Computing Aggregate Queries in Raster Image Databases Using Pre-Aggregated Data

Angélica García Gutiérrez, Peter Baumann *

Abstract— **Computing multidimensional aggregates in raster image databases is a challenging task for geo-raster applications. In this paper, we exploit the techniques in OLAP to speed up aggregate query processing in raster image databases. We focus on a special case of the aggregation problem: computation of the basic aggregation functions *add, average, count, maximum,* and *minimum*. Experimental evaluation shows that the application of the pre-aggregation framework and the resulting algorithms give much better performance compared to straightforward methods.**

Keywords: Aggregation, Query-Processing, Raster Databases

1 Introduction

Remote sensing data has proved to be useful in describing information about the Earth's surface and atmosphere. It is often collected by aircraft and satellites in form of images. These images are taken at regular time intervals and cover large areas of the globe, thus allowing a practical tracking of developments such as deforestation, desertification, environmental contamination, and natural hazards. Moreover, comparisons of satellite images from different times make these phenomena easier to understand. Hence, the nature of raster image data is often multidimensional: it can include 3-D image time series (x/y/t), 3-D exploration data (x/y/z), and 4-D climate models (x/y/z/t), to name few examples. Notably, the long-term availability of remote sensing data is indispensable to many types of analysis in Geo-applications, e.g., to study water, energy, and mineral resource problems; therefore, archives of raster image data may surpass the Petabyte size.

Typically, raster image data is stored in the database by following a so called *tiling* process. That is, the image is decomposed into a set of tiles that are efficiently indexed and stored in the database. This allows that only the tiles affected by a query are retrieved from disk during query execution. In raster image databases, *aggregation* is a very important statistical concept used to represent a set of items by a single value, or to classify them into groups while determining a value for each group. The most widely used aggregations are *add, average, minimum* and *maximum*. Notably, performing aggregate operations on large volumes of multidimensional raster data pose several challenges to existing raster database technology. That is, while delivering small-sized results, large data portions have to be touched during query evaluation. In application domains such as Geographic Information Systems (GIS), the computation of many fundamental operations requires the usage of aggregate functions [3] but to the best of our knowledge, pre-aggregation support has been provided for one single operation, namely, *scaling*(zooming). In a different application domain, most On-Line Analytical Processing (OLAP) systems adopt a *pre-aggregation* approach for ensuring adequate response time during data analysis. Based on the observation that data structures and operations in both application domains share similarities, our approach for speeding up aggregate operations consists in taking existing knowledge of OLAP pre-aggregation as a basis for defining an intelligent pre-aggregation scheme in raster image databases. OLAP pre-aggregation addresses three main problems: View Selection, Query Rewriting and View Maintenance. Despite the high maturity of these technologies [4], it has never been attempted, to the best of our knowledge, to adapt them to the field of raster image databases.

In this paper, we focus on the problem of answering a query in the presence of pre-aggregated data. We present a pre-aggregation framework along with a cost model to study the effect of using pre-aggregation in the computation of aggregate queries in multidimensional raster image databases.

The remainder of the paper is organized as follows. In Section 2 we present the pre-aggregation framework. Section 3 presents the cost-model used to estimate the cost of using pre-aggregated data for computing aggregate queries. Section 4 describes the implementation of the framework in a raster database management system. In Section 5 we present experimental results on real-life raster image datasets. Finally, Section 6 presents the conclusions and a brief description of our current and future work.

*The work of Angélica García is supported by a grant of El Consejo Nacional de Ciencia y Technologia (CONACYT), scholarship 175628, by the German Academic Exchange Service (DAAD) and by Jacobs University Bremen.
Jacobs University Bremen, School of Electric Engineering and Computer Science, College Ring 1, 28759 Bremen, Germany. Email: a.garciagutierrez@jacobs-university.de

2 Framework

Hereby we describe the general framework used in our study on computing aggregate queries in multidimensional raster image databases.

2.1 Aggregation

2.1.1 Aggregate Functions

The queries that we consider can contain arbitrary aggregation functions. An aggregation function is formally defined as follows.

Definition 2.1 *An aggregation function maps a multiset of cell values in a dataset D to a single scalar value.*

We consider the aggregation functions *count* which for a multiset returns the number of cells; *add*, and *avg*, which return the sum and average, respectively, of the cell values of a multiset; *max*, which returns the maximum value among the cells of a multiset; and *min* which returns the minimum value among the cells of a multiset.

2.1.2 Aggregate Queries

An *aggregate query* $\mathcal{Q}$ is an aggregate operation whose query predicate may contain a multidimensional spatial component, namely *spatial domain* (sdom). We use the following notation to express a spatial domain:

$$sdom = [l_1 : h_1, \ldots, l_d : h_d] \tag{1}$$

where the functions l (low) and h (high) deliver lower and upper bound vectors respectively. We consider queries following the *select-from-where* paradigm without nested statements. In our implementation, we are using rasdaman as a raster database management system that offers a declarative interface, *rasql*. rasql is an SQL-based query language for multidimensional raster databases based on Array Algebra [2].

2.2 Pre-Aggregation

The term *pre-aggregation* refers to the process of precomputing and storing the results of aggregate queries for subsequent use in the same or similar requests.

Definition 2.2 *The pre-aggregated relation $\mathcal{P}$ is a relation with schema $\mathcal{P}(p_{id}$,aggregateOperation, subOperation, result, spatialDomain), whose tuples (pre-aggregates) consist of a set of pre-aggregated queries $p_1, p_2, \ldots, p_n$.*

The p_{id} attribute of $\mathcal{P}$ joins the child relations *selection*, and *intervals*. The relation *selection* has the schema $selections(p_{id}, sel\_conditions)$, whereas the relation *intervals* has the schema $intervals(p_{id}, sdom)$.

2.3 Query and Pre-Aggregate Equivalence

Definition 2.3 *An aggregate query $\mathcal{Q}$ and a pre-aggregate p_i are equivalent if and only if all the following conditions hold true.*

1. The aggregate operation of the query $\mathcal{Q}$ is the same as the aggregate operation defined for the pre-aggregate p_i.
2. The aggregate operation of the query $\mathcal{Q}$ and the pre-aggregate p_i must be applied over the same raster objects.
3. The same logical and boolean conditions, if any, apply for both the query and the pre-aggreate.
4. In aggregate operations over a specific spatial domain of an object, the extent of the spatial domain for both the query $\mathcal{Q}$ and the pre-aggregate p_i is identical.

The decision of whether to use a pre-aggregate or not in answering a query is influenced by the structural characteristics of the query and the pre-aggregate. That is, by comparing the query tree structures between the pre-aggregate and the input query, one can determine if the pre-aggregated result contributes *fully* or *partially* to the answer of the query. In the case of partial-matching, multiple pre-aggregates could be considered for answering a query and further analysis should be carried out to determine which of all candidate pre-aggregates compensate the increased overhead. To this end, we distinguish the following types of pre-aggregates: *inner*, *overlapped*, and *dominant*.

2.3.1 Inner Pre-Aggregates (IPAS)

Definition 2.4 *A set of pre-aggregates is called inner if the spatial domain of $\mathcal{Q}$ contains the spatial domain (sd) of the pre-aggregates. For simplicity, we assume the pre-aggregates do not intersect with each other.*

$$\text{IPAS} = \{p_1, \ldots, p_n \mid p_{i.sd} \subseteq \mathcal{Q}_{.sd},\, p_{i.sd} \cap p_{j.sd} = \emptyset\}\,. \tag{2}$$

2.3.2 Overlapped Pre-Aggregates (OPAS)

Definition 2.5 *A set of pre-aggregates is called overlapped if there exist an intersection between the spatial domain (sd) of $\mathcal{Q}$ and those of the pre-aggregates.*

$$OPAS = \{p_1, p_2, \ldots, p_n \mid p_{i.sd} \cap \mathcal{Q}_{.sd} \neq \emptyset\}\,. \tag{3}$$

2.3.3 Dominant Pre-Aggregates (DPAS)

Definition 2.6 *A set of pre-aggregates is called dominant if the spatial domain (sd) of $\mathcal{Q}$ is contained in the spatial component of the pre-aggregates.*

$$DPAS = \{p_1, p_2, \ldots, p_n \mid \mathcal{Q}_{.sd} \subseteq p_{i.sd}\}. \quad (4)$$

Moreover, given an *ascendant-ordered* DPAS

$$\text{DPAS}_{asc} = \{p_1, p_2, \ldots, p_n \mid \mathcal{Q}_{.sd} \subseteq p_{1.sd} \subseteq \ldots \subseteq p_{n.sd}\}, \quad (5)$$

the *closest dominant pre-aggregate* (p_{cd}) to $\mathcal{Q}$ is given by p_1, i.e., $p_{cd} = p_1$. Note that dominant pre-aggregates are only considered for answering queries using the aggregate functions: *add, count,* and *average.*

3 Cost Model

In this section, we introduce a cost model that allows to estimate the cost (in terms of execution time) of computing a query considering the presence of pre-aggregates as well as from raw data. The cost is driven by the number of disk I/Os required and memory accesses. These parameters are influenced by the number of tiles that are required to answer the query as well as by the number and size of the cells in the datasets. We assume that it takes the same time to retrieve a tile from disk as it will take to retrieve any other tile.Similarly, we consider the time taken to access a given cell (pixel) on main memory to be the same as for any other cell.

3.1 Computing a Query from Raw Data

The cost of computing an aggregate query $\mathcal{Q}$ (or sub-partitions of pre-aggregates) from raw data, (C_r), is given by

$$C_r = \sum_{i=1}^{Ntiles} C_{acc}(tile_i) + \sum_{i=1}^{Ncells} C_{agg}(cell_i) \quad (6)$$

where C_{acc} is the cost of retrieving a tile required to answer $\mathcal{Q}$, and C_{agg} is the cost of accessing and aggregating the cells corresponding to the spatial domain of the query.

3.2 Computing a Query using Inner Pre-aggregates

The cost of answering an aggregate query considering the use of inner pre-aggregated results is given by

$$C_{IPAS} = C_{fin}(\mathcal{Q}, P) + \sum_{i=1}^{|IPAS|} C_{acc}(p_i) + C_{SP}, \quad (7)$$

where C_{fin} is the cost of finding the pre-aggregates $\in IPAS$ in the pre-aggregated relation P, C_{acc} is the accumulated cost of retrieving the results of the pre-aggregates, and C_{SP} is the cost of decomposing the query $\mathcal{Q}$ into a set of sub-partitions and aggregating each from raw data.

3.3 Computing a Query using Overlapped Pre-aggregates

The cost of answering an aggregate query by using overlapped pre-aggregated results is given by

$$C_{OPAS} = C_{fin}(\mathcal{Q}, P) + \sum_{i=1}^{|OPAS|} C_{dec}(p_i) + \sum_{i=1}^{|S|} C_r(s_i) + C_{SP}, \quad (8)$$

where C_{fin} is the cost of finding the pre-aggregates $\in OPAS$ in the pre-aggregated relation P, C_{dec} is the cost of decomposing the spatial domain of each pre-aggregate into a set of sub-partitions S such that the spatial domain of the partitioned pre-aggregate corresponds to $p_{i.sdom} - (p_{i.sdom} \cap \mathcal{Q})$, C_r is the cost of aggregating each resulting sub-partition $s_i \in S$ from raw data, and C_{SP} is the cost of decomposing the query $\mathcal{Q}$ into a set of sub-partitions and aggregating each from raw data.

3.4 Computing a Query using Inner and Overlapped Pre-aggregates

The cost of answering an aggregate query considering the use of inner and overlapped pre-aggregated results is given by

$$C_{IOPAS} = C_{IPAS} + C_{OPAS} + C_{SP}, \quad (9)$$

where C_{IPAS} and C_{OPAS} are the costs of retrieving the results of inner and overlapped pre-aggregates, respectively, and C_{SP} is the cost of decomposing the query $\mathcal{Q}$ into a set of sub-partitions and aggregating each from raw data.

3.4.1 Cost of inner pre-aggregates

The cost of retrieving the results of inner pre-aggregates (C_{IPAS}) is given by

$$C_{IPAS} = C_{fin}(\mathcal{Q}, P) + \sum_{i=1}^{|IPAS|} C_{acc}(p_i) \quad (10)$$

where C_{fin} is the cost of finding the pre-aggregates $\in IPAS$ in the pre-aggregated relation P, and C_{acc} is the accumulated cost of retrieving the results of the pre-aggregates.

3.4.2 Cost of overlapped pre-aggregates

The cost of retrieving the results of overlapped pre-aggregates (C_{OPAS}) is given by

$$C_{OPAS} = C_{fin}(\mathcal{Q}, P) + \sum_{i=1}^{|OPAS|} C_{dec}(p_i) + \sum_{i=1}^{|S|} C_r(s_i) \quad (11)$$

where C_{fin} is the cost of finding the pre-aggregates $\in OPAS$ in the pre-aggregated relation P, C_{dec} is the cost of decomposing the spatial domain of each pre-aggregate into a set of sub-partitions S such that the spatial domain of the partitioned pre-aggregate corresponds to $p_{i.sdom} - (p_{i.sdom} \cap \mathcal{Q})$, and C_r is the cost of aggregating each resulting sub-partition $s_i \in S$ from raw data.

3.4.3 Cost of aggregating sub-partitions of the query

The cost of aggregating each sub-partition in SP is given by

$$C_{SP} = C_{dec}(\mathcal{Q}) + \sum_{i=1}^{|SP|} C_r(s_i), \tag{12}$$

where C_{dec} stands for the cost of decomposing $\mathcal{Q}$ into a set SP of sub-partitions, and C_r is the cost of aggregating each resulting sub-partition $s \in SP$ from raw data. Note that C_{dec} is influenced by the costs of accessing the tiles required to aggregate each of the sub-partitions, and by the cost of accessing the spatial properties of the pre-aggregates in IPAS and OPAS.

3.5 Computing a Query using a Dominant Pre-aggregate

The cost of computing an aggregate query $\mathcal{Q}$ by using a dominant pre-aggregate is given by

$$C_{DPAS} = C_{DP}(\mathcal{Q}, P) + C_{agg}(p_{cd}), \tag{13}$$

where C_{DP} is the cost of finding the pre-aggregates $\in$ DPAS in the pre-aggregated relation P and the cost of finding the closest dominant pre-aggregate p_{cd}, whereas C_{agg} is the cost of computing the aggregate difference of p_{cd} corresponding to $p_{cd.sdom} - \mathcal{Q}_{.sdom}$.

3.5.1 Cost of the closest-dominant pre-aggregate

The cost of aggregating sub-partitions of the closest dominant pre-aggregate, C_{agg}, can be calculated as follows

$$C_{agg}(p_{cd}) = C_{dec}(p_{cd}) + \sum_{i=1}^{|SP|} C_r(s_i), \tag{14}$$

where C_{dec} is the cost of decomposing p_{cd} into a set SP of sub-partitions, and C_r is the cost of aggregating each resulting sub-partition $s \in SP$ from raw data.

4 Implementation

This section describes the application of the pre-aggregation framework in a raster-database management system, namely, rasdaman. The query processing module of rasdaman was extended with the pre-aggregation framework, and it has been implemented as part of the optimization and evaluation phases.

The QUERYCOMPUTATION procedure returns either the result R or an execution plan for a given query $\mathcal{Q}$. The input of the algorithm is the query tree $\mathcal{Q}_t$ corresponding to a given aggregate query. The algorithm first verifies if there is a PERFECT-MATCHING between the input query and the pre-aggregated queries. If it finds a perfect matching then it returns the result retrieved from the pre-aggregated relation. Otherwisse, the algorithm searches for a PARTIALMATCHING between the query and the pre-aggregates allowing only to differ in the values for the spatial domain. If a partial-matching is found, then the decision of whether to answer the query from pre-aggregated results or from raw data will be driven by the plan that results with minimum cost. The algorithm makes use of the following auxiliary procedures.

- DECOMPOSEQUERY($\mathcal{Q}_t$) examine the nodes of the query tree $\mathcal{Q}_t$ and generates a standardized representation S_{qt} that can be treated via SQL statements.
- PERFECTMATCHING(S_{qt}) compares a standardized representation of the query tree S_{qt} against existing pre-aggregates saved in a relation P. The output of this procedure is the corresponding key of the matched pre-aggregate, if found.
- FETCHRESULT(key) retrieves the result R of the pre-aggregate identified with key.

Algorithm 1 QUERYCOMPUTATION

Require: A query tree Q_t and a relation P with k number of pre-aggregated queries.

```
1: initialize R = 0, i = 0, flag = false
2: S_qt = decomposeQuery(Q_t)
3: while i ≤ k ∧ !flag do
4:    key = perfectMatching(S_qt, p_i)
5:    if key then
6:       R = fetchResult(key)
7:       return R;
8:    end if
9:    i++
10: end while
11: if !flag then
12:    plan = partialMatching(S_qt)
13:    return plan;
14: end if
```

The algorithm PARTIALMATCHING identifies aggregate nodes in a query tree $\mathcal{Q}_t$ that may be substituted by pre-aggregated nodes. It first identifies an *aggregate* sub-expression in the given query tree and then it searches for pre-aggregates that fulfill conditions 1, 2 and 3 but not condition 4 for query equivalence. That is, it considers the usage of pre-aggregates that may only contribute *partially* to the answer of a query sub-expression.

Upon evaluating the cost of using the different candidate pre-aggregates, the algorithm returns the plan with the cheapest cost for the computation of the query.

Algorithm 2 PARTIALMATCHING

Require: A standardized query tree $\mathcal{Q}_t$ with n number of nodes.

1: initialize $IPAS, OPAS, DPAS = \{\}, plan = "raw"$
2: **for** each node n of $\mathcal{Q}_t$ **do**
3: **if** $aggregateOp(node[n])$ **then**
4: $\mathcal{Q}' = getSubtree(\mathcal{Q}_t, node[n])$
5: $op = getOperation(\mathcal{Q}')$
6: $ro = getRasterObject(\mathcal{Q}')$
7: $sd = getSpatialDomain(\mathcal{Q}')$
8: $result = findPreaggregate(op, ro, sd)$
9: **if** $result = \Phi$ **then**
10: $IPAS = findIpasPreaggregates(op, ro, sd)$
11: $OPAS = findOpasPreaggregates(op, ro, sd)$
12: $DPAS = findDpasPreaggregates(op, ro, sd)$
13: **end if**
14: $plan = selectPlan(\mathcal{Q}', IPAS, OPAS, DPAS)$
15: **end if**
16: **end for**
17: **return** *plan*;

The *aggregateOp*() procedure compares a node n of a given query tree $\mathcal{Q}_t$ against a list of pre-defined aggregate operations, e.g, *add_cells*, *count_cells*, *avg_cells*, *max_cells*, and *min_cells*. If the node matches any of the aggregate operations then it returns a true value.

The *getSubtree*() procedure receives as parameter a query tree $\mathcal{Q}_t$ and a pointer to an aggregate node. If the aggregate node has children, then it creates a subtree $\mathcal{Q}'$ where the root node corresponds to the aggregate node.

The $findPreaggregate()$ procedure receives as parameters an aggregate operation *op*, a raster object identification *ro*, and a spatial domain *sd*. Then it submits an SQL query with these parameters against the pre-aggregated relation P to search for a pre-aggregate that matches such parameters. If found, it returns the result of the matched pre-aggregate.

The $findIpasPreaggregates()$ procedure receives as a parameter a subtree $\mathcal{Q}'$ and verifies if there are pre-aggregates that satisfy conditions 1, 2 and 3 of equivalence among a query and pre-aggregates. For these pre-aggregates, it identifies those whose spatial domain are contained in the spatial domain of the query (inner pre-aggregates). The output of this procedure is the set of the identified pre-aggregates.

The $findOpasPreaggregates()$ procedure receives as a parameter a subtree $\mathcal{Q}'$ and verifies if there are pre-aggregates that satisfy conditions 1, 2 and 3 of equivalence among a query and pre-aggregates. For these pre-aggregates, it identifies those whose spatial domain intersects with the spatial domain of the query (overlapped pre-aggregates). The output of this procedure is the set of the identified pre-aggregates.

The $findDpasPreaggregates()$ procedure receives as a parameter a subtree $\mathcal{Q}'$ and verifies if there are pre-aggregates that satisfy conditions 1, 2 and 3 of equivalence among a query and pre-aggregates. For these pre-aggregates, it identifies those whose spatial domain dominate the spatial domain of the query (dominant pre-aggregates). The output of this procedure is the set of the identified pre-aggregates.

The *selectPlan*() procedure receives as parameters a subquery tree $\mathcal{Q}'$, a set of inner pre-aggregates $IPAS$, a set of overlapped pre-aggregates $OPAS$ and a set of dominant pre-aggregates $DPAS$. It calculates the costs of answering the query considering the different types of pre-aggregates as well as from raw data. The output of this procedure is an indicator of the best plan for executing the query.

4.1 Query Evaluation

The task of the query optimizer module is completed by providing an optimized query tree along with the plan suggested for the computation of the query to the final phase, evaluation. Typically, the evaluation phase will identify the tiles affected by an aggregate query and then it will proceed to execute the aggregate operation on each tile, and finally it will combine the results accordingly to generate the answer of the query. With the extension of pre-aggregation in the optimizer, the traditional process differs in such a way that the selected plan is considered before proceeding to execution. If the plan corresponds to *raw*, then the computation of the query will be entirely done from raw data. Otherwise, it will execute the aggregated operation only on those tiles for which there are not pre-aggregated results.

5 Experimental Results

In this section, we present the performance of our algorithms on real-life raster image datasets. These experiments were performed on a Intel Pentium 4 -CPU 3.00 Ghz PC running SuSe Linux 9.1. The workstation had a total physical memory of 512 MB. The datasets were stored in a raster database management system, rasdaman.

The experiments consisted of a set of queries run 200 times against the database, shown in Table 1. Note that the formulation is according to rasql, the declarative query interface provided by rasdaman. We performed a warm test, that is, the queries were run sequentially. The dataset consist of several raster objects each distinguished with and object identifier (*oid*). We performed the experiments on a dataset called *blacksea*, a two-dimensional

raster object of 260 Mb size that represents a portion of the Black Sea. The object consists of 100 indexed (B-tree) tiles. We created 24 pre-aggregates that could be used for answering the queries in our experiment. They are stored in a relation that contains a total of 5000 pre-aggregates.

Note that the values of the spatial domain in the queries were purposely chosen such that we could measure the impact of using pre-aggregation in the following scenarios.

- The computation of *some* of the tiles needed to answer queries *Q1, Q2* and *Q3* could be omitted by rather using the result of existing pre-aggregates.
- The computation of *all* tiles needed to answer queries *Q4, Q5* and *Q6* could be omitted by rather combining the result of two or more pre-aggregates.
- The computation of *all* the tiles needed to answer queries *Q7, Q8* and *Q9* could be omitted by rather using the result of one pre-aggregate. That is, there exists a perfect-matching between the query and one of the pre-aggregates.

Table 1: Database and queries of the experiment.

Q_id	Description
Q1	select add_cells(y[6000:10000, 29000:32000])
	from blacksea as y where oid(y) = 49153
Q2	select add_cells(y[7000:10000, 29000:31000])
	from blacksea as y where oid(y) = 49153
Q3	select add_cells(y[6700:10000, 28000:30000])
	from blacksea as y where oid(y) = 49153
Q4	select add_cells(y[7680:8191, 29000:31000])
	from blacksea as y where oid(y) = 49153
Q5	select add_cells(y[8704:9215, 29000:31000])
	from blacksea as y where oid(y) = 49153
Q6	select add_cells(y[9728:10000, 29000:31000])
	from blacksea as y where oid(y) = 49153
Q7	select add_cells(y[7680:8191, 29696:30207])
	from blacksea as y where oid(y) = 49153
Q8	select add_cells(y[8704:9215, 30720:31000])
	from blacksea as y where oid(y) = 49153
Q9	select add_cells(y[9216:9727, 30208:30719])
	from blacksea as y where oid(y) = 49153

Table 2: Comparing query evaluation costs using pre-aggreated data and purely raw data.

Q_id	#aff. tiles	#preagg. tiles	t_pre	t_ex	ratio
Q1	63	24	15.6	17.8	87%
Q2	35	24	6.9	9.3	74%
Q3	35	8	9.4	10	94%
Q4	5	5	1.02	1.55	65%
Q5	5	5	1.1	1.63	67%
Q6	5	5	0.74	1.01	73%
Q7	2	1	0.04	0.41	9%
Q8	2	1	0.04	0.45	8%
Q9	2	1	0.04	0.41	9%

Table 2 compares the CPU cost required for the computation of the queries by using pre-aggregated data and purely raw data. The CPU cost was obtained by using the *time* library of C++. The column *#aff. tiles* shows the number of tiles that would need to be considered for computing the given query, whereas *# preagg. tiles* represents the number of pre-aggregates that can be used to compute the query. The column *t_pre* shows the total CPU cost of computing the query considering pre-aggregated data. Finally, the column *t_ex* shows the time taken to execute the query entirely from raw data. The *ratio* column shows that the CPU time is always better when the computation of the queries considers pre-aggregated data.

6 Conclusions and Future Work

We have presented a framework for computing aggregate queries in raster image databases using pre-aggregated data. We distinguished among different types of pre-aggregates: inner, overlapped, and dominant. We showed that such a distinction is useful to identify those pre-aggregates, from all potential candidates, that can reduce the CPU cost required for the computation of the query. We proposed a cost-model to estimate the cost of using different pre-aggregates and thus select the best plan for evaluating the query in the presence of pre-aggregated data. The measurements on a real-life raster image dataset showed that the computation of the queries is always faster with our algorithms over straightforward methods. We focused on queries using the basic aggregate functions. This covers a large number of operations in Geo-raster applications but it remains the challenge to support more complex aggregate operations, e.g., scaling and edge-detection, which are under current investigation within our group.

References

[1] E. Adelson, C. Anderson, J. Bergen, P. Burt, and J. Odgen. Pyramid methods in image processing. *RCA Engineer*, (29-6):9, November/December 1984.

[2] P. Baumann. A database array algebra for spatio-temporal data and beyond,. *The Fourth International Workshop on Next Generation Information Technologies and Systems (NGITS)*, (Lecture Notes on Computer Science 1649, Springer Verlag):76–93, July 5-7 1999.

[3] A. Garcia-Gutierrez and P. Baumann. Modeling geo-raster operations with array algebra. In *Seventh International Conference on Data Mining Workshops Proceedings*, pages 607–612. ICDMW, 2007.

[4] A. Gupta and I. S. Mumick. *Materialized Views - Techniques, Implementations, and Applications*. The MIT Press, Cambridge, Massachusetts, 2001.