RawVis
Visual Exploration over Raw Data

Nikos Bikakis [1] Stavros Maroulis [1,2]
George Papastefanatos [1,2] Panos Vassiliadis [1]

[1] University of Ioannina, Greece
[2] IMSI Institute, ATHENA R.C., Greece

Ευρωπαϊκή Ένωση
European Social Fund

Operational Programme
Human Resources Development, Education and Lifelong Learning
Co-financed by Greece and the European Union

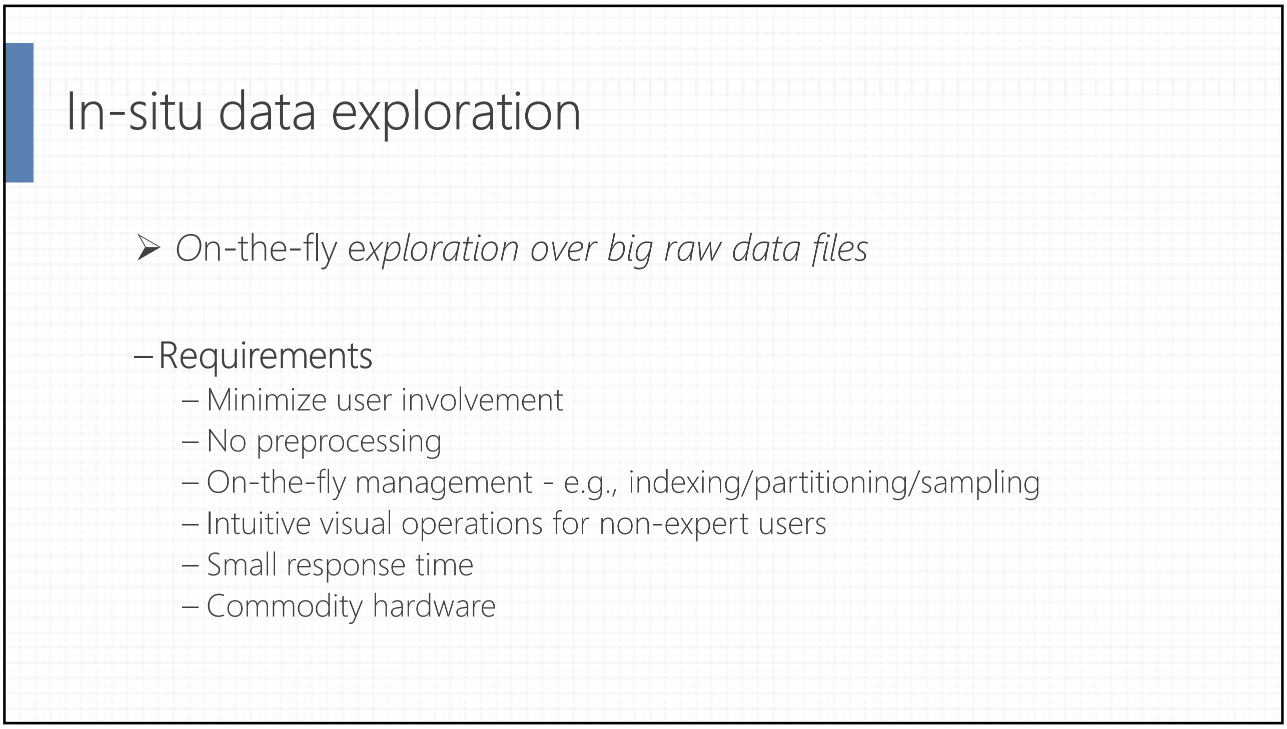
ΕΣΠΑ 2014-2020
ανάπτυξη - εργασία - αλληλεγγύη

Intro

- Today
  - Large & dynamic datasets
- Traditional DBMS
  - loading & indexing → long data-to-query time
- Distributed approaches
  - Expensive
  - Not accessible to most (non-expert) users

Challenge

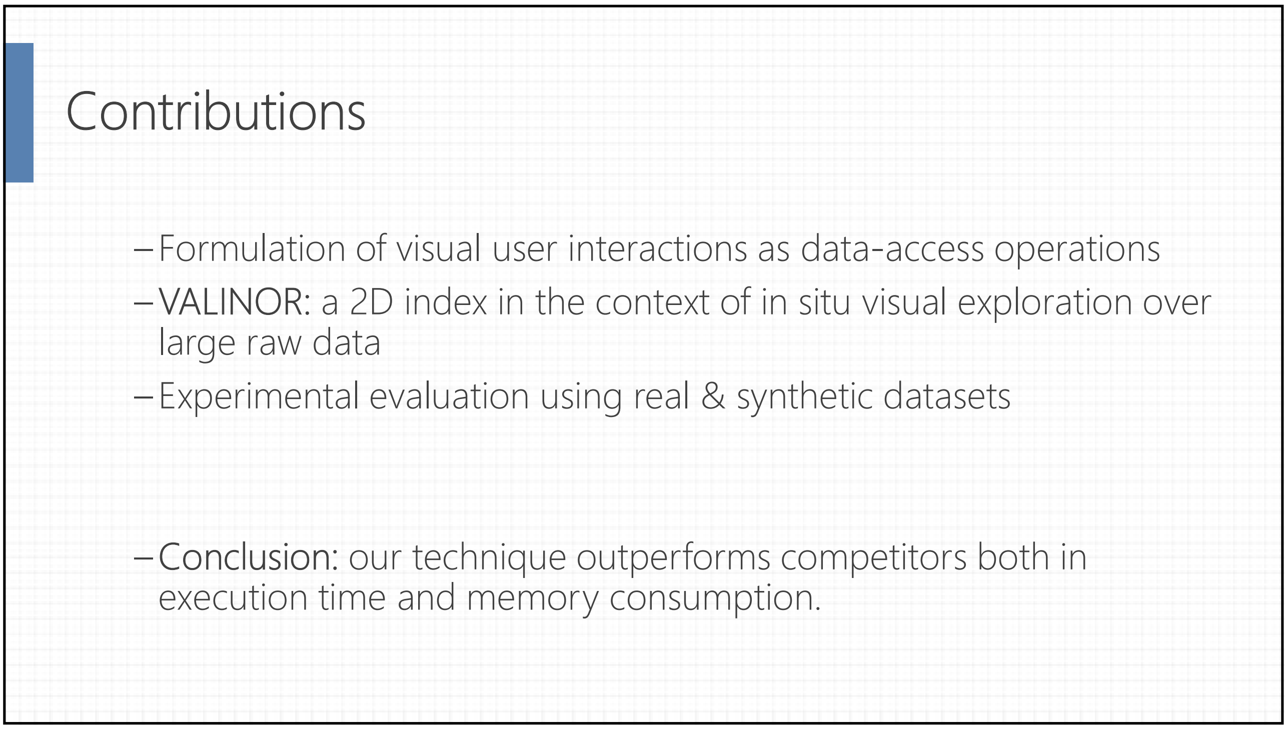
On-the-fly Visual Exploration over Raw Data using commodity hardware



In-situ data exploration

➢ On-the-fly *exploration over big raw data files*

– Requirements
  – Minimize user involvement
  – No preprocessing
  – On-the-fly management - e.g., indexing/partitioning/sampling
  – Intuitive visual operations for non-expert users
  – Small response time
  – Commodity hardware



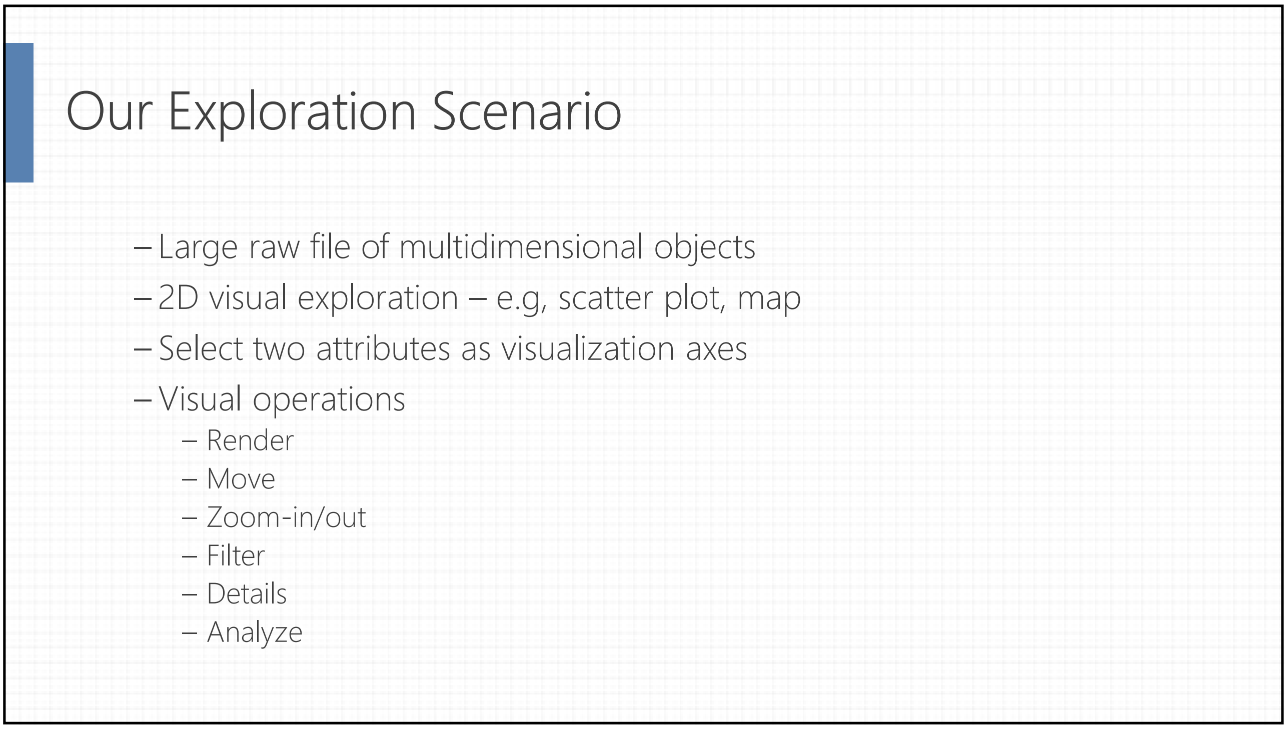
Contributions

– Formulation of visual user interactions as data-access operations

– VALINOR: a 2D index in the context of in situ visual exploration over large raw data

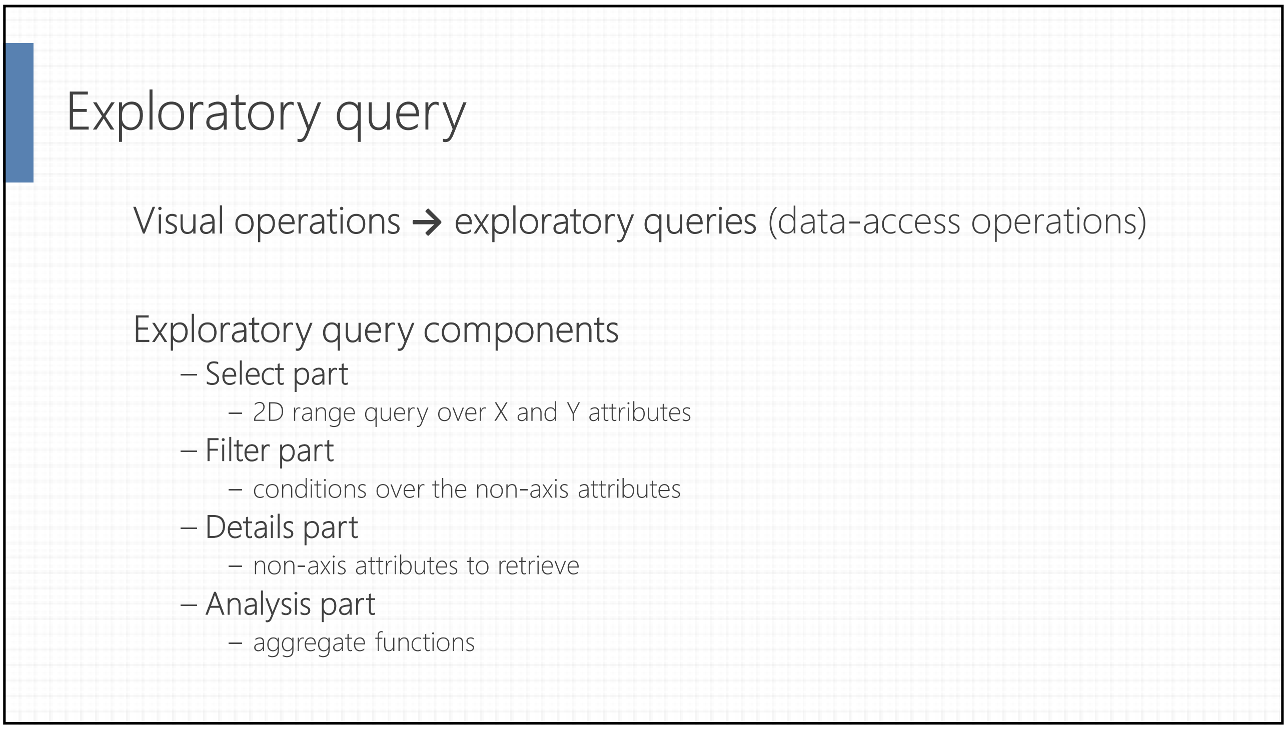
– Experimental evaluation using real & synthetic datasets

– Conclusion: our technique outperforms competitors both in execution time and memory consumption.



Our Exploration Scenario

- Large raw file of multidimensional objects
- 2D visual exploration – e.g, scatter plot, map
- Select two attributes as visualization axes
- Visual operations
  - Render
  - Move
  - Zoom-in/out
  - Filter
  - Details
  - Analyze

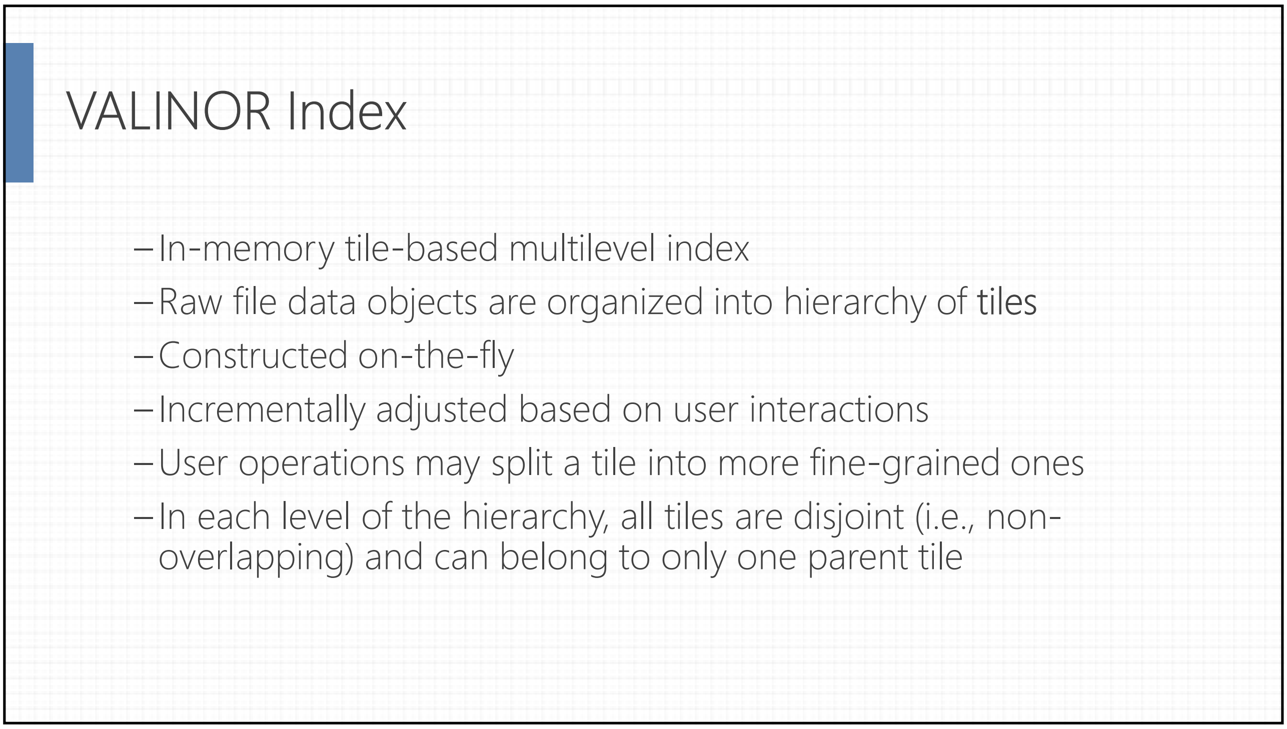


Exploratory query

Visual operations → exploratory queries (data-access operations)

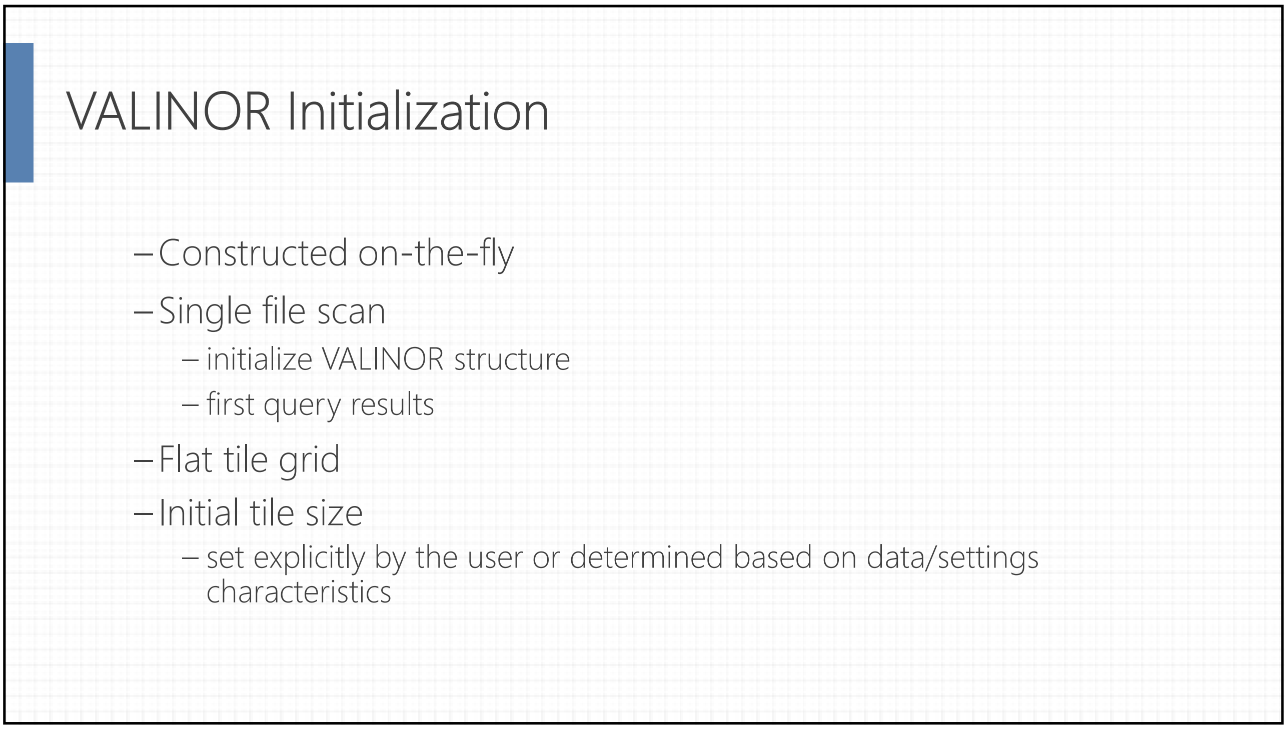
Exploratory query components

- Select part
  - 2D range query over X and Y attributes
- Filter part
  - conditions over the non-axis attributes
- Details part
  - non-axis attributes to retrieve
- Analysis part
  - aggregate functions



VALINOR Index

- In-memory tile-based multilevel index
- Raw file data objects are organized into hierarchy of tiles
- Constructed on-the-fly
- Incrementally adjusted based on user interactions
- User operations may split a tile into more fine-grained ones
- In each level of the hierarchy, all tiles are disjoint (i.e., non-overlapping) and can belong to only one parent tile



VALINOR Initialization

- Constructed on-the-fly
- Single file scan
  - initialize VALINOR structure
  - first query results
- Flat tile grid
- Initial tile size
  - set explicitly by the user or determined based on data/settings characteristics

VALINOR Initialization

Raw File

Attributes

Objects

$$\begin{array}{c|ccccccc|} & A_1 & \ldots & A_x & \ldots & A_y & \ldots & A_d \\ \hline o_1 & a_{1,1} & \ldots & a_{1,x} & \ldots & a_{1,y} & \ldots & a_{1,d} \\ o_2 & a_{2,1} & \ldots & a_{2,x} & \ldots & a_{2,y} & \ldots & a_{2,d} \\ & & & & \ldots & & & \\ o_1 & a_{n,1} & \ldots & a_{n,x} & \ldots & a_{n,y} & \ldots & a_{n,d} \\ \hline \end{array}$$

VALINOR Initialization

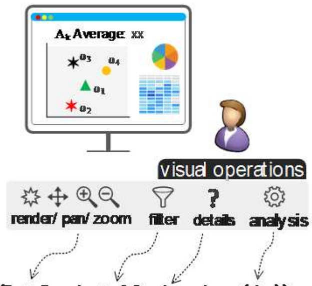
Raw File

Attributes

Objects

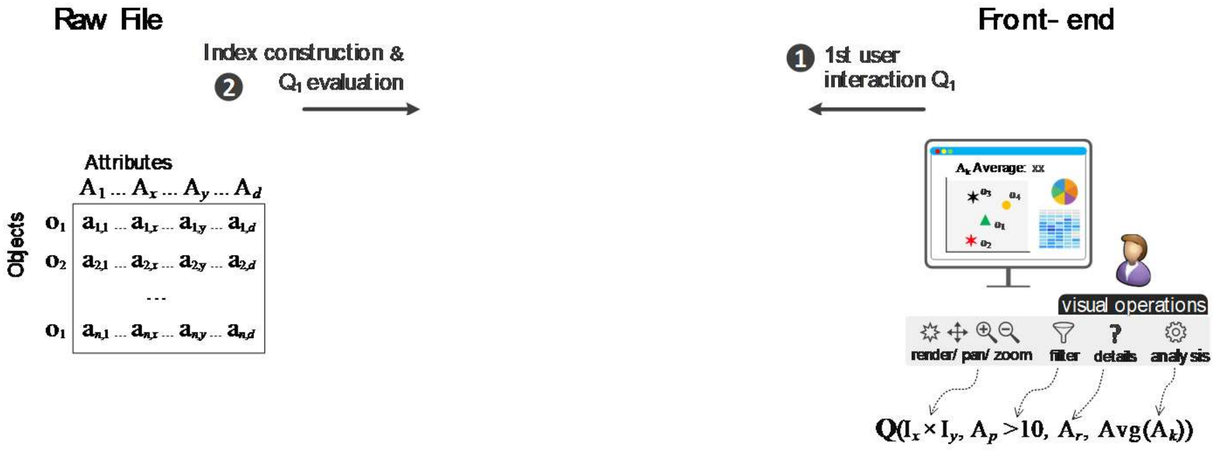
$$\begin{array}{c|ccccccc|} & A_1 & \ldots & A_x & \ldots & A_y & \ldots & A_d \\ \hline o_1 & a_{1,1} & \ldots & a_{1,x} & \ldots & a_{1,y} & \ldots & a_{1,d} \\ o_2 & a_{2,1} & \ldots & a_{2,x} & \ldots & a_{2,y} & \ldots & a_{2,d} \\ & & & & \ldots & & & \\ o_1 & a_{n,1} & \ldots & a_{n,x} & \ldots & a_{n,y} & \ldots & a_{n,d} \\ \hline \end{array}$$

Front- end



$Q(I_x \times I_y,\ A_p > 10,\ A_r,\ Avg(A_k))$

VALINOR Initialization



Raw File

2 Index construction & Q_1 evaluation

Front-end

1 1st user interaction Q_1

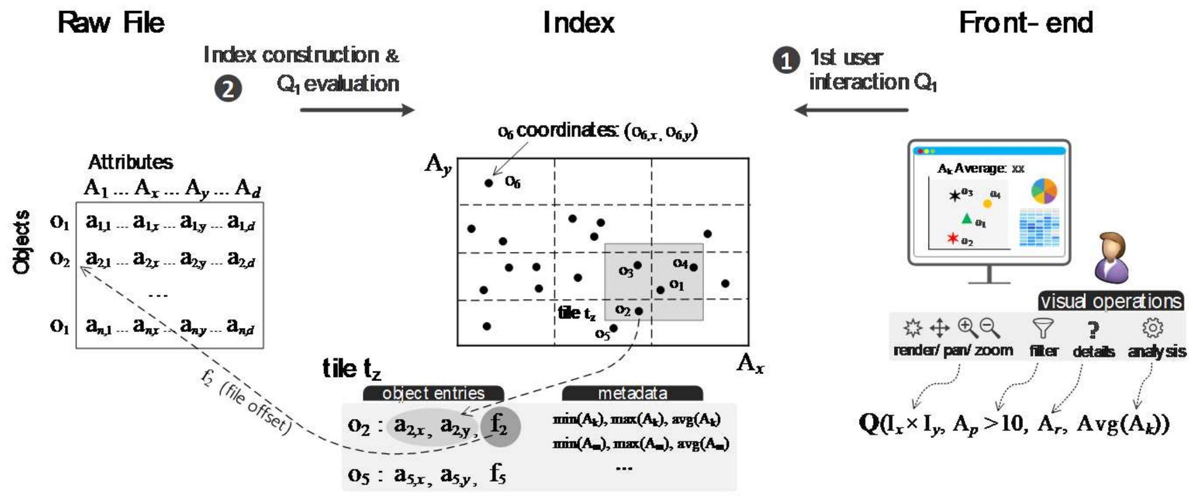
Attributes

$$\begin{array}{c|c} & A_1 \dots A_x \dots A_y \dots A_d \\ \hline o_1 & a_{1,1} \dots a_{1,x} \dots a_{1,y} \dots a_{1,d} \\ o_2 & a_{2,1} \dots a_{2,x} \dots a_{2,y} \dots a_{2,d} \\ & \dots \\ o_1 & a_{n,1} \dots a_{n,x} \dots a_{n,y} \dots a_{n,d} \end{array}$$

Objects

$Q(I_x \times I_y, A_p > 10, A_r, Avg(A_k))$

VALINOR Initialization



Raw File

Index

2 Index construction & Q_1 evaluation

Front-end

1 1st user interaction Q_1

Attributes

$$\begin{array}{c|c} & A_1 \dots A_x \dots A_y \dots A_d \\ \hline o_1 & a_{1,1} \dots a_{1,x} \dots a_{1,y} \dots a_{1,d} \\ o_2 & a_{2,1} \dots a_{2,x} \dots a_{2,y} \dots a_{2,d} \\ & \dots \\ o_1 & a_{n,1} \dots a_{n,x} \dots a_{n,y} \dots a_{n,d} \end{array}$$

Objects

o_6 coordinates: $(o_{6,x}, o_{6,y})$

tile t_z

object entries

o_2 : $a_{2,x}$, $a_{2,y}$, f_2

o_5 : $a_{5,x}$, $a_{5,y}$, f_5

metadata

min(A_k), max(A_k), avg(A_k)

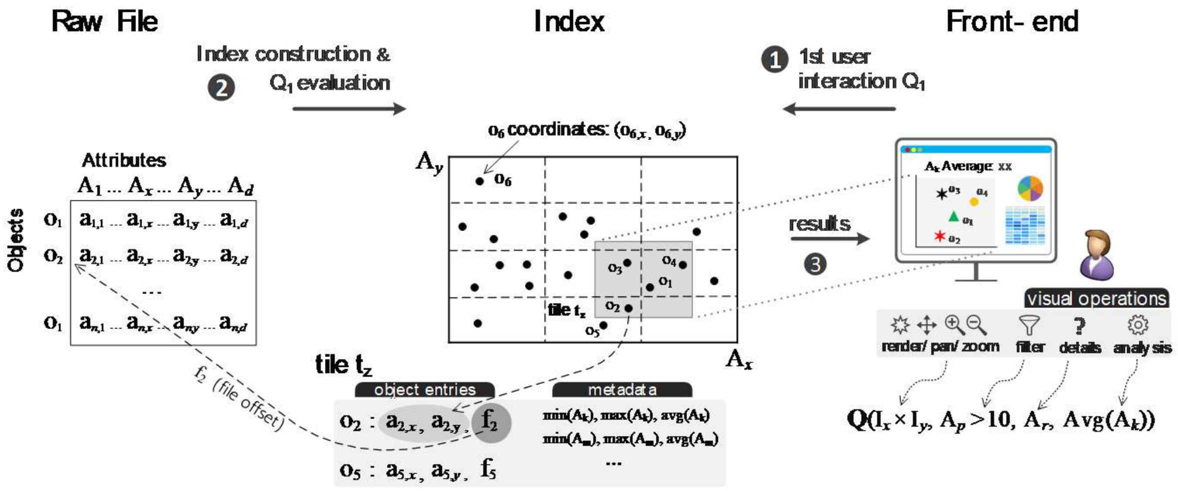
min(A_m), max(A_m), avg(A_m)

...

f_2 (file offset)

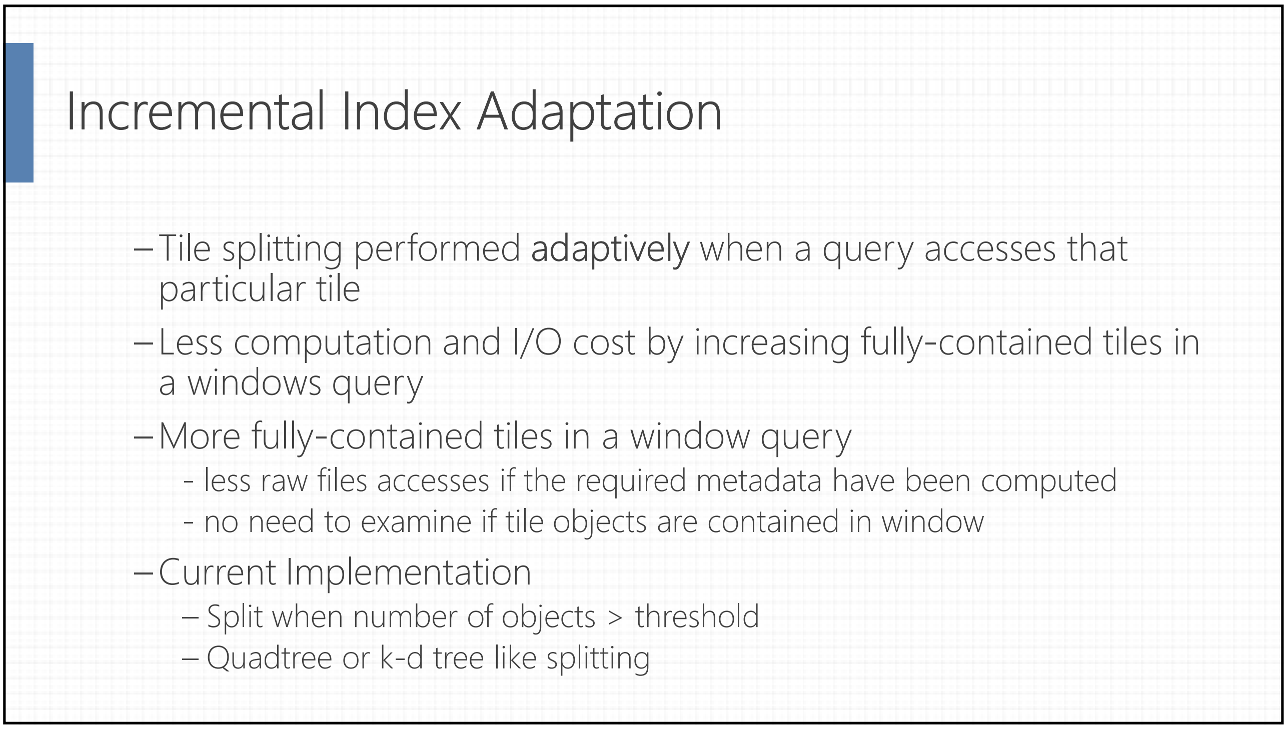
$Q(I_x \times I_y, A_p > 10, A_r, Avg(A_k))$

VALINOR Initialization



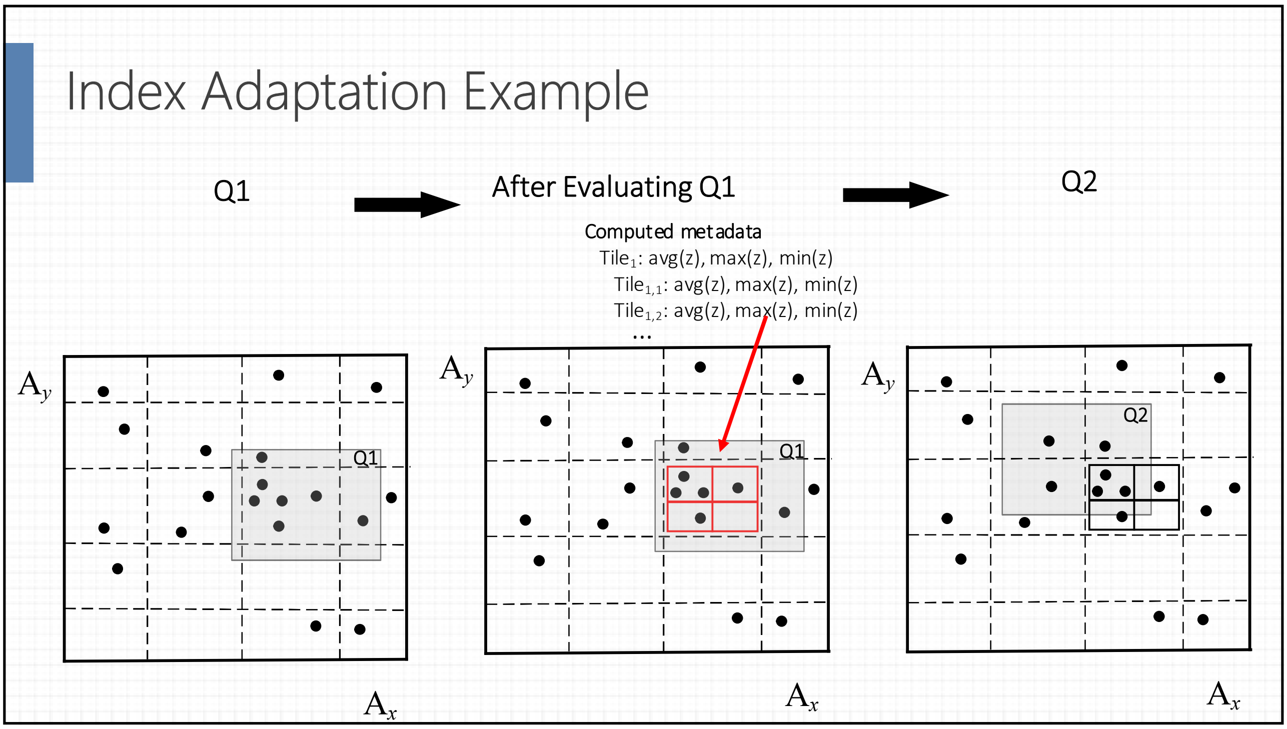
Query Evaluation

- Select part
  - Determine the leaf tiles that overlap with the query
  - For partially-contained tiles find the objects contained in the window
- Details
  - Details part always requires file access
- Analysis & Filter part
  - Metadata computed in previous queries may be used
- Tile Metadata
  - Computed incrementally when fetching non-axis attribute for fully-contained tiles
  - Improves performance of filter & analysis expressions



Incremental Index Adaptation

- Tile splitting performed **adaptively** when a query accesses that particular tile
- Less computation and I/O cost by increasing fully-contained tiles in a windows query
- More fully-contained tiles in a window query
  - less raw files accesses if the required metadata have been computed
  - no need to examine if tile objects are contained in window
- Current Implementation
  - Split when number of objects > threshold
  - Quadtree or k-d tree like splitting



Index Adaptation Example

Q1

After Evaluating Q1

Q2

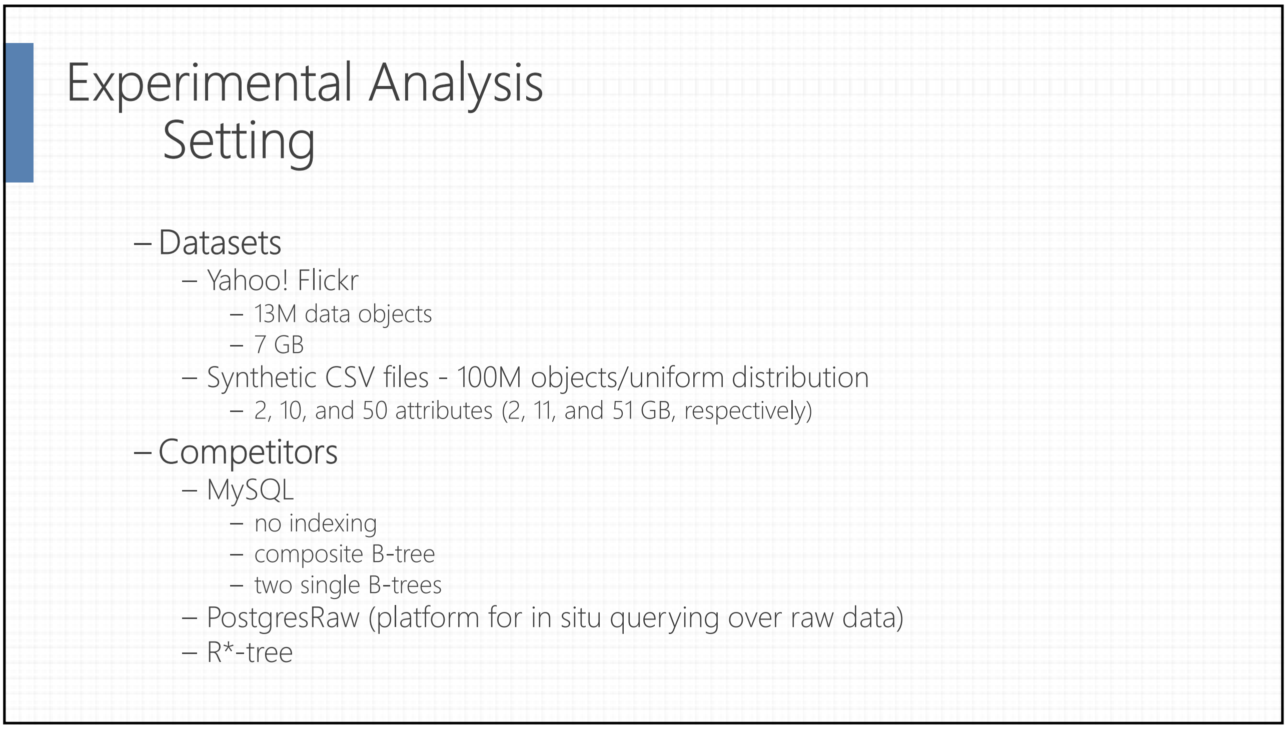
Computed metadata

Tile$_1$: avg(z), max(z), min(z)

Tile$_{1,1}$: avg(z), max(z), min(z)

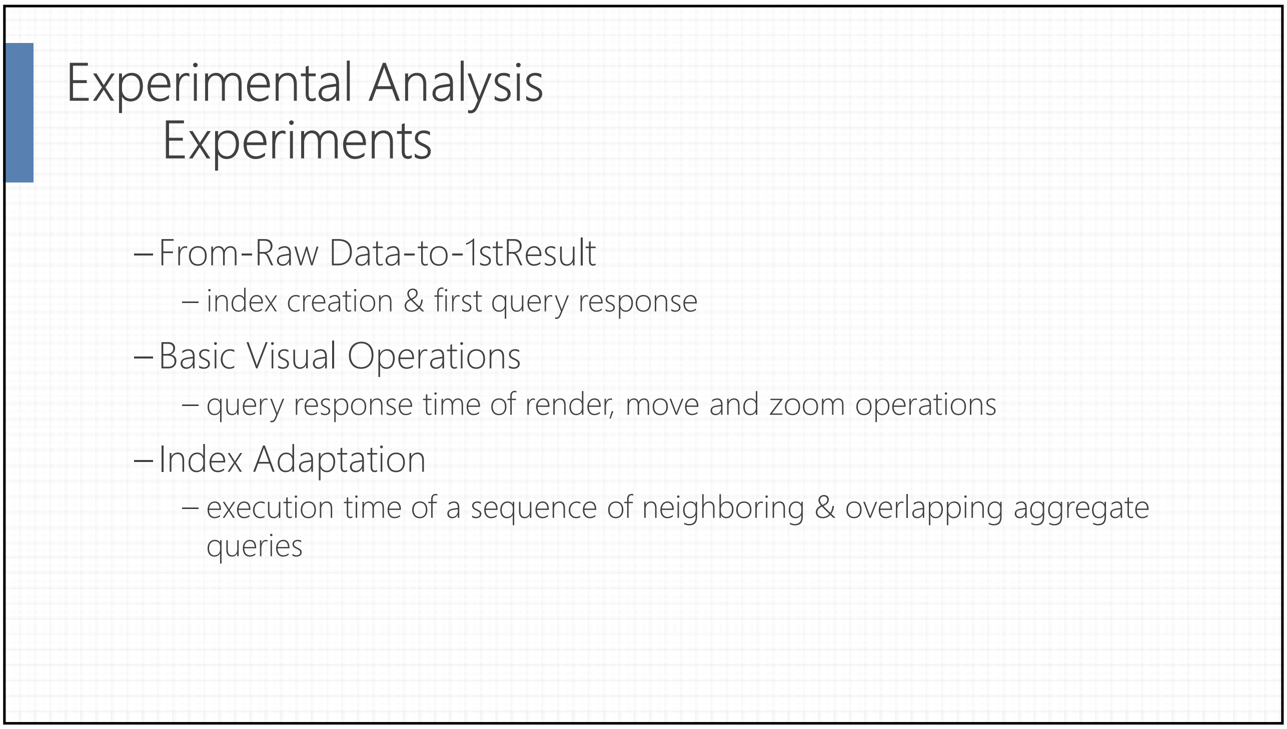
Tile$_{1,2}$: avg(z), max(z), min(z)

...



Experimental Analysis
Setting

- Datasets
  - Yahoo! Flickr
    - 13M data objects
    - 7 GB
  - Synthetic CSV files - 100M objects/uniform distribution
    - 2, 10, and 50 attributes (2, 11, and 51 GB, respectively)
- Competitors
  - MySQL
    - no indexing
    - composite B-tree
    - two single B-trees
  - PostgresRaw (platform for in situ querying over raw data)
  - R*-tree

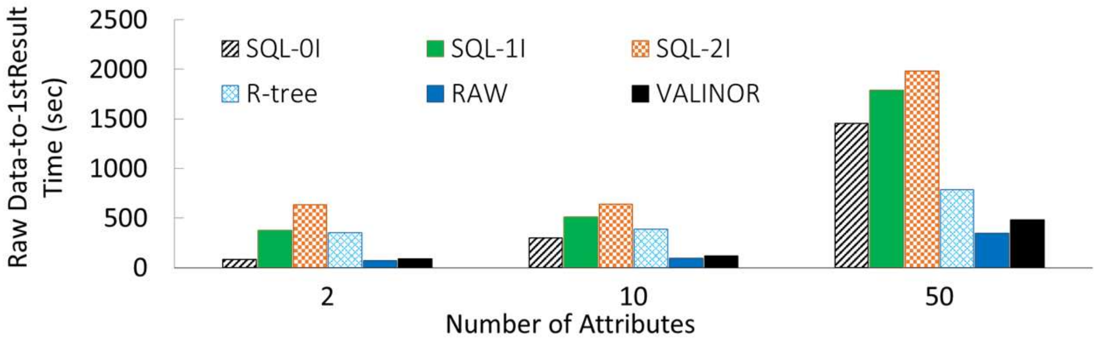


Experimental Analysis
Experiments

- From-Raw Data-to-1stResult
  - index creation & first query response
- Basic Visual Operations
  - query response time of render, move and zoom operations
- Index Adaptation
  - execution time of a sequence of neighboring & overlapping aggregate queries

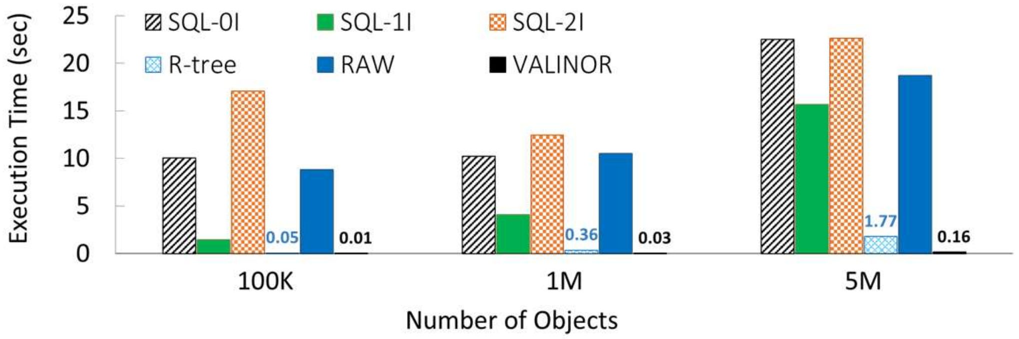
From-Raw Data-to-1stResult Time

Synthetic Data



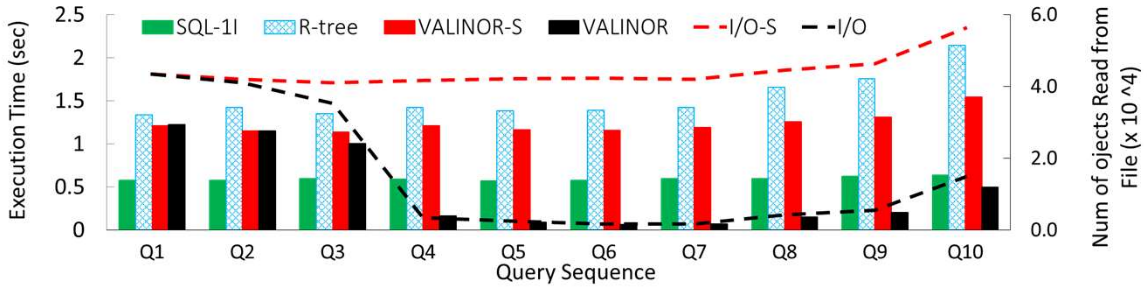
Execution Time for Basic Visual Operations

Synthetic Data



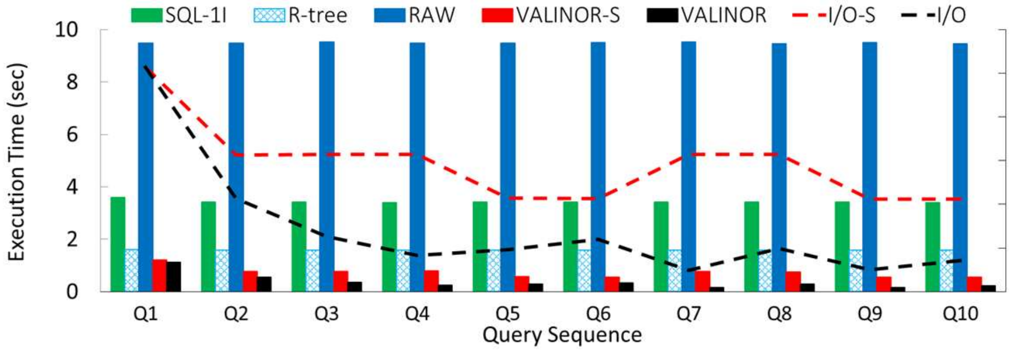
Index Adaptation

Read Data (YAHOO)

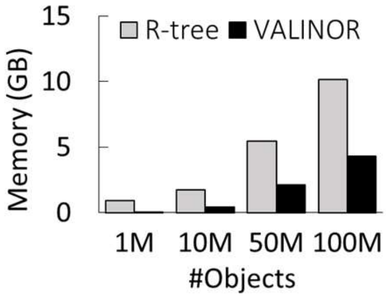


Index Adaptation

Synthetic Data



Memory Consumption



Conclusions

- VALINOR
  - lightweight main memory index for 2D visual exploration of large raw data files
  - Constructed on-the-fly and adapted to user operations
- Formulation of visual user interactions as query operators on VALINOR
- Experimental evaluation using real & synthetic datasets

  our technique outperforms competitors both in execution time and memory consumption.

Thank you!

http://www.cs.uoi.gr/~pvassil/projects/ploigia/