Hardware Acceleration in Commercial Databases: A Case Study of Spatial Operations*

Nagender Bandi† Chengyu Sun‡ Divyakant Agrawal† Amr El Abbadi†

†University of California, Santa Barbara {nagender, agrawal, amr}@cs.ucsb.edu
‡California State University, Los Angeles {csun}@cs.calstatela.edu

Abstract

Traditional databases have focused on the issue of reducing I/O cost as it is the bottleneck in many operations. As databases become increasingly accepted in areas such as Geographic Information Systems (GIS) and Bio-informatics, commercial DBMS need to support data types for complex data such as spatial geometries and protein structures. These non-conventional data types and their associated operations present new challenges. In particular, the computational cost of some spatial operations can be orders of magnitude higher than the I/O cost. In order to improve the performance of spatial query processing, innovative solutions for reducing this computational cost are beginning to emerge. Recently, it has been proposed that hardware acceleration of an off-the-shelf graphics card can be used to reduce the computational cost of spatial operations. However, this proposal is preliminary in that it establishes the feasibility of the hardware assisted approach in a stand-alone setting but not in a real-world commercial database. In this paper we present an architecture to show how hardware acceleration of an off-the-shelf graphics card can be integrated into a popular commercial database to speed up spatial queries. Extensive experimentation with real-world datasets shows that significant improvement in the performance of spatial operations can be achieved with this integration. The viability of this approach underscores the significance of a tighter integration of hardware acceleration into commercial databases for spatial applications.

1 Introduction

The cost of a DBMS query consists of two factors: *I/O cost*, the time spent in loading the data from the secondary storage into the main memory, and *computational cost*, the time spent by the DBMS in processing this data and returning the result. Traditionally, database research has focused on reducing I/O cost during query processing as it is the major bottleneck in many operations.

With the increased acceptance of databases in various areas such as Geographic Information Systems (GIS) and Bio-informatics, commercial DBMS, which historically handled simple data types like numbers and alpha-numeric characters, need to support complex data types. These new data types and the associated operations present new challenges for DBMS researchers. One such case is the support for efficient storage and retrieval of spatial data, which typically consists of large complex datasets representing real world GIS and CAD information.

Spatial database queries are typically evaluated in two steps: the *filtering* step and the *refinement* step. In the *filtering* (also referred to as *primary filtering*) step, the Minimum Bounding Rectangles (MBRs) of the objects and spatial indexes such as R-tree [9] are used to quickly determine a set of candidate results. In the *refinement* step (also referred to as *secondary filtering* step), the final results are determined by retrieving the actual geometries of the candidates from the database, and comparing them to either a query geometry or to each other. For complex geometries such as polygons, the cost of the *secondary filtering*

*This work is supported by the NSF grants under IIS 02-20152 and EIA 00-80134

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

step usually dominates the query cost due to the complexity of the underlying computational geometry algorithms in this step. The ratio of the *computational cost* of comparing the geometries to the *I/O cost* for loading the geometries varies significantly depending on the type of spatial queries and the complexity of the geometries, which can be roughly characterized by the number of vertices of a geometry. Generally speaking, the more complex the data, the higher the *computational cost*. For instance, a recent study on spatial selections [13] shows that for point geometries, the *I/O cost* is the dominant factor, but for polygon geometries, both costs are significant. In the case of a spatial join, the *computational cost* could be several orders of magnitude higher than the I/O cost. This is because once a geometry is loaded, it is buffered in the main memory and compared to many other geometries.

In order to reduce the *computational cost* of the *secondary filtering* step, various *intermediate filtering* techniques have been proposed. For intersection queries, Brinkhoff et al. [4] proposed using simple geometries such as convex hulls, n-corners, and maximum enclosing rectangles to approximate complex polygons. These simple geometries serve as an *intermediate filtering* step, in addition to the MBR filtering, and can identify a significant number of false positive hits without performing the costly geometry-geometry comparison. Recent work has proposed several tiling based intermediate filters [24, 2, 13], which approximate polygons with rectangular tiles.

With recent advancements in the graphics hardware technology, several proposals [19, 10, 11, 1, 15, 8] have been made to use commodity Graphics Processor Unit (GPU) to speed up conventionally computation-intensive applications. It is believed that the performance enhancement reported in these works will further improve because the peak performance of graphics processors is increasing at the rate of 2.5 - 3.0 times a year: much higher than the corresponding rate for the central processing units (CPUs) [14].

In the spatial database context, Sun et al. [19] developed a hardware-assisted intersection test as an intermediate filter for spatial database operations. However, this work was preliminary in that it established the feasibility of the approach, referred to as the *hardware filter*, in a stand-alone setting and not in a real-world commercial database where different storage layouts, index structures, query plans and proprietary optimizations may effect the effectiveness of this technique.

In this paper, we show how hardware acceleration of a commodity graphics card can be integrated into a commercial DBMS. We use Oracle 9I as a representative of a commercial DBMS. We present various approaches of integration provided by Oracle's extensibility architecture [7] and discuss their suitability for integration with the hardware filter. We chose to integrate the hardware filter as an external procedure [7] and the hardware filter itself was implemented using OpenGL because OpenGL provides a generic high level interface to any graphics hardware. We built spatial query operators using the integrated hardware filter and Oracle's primary and secondary filters thus providing similar functionality as Oracle's spatial operators [16]. We provide a cost analysis of different query operators and use it for discussing the results of the experimental section where we compare the performance of our spatial query operator against Oracle's corresponding operators. Through a detailed analysis of the experimental results, we not only validate the effectiveness of the hardware filter but also intend to provide feedback which will be helpful to database designers in integration of hardware acceleration.

The main contributions of this paper are:

- We discuss various integration options provided by Oracle and develop a system framework for integrating hardware acceleration into a commercial DBMS.
- Though many techniques have been proposed to use graphics hardware for non-visualization applications, few of them [19, 1] have made the effort to compare hardware-assisted techniques with leading software solutions. To the best of our knowledge, this paper is the first work which implements and evaluates a hardware acceleration technique in a commercial database setting.
- We conducted extensive experimentation using real world datasets, indexed by both R-tree as well as fine-tuned Quadtree indexes. The performance results of spatial selection and join confirm the advantage of the hardware filter [19], even against a preprocessing filter in most cases. The analysis also gives more insights regarding filtering techniques at different stages of spatial processing in a complex commercial database environment.

The rest of the paper is organized as follows. In Section 2, we summarize the hardware-assisted intersection test proposed by Sun et al. [19]. In Section 3, we present the details of Oracle's support for extensibility. In Section 4, we propose our architectures for integration and discuss its details. Section 5 presents the details of the data and query models of *Oracle Spatial*. In Section 6, we analyze the performance of the hardware filter for spatial selection and spatial join queries. Section 7 concludes the paper.

2 Hardware Acceleration of Spatial Operations

In [19], Sun et al. propose using graphics hardware to speed up the refinement step in spatial query operations. The technique is based on the observation that most low-level algorithms used in spatial databases have been well studied by the computational geometry community. Under current computer architectures, it is unlikely that algorithmic advances will significantly reduce the cost of geometry-geometry comparison. On the other hand, the last few years have seen tremendous advances in graphics hardware technologies. Off-the-shelf graphics cards are capable of handling thousands of polygons in real time, and are widely used in computer games, 3D modeling, and virtual reality applications. Since both graphics hardware and spatial databases work on geometries such as points, lines, and polygons, and they both deal with geometric relations such as intersection and containment in a 2D or 3D space, it is only natural to exploit the computational power of graphics hardware to speed up spatial database operations.

The idea is based on the intuitive notion of intersection of two polygons in the digital domain:

1. Render the first polygon with color c_1.

2. Render the second polygon with color c_2 adding the pixel-to-pixel color values of the second polygon to the first polygon.

3. Search for overlapping pixels with color $c_1 + c_2$. If such pixels exist, these two polygons intersect.

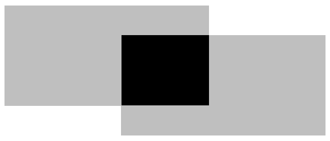

Figure 1: Hardware Intersection Test

An example of this strategy is illustrated in Figure 1, where the two polygons are rendered with color *gray*, and the overlapping pixels are *black*. However, it should be noted that a naive implementation of this approach will lead to both false hits and false dismissals due to the limited window resolution of the graphics hardware. So in [19], the hardware intersection test is implemented with a combination of a software point-in-polygon test and the rendering of only the polygon boundaries. Because of the rendering property of the anti-aliased line segments, this hardware test guarantees no false dismissals at any window resolution. However, false hits may exist because two disjoint objects might be mapped to the same pixel. Therefore, the hardware test is used as an intermediate filter as a part of the following 3-step filtering setup:

1. *Primary filtering* where MBR filtering and point-in-polygon tests are performed to determine a candidate set.

2. *Intermediate filtering* with graphics hardware to further reduce the candidate set.

3. *Secondary filtering* where a software intersection test is performed to determine the final results.

In the intermediate filtering step, which we will refer to as *hardware filtering*, the two geometries are rendered using graphics hardware, and the frame buffer is searched for overlapping pixels using the efficient *hardware minmax* test [18]. Using this hardware filter, [19] showed that significant reductions in the computational costs of spatial operations for complex queries resulted. In the experimental setup, the spatial data was stored in flat files and simple MBR comparison was used for primary filtering. In this paper, we evaluate the effectiveness of the hardware filter in a more realistic commercial database setup. In particular, we use Oracle's R-tree and Quadtree in the primary filtering step, and the geometry comparison functions in the refinement step. In the following sections, we first present how to integrate the hardware filter with Oracle Spatial, then report on the experimental setup and results.

3 Database Extensibility

In addition to the efficient and secure management of data specified by the relational model, commercial databases like Oracle and DB2 now provide support for data organized under the object model. Object types and other features such as large objects (LOBs), *external procedures*, extensible indexing and query optimization can be used to build powerful, reusable server-based components called data cartridges by Oracle [7] and Data Blades by DB2 [6]. In this paper, we use a data cartridge to integrate the hardware filter with the Oracle database engine. Through data cartridges, Oracle allows users to capture the business logic and the processes associated with domain-specific data in user-defined data types. It also allows them to build and integrate their own indexing and query optimization techniques into the database. For example, the Oracle extension for spatial data, popularly known as *Oracle Spatial*, is simply a data cartridge consisting of all the relevant spatial data types, as well as the associated indexes, functions, and operators [16].

The data types and operations encapsulated in a data cartridge can be used in user queries written in PL/SQL, which is the query language in Oracle. PL/SQL itself is a powerful programming language, but is not suitable for implementing complex algorithms due to performance reasons. For example, a numerical routine is faster when implemented in C or Java. To support such special-purpose processing, PL/SQL provides an interface for calling routines written in other languages, which makes the strengths and capabilities of third generation languages (3GL) like C and Java available through calls from a database server. Such a 3GL routine, called an *external procedure* or a *stored procedure*, is stored in a shared library, registered with PL/SQL, and called from PL/SQL at runtime to perform special-purpose processing. For instance, suppose we have a database table X with two columns A and B, and a data cartridge Y which provides operation C. A user query selecting information from X can be written as

select /* */ from X where Y.C(A, B) = value;

In the rest of this section, we discuss the details of stored procedures and external procedures, which are later used for integrating the hardware filter into Oracle.

3.1 Stored procedures

Stored procedures are implemented on the server side and are written in an interpreted language such as Java. Oracle provides an area in the database address space for the execution of the stored procedure, called the Java Virtual Machine (JVM). Since a stored procedure implemented in Java runs in the same address space as Oracle, a stored procedure invocation causes a context switch from Oracle's run-time threads to the JVM. Although both PL/SQL and Java are interpreted languages, Java is preferable over PL/SQL for implementing efficient procedures because Java has hundreds of classes, allowing interfaces with diverse functionality. Oracle does not allow C stored procedures for normal developers because unlike Java programs where program failures are handled by the JVM, C program failures can lead to database failure which is not desirable. However, Oracle allows C stored procedures for trusted developers i.e., people who write Oracle's proprietary software.

3.2 External procedures

External procedures are written in compiled languages, such as C and C++, and executed in an external address space separate from the database server. The external address space is managed by a process known as *Listener*. This separation ensures that the database server is insulated from any program failures that might occur in external procedures and, under no circumstances, is an Oracle database corrupted by such failures. But at the same time, the execution of a procedure in a separate address space implies an Inter Process Communication(IPC) overhead between Oracle and the external procedure. So implementing an algorithm as an external procedure is efficient only if the IPC overhead is insignificant with respect to the CPU cost incurred by the algorithm. The architecture of a typical external procedure is given in Figure 2

An important detail in the discussion of external procedure is the mapping of the PL/SQL data types of Oracle to the data types of the external procedure language, e.g., C. While conventional PL/SQL data types like numbers and varchar have corresponding mapping data types in C, complex data types like Large OBjects (LOB) do not have a simple mapping. These LOBs are transferred to the external procedure in a complex format and the external procedure needs to map these LOBs into its data structures using the Oracle Call Interface (OCI) [17] data manipulation callbacks. These callbacks are particularly useful for processing LOBs such as the Spatial Data Objects (SDO) [16]. For example, by using callbacks, an external procedure can perform piece-wise reads or writes of the SDOs stored in the database.

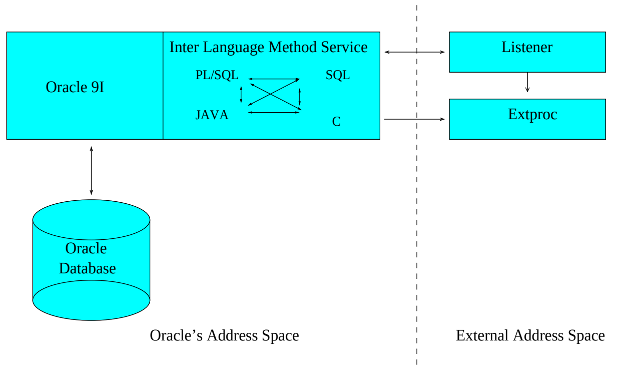

Figure 2: External Procedure - Architecture

3.3 Comparison

The decision of whether to use an external procedure or a stored procedure depends on various factors. Java class procedures are generally slower than a compiled C external procedure because of the interpreted nature of Java which makes external procedures obvious choice if speed is the main concern. If the IPC overhead is more significant than the processing cost, then stored procedures are preferable. One other factor is the amount of library support offered by the programming language. If the user wants to integrate hardware acceleration into the database, he/she would prefer to use a language like C. This is because current state of the art interfaces to access most hardware devices are more widely available in C than in Java. In particular, we use OpenGL for interacting with the graphics hardware. OpenGL provides a high level interface to the developers while hiding driver level details of the graphics hardware. OpenGL is a widely accepted standard and typically every graphics card manufacturer provides an implementation for this interface. Despite the IPC overhead drawback, we chose to integrate the hardware filter as an external procedure because C interfaces to OpenGL are ubiquitous; Java interfaces are not yet standardized and are not widely available.

4 Hardware Filter Integration

Traditional OpenGL programs are designed to run in an infinite loop waiting for interactive events to occur. Typically these event-driven programs consist of a set of *initialization* operations followed by a recurring set of *rendering* operations which handle event occurrences. The hardware filter consists of the following operations in sequential order: data retrieval, MBR filtering, point-in-polygon test, rendering the polygons and doing the minmax test. As the hardware filter requires *rendering* operations for performing the hardware test, *initialization* must be completed before the rendering operations can be performed. Since a typical spatial query would require the *rendering* operations (for the hardware tests) to be performed a number of times, it is not desirable to execute the *initialization* operations every time as these operations are very expensive. In order to avoid this *initialization* overhead, we propose to separate the process of accessing the graphics hardware from the process of data retrieval and software tests. This separation will make the hardware filter efficient because *initialization* is done only once for any query.

We present an architecture, referred to as Dual thread architecture, where we separate the hardware access and implement it as a separate thread within the address space of the external procedure. The external procedure consists of two threads (primary thread and graphics thread) running synchronously to perform the intersection test (Figure 3). The primary thread deals with the data retrieval and the initial filtering tests like the MBR test and the Point in the Polygon test. The graphics thread performs the OpenGL initializations and the hardware test. While the graphics thread is created at the beginning of the query and remains alive as long as the query runs, the primary thread is loaded in a nested manner. The two threads share data using global variables. Synchronization between the threads is done using the system calls provided by the pthread library. We describe the dual thread architecture below.

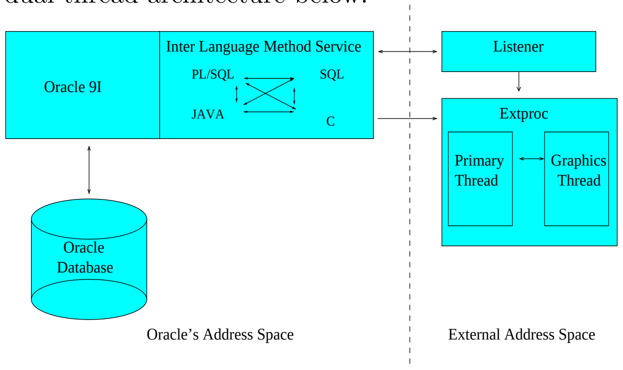

Figure 3: Dual Thread Architecture

The primary thread performs the following operations in sequential order:

1. Retrieve the data corresponding to the Spatial Data Objects (SDO) using Oracle Call Interface (OCI) callbacks into global variables.

2. Test the query polygons for Minimum Bounding Rectangle (MBR) intersection. If this test succeeds, go to step 3 else the two polygons do not intersect.

3. Test the query polygons for the Point in the Polygon condition. If this test succeeds the two polygons intersect, else go to step 4 for the hardware test.

4. Inform the graphics thread, which is waiting for a signal from the primary thread (as will be described below), so that the polygons are rendered.

5. Wait for a signal from the graphics thread. Upon receiving the signal from the graphics thread check the result of the hardware intersection test given by the graphics thread. If the result is true the software intersection test needs to be done. Otherwise the polygons do not intersect.

The graphics thread performs the following operations in sequential order:

1. Wait for the primary thread to generate the required data so that polygons can be rendered for doing the hardware test.

2. Upon receiving the signal from the primary thread, render the polygons and check for the intersection condition using hardware minmax test. Store the result of this test in a global variable and signal the waiting primary thread. Go back to step 1.

5 Oracle Spatial

As described in the previous section, Oracle Spatial [16] is an integrated set of functions and procedures that enables spatial data to be stored, accessed, and analyzed quickly and efficiently in an Oracle database. In this section, we give the details of the data and query models of Oracle Spatial.

5.1 Data Model

Oracle Spatial's data model is a hierarchical structure consisting of elements, geometries, and layers, which correspond to representations of spatial data. An element is the basic building block of a geometry. The supported spatial element types are points, line strings, and polygons. A geometry (or geometry object) is the representation of a spatial feature, modeled as an ordered set of primitive elements. A layer is a collection of geometries having the same attribute set. Each layer's geometries and the associated spatial index are stored in the database in standard tables.

5.2 Indexes and Query Model

Oracle provides support for both linear Quadtree and R-tree indexes. These indexes are implemented using the extensible framework of Oracle [7, 16]. The linear Quadtree (or Quadtree for short) computes tile approximations for the interior and boundary of geometries and uses existing B-tree indexes for performing spatial search and other operations. The R-tree indexing in Oracle is implemented logically as a tree and physically uses tables inside the database. The search involves recursive SQL from the root to the relevant leaves.

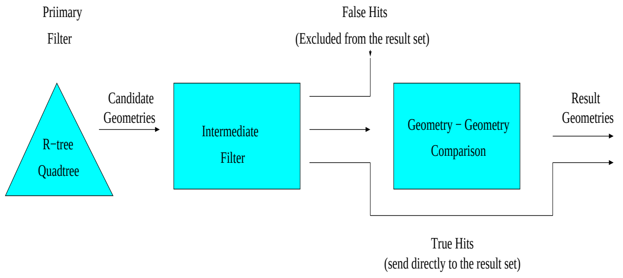

Figure 4: Oracle Spatial Query Model

Oracle Spatial uses a multi-stage query model as shown in Figure 4. In the first stage, referred to as the *primary* filter, the spatial index is used for query filtering. Candidate geometries that may satisfy a given query criterion are first identified in this stage with the help of exterior approximations in the spatial index. In the case of a Quadtree, Quadtree tiles are used as exterior approximation and in the case of an R-tree, minimum bounding rectangles (MBRs) are used. In the *intermediate* stage, candidate geometries from the primary filtering step are compared with the query geometry using a sorted list of interior tiles approximating the interior of the query geometry. This is referred to as the *interior approximation* filter [13] and is used either to accept or reject candidate geometries based on the query criterion. The rest of the geometries whose interaction are not determined in the *intermediate* filter are then passed through to the final stage, referred to as the *secondary* filter, and the exact result set is determined and returned to the user. Whereas the secondary filter uses computational geometry algorithms to determine the interaction between query and candidate geometries, the primary and intermediate filters use the exterior and interior approximations of query and data geometries (from the index).

As described above, during the intermediate filtering stage, Oracle uses the interior approximation filter [13] to reduce the candidate set for the secondary filter. If Quadtree indexing is used then interior approximations of both the query and also the data geometries are used during the interior filtering step. This is because, in the case of Quadtree indexing, the interior approximations of the datasets are calculated statically during index creation and stored inside the index tables. For queries that use R-trees, the interior approximations are calculated during run-time and hence are done only for query geometries. For R-trees, the choice of tiling level for interior approximations is decided by the Oracle run time optimizer.

5.3 Spatial Operators and Functions

In order to define a spatial query in Oracle three parameters must be defined: two for the geometries used in the spatial query, and the third for parameters defining the type of spatial query (selection or join). Given below are two spatial operators and a function provided by Oracle Spatial.

1. SDO_FILTER : This operator functions as a primary filter and returns a super-set of the actual query result.

2. SDO_RELATE : This operator performs both the primary and the secondary filtering operations and returns the exact query result. In addition to using the indexing for primary filtering, it also performs interior approximation filtering [13] before secondary filtering. This operator has two options specified by the third argument:

    - **WINDOW**: When this option is enabled, it performs a selection of the query geometry (second parameter) over the data geometries.
    - **JOIN**: When this option is enabled, it performs a spatial join of the geometry columns specified by the two parameters. Currently, Oracle allows join queries only when the geometries are indexed using Quadtrees. In the case of R-trees, it performs a nested join in the same way as a selection query. This operator requires the second column to be indexed using the same tiling level as the first column. When both data columns are indexed using Quadtrees, it performs a *hash* join during the primary filtering step followed by a direct application of the secondary filter. If the second column is not indexed, a *nested* join is performed as in the case of selection queries.

3. SDO_GEOM.RELATE : This function (not operator) applies exact computational geometry to find out the kind of interaction between two given geometries and is used as the secondary filter in SDO_RELATE.

6 Performance Evaluation

In this section, we evaluate the effectiveness of the hardware filter integrated with an Oracle database, and compare the performance of intersection queries with and without the hardware filter. We build a hardware intersection operator by applying the hardware filter to the result set produced by the primary filter (SDO_FILTER) and then apply the secondary filter (SDO_GEOM.RELATE). We call this the *hardware* operator. We also give a conservative estimate for the performance of the hardware operator, if the hardware filter is implemented right after the *primary filter* as a part of Oracle's Spatial data cartridge and not as an external procedure. This is done by deducting the IPC overhead which occurs because of transferring the data to the external procedure. We refer to the *hardware* operator without the IPC overhead as *hardware-no-ipc* operator. In the rest of this section we discuss the experimental setup, describe a theoretical cost analysis for the *hardware* operator, and analyze the performance of selection and join operations.

6.1 Experimental Setup

The experiments were performed on a desktop PC with an AMD AthlonXP 1800+ CPU and 1GB Double Data Rate (DDR) memory. The graphics card is equipped with an NVIDIA GeForce4 Ti4600 processor and 128MB on-board memory. Experiments are performed on an Oracle database (version 9.2.0.1) running on Linux Operating System. The hardware filter is coded in C++, compiled to a shared library using g++ and integrated with Oracle using the *Dual Thread* architecture.

The experiments are conducted with the following real world datasets:

- **PRISM** [5]. Average annual precipitation in the contiguous United States at 1:2,000,000 scale for the climatological period 1961-1990.
- **HYDRO** [23]. Hydrological unit boundaries for the United States, Puerto Rico and the US Virgin Islands at 1:2,000,000 scale.
- **COUNTY** [22]. The boundaries of the US counties at 1:2,000,000 scale.
- **STATES50** [20]. The boundaries of the main land boundaries of the 50 US states at 1:2,000,000 scale.
- **LSOVER** [21]. The boundaries of Landslide Incidence and Susceptibility distribution in the United States at 1:2,000,000.

Some statistics of the datasets are summarized in Table 1, where N is the number of objects in a dataset.

Dataset	N	Number of Vertices Per Polygon		
		Min	Max	Average
STATES50	50	91	70238	4416
PRISM	6243	4	45854	94
HYDRO	5348	4	12450	218
COUNTY	4933	4	10838	139
LSOVER	2814	4	91752	92

Table 1: Statistics of experimental Datasets

6.2 Operator Cost Analysis

In this subsection, we describe the cost analysis for the hardware operator and Oracle's software operator. Here, the cost refers to the total elapsed time for a spatial query. This cost analysis will provide the details of the various costs which constitute the total cost of hardware and software operators and will be used in later subsections to discuss the results.

6.2.1 Hardware operator

As described before, the hardware operator is built by applying an ordered sequence of filters: primary, hardware and secondary. It should be noted that for the primary and secondary filtering, we use Oracle's SDO_FILTER and SDO_GEOM.RELATE respectively. Since the hardware filter is implemented inside an external procedure, the total cost (t_{total}) of the hardware operator can be expressed as the sum of the costs of the following components:

1. cost of the primary filter ($t_{primary}$).
2. cost of the external procedure.
3. cost of the secondary filter ($t_{secondary}$).

The cost of the primary filtering step ($t_{primary}$) not only includes the cost of loading and using the index tables for calculating a superset of the actual result but also the cost of loading the geometries corresponding to the result set of the primary filter from secondary storage to Oracle's address space.

The cost incurred by the external procedure ($t_{extproc}$) can be partitioned into the following components.

1. cost of transferring data geometries to the address space of the external procedure ($t_{transfer}$).
2. cost of retrieval of the received data into local data structures using OCI function calls ($t_{retrieval}$).
3. cost of hardware filtering test which includes MBR test, Point-in-Polygon test, hardware test and also the thread synchronization overhead ($t_{hardware}$).

The cost of secondary filtering ($t_{secondary}$) comprises of the time for retrieval of data into internal data structures (same as $t_{retrieval}$ mentioned above) and the actual cost of comparing the data geometries using the computational geometry algorithms. It should be noted that both the external procedure and the secondary filter have to make OCI callbacks for retrieving data into local data structures before any processing. This implies that for those geometry pairs which successfully pass through the hardware filter, these OCI callbacks are made twice. This can be avoided if the external procedure is tightly integrated on the Oracle server itself as a part of the Oracle Spatial cartridge instead of being integrated as an external procedure. Since Oracle does not allow normal developers to modify its proprietary data cartridges, this overhead is currently unavoidable. Furthermore, $t_{retrieval}$ can be reduced to a great extent by efficient cache management which can be done once the hardware filter is tightly integrated into Oracle Spatial as a stored procedure. Overall, the cost of the hardware-no-ipc operator described in the beginning of this section gives a conservative estimate of the performance of the hardware operator if integrated into Oracle Spatial.

6.2.2 Software operator

We now describe the cost analysis for Oracle's software operator. The software operator is a sequence of two filters: the primary and the secondary filters and for certain operations, an interior filter in between. The cost of the primary ($t_{primary}$) and secondary ($t_{secondary}$) filters is defined in the same way as for the hardware operator. When the interior filter is used by the software operator, it uses interior approximations to filter out geometry pairs thereby reducing the processing cost of the secondary filter. The cost incurred by the interior filter ($t_{interior}$) is predominantly the time taken for contracting the interior approximations. Although a general breakdown of the total cost into primary filter and the remaining cost can be calculated, Oracle does not provide a breakdown of the remaining cost into interior and secondary filter cost. Since the interior filter is supposed to have insignificant cost [13], we assign the cost of the interior filter ($t_{interior}$) to 0.

We also add the costs of the intermediate (hardware and interior) filter and the secondary filter for both the hardware and software operators respectively and name the total cost as t_{comp}. We use t_{comp} because the total I/O cost which is incurred during the loading of the geometries is included inside the primary filter cost ($t_{primary}$). It should be noted that in the case of the hardware operator, we also add $t_{retrieval}$ to t_{comp} apart from the hardware and secondary filter costs while noting that this cost can be greatly reduced once the hardware filter is tightly integrated into Oracle Spatial. A comparison of t_{comp} for the hardware operator with that of the software operator gives a conservative estimate of the performance enhancement due to the hardware filter over the interior filter. We consider it conservative because in the case of the software operator, the interior filter does not need to retrieve the data geometries (by making OCI callbacks) for calculating the polygonal approximations (MBRs or interior approximations) as these approximations are precomputed and stored in the index structures.

6.3 Spatial Selections

In this subsection, we analyze the performance of spatial selection queries by measuring the time taken for selection queries using the following operators: SDO_RELATE, hardware and hardware-no-ipc. We refer to the selection query with the SDO_RELATE operator as *software selection* when discussing the results in this subsection. We evaluate the performance of these queries using the R-tree and Quadtree indexes. Using the boundaries of STATES50 of the United States, we perform the selection queries over three datasets, PRISM, HYDRO, and LSOVER. Based on the experimental results in [19], we chose a fixed 12x12 window resolution for the hardware filter and use it in all the experiments. In the following subsections we discuss the results for R-tree and Quadtree indexes over these data sets.

6.3.1 R-trees

The results of the R-tree selection for the above queries are shown in Figure 5. A breakdown of various costs incurred during the selection query for the hardware and the software operators are shown in the Tables 2 and 3. A comparison of the costs of primary ($t_{primary}$) and secondary filters ($t_{secondary}$) for the software operator (Table 3) shows that the primary filtering cost is minimal when compared to the secondary filtering cost. This implies that years of research efforts which focused on providing better spatial indexes have been able to reduce the primary filtering cost to the point where the secondary filtering cost becomes the bottleneck.

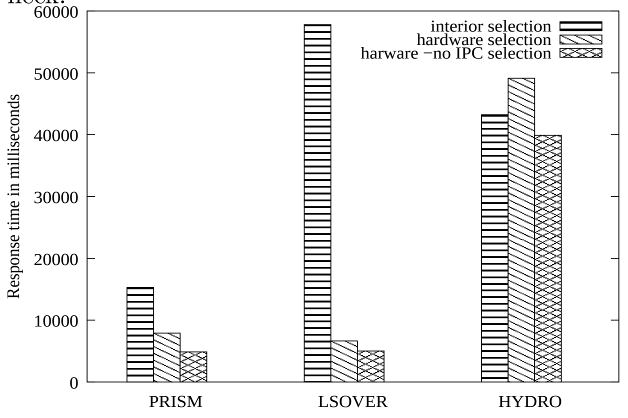

Figure 5: Selection results for R-trees

For the PRISM dataset, the hardware and the hardware-no-ipc selections improve the overall performance of the selection queries by 48.3% and 68.84% respectively. A comparison of computation cost (t_{comp}) values in Tables 2 and 3 shows that the hardware operator improves the performance of the software operator's computation cost for the PRISM dataset by 78.9%. Results for the case of selection over the LSOVER dataset show a computation cost improvement of 93% for the hardware operator over the software operator. These results also confirm the results reported in [19] where similar improvements are reported for the computation cost.

Cost parameter		PRISM	LSOVER	HYDRO
$t_{primary}$		1156	470	3000
$t_{extproc}$	$t_{transfer}$	3137	1627	9294
	$t_{retrieval}$	310	349	156
	$t_{hardware}$	304	160	1019
$t_{secondary}$		2980	3774	36291
Total Cost		7903	6647	49117
t_{comp}		3594	4444	37466

Table 2: Cost breakdown of hardware operator for R-Tree selections (milliseconds)

For the HYDRO dataset, the hardware-no-ipc selection performs marginally better than the software selection. It should be noted that although the hardware filter *can* assert when two polygons intersect by containment and also when two polygons *do not* intersect, it *cannot* assert when two polygons intersect by overlapping. In the case of datasets such as HYDRO where many data polygons have an overlapping intersection with the query polygon, the hardware operator has to go through the secondary filtering step to check for this condition in which case the hardware filter becomes an overhead. However, a close look at Table 2 indicates that in all the experiments, the hardware filtering step ($t_{hardware}$) consumes less than 3% of the total query response time in the worst case (2% for HYDRO). This implies that the hardware filter is a perfect run-time filter which can potentially enhance the performance of spatial queries without incurring any significant overhead.

For the PRISMS dataset, the large difference between hardware and hardware-no-ipc selections shows that IPC overhead ($t_{transfer}$) can account for a significant proportion of the total query cost. In the above result, $t_{transfer}$ accounts for 40% of the overall hardware operator's cost. As discussed before, the sum of the costs of IPC overhead ($t_{transfer}$) and the retrieval time ($t_{retrieval}$) gives an estimate of the amount of cost reduction that can be achieved if the hardware filter is tightly integrated into Oracle as a stored procedure. For the PRISM, HYDRO and LSOVER datasets, this cost accounts for 43%, 19% and 30% respectively of the total cost thus underscoring the need for a tighter integration.

In the current query model of Oracle Spatial, the interior filter is tightly integrated into the database where it has direct access to the polygonal approximations (MBRs) stored inside the index structures. This means that the interior filter incurs very little cost because it directly operates on the MBRs from the index and saves on the I/O and retrieval costs of loading the actual data geometries and calculating the MBRs respectively. We find this tightly integrated interior filter to be complimentary to the hardware filter because the interior filter uses the MBRs stored in the R-tree index, and reduces the I/O cost of retrieving the actual geometries, while the hardware filter operates on the geometries that are already loaded in memory, and thus reduces the computation required for the secondary filtering. So an ideal setup would integrate the hardware filter right after the interior filter inside the Oracle Spatial data cartridge.

Cost parameter	PRISM	LSOVER	HYDRO
$t_{primary}$	1156	470	3000
$t_{secondary}$	14137	57784	40235
t_{comp}	14137	57784	40235

Table 3: Cost breakdown of software operator for R-Tree selections (milliseconds)

6.3.2 Quadtrees

In this subsection, we analyze the performance of selection queries when the data geometries are indexed using Quadtrees. When Quadtrees are used, a $2^n * 2^n$ grid of tiles is used to approximate the interior and boundaries of geometries, where n is a user specified value usually referred to as the Quadtree tiling level. These interior and boundary approximations are calculated during the index creation step and stored in the index tables. Quadtree interior filtering has the advantage of using these preprocessed interior approximations during the filtering process. However, these approximations become very expensive to calculate and consume a lot of disk storage when the tiling levels become high. Timing and storage statistics of Quadtree index structures for the PRISM and HYDRO datasets shown in Figures 6 and 7 suggest an exponential growth for index creation time and disk utilization with increasing tiling levels. These statistics can be compared with the timing and storage details of the corresponding R-tree index structures in Table 4.

Intuitively, queries using Quadtree indexes have better query performance than R-tree indexes because the interior and the boundary tiles provide a better approximation of a geometry than the MBR, hence more results can be identified without accessing the geometries in the database table. However, this advantage of query performance comes at the costs of longer index creation time, larger index storage, as well as degraded update performance. Ideally, it would be desirable to have a filter which has the high performance of a preprocessing filter without incurring any preprocessing overhead.

It should be noted that the R-tree hardware filter (hardware filter using R-tree index for primary filtering) discussed in the previous subsection is absolutely run-time because the required R-tree index can be built very quickly (Table 4) and the hardware filter is inherently run-time. In the rest of this subsection, we analyze the performance of selection queries using the run-time R-tree hardware filter and compare the performance with the preprocessed Quadtree software (interior) filter.

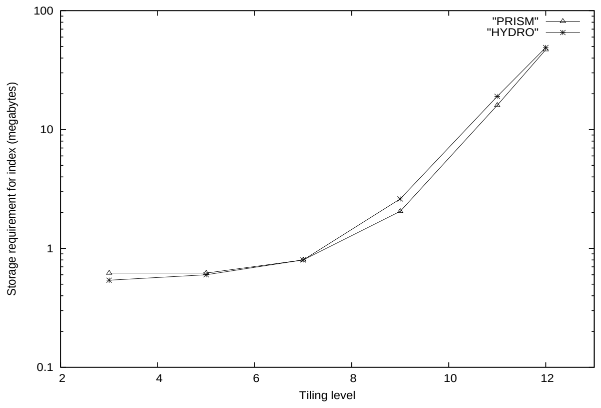

Figure 6: Quadtree index storage for PRISM and HYDRO (in log scale)

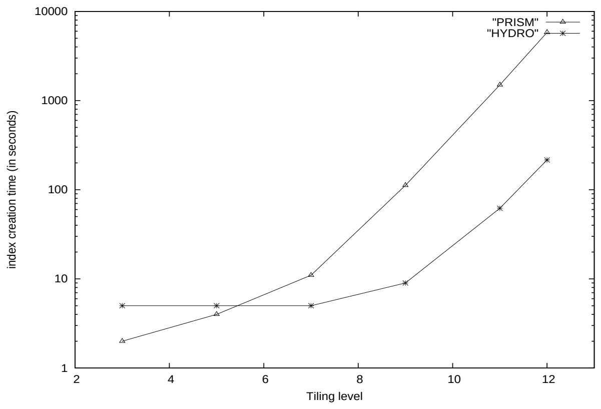

Figure 7: Quadtree index creation time for PRISM and HYDRO (in log scale)

type	PRISM	HYDRO
Index Creation time (seconds)	5	5
Index Storage cost (Megabytes)	0.5625	0.5

Table 4: Index statistics for R-Trees

The Quadtree software selection queries are performed for Quadtree tiling levels between 3 and 11. Results for selection queries over the PRISM dataset are given in the Figure 8. It can be observed that the response time for the software selection query decreases as the tiling level increases. This is because the candidate set returned by the primary filter gets increasingly refined with the increase in tiling level thus requiring less disk I/O during the ensuing filtering steps. As the tiling level is not defined for R-trees, the performance of the R-tree hardware filter appears as a straight line. For lower tiling levels, the hardware and the hardware-no-ipc selections outperform the preprocessed selection query. At higher tiling levels, the difference in the response time of hardware assisted selection queries and the preprocessed selection query decreases. This is because of the increase in effectiveness of the Quadtree interior filter due to improved approximation of interior of data geometries. But at higher tiling levels, the Quadtree indexes incur very high preprocessing cost which is not reflected in the performance cost of the selection query. These results suggest that the hardware filter coupled with the inexpensive R-tree indexing can significantly improve the performance of spatial intersection queries for complex datasets without incurring any preprocessing overhead. Results for the selection query over the LSOVER dataset (can be found in [3]) show that the R-tree hardware filter outperforms the static interior filter for all tiling levels thus supporting our argument that, the R-tree hardware filter is a perfect replacement for the Quadtree interior filter.

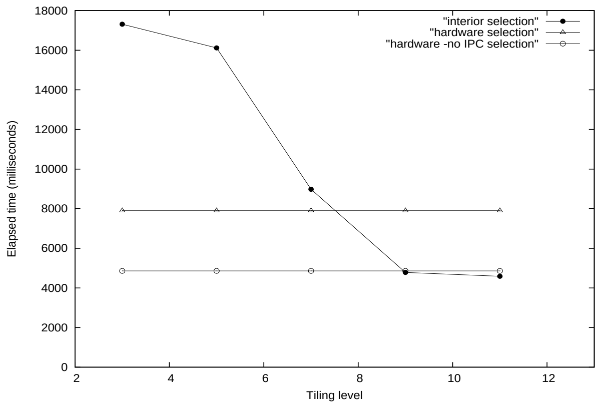

Figure 8: Timing results for selection over PRISM

Results for selection queries over the HYDRO dataset are shown in the Figure 9. As in the case of R-tree selection, the hardware-no-ipc operator performs at par with the Quadtree software operator. A closer look at the performance of the nested indexed join suggests that its query performance tends to converge to an asymptotic value. This supports our earlier argument made for the hardware filter that the intersection of certain pairs of geometries cannot be asserted by the intermediate filter (here the interior filter) and they have to go through the secondary filter, and this asymptotic value is the cost of the secondary filtering step.

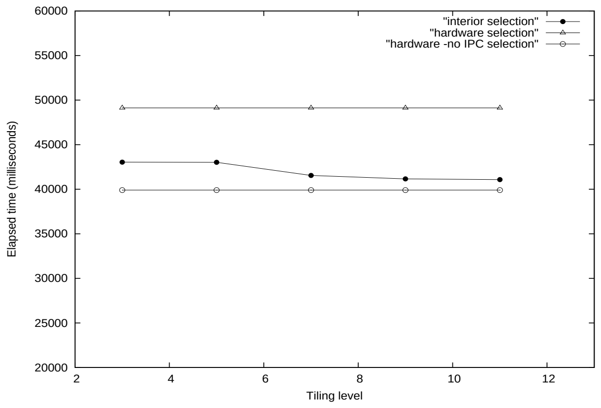

Figure 9: Timing results for selection over HYDRO

6.4 Spatial Joins

In this subsection, we analyze the performance of spatial join operations when assisted by the hardware filter. Here, we compare the performance of the *R-tree hardware join* against software joins of both R-trees and Quadtrees. For R-tree indexed cases, Oracle9i currently supports only a indexed nested join, which we refer to as *R-tree software join*. If both geometry columns are indexed using Quadtree tiles, SDO_RELATE performs a hash join during primary filtering followed by a direct application of the secondary filter. We refer to this join as the *Quadtree hash join*. If only the first column is indexed, then it performs the nested loop join using primary, interior and secondary filters. We call this the *indexed nested* join. In the rest of this subsection, we analyze the performance of a spatial join query by measuring the time taken for the following approaches: hash join, indexed nested join, hardware join and hardware-no-ipc join. We consider the join of the datasets COUNTY and HYDRO, PRISM and HYDRO followed by join on the datasets COUNTY and PRISM.

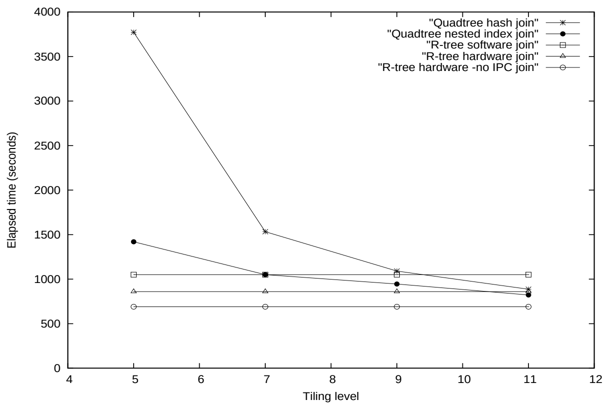

Figure 10: Timing results for join of COUNTY X HYDRO

Figure 10 shows the results of the above spatial join queries over COUNTY and HYDRO with various levels of tiling for the Quadtree indexing. These results show that the R-tree hardware-no-ipc join outperforms the R-tree software join and also both the Quadtree software joins for all the tiling levels. Among the Quadtree software joins, the indexed nested join performs better than the hash join because of the effectiveness of the interior filter which is not used in the latter. Results of the join of the PRISM and HYDRO datasets are given in Figure 11. It can be noted that at higher tiling levels, the indexed nested join performs well because the interior filter uses more accurate preprocessed approximations which identify more positive results, thereby reducing the I/O required for loading the actual geometries. These results show that although the hardware filter significantly improves the performance of Quadtree joins at lower tiling levels, the interior filter can be efficient

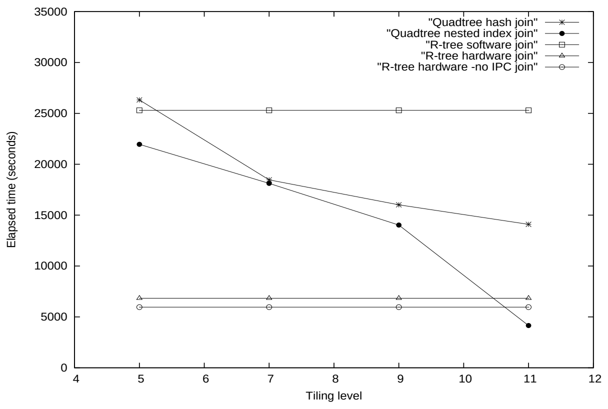

Figure 11: Timing results for join of HYDRO X PRISM

enough at higher tiling levels. However as shown in Figures 6 and 7, higher tiling levels have very expensive preprocessing and storage costs which is the reason why Oracle, in general, recommends using R-trees over Quadtrees [12]. Results for the join of COUNTY and PRISM (shown in [3]) also confirm that the R-tree hardware filter is very effective in improving the performance of joins.

7 Conclusion and Future Work

In this paper, we addressed the problem of integrating hardware acceleration into a commercial databases. We analyzed various approaches of integration provided by Oracle and integrated hardware acceleration for spatial operations as an external procedure. Using the hardware filter and the primary filter of an inexpensive R-tree index, we developed a run-time spatial intersection operator which has similar functionality as Oracle's software intersection operator. We analyzed the performance of this operator for spatial selection and join operations through extensive experimentation over real-world datasets and compared its performance with Oralce's run-time and preprocessed intersection operators. Our experimentation demonstrates that the hardware operator not only can improve the performance of the Oracle's run-time software operator significantly, but also can perform as well if not better than the preprocessed intersection operator without incurring any preprocessing and storage overhead. Since the hardware operator uses an R-tree index which has very low index update, creation and storage costs, we achieve the best of both worlds: low storage requirement and very low processing time. We also suggest that performance improvements can be expected if the hardware filter is integrated after the R-tree interior filter, as part of the DBMS itself. In the near future, we plan to explore the use of hardware filter integration for alternative complex queries.

8 Acknowledgment

We would like to thank Dr. Ravikanth Kothuri from Oracle for providing us necessary information about Oracle Spatial as well as giving valuable feedback on this work.

References

[1] P. Agarwal, S. Krishnan, N. Mustafa, and S. Venkatasubramanian. Streaming geometric optimization using graphics hardware. In *ESA 2003, Proceedings of Annual Meeting, Budapest, Hungary, September 15-20, 2003*.

[2] W. M. Badawy and W. G. Aref. On local heuristics to speed up polygon-polygon intersection tests. In *Proceedings of the 7th International Symposium on Advances in Geographic Information Systems (ACM-GIS '99)*, pages 97–102, 1999.

[3] N. Bandi, C. Sun, D. Agrawal, and A. El Abbadi. Hardware acceleration in commercial databases: A case study of spatial operations. Technical report, Computer Science Department, University of California, Santa Barbara, 2004. http://www.cs.ucsb.edu/research/trcs/docs/2004-15.pdf.

[4] T. Brinkhoff, H.-P. Kriegel, R. Schneider, and B. Seeger. Multi-step processing of spatial joins. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'94)*, pages 197–208, 1994.

[5] C. Daly and G. Taylor. *United States Average Annual Precipitation, 1961-1990.* Spatial Climate Analysis Service, OSU, 2000.

[6] Data Blades. http://www-3.ibm.com/software/data/informix/blades/, 2002.

[7] Data Cartridges. http://downloadwest.oracle.com/docs/cd/b10501_01/appdev.920/a96595/toc.htm, 2002.

[8] N. Goodnight, C. Woolley, G. Lewin, D. Luebke, and G. Humphreys. A multigrid solver for boundary value problems using programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware (SIGGRAPH'03)*, pages 102–111, 2003.

[9] A. Guttman. R-trees: a dynamic index structure for spatial searching. pages 599–609. Morgan Kaufmann Publishers Inc., 1988.

[10] K. E. Hoff III, T. Culver, J. Keyser, M. C. Lin, and D. Manocha. Fast computation of generalized voronoi diagrams using graphics hardware. In *Proceedings of the Annual Conference on Computer Graphics (SIGGRAPH'99)*, pages 277–286, 1999.

[11] K. E. Hoff III, A. Zaferakis, M. Lin, and D. Manocha. Fast and simple 2d geometric proximity queries using graphics hardware. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 145–148. ACM Press, 2001.

[12] R. K. Kothuri. Personal commnunications, 2004.

[13] R. K. Kothuri and S. Ravada. Efficient processing of large spatial queries using interior approximation. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD'01)*, pages 404–421. ACM Press, 2001.

[14] Macedonia, Michael. The gpu enters computing's mainstream. *Computer*, 36(10):106–108, 2003.

[15] K. Moreland and E. Angel. The fft on a gpu. In *Proceedings of the ACM SIGGRAPH/EURO GRAPHICS conference on Graphics hardware (SIGGRAPH'03)*, pages 112–119, 2003.

[16] Oracle Spatial 9.2.0.1. http://download-west.oracle.com/docs/cd/b10501_01/appdev.920/a96630/toc.htm, 2002.

[17] OracleCallInterface. http://download-west.oracle.com/docs/cd/b10501_01/appdev.920/a96584/toc.htm, 2002.

[18] M. Segal and K. Akeley. *The OpenGL Graphics System: A Specification (Version 1.2.1).* Silicon Graphics, Inc., April 1999.

[19] C. Sun, D. Agrawal, and A. El Abbadi. Hardware acceleration for spatial selections and joins. In *Proceedings of the ACM SIGMOD international conference on on Management of data (SIGMOD'03)*, pages 455–466. ACM Press, 2003.

[20] U. G. Survey. *State Boundaries of the United States.* U.S. Geological Survey, November 1999.

[21] U. G. Survey. *Landslide Incidence and Susceptibility distribution in United States.* U.S. Geological Survey, February 2001.

[22] J. Watermolen. *1:2,000,000-Scale County Boundaries.* U.S. Geological Survey, 2001.

[23] J. Watermolen. *1:2,000,000-Scale Hydrologic Unit Boundaries.* U.S. Geological Survey, 2002.

[24] G. Zimbrao and J. M. de Souza. A raster approximation for processing of spatial joins. In *Proceedings of 24rd International Conference on Very Large Data Bases (VLDB'98)*, pages 558–569. Morgan Kaufmann, August 24-27, 1998.