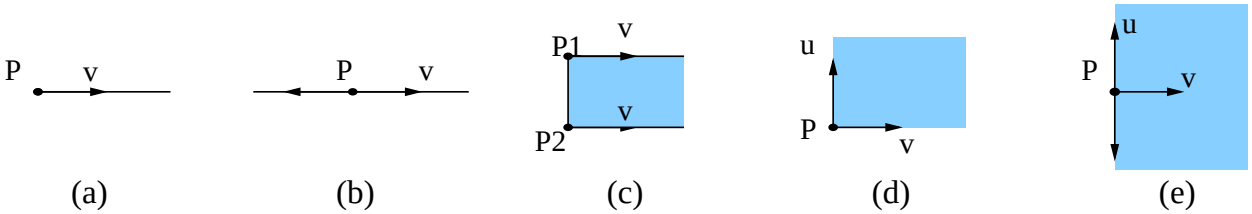
    $(O \cap C \neq C)$ `and` $(interior(O) \cap interior(C) \neq \emptyset)$
- interiorIntersects
  - O.interiorIntersects(C) = true iff

    `interior(C)` $\cap$ `interior(O)` $\neq \emptyset$

    - Covers both contains and overlaps

Defining Open Shapes

- Examples



(a) (b) (c) (d) (e)

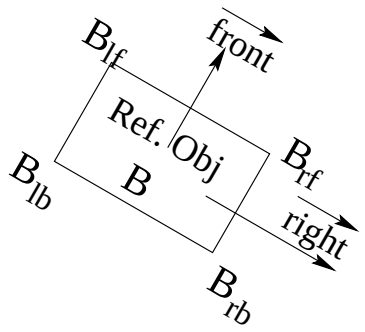
- Defining Open Rectangles

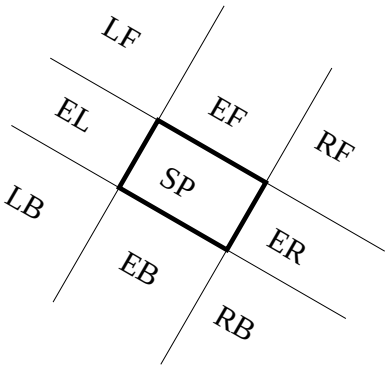
ADT	attributes	constructors
OpenLine1	$startPoint : Point$; $dir : Direction$;	OpenLine1(Point, Direction);
OpenLine2	$interPoint : Point$; $dir : Direction$;	OpenLine2(Point, Direction);
OpenRect1	$vertex_1 : Point$; $vertex_2 : Point$; $dir : Direction$;	OpenRect1(Point, Point, Direction);
OpenRect2	$startPoint : Point$; $dir1 : Direction$;	OpenRect2(Point, Direction, Direction);
OpenRect3	$line : OpenLine2$; $dir : Direction$;	OpenRect3(OpenLine2,Direction);

(a) OpenLine1(P,$\vec{v}$); (b) OpenLine2(P, $\vec{v}$)

(c) OpenRect1($P_1, P_2, \vec{v}$); (d) OpenRect2($P, \vec{u}, \vec{v}$);

Interpreting Direction Predicates Using OpenShape



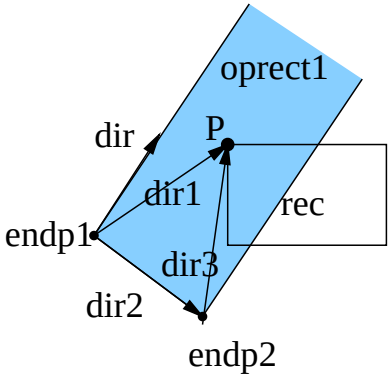


- Interpreting predicates using OpenShape

Predicates	Open Shape	Predicate true iff A interiorIntersects the following open regions
$EF(A,B)$	OpenRect1	$OpenRect1(B_{lf}, B_{rf}, \vec{front})$
$EB(A,B)$		$OpenRect1(B_{lb}, B_{rb}, \vec{front}.reverse())$
$ER(A,B)$		$OpenRect1(B_{rf}, B_{rb}, \vec{right})$
$EL(A,B)$		$OpenRect1(B_{rf}, B_{rb}, \vec{right}.reverse())$
$RF(A,B)$	OpenRect2	$OpenRect2(B_{rf}, \vec{right}, \vec{front})$
$LF(A,B)$		$OpenRect2(B_{lf}, \vec{right}.reverse(), \vec{front})$
$RB(A,B)$		$OpenRect2(B_{rb}, \vec{right}, \vec{front}.reverse())$
$LB(A,B)$		$OpenRect2(B_{lb}, \vec{right}.reverse(), \vec{front}.reverse())$

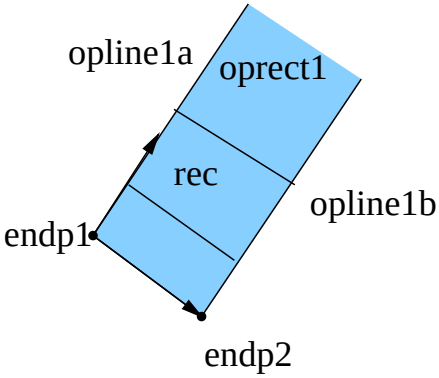
interiorIntersects Operation

- OpenRect1.interiorIntersects(rectangle)



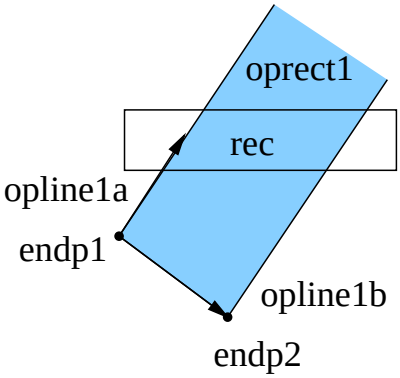
oprect1.contains(p)

(a) one endpoint inside



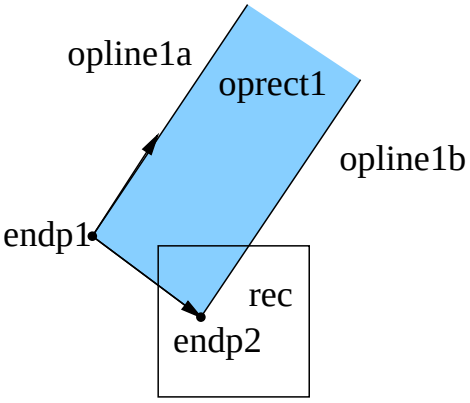
oprect1.contains(rec)

(b) all endpoints on bundary



opline1a.crosses(rec) ||
opline1b.crosses(rec)

(c) at least one ednpoint outside



rec.contain(endp2) ||
rec.contain(endp1)

(d)

OpenRect1.interiorIntersects(Rectangle)

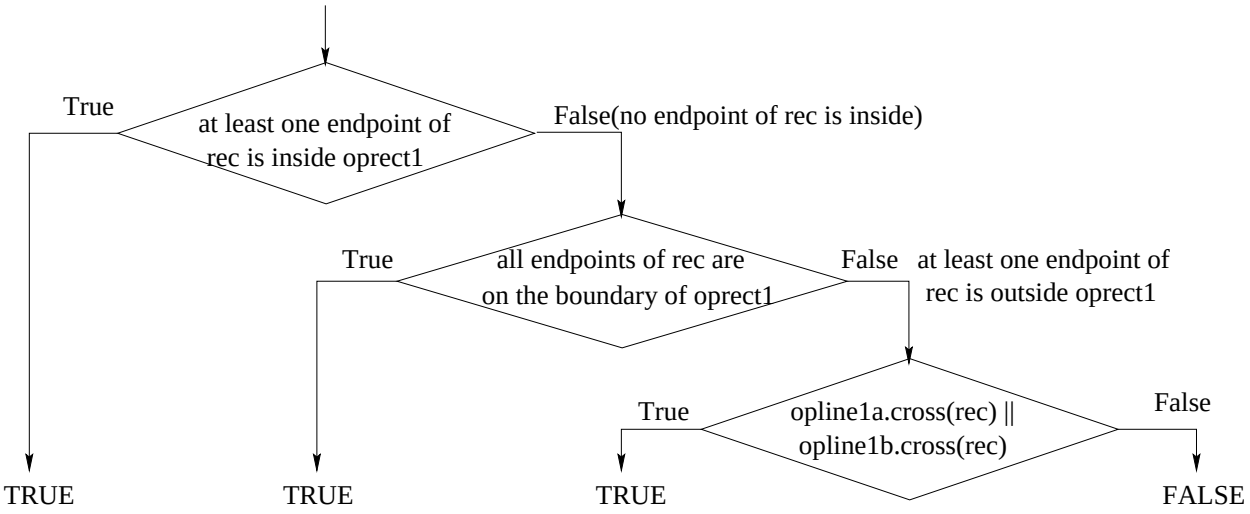
Input: rec is the rectangle that needs to be checked;
the current OpenRect1 object, which has attributes $(endp1, endp2, dir)$;
Output: $TRUE$ if overlaps,$FALSE$ otherwise

```
OpenRect1::overlaps(Rectangle rec) {
    for each ep ∈ endpoints of rec
        if (contains(ep))
            return TRUE;

    if (contains(rec))
        return TRUE;

    opline1a = OpenLine1(endp1, dir);
    opline1b = OpenLine1(endp2, dir);
    if ( opline1a.crosses(rec) || opline1b.crosses(rec) )
        return TRUE;

    return FALSE;
}
```

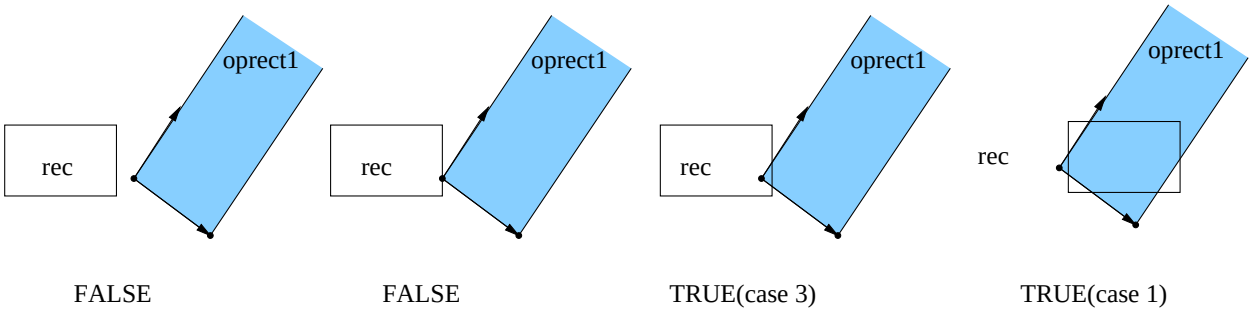
Completeness and Correctness

Lemma 3 ***Algorithm interiorIntersects is correct and complete.***

- Control flow

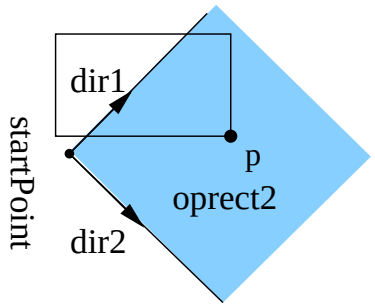


- Possible relationships(plane sweep argument)

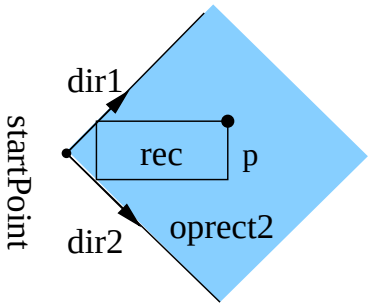


FALSE FALSE TRUE(case 3) TRUE(case 1)

OpenRect2.interiorIntersects(Rectangle)

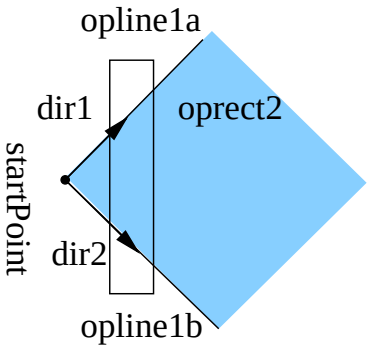


oprect2.contain(p);



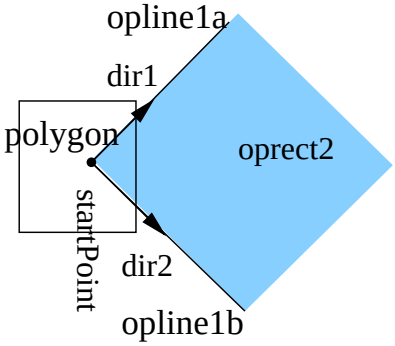
oprect2.contain(p);

(a) at lease one endpoint inside



opline1a.crosses(rec) ||
opline1b.crosses(rec)

(b) at least one endpoint outside



polygon.contain(startPoint)

(d)

OpenRect2.interiorIntersects(Rectangle)

Input: rec is the rectangle that needs to be checked;
the current OpenRect2 object with attributes $(startPoint, dir1, dir2)$;
Output: $TRUE$ if interiorIntersects, $FALSE$ otherwise

```
OpenRect2::interiorIntersects(Polygon aPolygon) {
    /* case 1, figure (a) */
    for each ep ∈ endpoints of aPolygon
        if (contains(ep))
            return TRUE;

   /* case 2, figure (b) */
    opline1a = OpenLine1(startPoint, dir1);
    opline1b = OpenLine1(startPoint, dir2);
    if ( opline1a.crosses(aPolygon) || opline1b.crosses(aPolygon) )
        return TRUE;

    return FALSE;
}
```

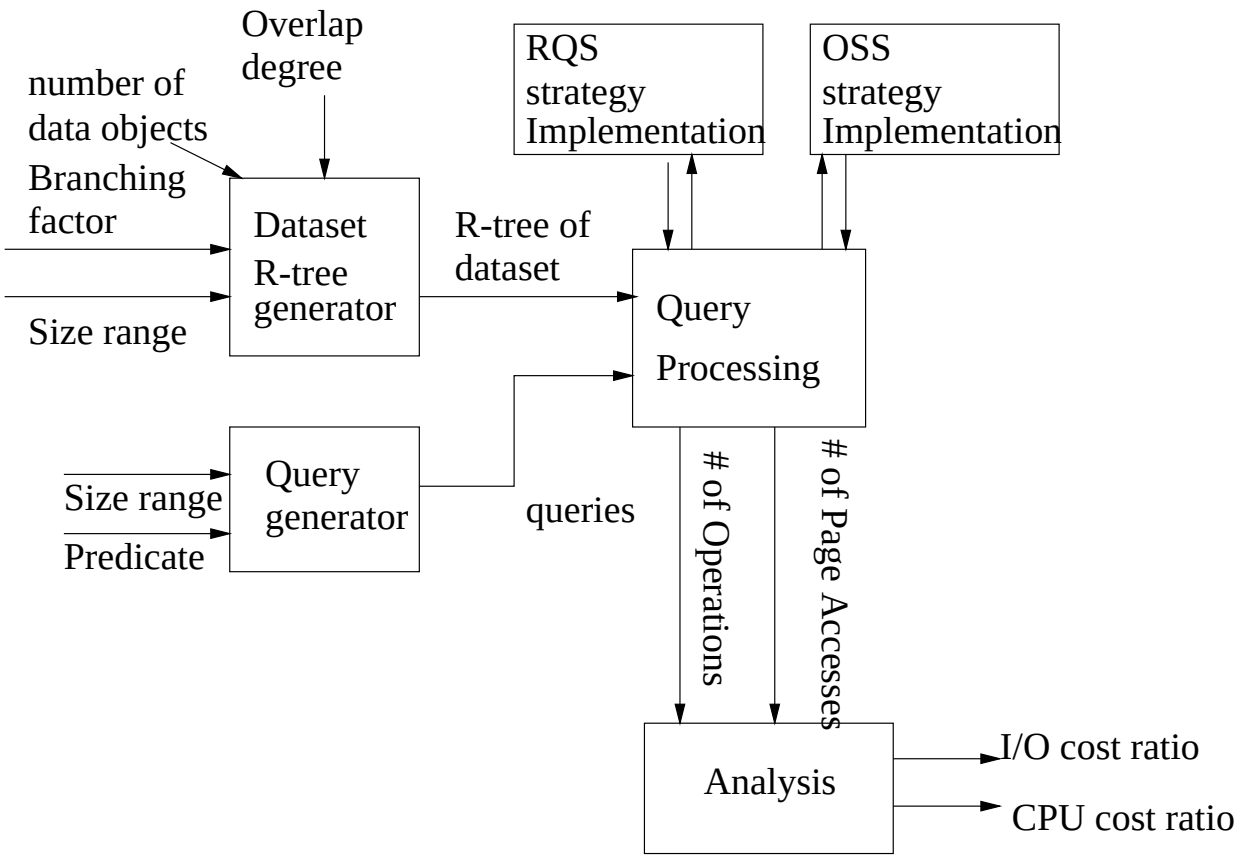
Outline

- Introduction
- Problem Formulation and Related Works
- Our Approach
- ⇒ Experimental Evaluation
  - Experiment Design
  - Experimental Results
- Conclusions and Future Work

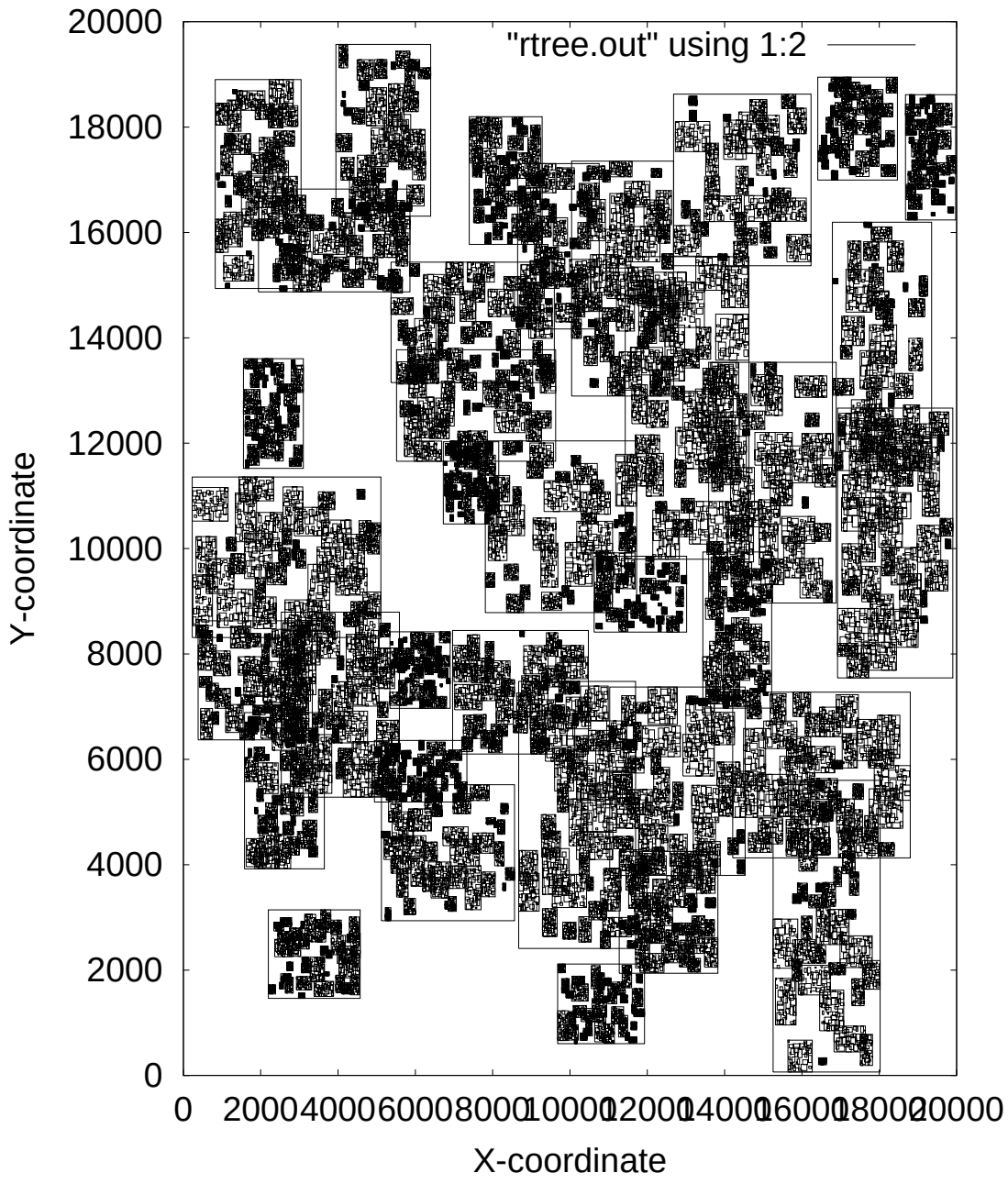
Experiment Design

- Strategies
  - OSS, RQS
- Data set
  - Randomly generated Data Sets
  - Randomly generated reference object set(1500)
- Variable parameters
  - The orientations of reference objects
  - The size of reference objects
  - The number of objects in data set
- Results
  - Disk pages accessed
  - number of operations performed

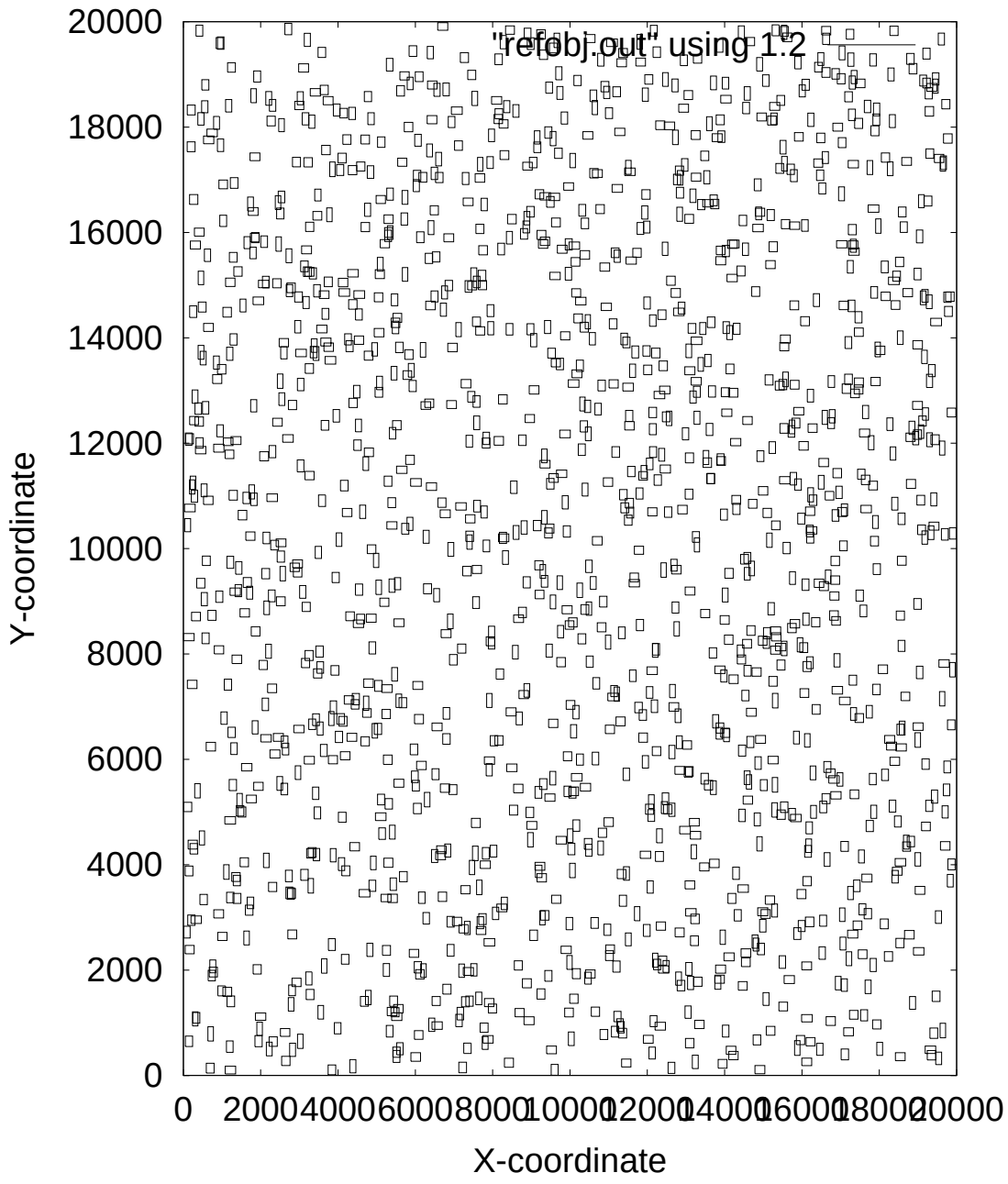
Experimental Setup



Example Data Set

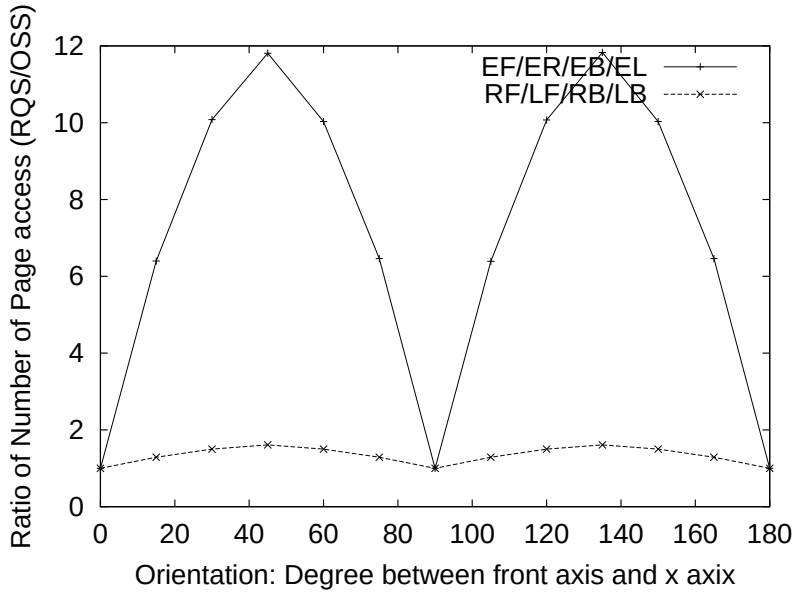


Example Reference Object



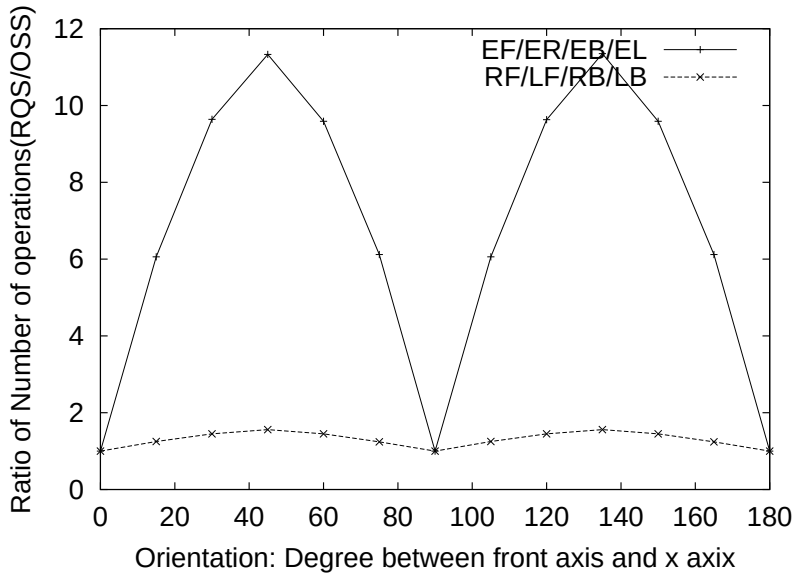
Effect of Reference Object's Orientation

- Fixed Parameters
  - Data set(Data R-tree, 42875 objects)
  - The locations and size of reference objects
- Variable Parameters
  - Orientation of reference objects
- Number of Page access



Effect of Reference Object's Orientation

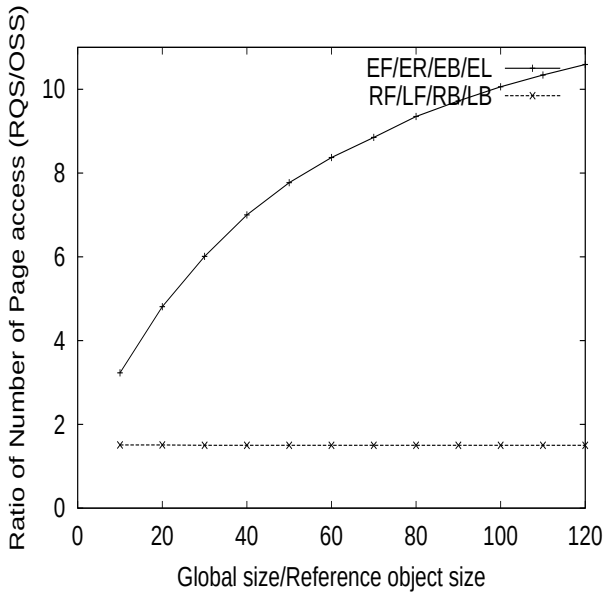
- Number of operations

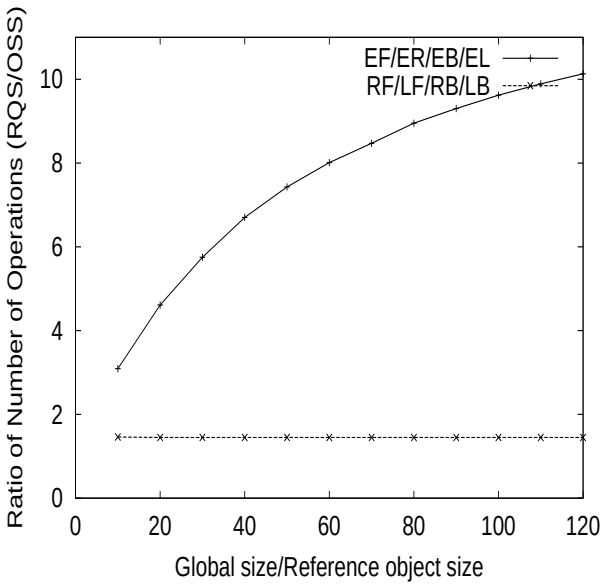


- Observations
  - The relative performance advantage of OSS varies cyclically
  - The performance ratio reaches maximum when the degree close to $\frac{\pi}{4} + k \times \frac{\pi}{2}$ (k is an integer). e.g. 45^0, 135^0
  - The performance is equal to 1 if the degree equals $\frac{\pi}{2} \times k$ (k is an integer) e.g. 0^0, 90^0, 180^0
  - OSS performs much better on average

Effect of the Size of Reference Objects

- Fixed Parameters
  - Data set(Data R-tree, 42875 objects)
  - The orientation degree = 30^0
- Variable Parameters
  - The Reference Objects with Different size.





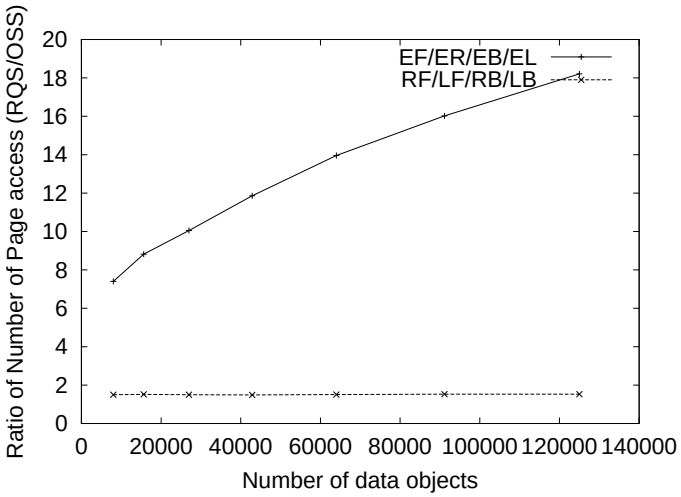
- Observations
  - OSS consistently outperforms RQS
  - The relative performance advantage for EF-type predicates increases steadily

Effect of the size of data set

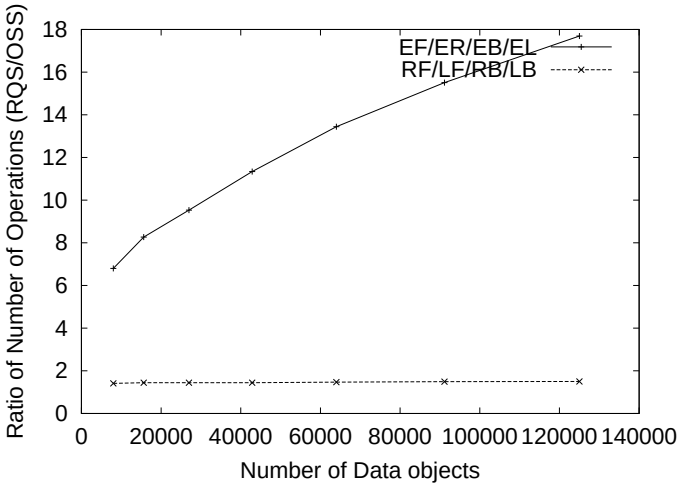
- The number of data objects in data set
- Fixed Parameters
  - The orientation degree = 30^0
  - Reference Objects (1 percent of global size)
- Variable Parameters
  - The size of data set from 20000 to 125000

Effect of the size of data set

- Number of page access



- Number of operations



- Observations
  - OSS consistently outperforms RQS
  - OSS scales better than RQS

Conclusions and Future work

- Conclusions
  - Object model of directions
  - New ADT for open shapes
  - New strategy: Open shape-based strategy(OSS)
  - Experiment Evaluation: OSS outperforms RQS
- Advantages
  - Improve filtering efficiency by eliminating false hits
  - Reduce unnecessary I/O and CPU cost
  - Eliminate postFiltering step for MBR objects
  - Do not need to have knowledge of world boundary
- Future Work
  - Viewer-orientation-based predicates
  - More complicated objects
  - Different direction predicate set
  - Join on direction predicates
  - Mobile applications
  - Robotics and motion planning applications

Reference

References

[PTS94] D. Papadias, Y. Theodoridis, and T. Sellis. The Retrieval of Direction Relations Using R-trees. *Proceedings of the 5th Conference on Database and Expert Systems Applications(DEXA)*, 1994.

[Te96] Yannis Theodoridis and etc. Supporting Direction relations in Spatial database Systems. 30(10):1249–1257, 1996.

[TP95] Y. Theodoridis and D. Papadias. Range Queries Involving Spatial Realtions: A performance Analysis. *Proceedings of the 2nd Conference on Spatial Information Theory(COSIT)*, 1995.