Scalable Interactive Middleware Components for Ubiquitous Fashionable Computers

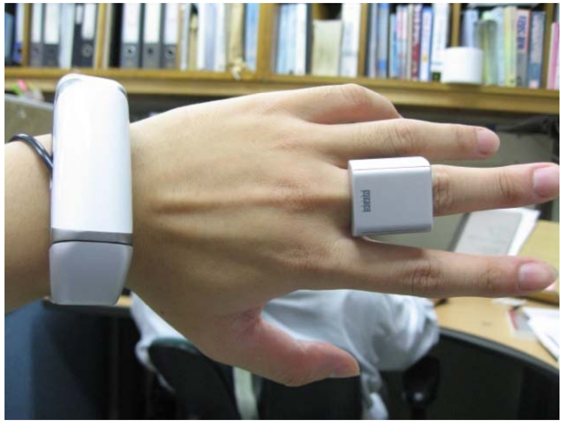
Gyudong Shim, Kyu-Ho Park
presented by Gyudong Shim
gdshim@core.kaist.ac.kr
2009.4.28

Contents

- Introduction
  - U-interactive system for Ubiquitous Fashionable Computers
- Motivation of scalability issues
- Scalable Interactive Middleware Components
  - Tuple indexing and query methods
  - Fan search
    - Target selection with angle and distance by space filling curves
- Performance Evaluation
- Conclusions

UFC

- UFC(Ubiquitous Fashionable Computer)

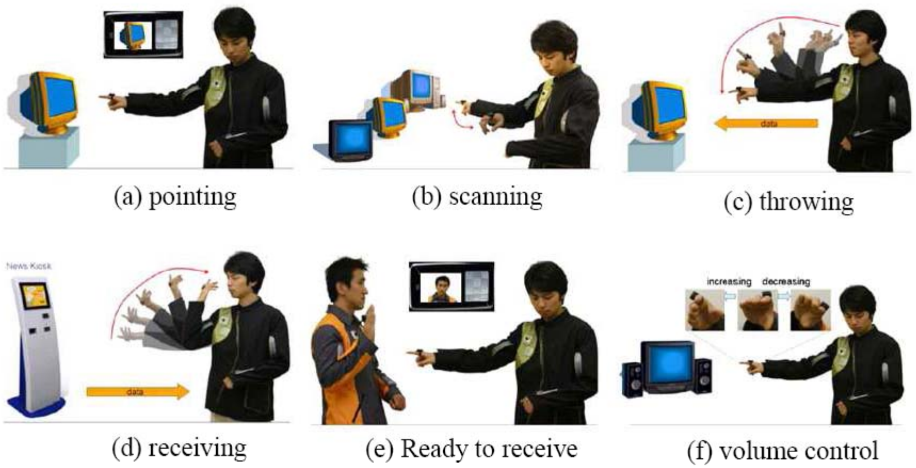


iThrow

Display(touch screen)

iThrow

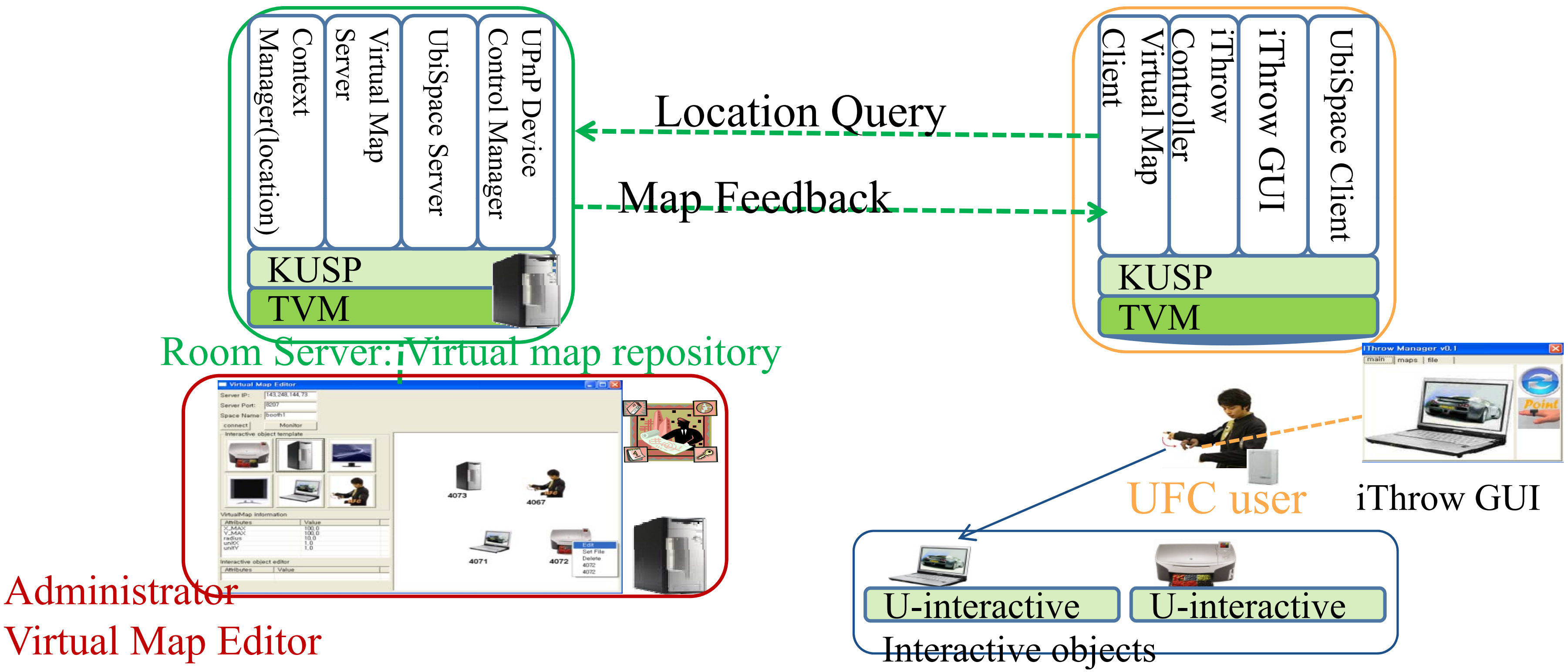
- *iThrow*
  - Intuitive gesture based interface for ubiquitous services
    - Control TV, DVD players
    - Uploading UCC after editing pictures
    - PowerPoint presenter



iThrow command sets

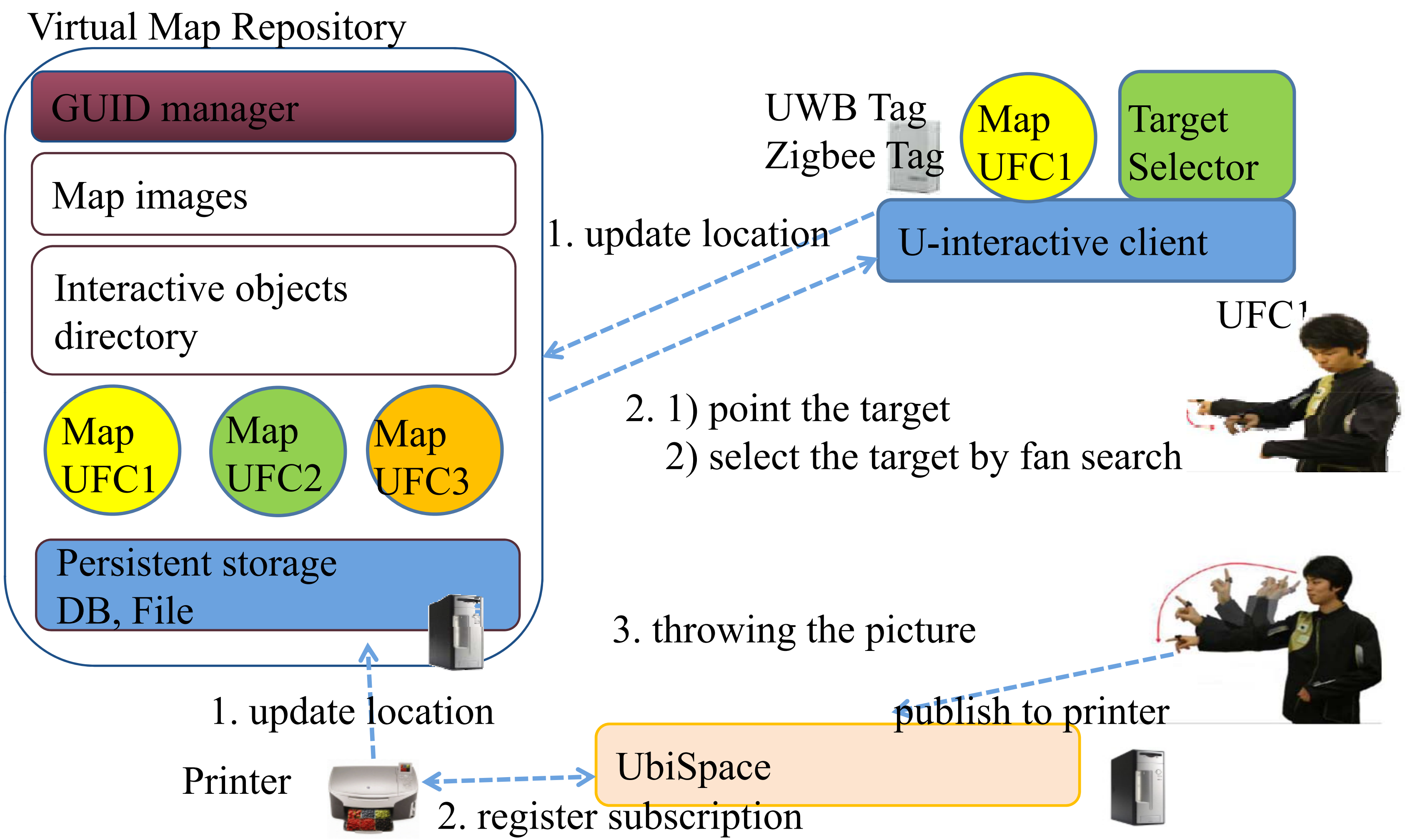
U-interactive System Architecture

- U-interactive middleware
  - Location based interaction with surrounding services by *iThrow* interface
  - Handles user commands, data transfers, location services



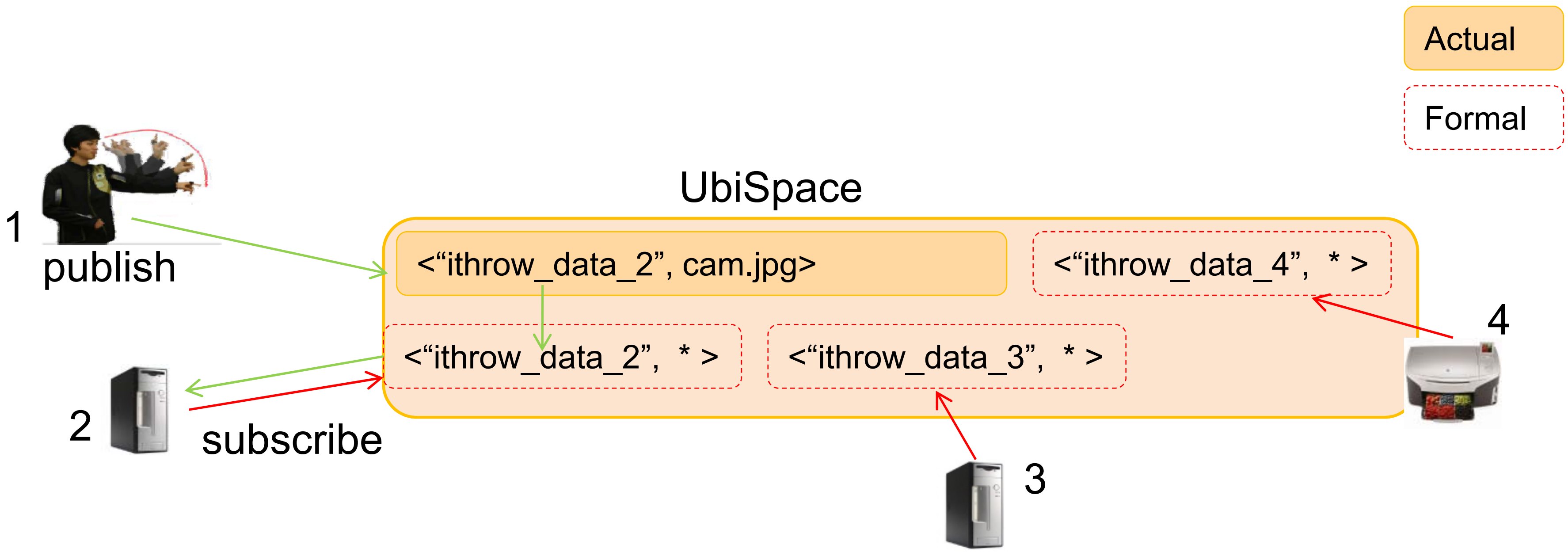
Example of U-interactive operation

- Print a File Scenarios



UbiSpace

- UbiSpace
  - Tuple space based coordination middleware
  - Java object and File sharing
  - String key based publish/subscribe system

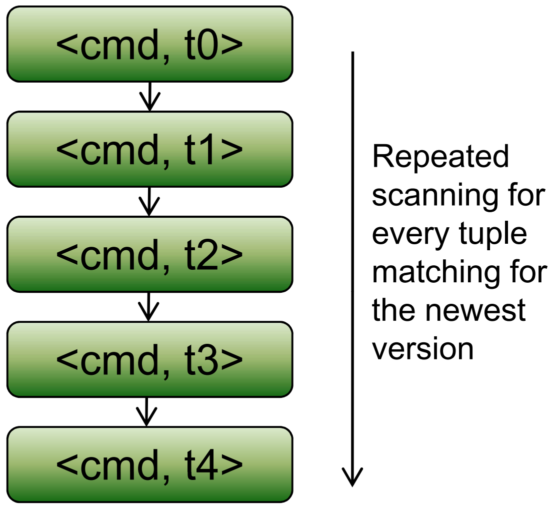


Motivation of SIMC

- Scalability of U-interactive for massive target environments
  - Museums, public stations with crowd users
  - More than thousands users and service objects
  - Frequent location updates and queries
  - A lot of control messages and files over active spaces
- Efficient data indexing and query processing for Our System
  - Tuple indexing in UbiSpace
  - Fan search by space filling curves with query optimization

1. Tuple Indexing Scheme

- Tuple matching pattern
  - Read the newest version of “KEY” in the tuple space - LIFO
  - Subscribe “KEY” from the tuple space
- Problem of T-Space[1]
  - Support FIFO ordering
  - Index tuples by template tuple through manual configuration
  - Exhaustive tuple scanning operation
- Tuple indexing for interactive spaces
  - Indexing by <name, time(reverse order)>



Tuple indexing by FIFO

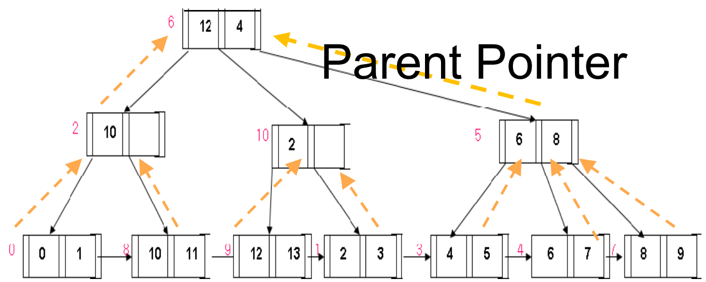
[1] Hitting the distributed computing sweet spot with TSpaces. In: Computing Networks, pp. 457–472 (2001)

1. Tuple Indexing Scheme

- Tuples are indexed by <tuple name, tuple id> composite keys for tuple matching
  - Tuple id is the serial number of tuple creation
  - The same name of tuples are indexed by tuple id by descending order
  - No repeated scanning overhead in tuple matching

Tuple Name	Tuple ID	Tuple
lcommand	12	
lcommand	11	
lcommand	10	
Ticket	4	
Ticket	2	
zoo	1	

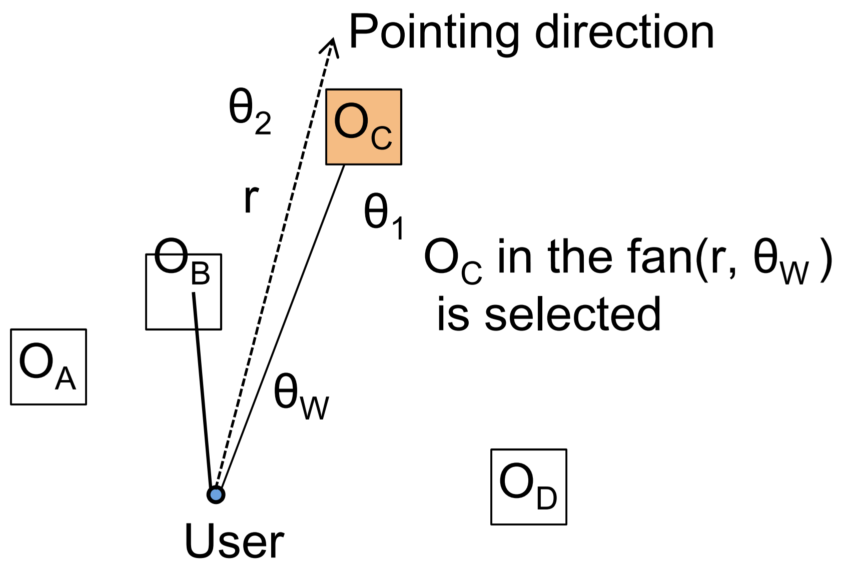
Tuple container<name, id>



B^+-Tree with composite key

2. Fan Search

- Fan search
  - Pointing direction θ_P
  - Allowed range θ_A
  - Select a target which is the closest to the θ_P within $[\theta_P - \theta_A .. \theta_P + \theta_A]$ and distance within r

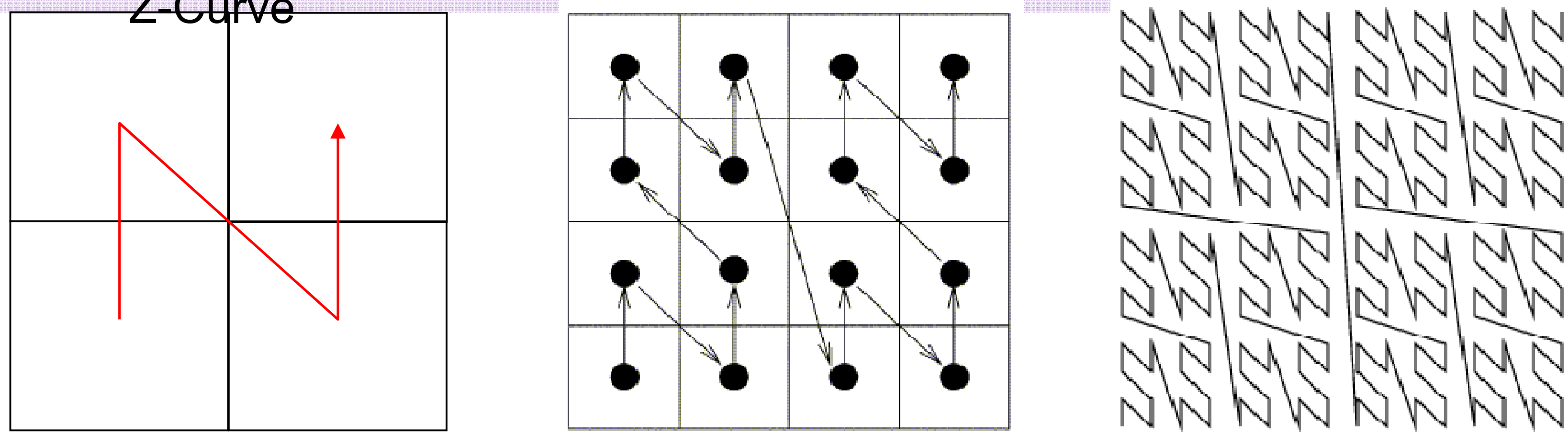


2. Fan Search with Space Filling Curves

- Efficient object indexing for location based queries
  - Exploit space filling curves for frequent location update and region queries
  - Query Optimization
    - Query region decomposition for space filling curves
    - Caching path stack for repeated tree-traversals
    - Query interval skip by leaf node data

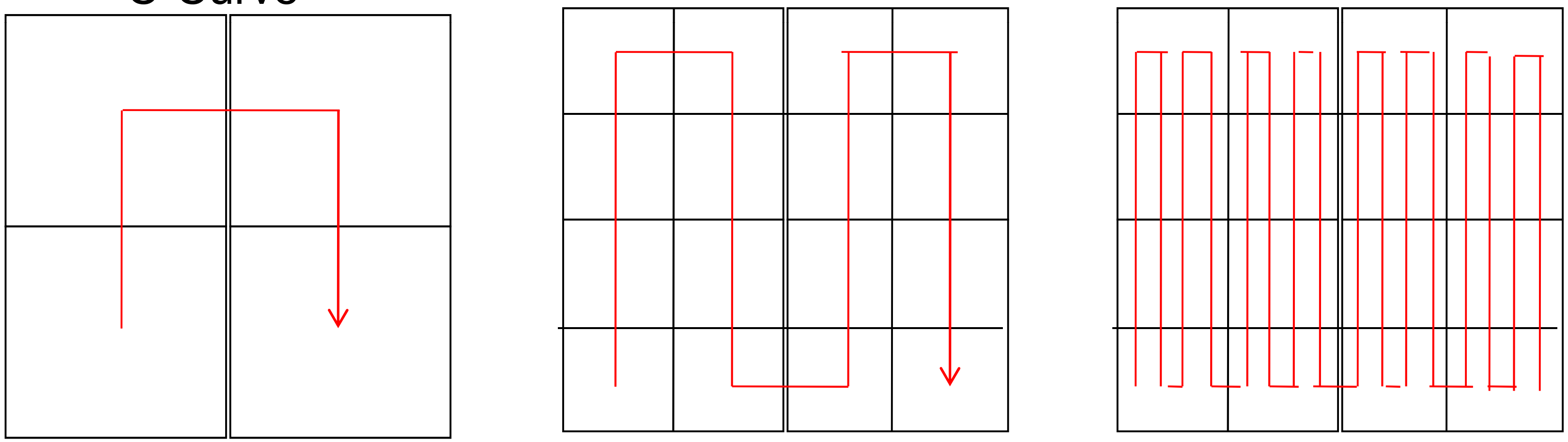
Space Filling Curves

Z-Curve



1st iteration 2nd iteration 4th iteration

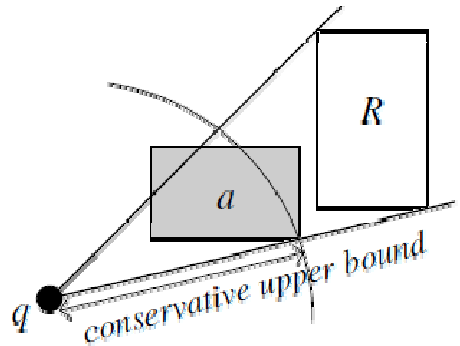
C-Curve

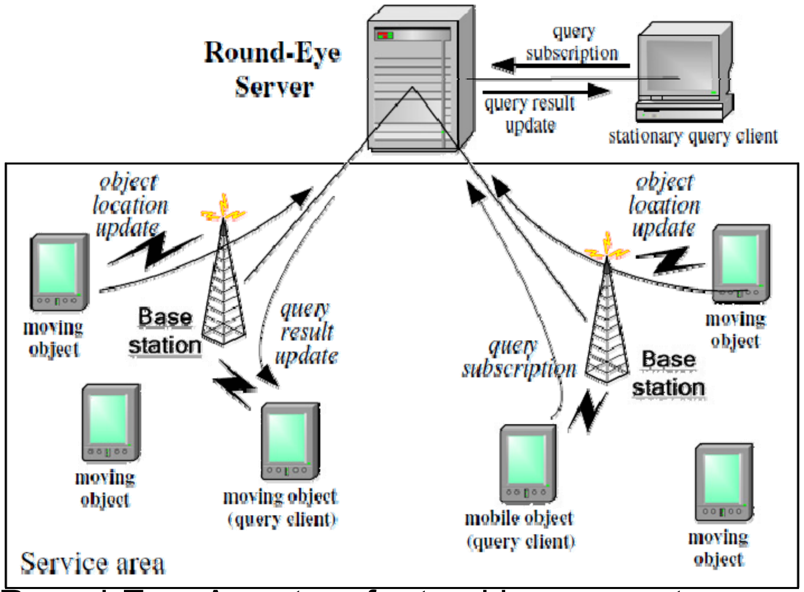


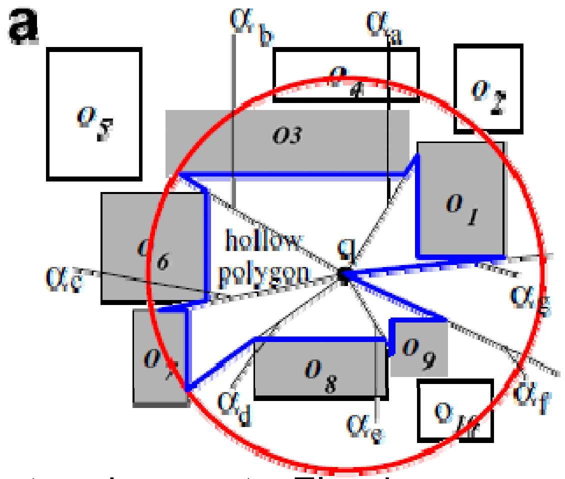
* Figures from JASS 2005 Saint Petersburg

Related Work : Round Eye[2]

- Nearest Surrounder Search
  - A set of nearest surroundings at given position
  - Application: Robot soccer
  - Query indexing for efficient tracking
  - R-Tree based Query, Object indexing





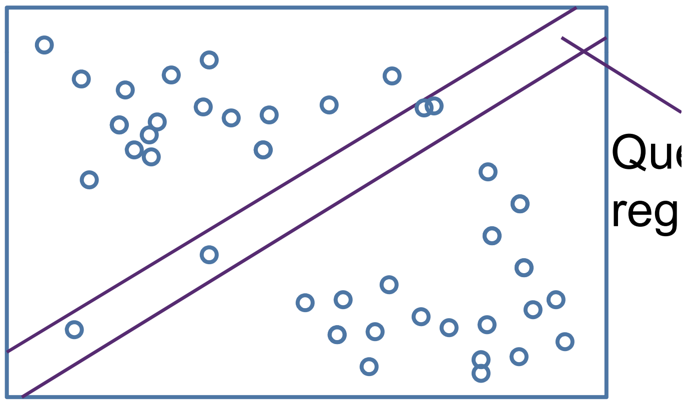


[2] Round-Eye: A system for tracking nearest surrounders in moving object environments. Elsevier Information Systems 80(12), 2063–2076 (2007)

2.1 Query Decompositions

- Motivation
  - Reduce false hits by single MBR for spatial queries such as line, curves, and fan
  - Selection of Space filling curves
    - Z-Curve(DRU) performance degradation by decomposition query region

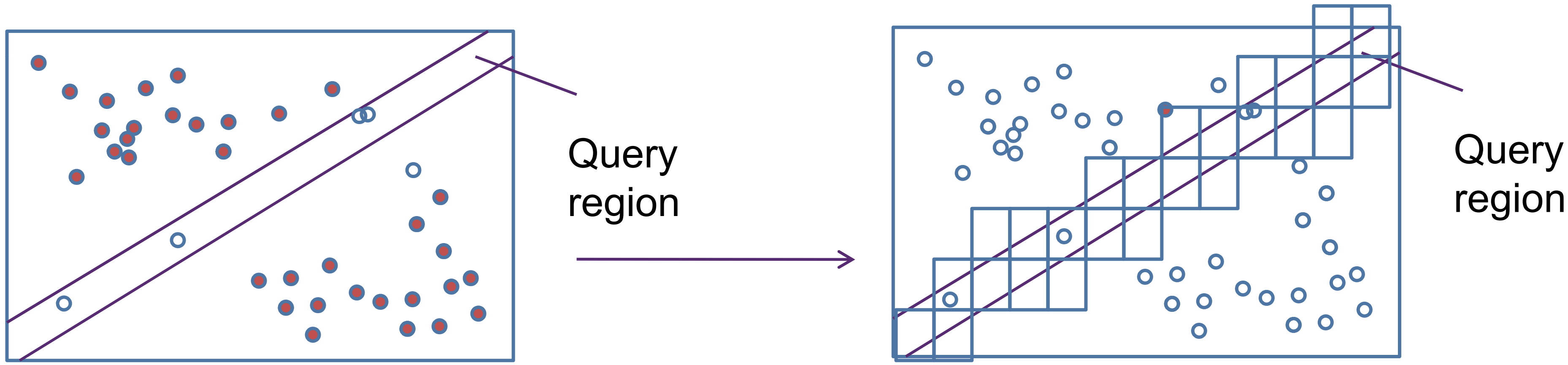
      duplicated outside query region checks
    - C-Curve with multi interval query



Road planning: lookup any buildings which intersect the new road

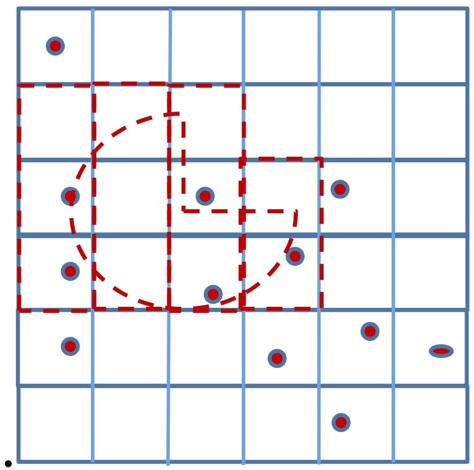
2.1 Integral Range Queries

- One MBR query vs sum of MBRs queries
  - Query region approximation by one MBR is inefficient by FALSE HIT



2.1 Integral Range Queries

1. MBRs Calculation
   - For each x_i intervals from x_{min} to x_{max} of query region
     calculate y_{min}, y_{max} value for $[x_i..x_{i+1}) \rightarrow R_{x_i}$

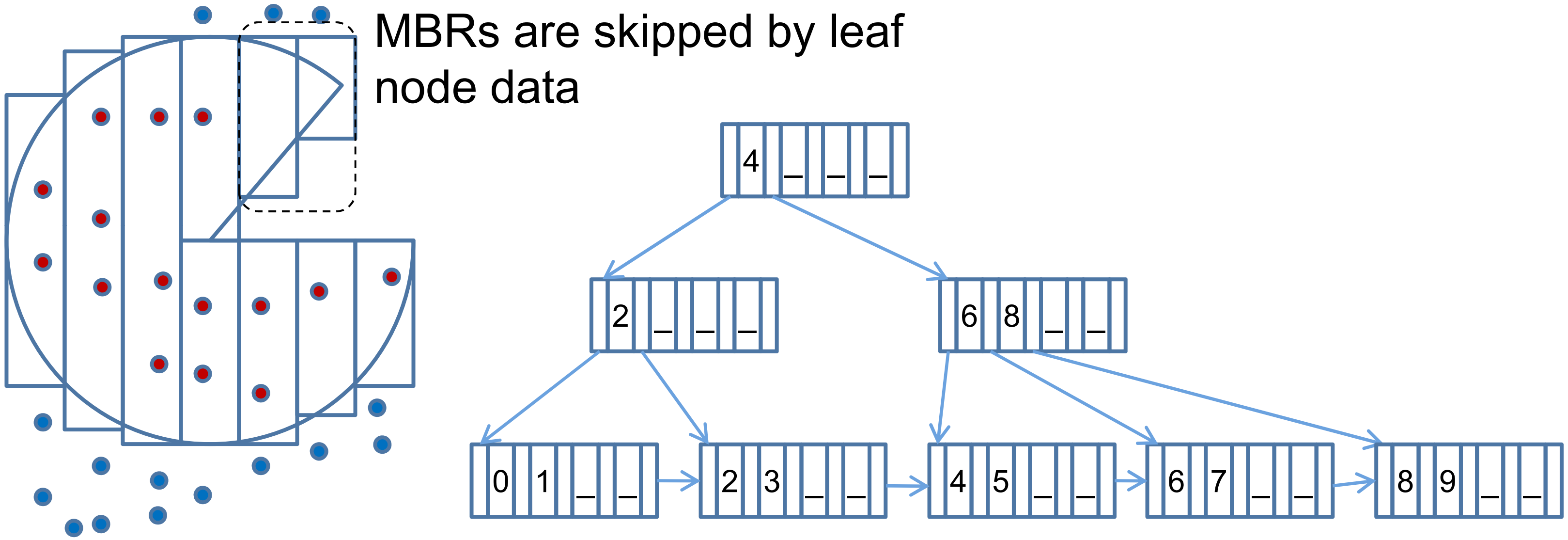


2. Range Query for multiple intervals
   - BPTree.rangeQuery($[R_{x_0} .. R_{x_k}]$)

3. Pruning candidate results
   - Check the object is inside the query region

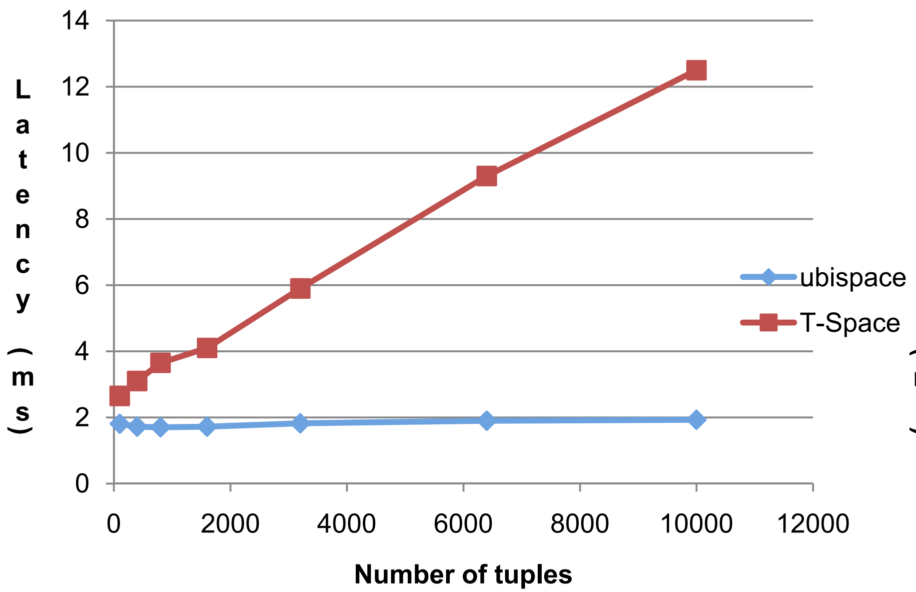
2.2 LeafNode jumps

- If an entry of leaf node exceeds the current range,
  - Skip MBRs which are behind the entry

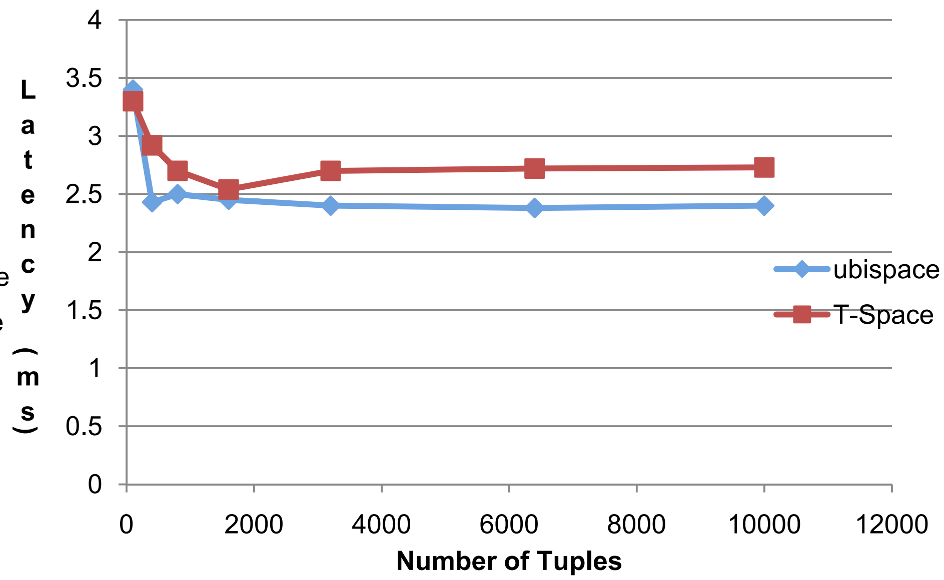


Performance Evaluation – Tuple indexing

- UbiSpace Tuple indexing effect
  - UbiSpace exploit the indexing effect
  - Bounded matching time to the number of tuples



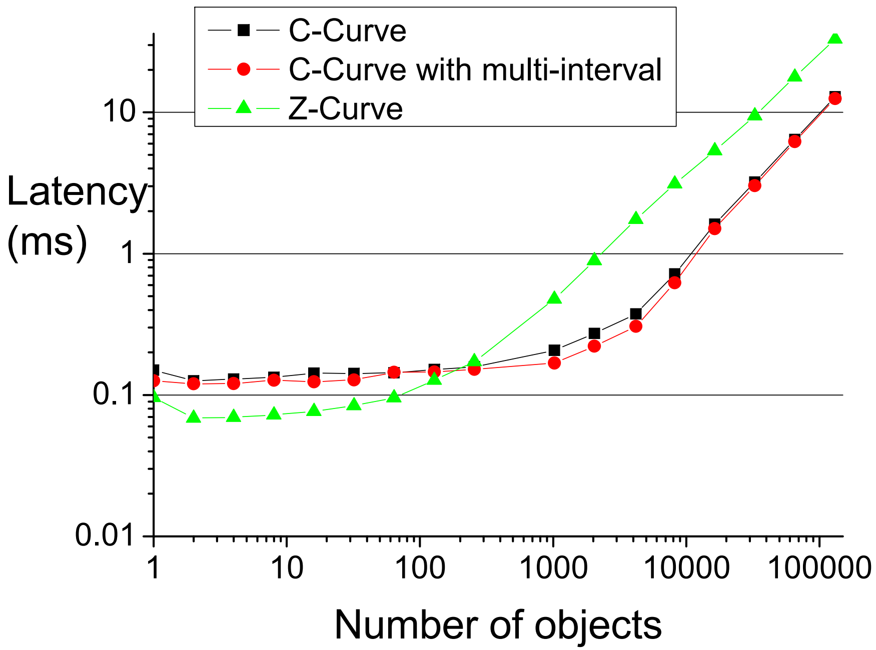
Average latency of read operations



Average latency of write operation

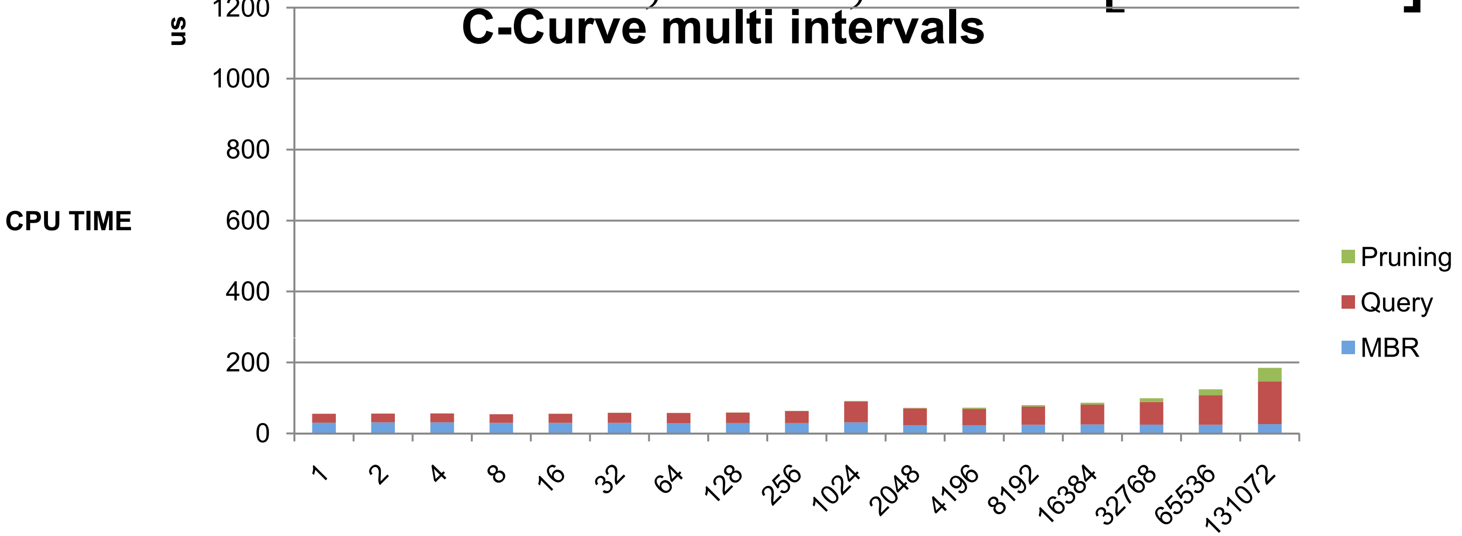
Performance Evaluation Fan Search Latency

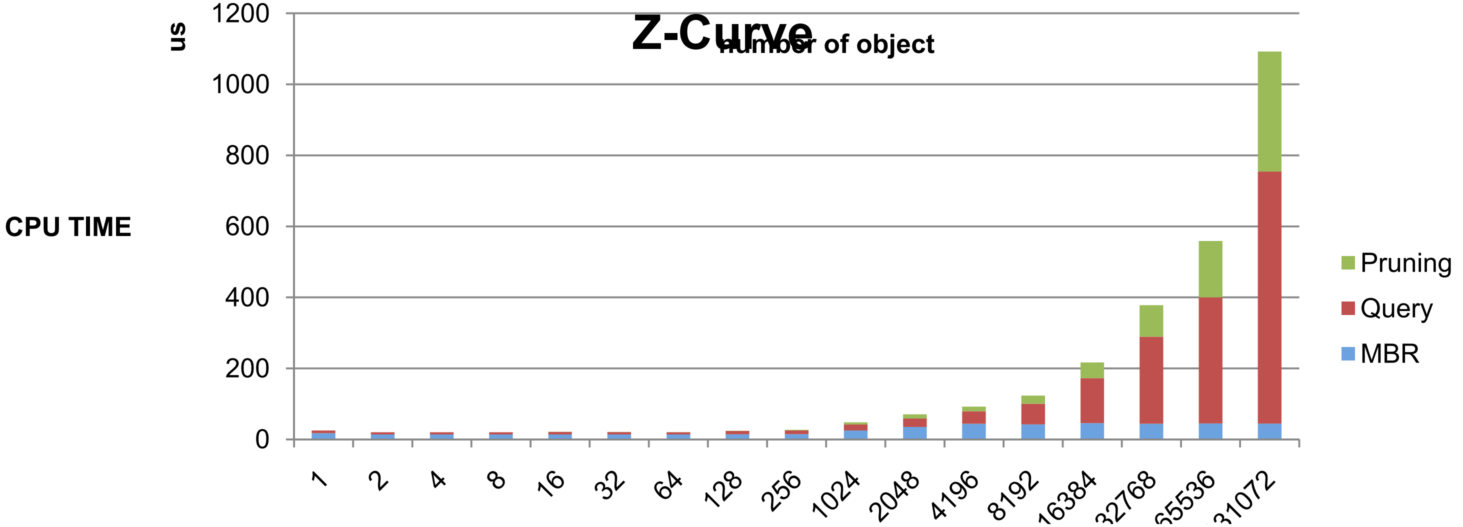
- Effect of Various space filling curves
  - In low density, Z-Curve outperforms
  - In high density, C-Curve outperforms due to less false hit on tree traverse
  - Path stack cache and leaf node jumps improves up to 5% latency



Performance Evaluation
Effect of Space Filling Curves

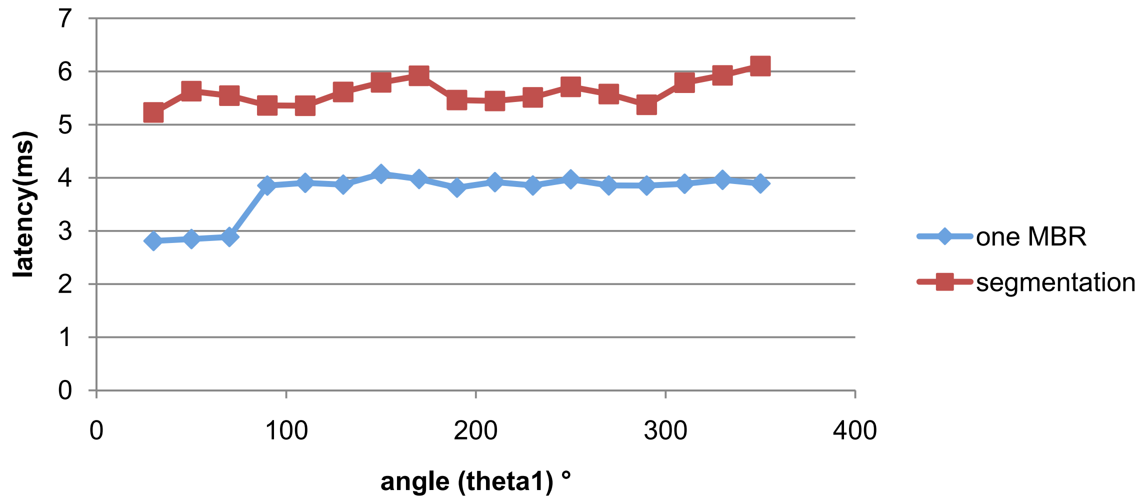
- Fan search 1km x 1km, r=50, theta: [40˚..50˚]

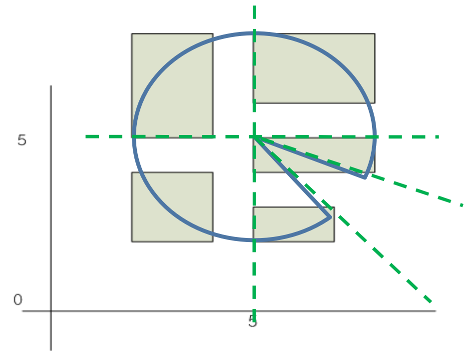


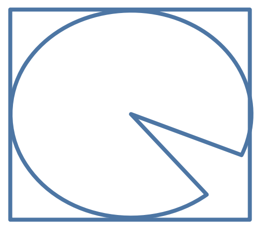


Effect of Query region decomposition

- Effect of Segmentation of Z-Curve
  - Segmentation cause poor performance
  - Candidate region + real query box region in each MBR,
  - Decomposition causes the more candidate region and duplicated comparison & region check overhead
  - In Z-curve we should query by an MBR which covers the fan

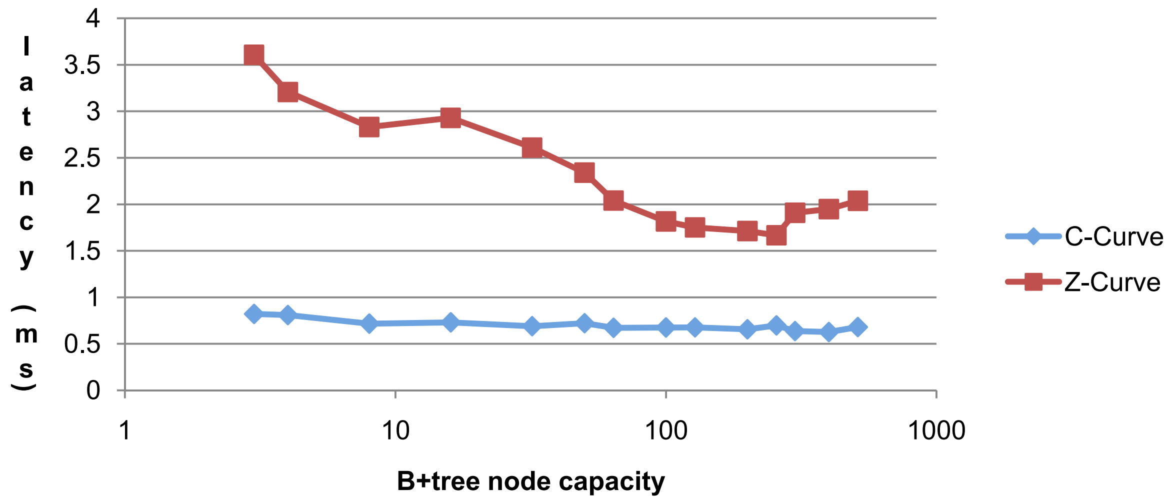






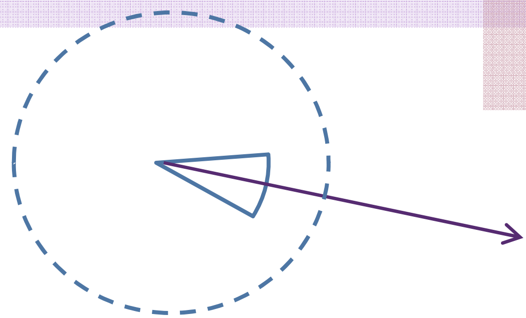
Performance Evaluation

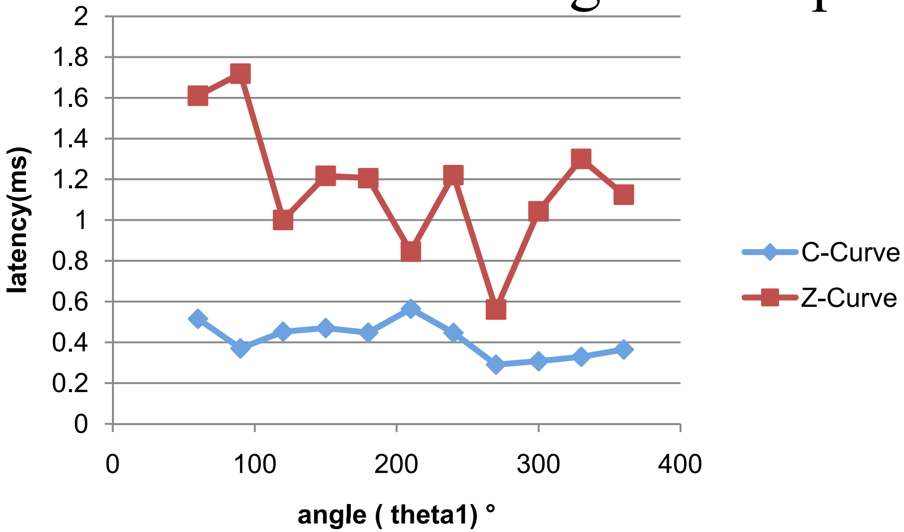
- Effect of B+tree node capacity
  - It is known that a node size should be almost of the page size (4KBytes) so normally 100 to 200
  - The result indicates that Z-Curve has time complexity of logC in small number of objects, best at 256
    - Z-Curve's DRU algorithm requires node interval resolve by tree traversal, As the n is smaller, the chance of tree traversal increases more
  - C-Curve has almost constant, best at 400

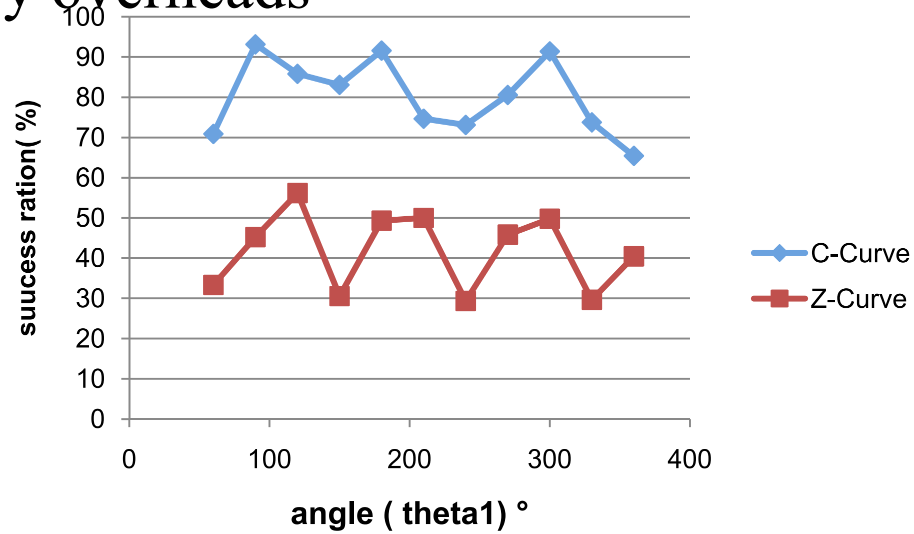


Performance Evaluation

- Effect of Angle direction
  - Almost same in C-Curve
  - Z-curve
    - not homogeneous query overheads

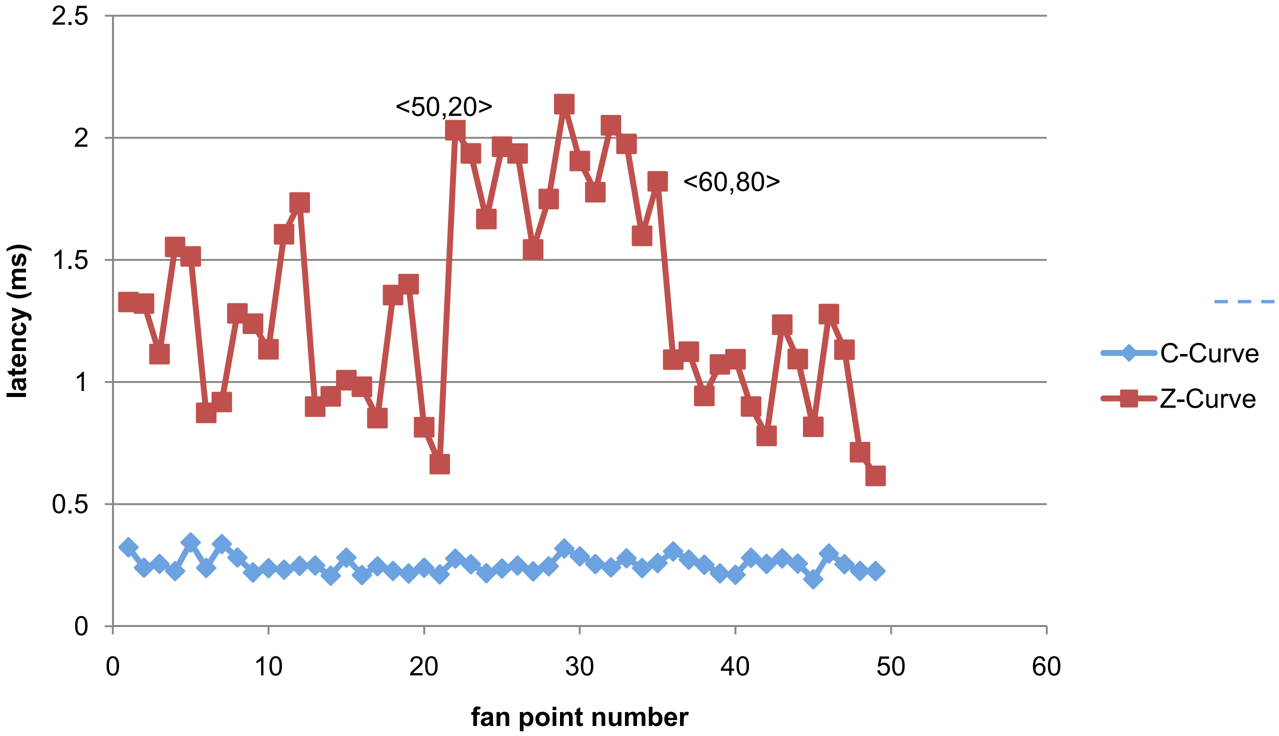


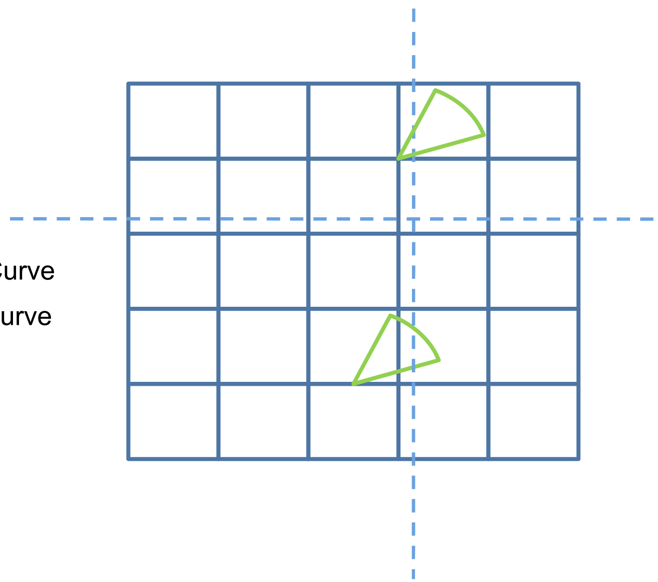




Performance Evaluation

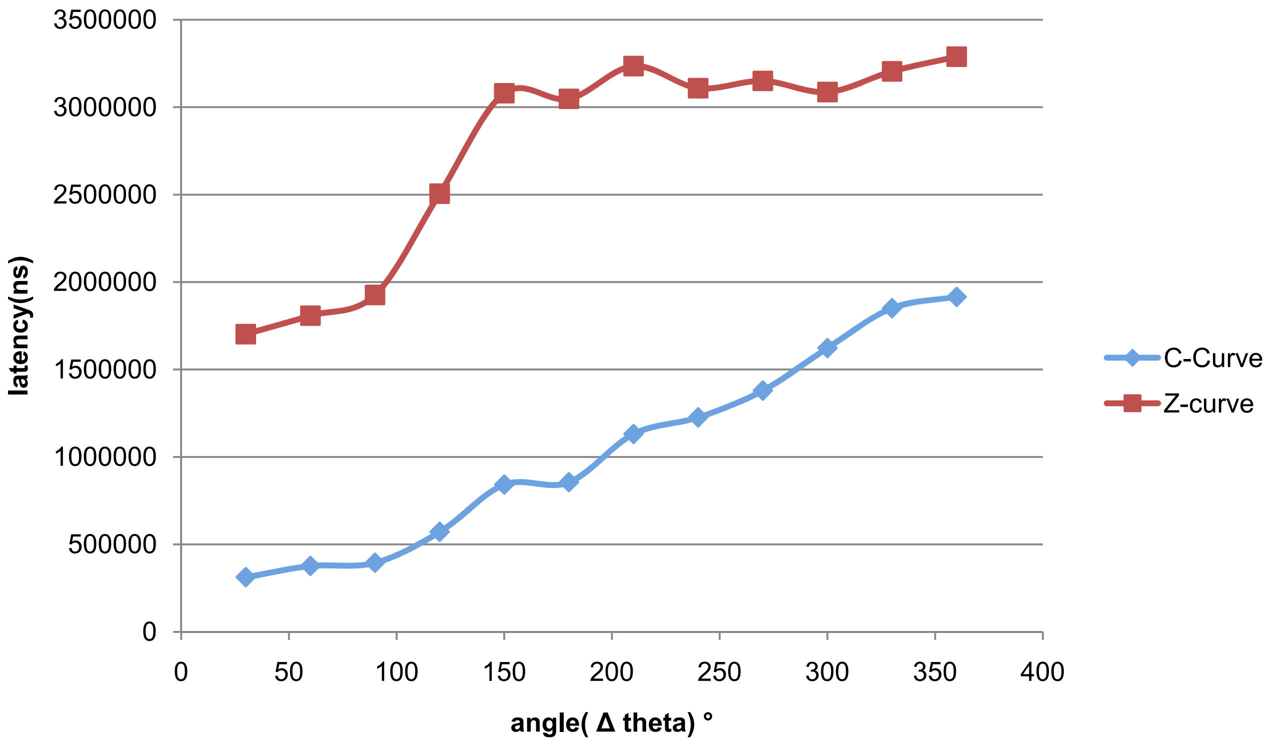
- Effect of Origin of the Fan
  - Does Z-curve has homogeneous query overhead in the space?
  - Answer : No! discontinuous points in space filling curves causes candidates regions





Performance Evaluation

- Effect of Fan angle width
  - Linear increase



Conclusions & Further works

- Conclusions
  - SIMC are designed and implemented to be scalable to the number of data by tuple indexing, fan search with SFC.
  - Fan Search with C-Curve provides better latency in large density of nodes than Z-Curve
- Further works
  - Network and System architectures should be tailored to be scalable in Massive Ubiquitous Environments