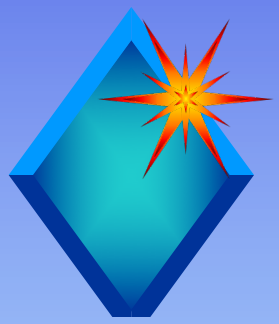
Optimization techniques for Improving Geospatial Semantic Web Query

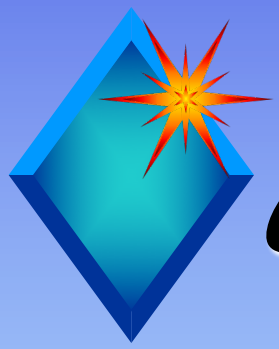
Chuanrong Zhang, Tian Zhao and Weidong Li

Department of Geography & Center of Environmental Sciences and Engineering, University of Connecticut, Storrs, USA
Department of Computer Science, University of Wisconsin, Milwaukee, USA



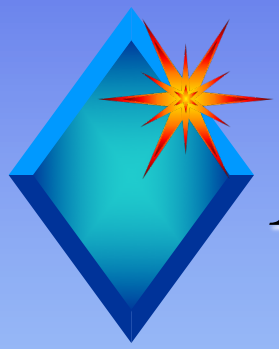
Geospatial Semantic Web

- Geospatial databases created from a variety of sources have syntactic, structural, and semantic heterogeneity problems.
- Geospatial web services have been developed to provide access to heterogeneous geographic information on the Internet.



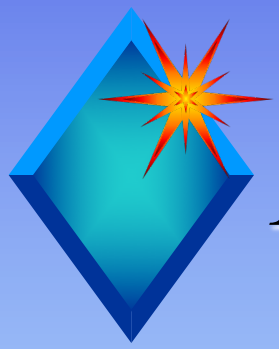
Geospatial Semantic Web

- By associating spatial data content from the Web with ontologies (that would supply context and meaning), the vision of Geospatial Semantic Web is to extract geospatial knowledge from the Web regardless of geospatial data formats or sources; thus it can facilitate transparent geospatial data exchange, sharing, and query.



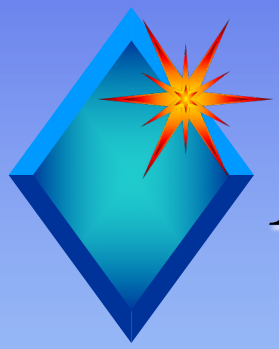
Introduction

- Geospatial Semantic Web promises better retrieval of geospatial information for the Web.
- It also promotes sharing and reuse of spatial data for a wide variety of applications by using standardized Semantic Web languages such as RDF.
- The USGS (United States Geological Survey) have been working on developing ontologies for The National Map for many years using RDF language.



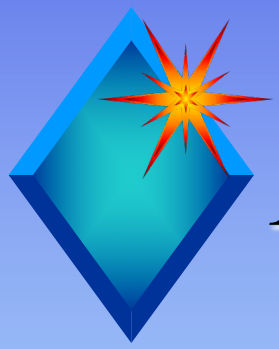
Introduction

- However, representing structured geospatial data in these languages can result in inefficient data access.
- One of the main obstacles that prevent efficient and distributed query on geospatial knowledge base is the lack of indexing on spatially related data objects.



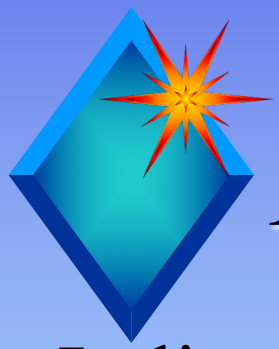
Introduction

- It is possible to recreate indices for RDF objects with spatial attributes in knowledge bases such as Parliament and Strabon.
- However, pre-computing spatial indices does not guarantee performance improvement since the RDF queries are much more flexible than database queries and it is difficult to predict which spatial objects should be indexed and how.



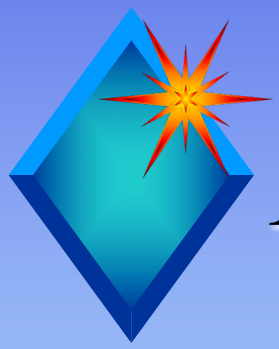
Introduction

- Geo-SPARQL queries are dominated by spatial join operations due to the fine-grained nature of RDF data model.
- Lack of spatial indices causes additional performance problems for Geo-SPARQL queries.



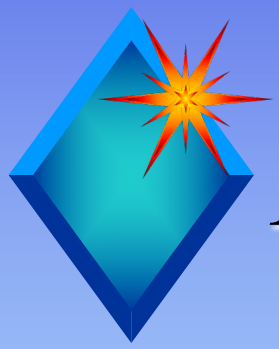
Introduction

- In literature, different approaches have been widely used for improving spatial joint performance for a long time.
- However, past research on improving query performance has been centered on offline relational databases.
- Optimizing techniques for offline spatial relational databases do not specialize on the triple model of RDF and triple patterns of GeoSPARQL queries.



Introduction

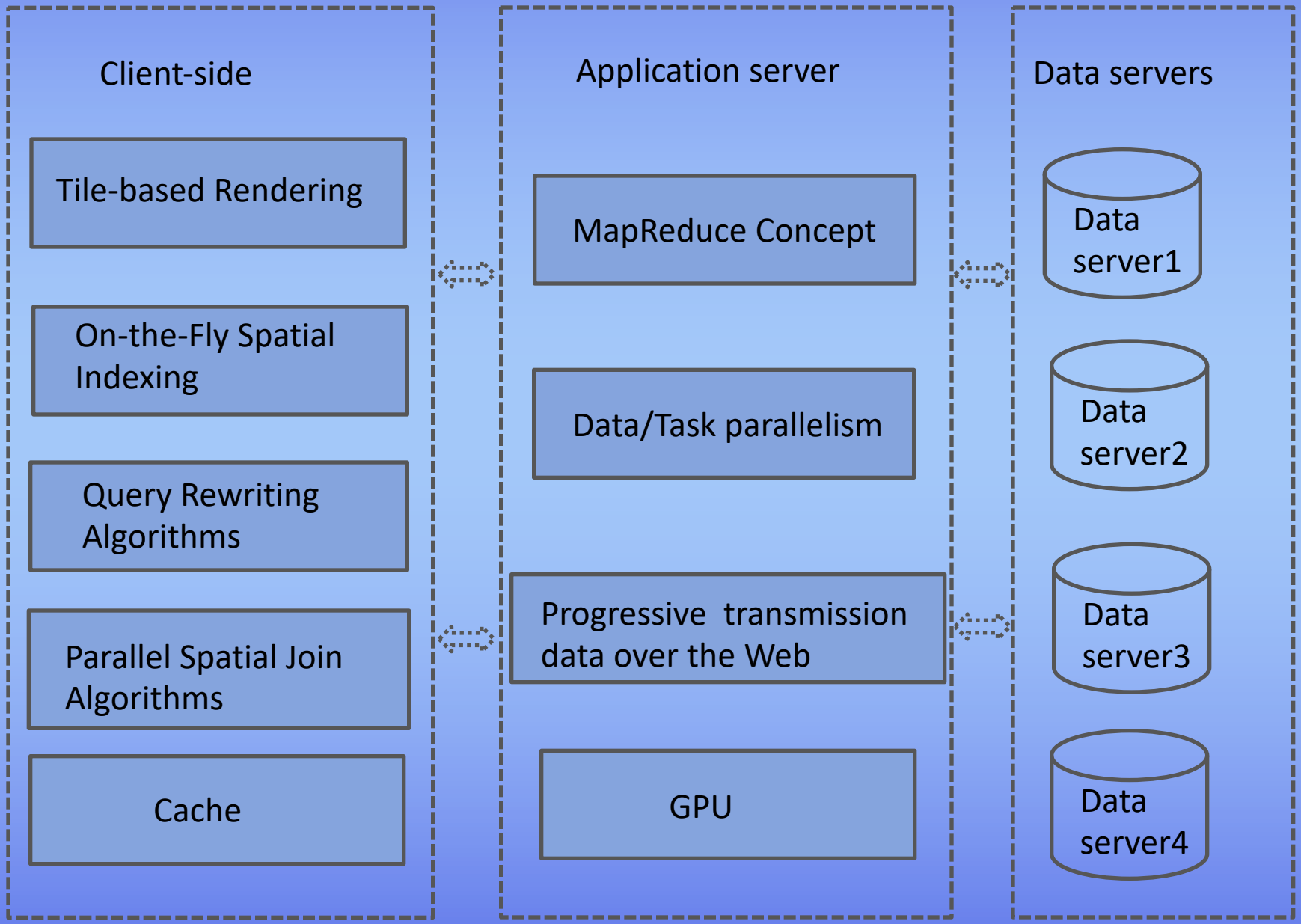
- To the best of our knowledge, there are few studies to explore support efficient spatial RDF query, which is an important issue for the development of Geospatial Semantic Web.

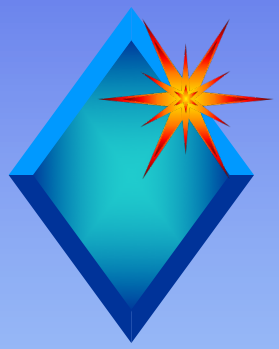


Research Objectives

- Explore approaches for efficient spatial knowledge queries of The National Map.
- The ultimate goal is to make spatial knowledge query flexible and efficient so that it can replace the current ad hoc Web interface of The National Map.

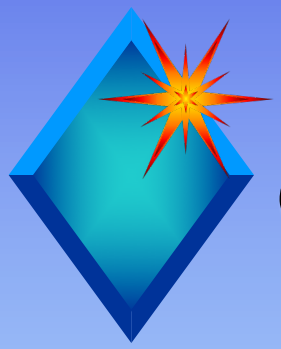
Methods


Client-side
Tile-based Rendering
On-the-Fly Spatial Indexing
Query Rewriting Algorithms
Parallel Spatial Join Algorithms
Cache
Application server
MapReduce Concept
Data/Task parallelism
Progressive transmission data over the Web
GPU
Data servers
Data server1
Data server2
Data server3
Data server4



Query rewriting algorithms

- To avoid the problems of data replication and minimize the bulk of transferred data from the data server, we do not create RDF ontology instances explicitly.
- Instead, we use a query rewriting approach to enable GeoSPARQL ontology query on spatial data available from WFS services and data stored in databases.



Query rewriting algorithms

Input: target query q

A set of inference rules I

For each triple t **in** q.body

For each inference rule I **in** I

If there exists a substitution s such that s (i.head) = t

Then replace t **in** q.body with s (i.body)

End for

End for

Output: q′ where q′.body has only triples **in** RDF mapping

(a)Rewrite GeoSPARQL queries to WFS queries algorithm part 1: apply inference rule to GeoSPARQL query

Query rewriting algorithms

Input: target query q

A set of mapping rules M

Initialize: apply algorithm 1 to q to obtain q′

Group triples **in** q′.body by subject name

Resulting a set of triple groups L

For each triple group t **in** L

For each mapping rule m **in** M

If there exists a substitution s such that

s(m.body) contains all triples **in** t

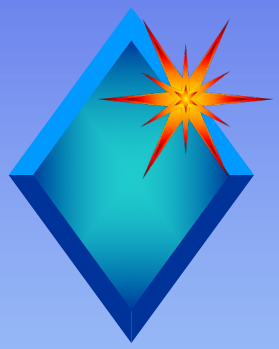
Then replace t **in** q′ with s (m.head)

End for

End for

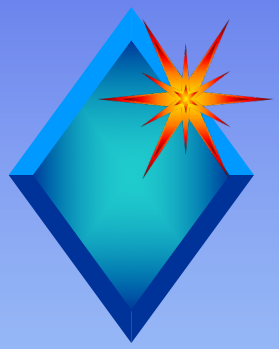
Output: q′ where q′.body contains WFS get Feature requests

(a)Rewrite GeoSPARQL queries to WFS queries algorithm part 2: query rewriting



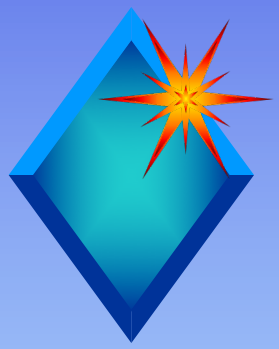
Tile-based rendering

- Two types of tile-based rendering techniques: vector tiles and raster tiles.
- Use a modified Ramer-Douglas-Peucker simplification algorithm (Hershberger and Snoeyink 1992) to recursively find the most important points in a polyline and discard unnecessary detail in each tile.



Tile-based rendering

- Geometries that are too tiny to be displayed on any particular zoom level are filtered using the pre-calculated values.
- The bounding box of each geographic feature are used to make fast clipping.
- The well-known Tile Pyramid structure based on a level-of-detail (LOD) model are adopted for efficient large map image rendering and fast tile scheduling.



Cache techniques

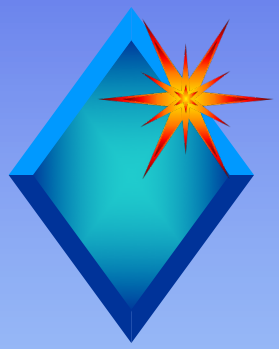
- In this study the client-side mobile devices/computers cache the tiles on their sides via the tile mechanism to reduce the user-perceived response time and improve user navigation and query efficiency.
- The commonly used tiles are cached in advance.
- While a client is viewing one of the commonly used tiles, the next possible tiles are pre-fetched by the system using a simple prefetching mechanism.

Cache techniques

- Based on the prefetching mechanism, tiles around the map view extent are requested from the application/data servers while the user requests the tiles inside the map view extent.
- Because the tiles that represent the next navigation/query of the client are already in the cache, seamless navigation or fast query can be provided to the client.

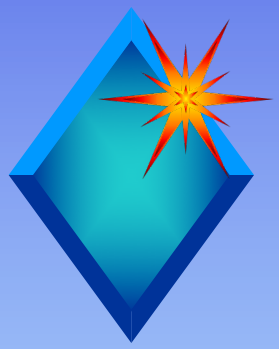
Separating spatial query from non-spatial query

- Instead of processing the triple statements of GeoSPARQL query in sequence, we separate the triple non-spatial statements from the filtering statements that involved spatial computations, which typically are very costly.
- This way a smaller set of ontology objects can be obtained after non-spatial sub-queries so that their spatial attributes can be cached for subsequent spatial computation including on-the-fly spatial indexing and spatial joins.



On-the-fly spatial indexing

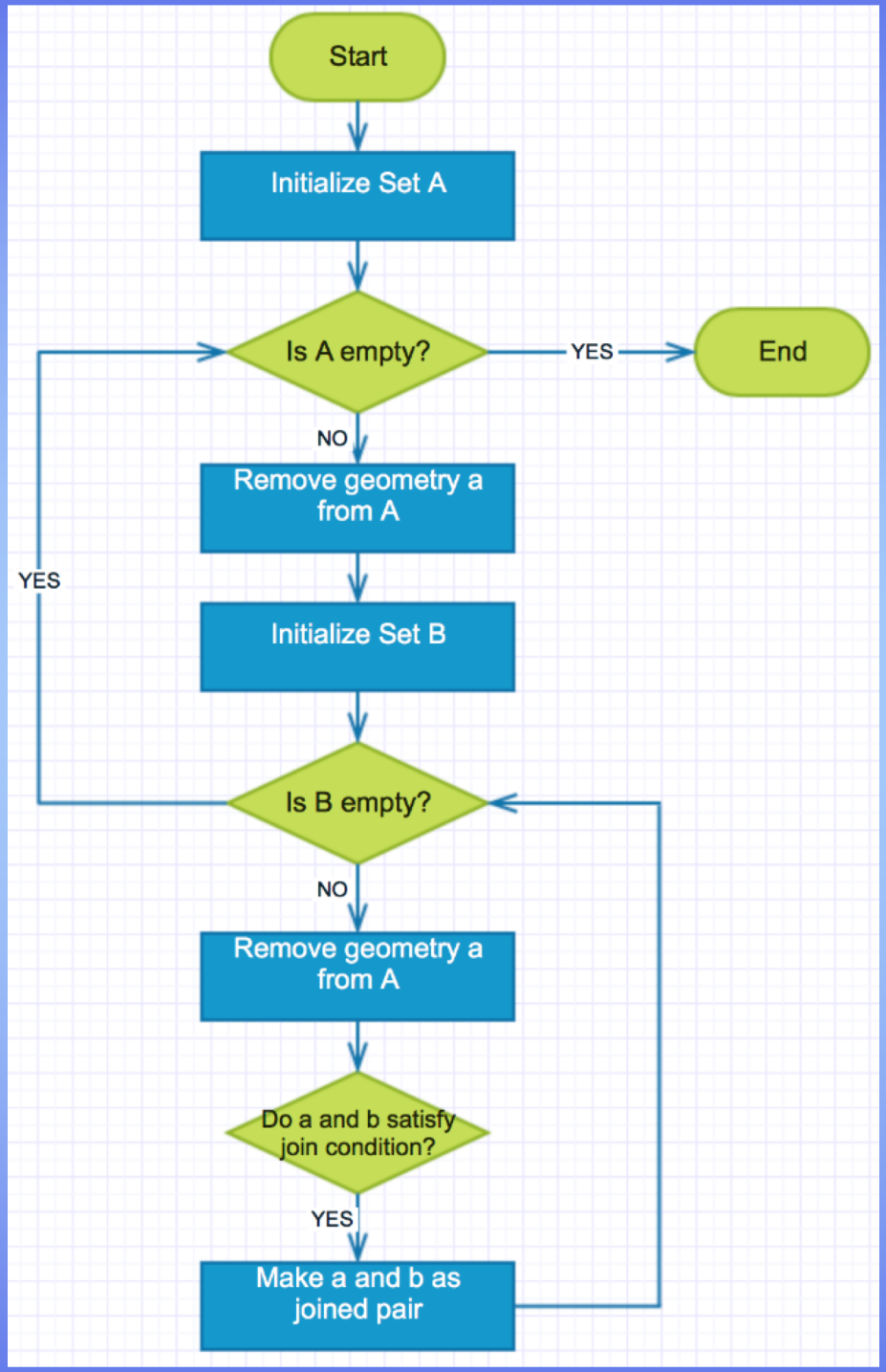
- Construct spatial indices on-the-fly for the spatial objects returned from the non-spatial component of the query.
- We employ the R-Tree technique to the on-the-fly spatial indexing for GeoSPARQL queries in the system due to simplicity of the R-Tree structure and its ability to handle spatial objects efficiently.



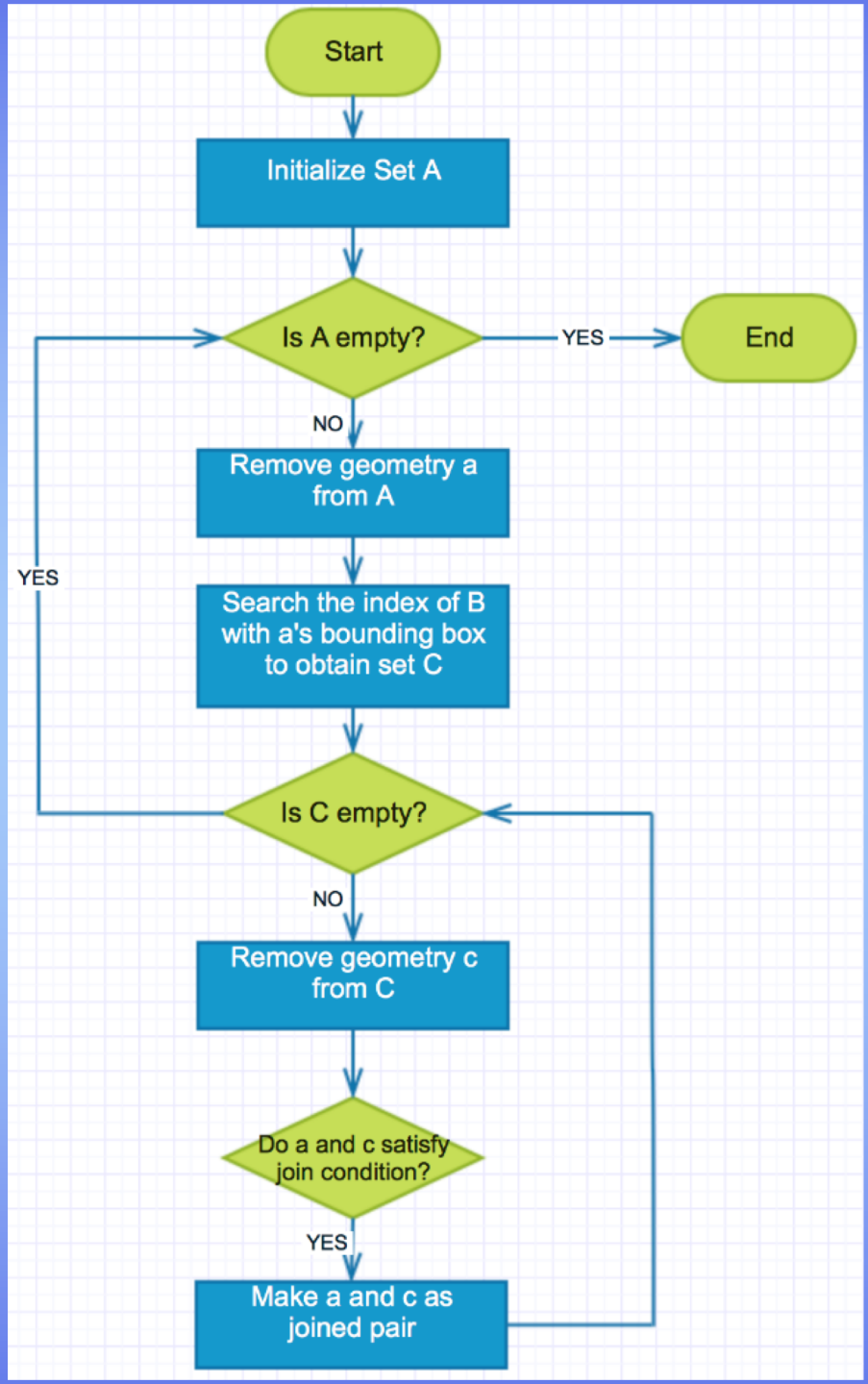
Spatial Join Algorithms

- The nested-loop join algorithm
- The index nested-loop (R-Tree) join algorithm
- The plane sweep join algorithm
- The hierarchical traversal join (R-Tree) algorithm

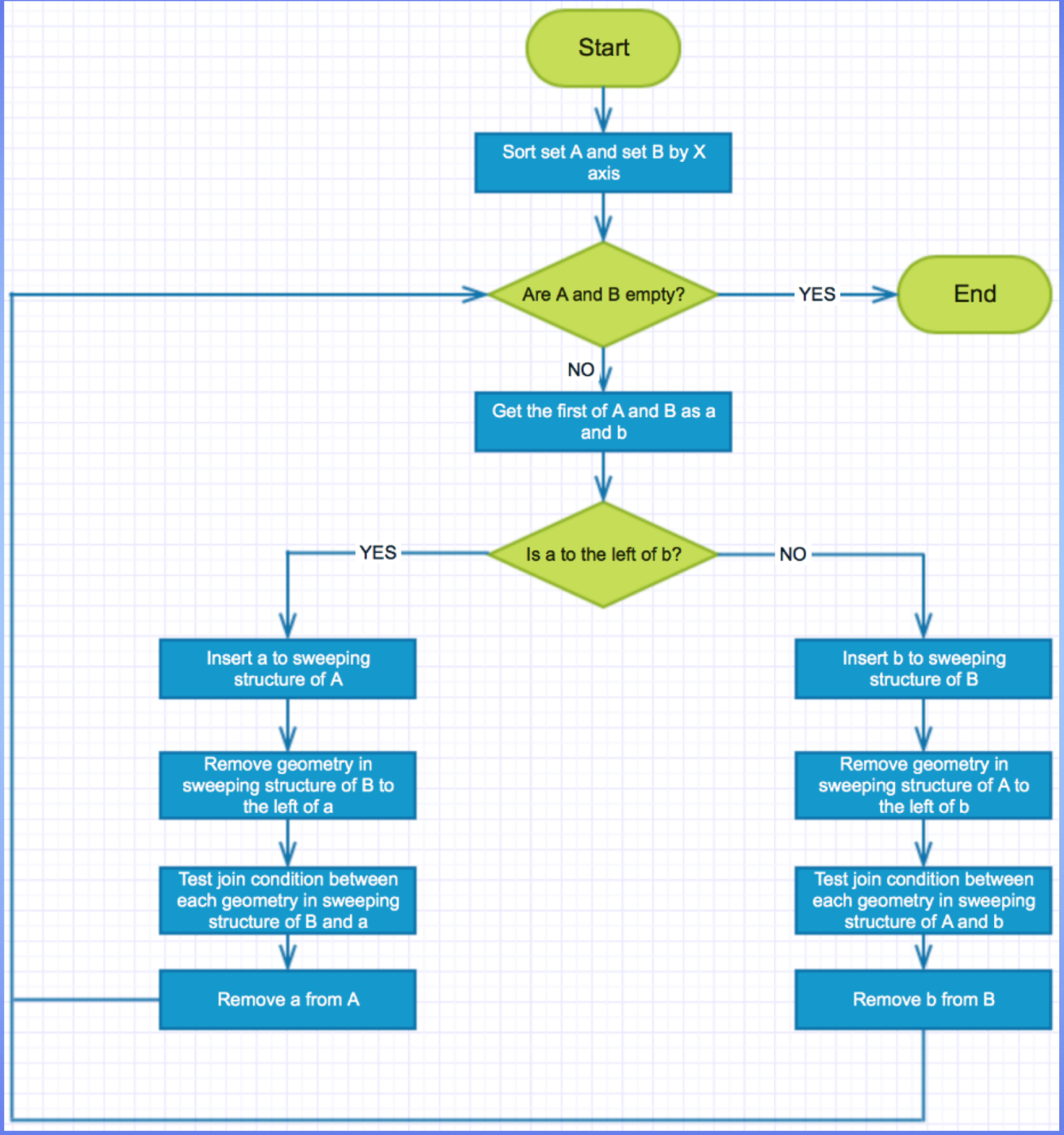
Flowchart for nested loop join algorithm



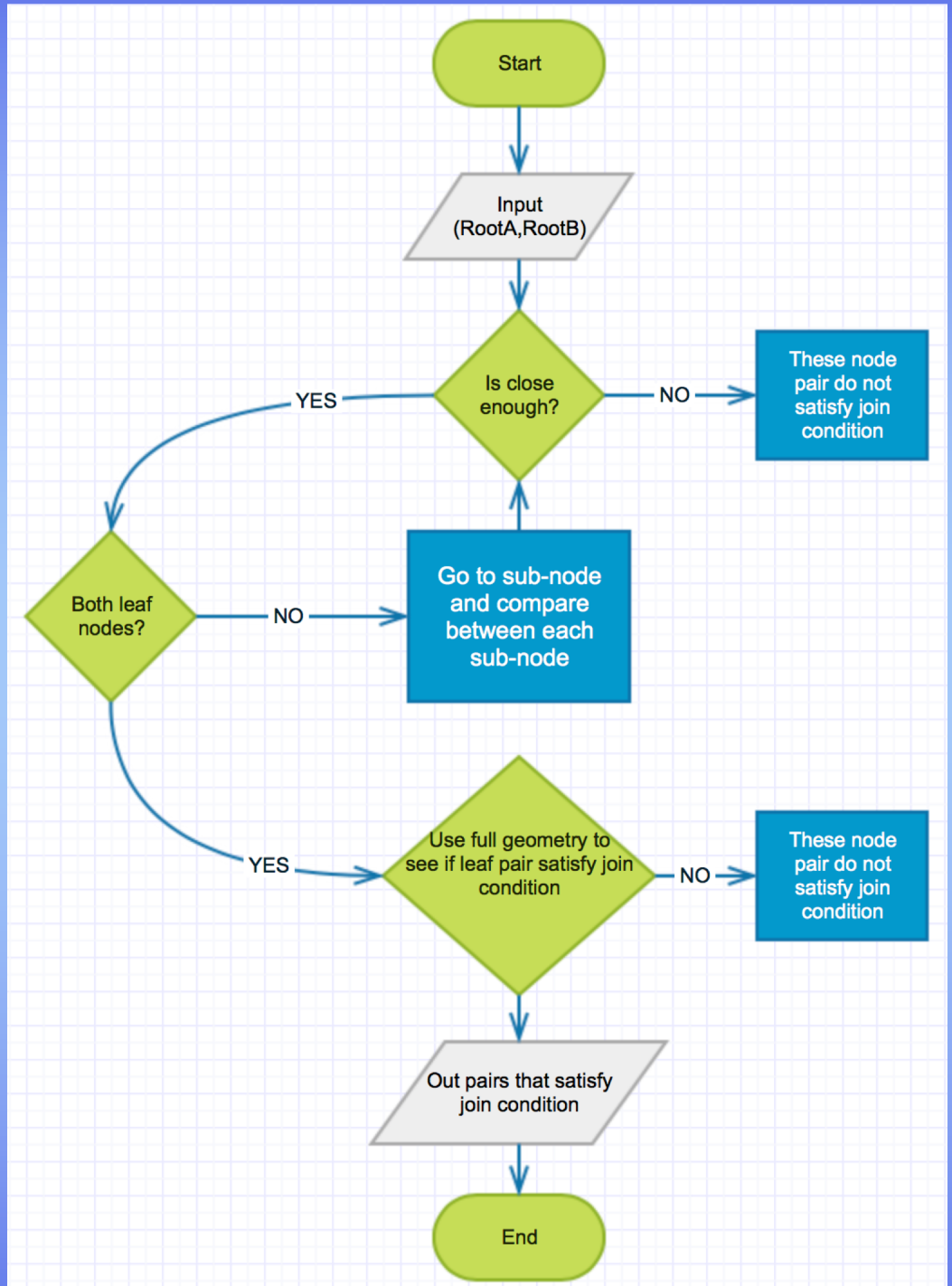
Flowchart for index nested-loop join algorithm



Flowchart for plane sweep join algorithm

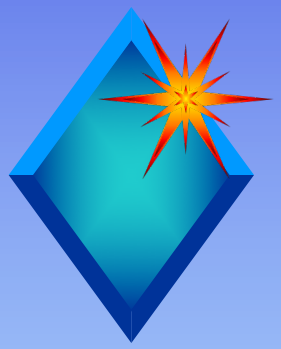


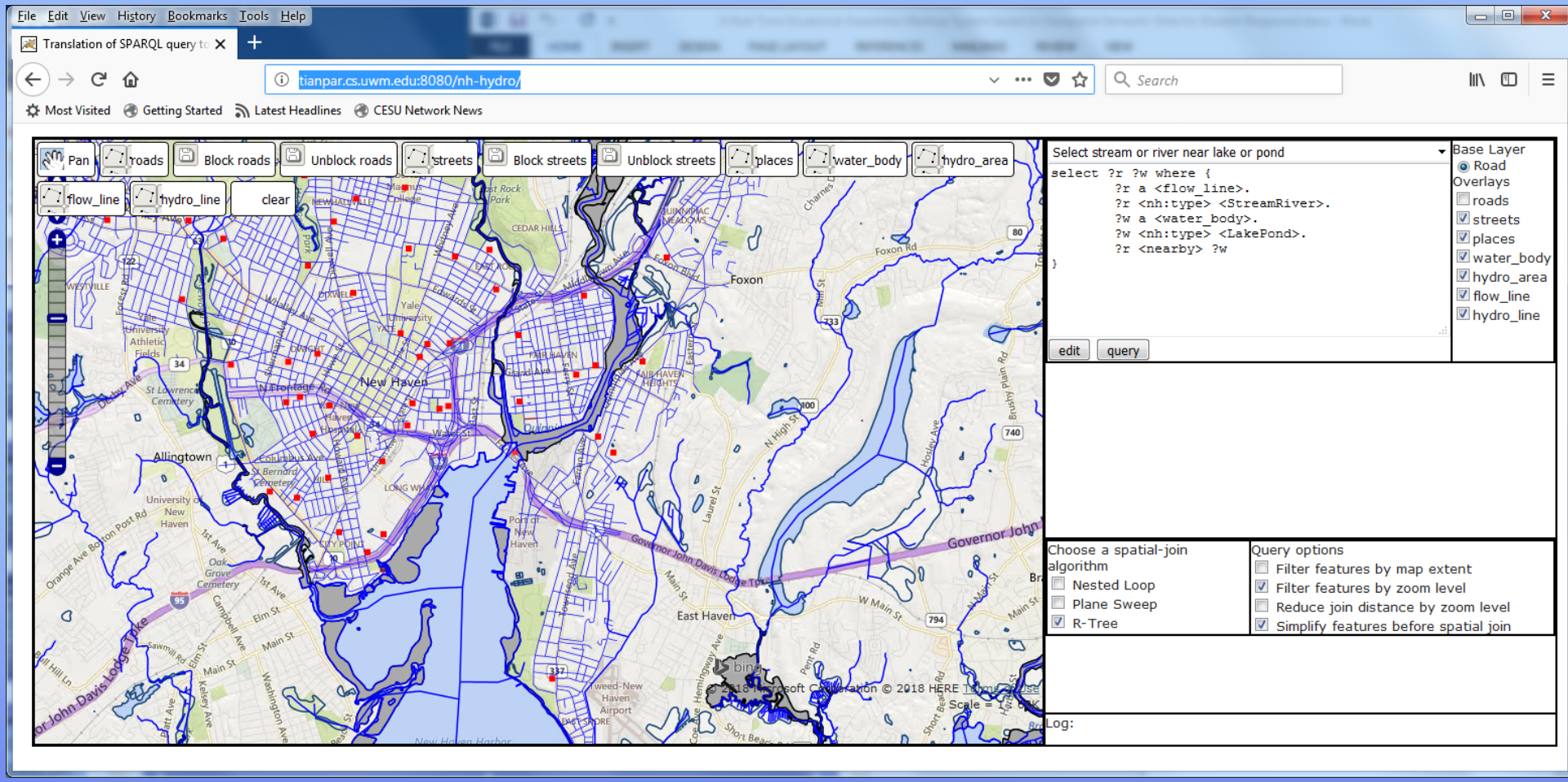
Flowchart for hierarchical traversal join algorithm

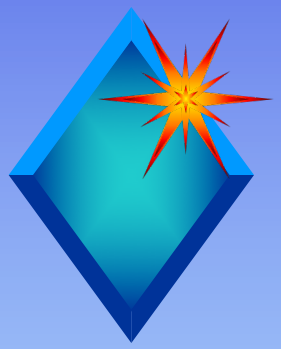


Some experimental results

- The implemented prototype can be accessible from the website:
  http://tianpar.cs.uwm.edu:8080/nh-hydro/
  http://tianpar.cs.uwm.edu:8080/usgs/




File Edit View History Bookmarks Tools Help
Translation of SPARQL query
tianpar.cs.uwm.edu:8080/nh-hydro/
Search
Most Visited
Getting Started
Latest Headlines
CESU Network News
Pan
roads
Block roads
Unblock roads
streets
Block streets
Unblock streets
places
water_body
hydro_area
flow_line
hydro_line
clear
Select stream or river near lake or pond
select ?r ?w where {
?r a <flow_line>.
?r <nh:type> <StreamRiver>.
?w a <water_body>.
?w <nh:type> <LakePond>.
?r <nearby> ?w
}
edit
query
Base Layer
Road
Overlays
roads
streets
places
water_body
hydro_area
flow_line
hydro_line
Choose a spatial-join algorithm
Nested Loop
Plane Sweep
R-Tree
Query options
Filter features by map extent
Filter features by zoom level
Reduce join distance by zoom level
Simplify features before spatial join
Log:
New Haven
Foxon
Allingtown
East Haven
bing



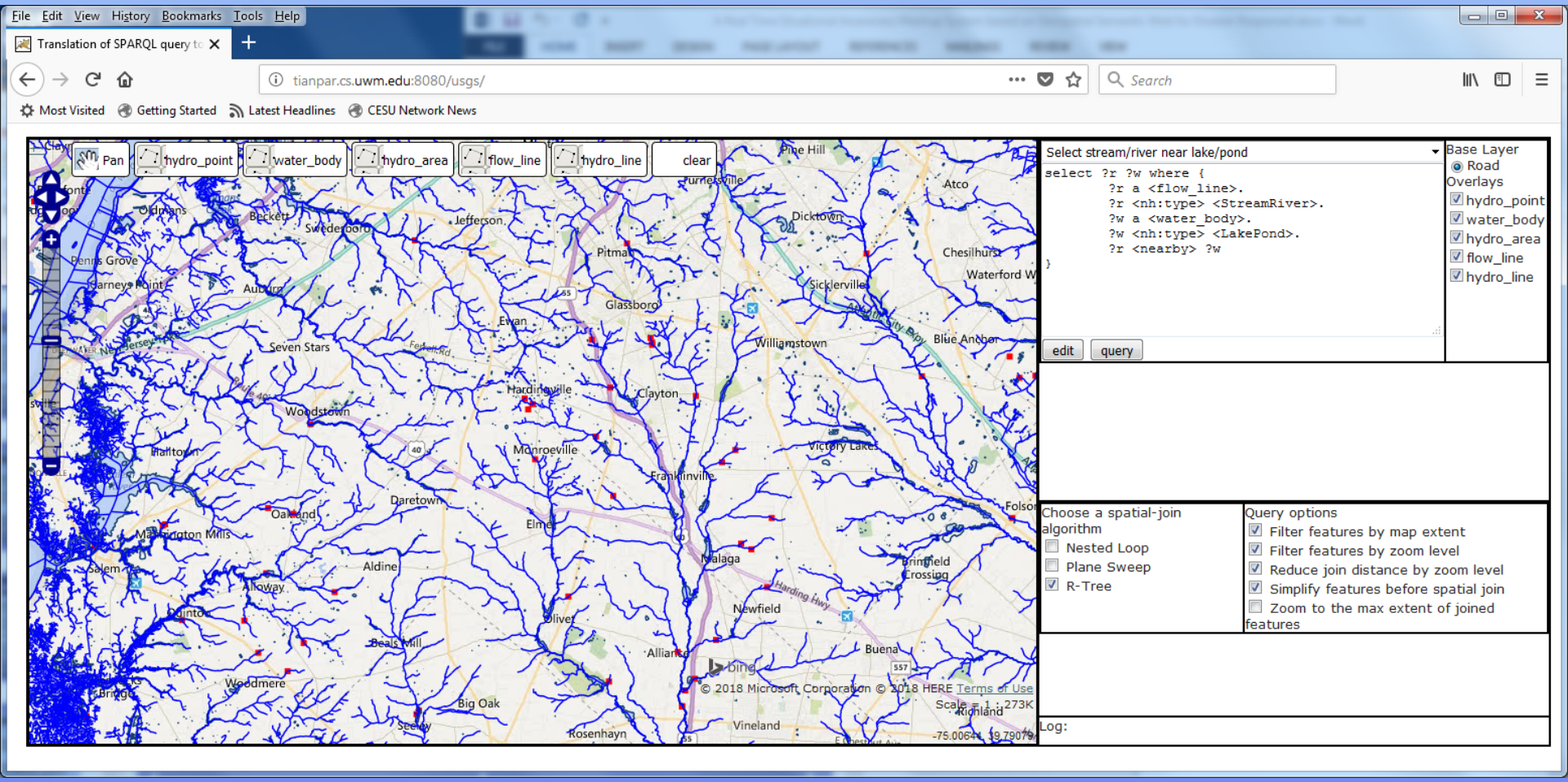

File Edit View History Bookmarks Tools Help
Translation of SPARQL query
tianpar.cs.uwm.edu:8080/usgs/
Search
Most Visited
Getting Started
Latest Headlines
CESU Network News
Pan
hydro_point
water_body
hydro_area
flow_line
hydro_line
clear
Select stream/river near lake/pond
select ?r ?w where {
?r a <flow_line>.
?r <nh:type> <StreamRiver>.
?w a <water_body>.
?w <nh:type> <LakePond>.
?r <nearby> ?w
}
edit
query
Base Layer
Road
Overlays
hydro_point
water_body
hydro_area
flow_line
hydro_line
Choose a spatial-join algorithm
Nested Loop
Plane Sweep
R-Tree
Query options
Filter features by map extent
Filter features by zoom level
Reduce join distance by zoom level
Simplify features before spatial join
Zoom to the max extent of joined features
Log:
Pine Hill
Atco
Dicktown
Chesilhurst
Waterford W
Sicklerville
Glassboro
Williamstown
Blue Anchor
Victory Lakes
Folsom
Brimfield Crossing
Newfield
Buena
Vineland
Richland
Rosenhayn
Big Oak
Seeley
Woodmere
Beals Mill
Alloway
Aldine
Salem
Oakland
Daretown
Monroeville
Hardingville
Clayton
Woodstown
Seven Stars
Ewan
Auburn
Harrys Point
Penns Grove
Jefferson
Swedesboro
Beckett
Oldmans
Pitman
Franklinville
Elmer
Malaga
Olivet
Alliance
Harding Hwy
© 2018 Microsoft Corporation © 2018 HERE Terms of Use
Scale = 1 : 273K
-75.00644, 39.79079

Table 1. The total number of features for each of the involved spatial data layers.

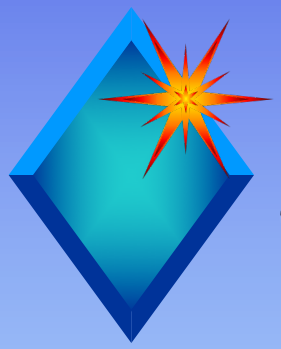
Layers	Number of features
Flow_line	10654
Water_body	4995
Streets	3449
Roads	1537
Places	54

We conducted six GeoSPARQL queries on the client side computer as listed below:

1. Select flow line near water body, which involves 4995 waterbody polygon features and 10654 flowline line features for the spatial joint query after the initial filtering using non-spatial query.
2. Select stream or river near lake or pond, which involves 2522 waterbody polygon features and 4818 flowline features for the spatial joint query after the initial filtering using non-spatial query.
3. Select streets near high schools, which involves 9 school point features and 3449 street line features for the spatial joint query after the initial filtering using non-spatial query.
4. Select streets near “Fair Haven School” and less than 500 feet, which involves 1 school point feature and 2481 street line features for the spatial joint query after the initial filtering using non-spatial query.
5. Select roads near middle schools, which involves 4 school point features and 1537 road line features for the spatial joint query after the initial filtering using non-spatial query.
6. Select high schools near state highways, which involves 9 school point features and 29 highway line features for the spatial joint query after the initial filtering using non-spatial query.

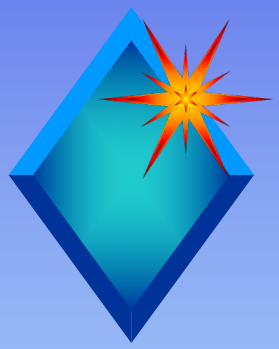
Table 2. Performance of spatial joint query examples.

	Nested Loop algorithm	R-Tree spatial index	Tiled based rendering	Cache
Select flow line near water body (4995 polygons and 10654 lines)	728244ms (12 minutes)	23034ms (23 sec)	7457ms (7.5 sec)	3339ms (3.3 sec)
Select stream or river near lake or pond (2522 polygons and 4818 lines)	174998ms (3 minutes)	3698ms (3.7 sec)	1977ms (2 sec)	715ms (0.7 sec)
Select streets near high schools (9 points and 3449 lines)	2234ms (2.2 sec)	2200ms (2.2 sec)	2172ms (2.1 sec)	155ms (0.1 sec)
Select streets near “Fair Haven School” and less than 500 feet (1 points; 2481 lines)	1786ms (1.8 sec)	1690ms (1.7 sec)	1760ms (1.8 sec)	19ms (0.02 sec)
Select roads near middle schools (4 points and 1537 lines)	879ms (0.9 sec)	871ms (0.9 sec)	874ms (0.9 sec)	36ms (0.04 sec)
Select high schools near state highways (9 points and 29 lines)	217ms (0.2 sec)	211ms (0.2 sec)	226ms (0.2 sec)	18ms (0.02 sec)



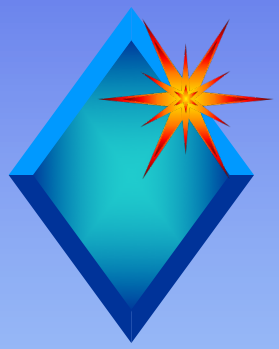
Some Query examples

Query examples	Nested loop	Plane sweep	Index nested-loop	Hierarchical traversal join
Select stream or river near lake or pond (3818)	4431ms	366ms	315ms	368ms
Select flow line intersecting water body (3035)	2295ms	594ms	450ms	483ms
Select streets near high schools(3440)	167ms	99ms	107ms	105ms
Select elementary schools near “Connecticut Tpke” and intersecting streets(3447)	201ms	155ms	133ms	130ms
Select streets near ”Fair Haven School” and less than 500 feet (2479)	59ms	84ms	72ms	196ms



Conclusion

- Our experimental results show that the developed prototype can greatly reduce the runtime costs of GeoSPARQL queries through on-the-fly spatial index, tiled based rendering, and cache techniques, especially for those spatial joint GeoSPARQL queries involving a large number of spatial features and heavy geometric computations.



Conclusion

- There are some limitations in the currently developed prototype:
  - The on-the-fly R-tree index has its own limitation due to its poor scaling characteristic.
  - These pre-fetched tiles may consume significant space in memory.
  - A client might be using the stale cached data because of a lack of proper updating.

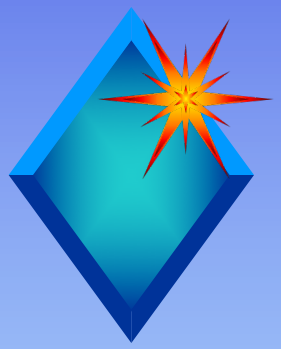
Acknowledgement

- This work is supported by the U.S. Geological Survey (USGS) (funding G17AS00057).

USGS science for a changing world

UCONN

UNIVERSITY of WISCONSIN UWMILWAUKEE



Thank you!