Querying and Indexing of moving objects

Nirmesh Neema
Guided By: Prof. N.L. Sarda

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

June 24, 2011
Outline

Introduction
Need for Spatio-Temporal Data type and Predicates
PostGIS

Spatio-temporal Data Type and Predicates
Modeling of Spatio-Temporal Data type
Basic Spatio-Temporal Predicates

Indexing Techniques
Spatio-temporal Indexing

Implementation
Datatype: mpoint
Spatio-Temporal Functions
Spatio-Temporal Indexing

Performance and Comparison
Index Size
Index Performance

Conclusion and Future Work
Outline

Introduction

Need for Spatio-Temporal Data type and Predicates
PostGIS

Spatio-temporal Data Type and Predicates
Modeling of Spatio-Temporal Data type
Basic Spatio-Temporal Predicates

Indexing Techniques
Spatio-temporal Indexing

Implementation
Datatype: mpoint
Spatio-Temporal Functions
Spatio-Temporal Indexing

Performance and Comparison
Index Size
Index Performance

Conclusion and Future Work
Introduction

- Moving object application has gained a lot of interest.
- Vehicle Tracking is one of the most important application.
- Moving object requires some frequent updates in databases regarding their position.
- Existing databases are not well equipped to handle moving object data.
- Need of moving object databases that can capture both spatial and temporal movement of an object.
Spatio-Temporal Data type and Predicates

Modeling of Spatio-Temporal Data types

- How to represent moving objects.
- Need to define the spatio-temporal data types.

Quering

- Spatio-Temporal queries are difficult to write, with present SQL
- All moving objects that are passing through a moving region.
  "Determine the flights that have passed through a hurricane."
- All moving objects that have entered into a region.
  "Find the vehicles that entered into IIT campus."

Need to define some spatio-temporal predicates
Spatio-Temporal Data type and Predicates

Modeling of Spatio-Temporal Data types

- How to represent moving objects.
- Need to define the spatio-temporal data types.

Quering

- Spatio-Temporal queries are difficult to write, with present SQL
  - All moving objects that are passing through a moving region.
    “Determine the flights that has pass through a hurricane.”
  - All moving objects that has entering into a region.
    “Find the vehicles that entered into IIT campus.”
- Need to define some spatio-temporal predicates[3]
Brief Review of PostGIS[8]

- It is a Spatial extension of PostgreSQL.
- Various Spatial functions are available.
  - `st_distance` (geometry, geometry) - returns the distance between two geometries
  - `st_intersect` (geometry, geometry) - returns boolean whether two geometries are intersecting each other.
- Spatial indexing techniques like GIST available.
Brief Review of PostGIS[8]

- It is a Spatial extension of PostgreSQL.
- Various Spatial functions are available.
  - `st_distance (geometry, geometry)` - returns the distance between two geometries
  - `st_intersect (geometry, geometry)` - returns boolean whether two geometries are intersecting each other.
- Spatial indexing techniques like GIST available.

Features not supported

- Doesn't have spatio-temporal functions.
- Doesn't have spatio-temporal indexing technique.
Outline

Introduction
Need for Spatio-Temporal Data type and Predicates
PostGIS

Spatio-temporal Data Type and Predicates
Modeling of Spatio-Temporal Data type
Basic Spatio-Temporal Predicates

Indexing Techniques
Spatio-temporal Indexing

Implementation
Datatype: mpoint
Spatio-Temporal Functions
Spatio-Temporal Indexing

Performance and Comparison
Index Size
Index Performance

Conclusion and Future Work
Modeling of Data Type[2]

- For defining Spatio-temporal data type
  - Consider two basic spatial data type *point* and *region*
  - Also consider the time dimension
  - Need to capture both space and time.
  - Taking space and time together for a point can be viewed as a moving point i.e. *mpoint*
  - Similarly, for moving region, it is *mregion*

![Figure 1: Moving point](image.png)
Modeling of Data Type and operations

- Basic data type

  \[ mpoint = time \rightarrow point \]
  \[ mregion = time \rightarrow region \]

- For Generic data type,

  \[ \tau(\alpha) = time \rightarrow \alpha \]

- Some generic operation

  \[ \tau(\alpha) \times time \rightarrow \alpha \quad \text{at} \]
  \[ \tau(\alpha) \rightarrow \text{real duration} \]

- Some other operations

  \[ mpoint \rightarrow \text{line trajectory} \]
  \[ line \rightarrow \text{real length} \]
Topological Predicates for Spatial Objects

- In modeling of data type, emphasis is given on point and region.
- Dividing a region into 3 parts, i.e. boundary $\partial A$, its interior $A^\emptyset$ and its exterior $A^-$.  

**Topological Relations**

- Between two points.
- Between point and region.
- Between two regions.

- This gives overall defining of eight predicates.
  - *equal, disjoint, coveredBy, covers, overlap, meet, inside and contains*
Temporally Lifting of Topological Predicate

- Predicates defined in previous section are required to be lifted by time.
- Spatial predicate version

\[
\text{inside} : \text{point} \times \text{region} \rightarrow \text{bool}
\]  

(1)

- Lifted version of this predicate

\[
\text{Inside} : \text{mpoint} \times \text{mregion} \rightarrow \text{mbool}
\]  

(2)

- True when the moving point is in moving region and is undefined when one of them is undefined \(\bot\). Otherwise it is false.
Basic Spatio-Temporal Predicates

- Some basic Spatio-Temporal Predicates [3] that are derived from lifting the spatial topology predicates.

\[
\begin{align*}
\text{Disjoint} & := \forall \cap . \text{disjoint} \\
\text{Meet} & := \forall \cup . \text{meet} \\
\text{Overlap} & := \forall \cup . \text{overlap} \\
\text{Equal} & := \forall \cup . \text{equal} \\
\text{Covers} & := \forall_{\pi_2} . \text{covers} \\
\text{Contains} & := \forall_{\pi_2} . \text{contains} \\
\text{CoveredBy} & := \forall_{\pi_1} . \text{coveredBy}
\end{align*}
\]

- Here, Disjoint means two spatio-temporal objects never intersect or touch each other in the common lifetime of both the objects.
Development of Spatio-temporal Predicate

- Combining existing spatial predicates, capturing spatial changes over time.

- For example: P \textit{PassThrough} R
  Series of combination of \textit{Disjoint} at \( t_1 \), then P \textit{Inside} R at \( t_2 \) and then again \textit{Disjoint} at \( t_3 \), provided \( t_1 < t_2 < t_3 \)

- An operator \( \triangleright \) (infix operator)\cite{11} can be used between two predicates given by a mapping function \( C \). Defined as:

\[
\begin{align*}
C(P) & = P \\
C(P \triangleright p) & = P \text{ until } p \\
C(p \triangleright P) & = p \text{ then } P \\
C(P \triangleright p \triangleright \Pi) & = P \text{ until } p \text{ then } C(\Pi) \\
C(p \triangleright P \triangleright \Pi) & = p \text{ then } C(P \triangleright \Pi)
\end{align*}
\]

Some Spatio-Temporal Predicates can be defined by the sequence of these predicates:

- **Enter** = \( \text{Disjoint} \triangleright \text{meet} \triangleright \text{Overlap} \triangleright \text{coveredBy} \triangleright \text{Inside} \)
- **Release** = \( \text{Enter}^{-1} \)
- **PassThrough** = \( \text{Enter} \triangleright \text{Leave} \)
- **Bypass** = \( \text{Disjoint} \triangleright \text{Meet} \triangleright \text{Disjoint} \)
- **Remeets** = \( \text{Meet} \triangleright \text{Disjoint} \triangleright \text{Meet} \)
- **Separate** = \( \text{Equal} \triangleright \text{Overlap} \triangleright \text{meet} \triangleright \text{Disjoint} \)

From the above predicates, **Enter**, **Release**, **PassThrough** are implemented.
Spatio-temporal queries with PassThrough

Schema
Flights(flightid:string, Route: mpoint)
Hurricane(kind:string, Extent: mregion)

Query
Determine the flights that has pass through the hurricane?

SQL Syntax
Select ST_passthrough(Extent, Route, flightid)
from Flight, Hurricane
where mpoint_ll_coord(Route, 3)=mpoint_ll_coord(Extent, 3)
order by mpoint_ll_coord(Route, 3);
Spatio-temporal queries with Enter

**Schema**
vehicle_st1(vehregno:string, stpoint:mpoint)

**Query**

**SQL Syntax**
```sql
FROM vehicle_st1 
WHERE mpoint_overlap('((0, 0, 2007/06/09 01:11:23), (1000, 1000, 2007/07/08 11:11:11))', stpoint) 
ORDER BY mpoint_ll_coord(stpoint, 3);
```
Outline

Introduction
   Need for Spatio-Temporal Data type and Predicates
   PostGIS

Spatio-temporal Data Type and Predicates
   Modeling of Spatio-Temporal Data type
   Basic Spatio-Temporal Predicates

Indexing Techniques
   Spatio-temporal Indexing

Implementation
   Datatype: mpoint
   Spatio-Temporal Functions
   Spatio-Temporal Indexing

Performance and Comparison
   Index Size
   Index Performance

Conclusion and Future Work
Need for Spatio-temporal Indexing

- Currently, PostgreSQL provides spatial and temporal indexing separately.
- Both spatial and temporal index needs to be scanned for spatio-temporal queries.
- Need to join the result from both indexes.
Need for Spatio-temporal Indexing

- Currently, PostgreSQL provides spatial and temporal indexing separately.
- Both spatial and temporal index needs to be scanned for spatio-temporal queries.
- Need to join the result from both indexes.

Disadvantage of Spatial and Temporal Indexing

- Useless when most of the elements are not found in intersection set.
- Two indexes needs to be scanned rather than one.
Related Work

3DRtree[1]

- Extension of R-tree with time as a third dimension.
- Works in same way as 2D Rtree.
- Efficient for time interval queries.
Related Work

3DRtree[1]

▶ Extension of R-tree with time as a third dimension.
▶ Works in same way as 2D Rtree.
▶ Efficient for time interval queries.

STR-tree[6]

▶ Extends R-Tree supporting efficient query processing of trajectories of moving points.
▶ Organizes line segment according to spatial proximity considering trajectory preservation.
▶ Efficient for trajectory queries.
Indexing in PostgreSQL

Operator Class[10]

- Specifies Operator and index method.
- Operator needs to be defined on data type.
- Index procedure must be defined for creating Operator Class.
Indexing in PostgreSQL

Operator Class[10]

- Specifies Operator and index method.
- Operator needs to be defined on data type.
- Index procedure must be defined for creating Operator Class.

GIST[4]

- GIST is used to implement various tree structure.
- GIST methods have to defined to implement indexing technique.

  - Consistent, Union and Same - Correctness of index.
  - Penalty and PickSplit - Efficiency (speed and size).
  - Compress and Decompress - Internal tree data of different type.
Outline

Introduction
Need for Spatio-Temporal Data type and Predicates
PostGIS

Spatio-temporal Data Type and Predicates
Modeling of Spatio-Temporal Data type
Basic Spatio-Temporal Predicates

Indexing Techniques
Spatio-temporal Indexing

Implementation
Datatype: mpoint
Spatio-Temporal Functions
Spatio-Temporal Indexing

Performance and Comparison
Index Size
Index Performance

Conclusion and Future Work
Implementation of mpoint

- Need to define input and output method.
- *mpoint* internally stored in the form of cube

**Type declaration**

```sql
CREATE OR REPLACE FUNCTION mpoint_in(cstring) RETURNS mpoint AS
'${libdir}/mpoint', 'mpoint_in' LANGUAGE C IMMUTABLE STRICT;

CREATE OR REPLACE FUNCTION mpoint_out(mpoint) RETURNS cstring AS
'${libdir}/mpoint', 'mpoint_out' LANGUAGE C IMMUTABLE STRICT;

CREATE TYPE mpoint (internallength = variable,
input = mpoint_in,
output = mpoint_out);
```

**Two ways of input**

Spatio-Temporal Functions: ST_Enters

\[ \text{Enters} = \text{Disjoint} \triangleright \text{Contains} \]

**Algorithm 1** Pseudo code of ST_Enters (mregion, mpoint, vid)

1. if mpoint_overlaps(mregion, mpoint) = false then
2.   clear the data from temp table of vehicleid = vid;
3.   insert new entry of \( \text{vid} \) with flag = ‘disjoint’ in temp table
4. else
5.   retrieve the data from temp data of \( \text{vid} \)
6.   if mpoint_contains(mregion, mpoint) = \text{true} and flag is \text{disjoint} then
7.     clear entry from temp table for that \( \text{vid} \)
8.     return \( \text{vid} \) along with position anytime
9. end if
10. end if
Example for ST_Entered

**Schema**

vehicle_st (regno text, stpoint mpoint)

**Query**


**SQL Syntax**

```
where mpoint_overlap('((0,0,2007/06/09 01:11:23),
(1000,1000,2007/07/08 11:11:11'),stpoint)
order by mpoint_ll_coord(stpoint,3);
```

**Output**

```
MH01-0111, "(116.414467,39.947250,2007-06-10 09:46:24+05:30)"
MH01-0114, "(116.331133,39.975733,2007-06-22 03:51:01+05:30)"
MH01-0114, "(116.331433,39.997600,2007-06-25 01:50:19+05:30)"
..........
```
Steps for defining New Indexing Structure in PostgreSQL

- Select datatype on which index has to be created.
- Define operators on this datatype to perform some operation.
- Define GIST methods for correct, efficient and proper implementation of index.
- Declare all the operators and methods in Operator class.
Data type on which Index is created

- Data type for ST indexing is mpoint.
- mpoint internally stores in form of cube[9].
- Base element is cube having three dimension \((x, y, t)\).

Figure 2: Cube[1]
Operators on mpoint

Need to define operators on \textit{mpoint}

\begin{verbatim}
CREATE OPERATOR && (  
    LEFTARG = mpoint, RIGHTARG = mpoint, PROCEDURE = mpoint_overlap,  
    COMMUTATOR = '&&',  
    RESTRICT = areasel, JOIN = areajoinsel
);
\end{verbatim}

\begin{verbatim}
CREATE OR REPLACE FUNCTION mpoint_overlap(mpoint, mpoint) RETURNS bool AS  
    '$libdir/mpoint', 'mpoint_overlap' LANGUAGE C IMMUTABLE STRICT;
\end{verbatim}

Similarly, other operators like $=, @ \geq, < @$ are defined.
GIST Implementation

- Seven methods need to be defined on *mpoint*.
- For efficiency of an index, Penalty and PickSplit are used[5].

Penalty

- Cost of insertion of an index entry in tree.
- \[ \text{Penalty} = \text{Size of New Cube after adding new entry} - \text{Size of Original Cube} \]
- Least Penalty = Minimum difference of Size of cubes.
Cube’s PickSplit Method

Used to Split a node of index tree.

Algorithm 2 Pseudo code of Cube’s PickSplit[7]

1: Put all entries in set $E$.
2: Choose two data items such that dead space created by this two items is maximize. Dead Space is defined as difference of MBB of $e_1$ and $e_2$ with MBB of $e_1$ and with MBB of $e_2$.
3: for each entry $e'$ in remaining elements,
   
   **Least enlargement** $(mbb(e_1, e') - mbb(e_1), mbb(e_2, e') - mbb(e_2))$ whichever require least enlargement, add elements in that group.
4: Put $G_1$ elements on one node and $G_2$ elements on another.
Cube’s PickSplit Method
Cube’s PickSplit Method
Modified Cube’s PickSplit

Algorithm 3 Pseudo code of Modified Cube’s PickSplit

1: ....
2: ..... 
3: ...... 
4: for each entry $e'$ in remaining elements do 
5: if size of both group is less than '$m$' then 
6: Least enlargement ($mbb(e_1, e') - mbb(e_1), mbb(e_2, e') - mbb(e_2)$) 
7: whichever require least enlargement, add elements in that group. 
8: else 
9: put remaining elements in other group. 
10: end if 
11: end for 
12: Put $G_1$ elements on one node and $G_2$ elements on another.
Operators Class[10]

```sql
CREATE OPERATOR CLASS gist_mpoint_ops
DEFAULT FOR TYPE mpoint USING gist AS
  OPERATOR 3 &&,
  OPERATOR 6 =,
  OPERATOR 7 @>,
  OPERATOR 8 <@ ,

FUNCTION 1 g_mpoint_consistent (internal, mpoint, int, oid, internal),
FUNCTION 2 g_mpoint_union (internal, internal),
FUNCTION 3 g_mpoint_compress (internal),
FUNCTION 4 g_mpoint_decompress (internal),
FUNCTION 5 g_mpoint_penalty (internal, internal, internal),
FUNCTION 6 g_mpoint_picksplit (internal, internal),
FUNCTION 7 g_mpoint_same (mpoint, mpoint, internal);
```
Operators Class[10]

CREATE OPERATOR CLASS gist_mpoint_ops
DEFAULT FOR TYPE mpoint USING gist AS
  OPERATOR 3 &&,
  OPERATOR 6 =,
  OPERATOR 7 @> ,
  OPERATOR 8 <@ ,

FUNCTION 1 g_mpoint_consistent (internal, mpoint, int, oid, internal),
FUNCTION 2 g_mpoint_union (internal, internal),
FUNCTION 3 g_mpoint_compress (internal),
FUNCTION 4 g_mpoint_decompress (internal),
FUNCTION 5 g_mpoint_penalty (internal, internal, internal),
FUNCTION 6 g_mpoint_picksplit (internal, internal),
FUNCTION 7 g_mpoint_same (mpoint, mpoint, internal);

Creation of Index

Create index stpoint on tablename using gist(columnname)
where, columnname is of type mpoint
Outline

Introduction
  Need for Spatio-Temporal Data type and Predicates
  PostGIS

Spatio-temporal Data Type and Predicates
  Modeling of Spatio-Temporal Data type
  Basic Spatio-Temporal Predicates

Indexing Techniques
  Spatio-temporal Indexing

Implementation
  Datatype: mpoint
  Spatio-Temporal Functions
  Spatio-Temporal Indexing

Performance and Comparison
  Index Size
  Index Performance

Conclusion and Future Work
Details of GIST tree with PostgreSQL Bug

gist_tree

0(1:0) blk: 0 numTuple: 16 free: 7060b(13.48%) rightlink:4294967295 (InvalidBlockNumber)
1(1:1) blk: 123510 numTuple: 1 free: 8080b(0.98%) rightlink:123511 (OK)
1(1:2) blk: 120752 numTuple: 1 free: 8080b(0.98%) rightlink:120753 (OK)
1(1:3) blk: 118100 numTuple: 1 free: 8080b(0.98%) rightlink:118101 (OK)
1(1:4) blk: 115554 numTuple: 1 free: 8080b(0.98%) rightlink:115555 (OK)
1(1:5) blk: 113114 numTuple: 1 free: 8080b(0.98%) rightlink:113115 (OK)
1(1:6) blk: 110780 numTuple: 1 free: 8080b(0.98%) rightlink:110781 (OK)
1(1:7) blk: 108552 numTuple: 1 free: 8080b(0.98%) rightlink:108553 (OK)
1(1:8) blk: 106430 numTuple: 1 free: 8080b(0.98%) rightlink:106431 (OK)
1(1:9) blk: 104414 numTuple: 1 free: 8080b(0.98%) rightlink:104415 (OK)
1(1:10) blk: 102504 numTuple: 1 free: 8080b(0.98%) rightlink:102505 (OK)
# Details of GIST tree with PostgreSQL Bug Fixed

<table>
<thead>
<tr>
<th>level</th>
<th>blk</th>
<th>numTuple</th>
<th>free</th>
<th>rightlink</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(1:0)</td>
<td>0</td>
<td>8</td>
<td>7604b(6.81%)</td>
<td>4294967295 (InvalidBlockNumber)</td>
</tr>
<tr>
<td>1(1:1)</td>
<td>7226</td>
<td>79</td>
<td>2776b(65.98%)</td>
<td>7227 (OK)</td>
</tr>
<tr>
<td>1(1:2)</td>
<td>109</td>
<td>54</td>
<td>4476b(45.15%)</td>
<td>166 (OK)</td>
</tr>
<tr>
<td>1(1:3)</td>
<td>1</td>
<td>66</td>
<td>3660b(55.15%)</td>
<td>2 (OK)</td>
</tr>
<tr>
<td>2(1:3)</td>
<td>118</td>
<td>92</td>
<td>1892b(76.81%)</td>
<td>117 (OK)</td>
</tr>
<tr>
<td>3(1:3)</td>
<td>120</td>
<td>55</td>
<td>4408b(45.98%)</td>
<td>116 (OK)</td>
</tr>
<tr>
<td>4(1:3)</td>
<td>119</td>
<td>56</td>
<td>4340b(46.81%)</td>
<td>121 (OK)</td>
</tr>
<tr>
<td>5(1:3)</td>
<td>121</td>
<td>55</td>
<td>4408b(45.98%)</td>
<td>122 (OK)</td>
</tr>
<tr>
<td>6(1:3)</td>
<td>122</td>
<td>62</td>
<td>3932b(51.81%)</td>
<td>120 (OK)</td>
</tr>
<tr>
<td>7(1:3)</td>
<td>116</td>
<td>69</td>
<td>3456b(57.65%)</td>
<td>123 (OK)</td>
</tr>
<tr>
<td>8(1:3)</td>
<td>105</td>
<td>104</td>
<td>1076b(86.81%)</td>
<td>103 (OK)</td>
</tr>
</tbody>
</table>
Index Parameters Comparison

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Before Bug</th>
<th>After Bug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of levels</td>
<td>28</td>
<td>4</td>
</tr>
<tr>
<td>Number of pages</td>
<td>858 k</td>
<td>35.6 k</td>
</tr>
<tr>
<td>Number of leaf pages</td>
<td>393 k</td>
<td>35 k</td>
</tr>
<tr>
<td>Number of tuples</td>
<td>2,954 k</td>
<td>2,132 k</td>
</tr>
<tr>
<td>Number of leaf tuples</td>
<td>2,096 k</td>
<td>2,096 k</td>
</tr>
<tr>
<td>Total size of tuples</td>
<td>201 MB</td>
<td>139 MB</td>
</tr>
<tr>
<td>Total size of leaf tuples</td>
<td>140 MB</td>
<td>136 MB</td>
</tr>
<tr>
<td>Total size of index</td>
<td>6702 MB</td>
<td>278 MB</td>
</tr>
</tbody>
</table>
Index Size Comparison

![Index Size Comparison Graph](image)

**Figure 3**: Comparison
Sequential vs Index Scan

Schema
Vehicle_st1 (regno: text, stpoint : mpoint)

Query
Find the vehicles that are contains in this region ((116.3, 39.3),(117.2, 39.8)) between ‘2007/06/11 11:11:11’ and ‘2007/09/13 11:11:11’?

SQL Syntax
Select * from vehicle_st1
Index Performance

Sequential vs Index Scan

Query Plan for Sequential Scan

Seq Scan on vehicle_st1 (cost=0.00..50870.86 rows=2096 width=66)
(actual time=550.683.. 4189.592 rows=9069 loops=1)
   (117.200000,39.800000,2007-09-13 11:11:11+05:30) '::mpoint @> stpoint)
Total runtime: 4193.612 ms

Query Plan for Index Scan

Index Scan using st1 on vehicle_st1 (cost=0.00..8226.36 rows=2096 width=66)
(actual time=151.295..1197.213 rows=9069 loops=1)
   (117.200000,39.800000,2007-09-13 11:11:11+05:30) '::mpoint @> stpoint)
Total runtime: 1203.435 ms
Spatio-Temporal Index

Schema
Vehicle_st1 (regno: text, stpoint : mpoint)

Query

SQL syntax
Select * from vehicle_st1

Query Plan for ST Scan

Index Scan using st1 on vehicle_st1 (cost=0.00..35653.11 rows=10482 width=66)
(actual time=377.857..400.107 rows=125 loops=1)
(117.200000,39.800000,2008-01-13 11:11:11+05:30)’::mpoint && stpoint)
  Total runtime: 400.553 ms
Spatial and Temporal Index

SQL syntax

Select * from vehicle_geomts
where, spatial index is on geom and temporal index is on movts attribute.

QUERY PLAN

Index Scan using tsindex on vehicle_geomts(cost=0.00..72936.81 rows=2866 width=134) (actual time=811.686..1108.057 rows=125 loops=1)
  Filter:((’010300...........66A64340’::geometry & geom) AND _st_intersects(’010300002.............66A64340’::geometry, geom))
Total runtime: 1108.216 ms
Comparison done on basis of these 4 queries

Query 2

Query 3

Query 4
Find all vehicle that intersects with \(((116.3,39.3),(117,39.9))\) between ‘2007/12/11 11:11:11’ and ‘2008/05/13 11:11:11’?
Index Performance Comparison

Figure 4: Comparison between ST and spatial-temporal indexing
Outline

Introduction
Need for Spatio-Temporal Data type and Predicates
PostGIS

Spatio-temporal Data Type and Predicates
Modeling of Spatio-Temporal Data type
Basic Spatio-Temporal Predicates

Indexing Techniques
Spatio-temporal Indexing

Implementation
Datatype: mpoint
Spatio-Temporal Functions
Spatio-Temporal Indexing

Performance and Comparison
Index Size
Index Performance

Conclusion and Future Work

Nirmesh Neema
Guided By: Prof. N.L. Sarda
IIT Bombay
Querying and Indexing of moving objects
Conclusion and Future Work

- **Conclusion**
  - *mpoint* data type is defined for spatio-temporal data which gives information (position, time) about moving point.
  - Spatio-temporal predicates ease the writing of complex spatio-temporal queries as seen by *ST_Enters*, etc.
  - Spatio-temporal index is created by defining GIST methods in PostgreSQL.
  - Fixed a bug in PostgreSQL for index creation, which improves space utilization.

- **Future Work**
  - *mpoint* can be made as a geometry to allow for geometric operations.
Spatio-temporal indexing for large multimedia applications.

Abstract and discrete modeling of spatio-temporal data types.
Martin Erwig and Markus Schneider.
Developments in spatio-temporal query languages.

GIST.
Gist reference.

Joseph Hellerstein, Jeffrey Naughton, and Avi Pfeffer.
Generalized search trees for database systems (extended abstract).
References III


PostGIS. Postgis reference. 
http://postgis.refractions.net/documentation/manual-1.3/ch06.html.
References IV

Postgres.
Cube reference.

PostgreSQL.
Operator reference.

Markus Schneider Ralf Hartmut Gting.
Moving objects databases.
THANK YOU...