QBISM: Extending a DBMS to Support 3D Medical Images

Manish Arya[*] William Cody[*] Christos Faloutsos[*+] Joel Richardson[#] Arthur Toga[x]

[*]IBM Almaden Research Center [+]Univ. of Maryland, College Park
[#]The Jackson Laboratory [x]Univ. of California, Los Angeles

Abstract

We describe the design and implementation of QBISM (Query By Interactive, Spatial Multimedia), a prototype for querying and visualizing 3D spatial data. Our first application is in an area in medical research, in particular, Functional Brain Mapping. The system is built on top of the Starburst DBMS extended to handle spatial data types, specifically, scalar fields and arbitrary regions of space within such fields. In this paper we list the requirements of the application, discuss the logical and physical database design issues, and present timing results from our prototype. We observed that the DBMS' early spatial filtering results in significant performance savings because the system response time is dominated by the amount of data retrieved, transmitted, and rendered.

1: Introduction

The goal of the QBISM project is to study the extensions of database technology that enable efficient, interactive exploration of numerous large spatial data sets from within a visualization environment. In this work we focus on the logical and physical database design issues to handle 3-dimensional spatial data sets. We also present timing results collected from our prototype. As a first application area we have chosen the Functional Brain Mapping project. Our prototype serves as a tool medical researchers can use to visualize and to spatially query 3-d human brain scans in order to investigate correlations between human actions (e.g., speaking) and physiological activity in brain structures. The spatial techniques presented here could also be applied to other medical applications involving anatomic modelling, such as surgery or radiation treatment planning.

Many other application domains involve access to and visualization of large spatial databases. In particular, Geographic Information Systems (GIS) [26] (e.g., environmental and archeological [25] applications); scientific databases (e.g., molecular design systems); and multimedia systems [19] (e.g., image databases [20]). In these classes of applications it is essential to provide accurate and flexible data visualization as well as powerful exploration tools [5, 15].

The **scalar field** is a data type common to several of these applications. In particular, a 3-d scalar field is a collection of <x, y, z, value> tuples. In a medical database, the "value" could be a measure of glucose consumption at the <x, y, z> point in the brain as depicted in a PET[1] study; in a meteorological database, the value could be the temperature at a given point in the atmosphere; and in a chemical database, the value could be the charge at a point in a molecular model. Scalar fields can have other dimensionalities as well; for example, the price history of a stock can be represented as a 1-d scalar field of <time, price> samples. Furthermore, fields can also represent non-scalar data, such as wind velocity. More generally, an **n-d m-vector field** is a field of samples in n-d where the value is an m-dimensional vector. The techniques presented in this paper can be extended to handle fields of dimensionalities other than 3 in a straightforward manner, and to handle vector fields by simply storing vectors in place of scalars in the appropriate data structures.

We believe our results on medical image databases will be useful in many of the above applications because they all share some basic traits: (a) the principal data objects have spatial extent, (b) the users would like to ask ad-hoc queries in an exploratory, interactive format, (c) the users need visualization tools to view 3-d or higher dimensional data in a variety of ways, (d) the spatial data objects are large, and finally, (e) the number of spatial data objects over which the user wants to query is increasing. This last characteristic is especially important in our current work in which queries like "display the PET studies of 40-year old females that show high physiological activity inside the hippocampus" are essential for understanding struc-

1. Positron Emission Tomography. PET images generally show physiological activity.

tural and functional relationships in the brain over population groups.

To provide such a flexible query environment for non-traditional data, we utilized the extensibility features of the Starburst DBMS developed at IBM's Almaden Research Center and built an operational prototype. We added new data types and associated query processing operators. We studied compact representations for these data types and assessed their performance. We integrated IBM's Data Explorer/6000 into our prototype as a visual, query front-end. Finally, we populated our prototype with anatomic models and acquired human brain imagery from the Laboratory of Neuro-Imaging of the U.C.L.A. School of Medicine.

The remainder of the paper is organized as follows: Section 2 describes the particular medical research problem we studied and its query and data characteristics; Section 3 describes the logical database design; Section 4 analyzes compact representation schemes for the data; Section 5 describes our prototype implementation, concentrating on extensions to Starburst and Data Explorer/6000; Section 6 provides initial performance results derived from the prototype; and finally, Section 7 summarizes the paper and describes the overall project and its future directions.

2: The medical application

2.1: Problem definition

As mentioned above, we have chosen the brain mapping project as a sample application for QBISM. The goal of the brain mapping research is to discover spatial correlations between activity in the brain and functional behavior, e.g. speaking or arm movement. Such activity in the brain is frequently characterized by localized, non-uniform intensity distributions involving sections or layers of brain structures, rather than uniform distributions across complete structures. Discovering the precise locations of brain activity, correlating it with anatomy, and constructing functional brain atlases is the goal of an ongoing major medical research initiative [32]. Ultimately, this understanding has clinical applications in diagnosis and treatment planning, as well as scientific and educational value.

Our system must support queries across multiple medical image studies. A **study** is actually a "billing" term referring to a set of medical images collected for a single purpose on a single patient, such as a 50 slice MRI[2] study or three x-rays of a fractured elbow. Querying across collections of these will enable the return of statistical responses and support the visualization of multiple data sets [11]. This will extend the power of medical visualization environments which today typically deal with a single study at a time. The system we envision will provide query capability over large image databases in a very investigative, interactive and iterative fashion. The following scenario illustrates a sample session with such a system in which each step generates a database query:

- The medical researcher may start by selecting from a standard atlas [29] a set of brain structures for the system to render, for example those supporting the visual system.
- After repositioning the scene to a desired viewing angle, structures may be texture mapped with a patient's PET study to highlight activity along their surfaces.
- The intensity range may be histogram segmented and other regions in this PET study identified in the same range.
- An arbitrary region may be compared with the same (or a nearby) region from a previous PET study.
- Targeting electrodes or radiation beams to regions of interest may be calculated or simulated to visualize anatomical structures intersected.
- An individual PET (or other study) may be compared with data from a comparable subpopulation of the same demographic group.

The above scenario is representative of the queries that medical researchers (i.e., those at the U.C.L.A. Laboratory of Neuro Imaging) would like to ask. To help provide a general classification of these queries, we use the concept of a scalar field: a study is represented as a collection of <x, y, z, value> tuples, where "value" is an intensity level in our application. We then have the following classification of queries:

- *Spatial* queries specify a condition on the <x,y,z> part of a scalar field (e.g., show the intensity values in a given query region of a particular MRI study).
- *Attribute* queries specify a condition on the value part of a scalar field (e.g., show regions of high intensity in a PET study).
- *Mixed* queries involve both spatial and attribute specifications (e.g., show the regions of high intensity in the right brain hemisphere).
- *Data mining* queries (not part of the current work) seek to discover patterns and "association rules" [1] in subpopulation groups (e.g., find PET study intensity patterns that are associated with any neurological condition, such as focal epilepsy, in any subpopulation).

2. Magnetic Resonance Imaging. MRI images generally show soft-tissue structural information.

2.2: Data characteristics

Basically, the database will consist of a large, growing collection of static, 3-dimensional scalar fields and a collection of anatomic models (i.e., atlases) that describe the spatial extent of anatomical structures.

The 3-dimensional scalar fields correspond to the studies. These are collected via an assortment of medical imaging modalities used to capture structural (e.g., MRI, CT[3], histology[4]) and functional / physiological (e.g., PET, SPECT[5]) information about the human brain. Each of these studies results in a 3D "volume" of intensity readings that can consume 1-100 megabytes of storage using current spatial resolutions and image depths. This volume is essentially a scalar field comprised of 3 spatial coordinates and an associated scalar intensity value. As a reference point, for clinical purposes a medium sized hospital (e.g., 500 beds) typically performs about 120,000 radiological image studies a year, including standard X-ray film studies. If all this imagery were stored in digital form (as hospitals are beginning to do [33]), the size of this hospital's yearly radiological data is estimated to be about 2 terabytes uncompressed, or 1 terabyte after lossless compression. In our work we must save the raw data volumes from the tomographic modalities as well as considerable amounts of derived data. The derived data is generated as a result of transformations to align and register the raw data, to create models suitable for surface and volume rendering of the data, and to build database representations that enable exploratory query.

As mentioned above, the database also contains atlases of reference brains for each demographic group. These models provide anatomical access to the acquired imagery via computed spatial transformations stored in the database and the spatial query operators. Their use is illustrated in the previous scenario by the step in which a structure in the visual system is used to select a particular patient's PET data. The spatial extent of that structure from the appropriate reference atlas is used to drive selective spatial extraction of the functional data.

An important point is that a PET study of a patient is not perfectly aligned with the corresponding atlas. To solve this problem, spatial and statistical warping techniques [24, 30, 31] are used to derive affine transformations that allow a study to be registered to an appropriate atlas. Thus, when a study is loaded into the database, warping matrices are computed and stored along with the original and warped study. The details of the warping techniques are outside the scope of this paper. However, these automatic or semi-automatic warping algorithms are extremely important for this application. It is precisely this technology that permits anatomic access to acquired medical images as well as comparisons among studies, even of different patients, that have been warped to the same atlas. Furthermore, it enables the database to grow, and be queryable, with minimal human analysis of the data. The coordinate system of the original study is called *patient space* while that of the atlas (and therefore, warped study) is called *atlas space*.

3: Logical design

We discuss the logical data types, spatial operations, and database schema relevant to the medical application in this section. For implementation details, refer to Section 4 and Section 5.

3.1: Data types

The data types *REGION* and *VOLUME* are of particular importance in this application; we store instances of these types, as well as other large objects, in long fields (see Section 5.1). A REGION encodes the spatial extent of an arbitrarily shaped entity, such as an anatomical structure. A VOLUME encodes all values from a 3D scalar field (e.g., a PET study) sampled on a complete, regular, cubic grid (e.g., 128x128x128 positions evenly-spaced along each axis corresponding to a 20x15x30 cm. real-world scalar field); the samples are stored in a linearized form in an implied order. We discuss these representations at length in Section 4.

3.2: Spatial operations

To efficiently execute the queries discussed in Section 2, we need spatial operators to manipulate REGIONs and VOLUMEs. We defined and implemented the following useful subset:

- INTERSECTION(REGION r1, REGION r2) returns a REGION representing the spatial intersection of r1 and r2.
- CONTAINS(REGION r1, REGION r2) returns a boolean value indicating whether r1 is a spatial superset of r2.
- EXTRACT_DATA(VOLUME v, REGION r) returns a long field[6] containing exactly those intensity values from v that are inside r.

3. Computed Tomography. CT studies generally show hard-tissue structural information (e.g., bones).

4. Histology images are acquired by slicing and photographing tissue, one thin layer at a time.

5. Single Photon Emission Computed Tomography. SPECT studies, like PET studies, show physiological information.

6. A recent version of the prototype includes the data type DATA_REGION to represent the return value of EXTRACT_DATA(); it contains a REGION and data values for each point in the REGION.

Other spatial operations would be useful as well, such as UNION(r1, r2) and DIFFERENCE(r1, r2), and would be straightforward to implement.

3.3: Schema

In Figure 1 we present an E-R diagram capturing a subset of a full medical schema appropriate for our application. Each entity (in a rectangular box) corresponds to a

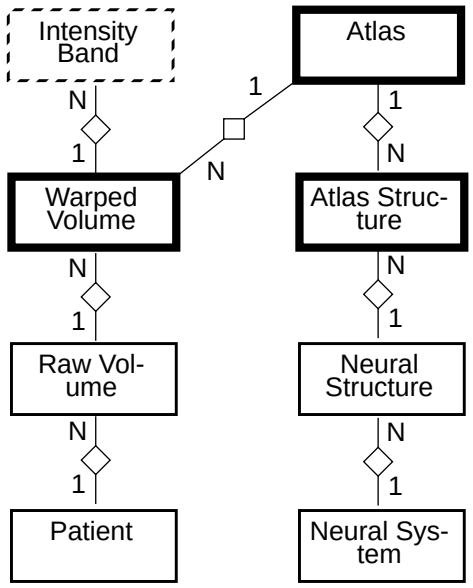


Figure 1. An entity-relationship diagram of the medical database schema. Darker boxes represent the most important entities that support spatial operations.

table in our extended relational DBMS implementation of the system. The *Neural System* and *Neural Structure* entities capture various neuro-anatomic data and relationships common to all human brains, (e.g., which structures comprise the visual system). The *Patient* entity records information pertinent to each individual (e.g., name and age). The *Raw Volume* entity captures information pertinent to a particular study of a patient, including the actual study data stored in scanline order in a long field. We will not discuss these entities in any further detail.

The *Warped Volume* entity is particularly important for our current work; its most significant attribute is a long field VOLUME that stores the warped study. As mentioned in Section 2.2, a Raw Volume can be warped to one or more atlas reference brains; we generate and store the warped volume here at database load time (rather than query time) since the computation is expensive. Additional attributes of the *Warped Volume* entity include the actual warping parameters, the raw study id, and the atlas id, among others. For the rest of this paper, the term VOLUME implies *warped* VOLUME, unless explicitly specified otherwise.

Another key entity is the *Atlas Structure* entity. Its most important attribute is a long field REGION, storing the spatial representation of the interior of the given structure in the specified atlas space. A second long-field column stores a triangular mesh representing the surface of the structure to support faster rendering of the structure itself, optionally with study data mapped onto its surface.

The *Atlas* entity has several string and numeric attributes, describing the characteristics of the reference population it represents and the coordinate system it defines (e.g., resolution and voxel size in real world units).

Finally, the *Intensity Band* entity serves as an index on the Warped Volume entity that allows rapid access to VOLUME data based on intensity (it is shown in a dotted box because it is redundant). We define an **intensity band** as a REGION representing the subset of the voxels in a VOLUME that have intensities in a particular interval (with fixed width and uniform spacing in our current prototype), such as 0-31 or 32-63. The most important attributes of an *Intensity Band* are the intensity interval end-points and a long field for the associated REGION.

3.4: Queries

To demonstrate how we use the schema and the spatial operators (see Section 5 for more details), we show below two Starburst Structured Query Language (SQL) queries that the system generates in response to the user query "retrieve the intensity values from study number 53 inside the putamen (a neural structure) from the Talairach atlas":

```
select a.n, a.x0, a.y0, a.z0, a.dx, a.dy, a.dz,
       a.atlasId, p.name, p.patientId, rv.date
from   atlas a, rawVolume rv,
       warpedVolume wv, patient p
where a.atlasId = wv.atlasId and
       wv.studyId = rv.studyId and
       rv.patientId = p.patientId and
       rv.studyId = 53 and a.atlasName = 'Talairach'

select as.region,
       extractVoxels(wv.data, as.region)
from   warpedVolume wv, atlasStructure as,
       neuralStructure ns
where wv.studyId = 53 and
       wv.atlasId = <from first query> and
       as.structureId = ns.structureId and
       ns.structureName = 'putamen'
```

The first query checks that an appropriate warped study exists and obtains information about the atlas coordinate space and patient (necessary for rendering and annotation), while the second one retrieves the actual region and data values.

For a more complicated user query, such as “retrieve the intensity values from some study inside some neural structure that are in the interval [100-200],” the SQL is similar, but includes a call to intersection() in the select list and additional joins.

4: Physical database design: static studies of representations

As mentioned before, there are two basic data types: VOLUMEs and REGIONs. A warped MRI study is an instance of a VOLUME, with intensity values defined over *all* the points of a 3D grid; an intensity band and an anatomical structure are instances of REGIONs. In the subsections that follow, we present methods of storing these data types so that the queries of interest can be answered efficiently.

In our discussion we present measurements from actual human brain data obtained from the Laboratory of Neuro Imaging at UCLA. The atlas was digitally extracted from the Talairach & Tournoux atlas [29] and represented 11 neuro-anatomic structures as REGIONs in a 128x128x128 atlas space grid. The radiological data consisted of 5 PET studies (each with 51 128x128 8-bit deep image slices) and 3 MRI studies (each with 44 512x512 8-bit deep image slices). Each study was warped and resampled to produce a 128x128x128 8-bit per voxel VOLUME and banded with uniformly spaced intensity intervals 32 units wide covering the range 0-255 to produce 8 intensity band REGIONs.

We make heavy use of space filling curves and specifically, of the Hilbert curve. On a 4x4 grid, Figure 3 shows a 2-dimensional example of the Peano curve (dotted line, also known as the Z curve [21], bit-shuffling, or Morton key [26]) and the Hilbert curve (solid line). The latter has been shown to have better spatial clustering properties [9]. Both curves require O(n) complexity to convert between locations on the curve and Cartesian coordinates where n is the number of bits used to store a position along the curve. A general algorithm for the Hilbert curve is presented in [4] and a simpler algorithm for 2 dimensions in [16].

Some terminology is necessary. Refer to Figure 2 and Figure 3 for examples. We give the definitions for the Z curve, using the prefix “z-”; the same terms with the “h-” prefix correspond to the Hilbert curve.

- The **z-id** of a voxel is its position in the Z ordering. Typically, it is considered as a binary string. In Figure 2, the z-id of the shaded 1x1 square is 2, or “0010”. Alternatively, one can compute the z-id by interleaving the voxel’s x and y coordinates; for the same shaded 1x1 square, x_1x_0=01 and y_1y_0=00, so the z-id=$x_1y_1x_0y_0$=0010.

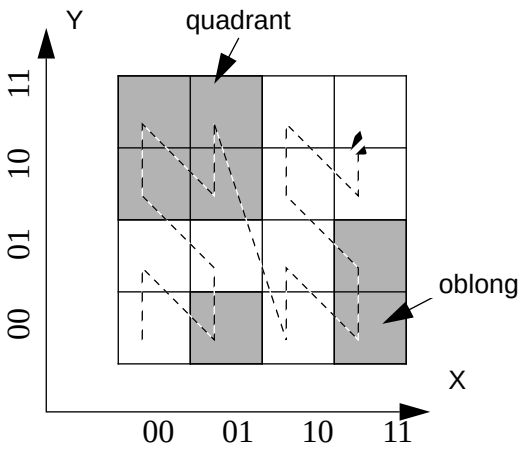


Figure 2. Illustration of (oblong) quadrants in 2D on the Z curve.

- An **octant** is a cube of maximal size that is the result of the recursive decomposition of space, and entirely inside some REGION of interest (e.g., the shaded upper-left square in Figure 2 is a quadrant, or 2D octant). More generally, an **oblong octant** (or **z-element**) of rank r is the complete set of 2^r voxels that have the same prefix in their z-ids, differing only in their r least significant bits (e.g., the shaded 1x2 rectangle in Figure 2). For a regular (cubic) octant in n-d, r must be a multiple of n.
- The **z-value** of an oblong octant is the common prefix of the z-ids of the constituent voxels (e.g., the upper-left quadrant in Figure 2 has “01**” as its z-value, where “*” stands for “don’t care”). Typically, the z-value is represented as a pair of the form <z-id, rank>, using the smallest z-id of the constituent voxels. Using bit operations, the two components can be packed into 4 bytes for grids as large as 512x512x512.
- A **z-delta** is a maximal set of voxels with consecutive z-ids all either entirely inside or outside a REGION. When these voxels are inside, we call it a **z-run**; when outside, we call it a **z-gap**. For example, one z-run in Figure 3 stretches from z-id 1100 to 1101.

4.1: Representation of a VOLUME

Our goal is to choose the best way to store a volume, with the following requirements:

1. *efficient random access*: spatial probes into a VOLUME should be fast and simple (e.g., “what is the value at point <10, 10, 10>”).
2. *good spatial clustering*: neighboring grid points in 3D should be stored close to each other on disk to reduce the number of random disk accesses into a VOLUME during extraction queries.

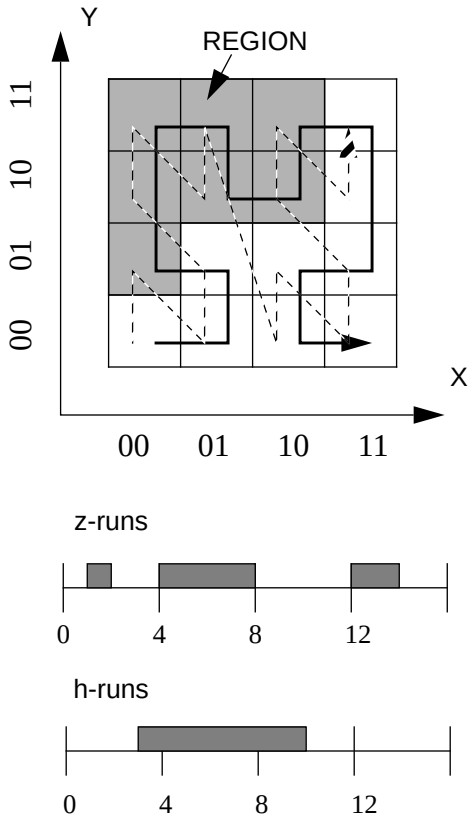


Figure 3. Illustration of h- and z-runs in 2D for the shaded REGION.

The first requirement makes compression methods unattractive; the second leads to "distance preserving" k-dimensional-to-1-dimensional mappings. Since the Hilbert curve has the best clustering properties among the known curves, we propose to store a volume by sorting the voxels in Hilbert order and storing only the intensities, since their positions are implied. We have implemented the Z ordering, too, but it gives inferior clustering (yielding about 27% more runs for each of the REGIONs we tried).

4.2: Representation of a REGION

Here we study the problem of storing REGIONs to efficiently support spatial operations such as intersections (e.g., "find the voxels that belong to the intersection of the hippocampus and the 32-63 intensity band") and "extractions" (e.g., "find the intensities in the hippocampus of Sue's last PET study"). We discuss alternative representations, compression methods, and spatial approximations.

Given the above operations, we have chosen a volumetric representation of the REGIONs. Surface models cannot support these spatial operations efficiently; Constructive Solid Geometry is not applicable since arbitrary REGIONs of interest do not necessarily have simple analytical descriptions. With a volumetric representation, we can tap the vast literature on quadtrees/octrees [13, 26] with a wealth of algorithms for indexing and spatial operations (e.g., the spatial join [22]). We use surface models only on atlas structures (in addition to the volumetric one) because they support faster, better quality rendering.

Using a volumetric representation, a REGION is typically encoded as a list of the z-values of its (oblong) octants. We propose two improvements:

- Use "runs" instead of (oblong) octants because they generally merge more voxels together. Note that every (oblong) octant is either a run or a part thereof, and every run consists of one or more (oblong) octants; therefore, the number of runs never exceeds the number of octants. Also note that most algorithms that efficiently process octants have close analogs that efficiently process runs, including the "spatial join" algorithm for computing intersections [22]. These algorithms operate by linearly scanning the runs or octants of two REGIONs in parallel (analogous to a merge of two sorted lists), optionally with some optimizations.
- Use Hilbert order, as opposed to Z order (i.e., h-runs instead of z-runs) because the Hilbert curve offers better spatial clustering and yields fewer runs.

Both of these proposals reduce the number of "pieces" algorithms must process, so they are expected to accelerate spatial operations. For illustration, Table 1 and Table 2 show how these methods represent the 2D shaded REGION of Figure 3.

TABLE 1. Z-curve encodings for REGION in Figure 3.

octants <z-id, rank>	<0001,0> <0100,2> <1100,0> <1101,0>
oblong octants <z-id, rank>	<0001,0> <0100,2> <1100,1>
runs <start, end>	<1,1> <4,7> <12,13>

TABLE 2. Hilbert-curve encodings for REGION in Figure 3.

octants <h-id, rank>	<0011,0> <0100,2> <1000,0> <1001,0>
oblong octants <h-id, rank>	<0011,0> <0100,2> <1000,1>
runs <start, end>	<3,9>

Our first step was to illustrate that h-runs do indeed produce fewer "pieces" than z-runs and (oblong) octants. For each of the various anatomic and intensity band REGIONs, we plotted the number of z-runs, octants, and oblong octants against the number of h-runs. As expected,

the Hilbert curve resulted in fewer runs than the Z curve. Furthermore, the scatter-plots were well approximated by lines: the correlation coefficients for the linear fits were 0.998, 0.974, and 0.991 for the z-runs, the octants, and the oblong octants respectively. In summary, we found that for typical brain regions, the number of h-runs, z-runs, octants, and oblong octants are in constant ratios:

(#h-runs):(#z-runs):(#oblong octants):(#octants) =
1: 1.27: 1.61: 2.42

It is interesting to note the similarity to the results reported in [9], in which the ratio for all possible 3-d rectangles is:

(#h-runs): (#z-runs) = 1: 1.20

Since h-runs consistently outperform z-runs in our experiments, as well as in all other published experiments of which we are aware, we concentrate mainly on h-runs throughout the rest of this paper.

Compression methods for runs: Compressed representations of REGIONs should improve system performance because, as we show in Section 6, the database component of the system is I/O bound. In this subsection, we seek to determine the most compact method to encode the h-run representation of a REGION on disk. The straightforward approach (termed "**naive"**) is to store the starting and ending h-ids each as long integers (4+4 bytes per run). For the example REGION in Figure 3, this method would store 1 run in 8 bytes.

When considering compression schemes, the most promising point of view is to envision the REGION as a set of runs and gaps (called "deltas") on the Hilbert curve, and to encode their lengths. To achieve optimal compression, we need to know the probability distribution of the length of a "delta". Our measurements showed that the distribution roughly obeys the relationship:

$$\text{count} = (\text{constant}) * (\text{length})^{(-a)} \qquad \text{(EQ 1)}$$

where a is ~1.5-1.7 for several atlas structure and intensity band REGIONs we tried. Thus, we should rule out all the compression methods that are tailored for geometric distributions, such as the "infinite Huffman codes" method [14, 12] and the "variable length - fixed increment" codes [28].

In light of the above, we have chosen the γ-code of Elias [8], and we will refer to it as "**elias**". It uses small codes for small numbers and achieves excellent compression because the majority of lengths are small. Let "log x" denote the binary logarithm of x. This method encodes the integer x as its length in unary followed by the binary representation of x. That is, it encodes $\lfloor \log x \rfloor + 1$ in unary (i.e., $\lfloor \log x \rfloor$ 0-bits followed by a 1-bit), followed by $x - 2^{\lfloor \log x \rfloor}$ in binary (i.e., the binary representation of x, except for the most significant digit, which is always "1"). For example:

1: 1
2: 0 1 0
3: 0 1 1
4: 00 1 00

We also calculate the **entropy** of the collection of deltas of a given REGION. If p_l is the fraction of length l deltas among the total, then the entropy theorem states that we cannot use less than

$$-\sum_l p_l \log p_l \qquad \text{(EQ 2)}$$

bits per delta. We use this lower bound as a "yardstick" to assess the performance of each method.

We compared the relative performance of these compression methods on h-runs by plotting their space requirements against the entropy bound for each REGION (see Figure 4); we also included the octant and oblong octant methods (using 4 bytes per octant). Interestingly, the linear regression fits were very good (ranging from 0.968 to 0.985), despite the variety of the REGIONs we used (atlas structures, MRI-bands, and PET-bands), and the ratios of average REGION sizes were:

(entropy):(h-run-elias):(h-run-naive):
(oblong-octant):(octant) =
1 : 1.17 : 9.50 : 10.4 : 17.8

This led us to the following conclusions:

- Without compression, the h-run-naive method outperforms the octant method, roughly by a factor of 2 (17.8/9.50), and it requires approximately the same amount of space as the oblong octant method.
- The h-run-elias method achieves the smallest storage, only 1.2 times the entropy bound, so we are confident that it is difficult to improve upon. Compared to the h-run-naive and the oblong octant methods, it achieves an 8-fold improvement.

Approximate representation of REGIONs: Additional space savings in REGIONs can be achieved by sacrificing some spatial accuracy. For the z- and h-run representations, we eliminate all the gaps that are shorter than some threshold ("mingap") by merging together the runs on each side. For the octant representation, we require that octants have a minimum size of GxGxG rather than 1x1x1, where G is a power of two, even when only one voxel of the octant is in the REGION (similar to the error-bound criterion described in [23]). Both techniques effectively increase the volume of a REGION by including outside space while simultaneously reducing the number of octants or runs required to represent it. Queries involv-

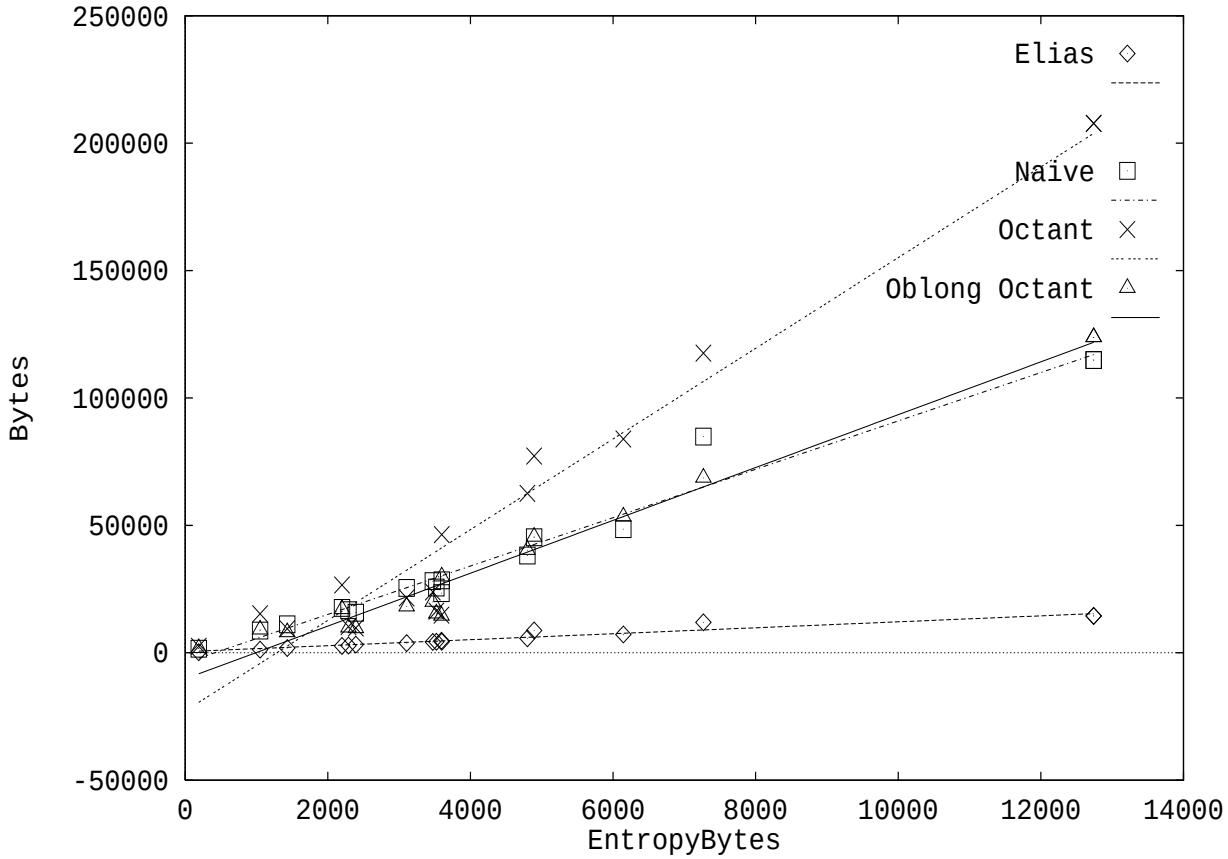


Figure 4. Comparison of REGION sizes for various methods (using the Hilbert curve) relative to the entropy limit.

ing such over-approximated REGIONs require post-processing with exact REGIONs; we do not consider them further in this paper.

4.3: Conclusions

From this section, we come to the following conclusions:

- We will store VOLUMEs as a list of intensity values, sorted on Hilbert order.
- We will store REGIONs as a compressed list of Hilbert runs. The "elias" method provides excellent space consumption results.
- We observe that the ratio of the number of h-runs to z-runs is approximately 1:1.2 in this application for various query regions.

5: System issues

5.1: Starburst extensions

Our prototype makes use of several extensibility features in Starburst [27, 18], most notably long fields and user-defined functions.

Long-fields: We store each large object, such as a REGION or VOLUME, in a separate long field. The Long Field Manager (LFM) stores long fields directly in an operating system disk device (not a file system) using a buddy allocation scheme to promote contiguity, thereby exploiting the clustering properties of the Hilbert curve. The LFM supports fast random I/O to arbitrary pieces of long fields directly to and from client memory without internal buffering. The long field is a Structured Query Language (SQL) data type in Starburst that SQL functions may accept and return. We rely heavily on these low-level features to reduce disk traffic and response time in our prototype. (Although the Starburst SQL query compiler sees

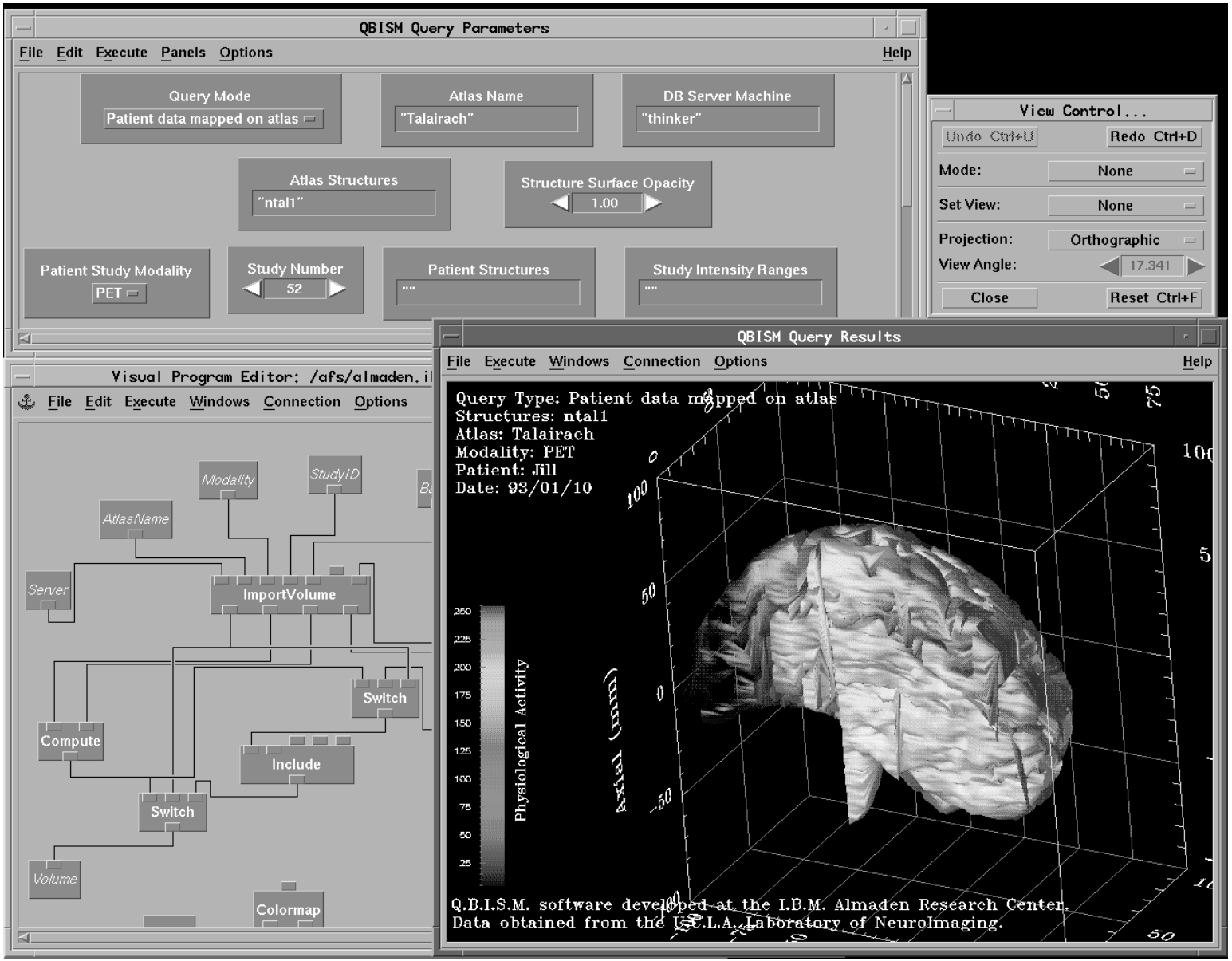
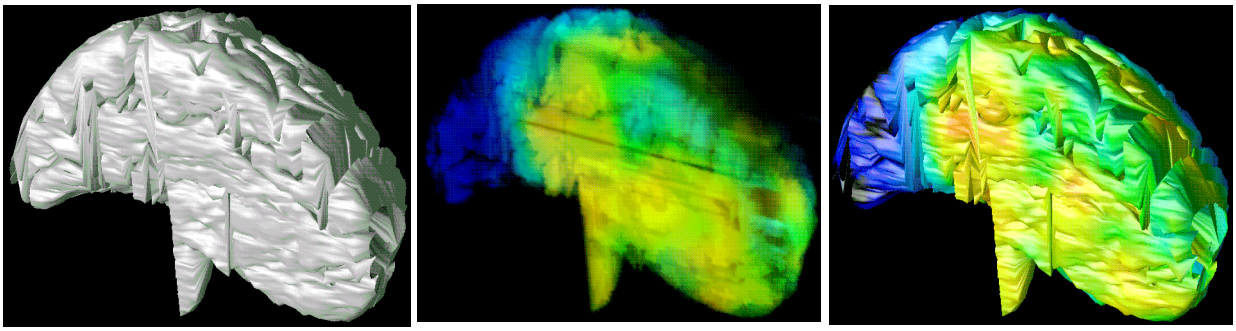


Figure 5. A sample QBISM session. After entering a query in the upper-left window, the user can see the results in the lower-right (and change the viewpoint with the controls in the upper-right corner). The partially-visible window on the lower-left shows a portion of the DX visual program, which is typically hidden from the user.



(a) (b) (c)

Figure 6. Sample query results. (a) The atlas structure "ntal1" (one hemisphere of the brain). (b) The intensity data from a PET study inside the same structure. (c) The PET data mapped onto the surface of the structure. Note the difference in shading between a and c, which is easier to see on a color screen.

our REGIONs and VOLUMEs as instances of the same long-field type, we "encapsulate" these "types" by using SQL functions to operate on them.)

User-defined SQL Functions: We implemented the operators of Section 3.2 in Starburst as user-defined SQL functions. Starburst embeds these operators (like all other SQL functions) within query execution plans at compile time and invokes them in the run-time environment. We can therefore use the complex predicate construction and query block nesting features of the SQL language to express and execute a wide variety of spatial queries, even over multiple studies.

5.2: System architecture

User interface: IBM Data Explorer/6000 (DX), a scientific visualization package [7, 6], provides the foundation for the end-user interface in our prototype. We wrote a DX "visual program" which accepts the user's query specifications through entry fields and renders the result in a variety of ways in 3D. Figure 5 shows the workstation screen during a sample session. The user can specify a study, some anatomical structures, and intensity values of interest (e.g., the data from Jane's last PET study with intensity above 200 in the right brain hemisphere). DX renders the selected information in a variety of ways: just the anatomical data, just the intensity data, both together, or a solid-textured mapping of the intensity data onto the surfaces of the structures (see Figure 6). The user can interact with the rendered picture to change the viewpoint and zoom factor, or further manipulate the selected data, by adding a cutting plane, computing a gradient field, or generating an animation, for example. Because of the caching mechanism built into DX, the user can quickly review and manipulate the results of several recently issued queries without necessitating a database reaccess.

Division of labor: Figure 7 depicts the overall architecture of the system. Its components perform the following functions:

- DX is responsible for all visualization tasks. It consists of a user-interface process for interacting with visual programs and an executive process for performing most of the computations. We added a new module called *ImportVolume* to the DX executive; it accepts the user's query and converts the spatially restricted data from the database into a DX object.

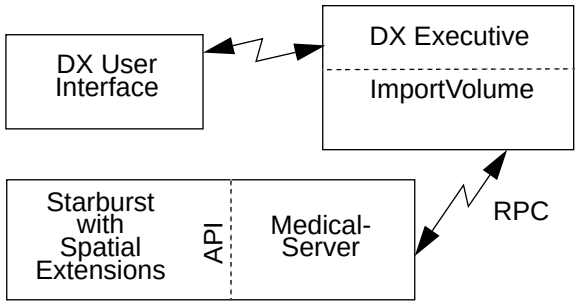


Figure 7. System architecture. Each box represents a process. The arrows represent the network.

- Starburst manages the medical data and performs the query processing, including the operations designed to spatially restrict the answer set.
- MedicalServer translates high-level query specifications it receives from DX into SQL (consider the example of Section 3.4), sends the query strings to Starburst, and then returns the results to DX. Medi-

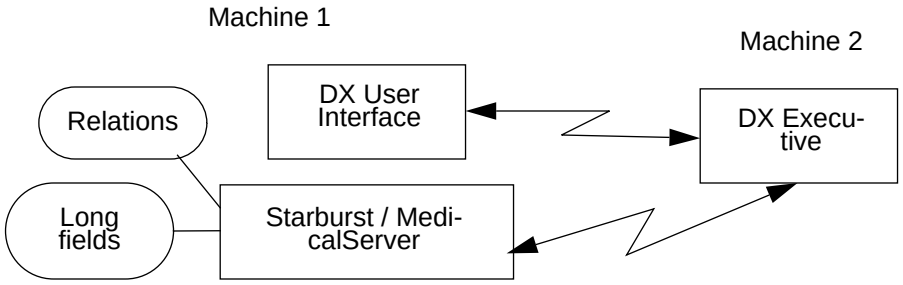


Figure 8. System configuration illustrating the assignment of storage and processes to machines.

calServer accesses Starburst through a shared-library application programming interface (API) and runs in the same process.

- The DX executive and Starburst/MedicalServer processes communicate with each other using Remote Procedure Calls (RPCs) and can thus run on separate machines (even with different low-level data formats).

6: Performance experiments

We conducted some experiments to see how our prototype performs, to identify bottlenecks, and to gather information that permits extrapolation of the results to other hardware, larger databases, and different methods. We describe the system configuration, the experiments on single studies and finally, experiments on multiple studies.

6.1: Experimental environment

Our system consisted of two IBM Risc System/6000 Model 530 workstations running AIX 3.2 (see Figure 8).

- On machine 1, with 64MB of memory, we ran the Starburst/MedicalServer process and the DX user-interface process; this machine did not utilize any special graphics rendering hardware. It held the relational data in a local AIX file system and the long field data in an AIX logical volume.
- On machine 2, with 48MB of memory, we ran the DX executive process. Running the Starburst/MedicalServer and DX executive processes on the same machine may improve performance. However, we believe that a real world system may benefit from separate dedicated visualization and database server machines and chose to conduct our experiments with a similar configuration. Note that the DX user interface process does not perform much processing, so we ran it on the database server machine rather than on a third workstation.
- Machine 1, on a 16Mbps Token Ring, communicated through a router with the second, on a 10Mbps Ethernet (*ping* reported a 4ms round-trip packet travel time).

We used the same data as in Section 4 and warped and banded it in advance to produce the schema shown in Figure 1. Since the atlas space had dimensions 128x128x128, each warped VOLUME consisted of 2 million, single-byte intensity values. We did not create indexes on any of the relation columns. Finally, for each query, unless otherwise mentioned:

- We used exact spatial REGIONs encoded as runs in Hilbert order with the "naive" 8 bytes-per-run representation scheme.
- We queried intensity ranges (e.g., 224-255) that exactly matched intensity bands stored in the database.
- We issued each query 4 times and reported the average measurements for the last 3 runs. The major components did not buffer data: we flushed the DX cache before each run (otherwise, it would buffer the database's query result), and Starburst's Long Field Manager performs no buffering anyway. Measurements varied little across runs.

6.2: Single-study queries

Table 3 shows the results of our single-study run-time experiments. The queries are all variations of "display the data from a particular PET study inside a particular REGION." Note that:

- The total execution time column shows elapsed time from start to finish, including database access and visualization of the result with an empty DX cache.

- The Starburst/MedicalServer column covers all database activity. The spatial extensions to Starburst (e.g., INTERSECTION() and EXTRACT_DATA()) and the LFM account for most of the cpu time. LFM I/O wait time accounts for the difference between the real and cpu times.
- The network column measures traffic between the MedicalServer and DX executive. It shows the number of network messages sent and their total real time cost, including both software time (e.g., RPC overhead) and "wire" time.
- The DX column covers all visualization activity. The "rendering +" time represents all processing in DX after ImportVolume is finished, primarily related to computing the 3D image. It includes some network communication between the DX user interface and executive processes, such as the transmission of the final image.
- The "other" column shows any other time the remaining columns do not measure. It consists mainly of time to run an atlas query that retrieves coordinate space information, time to compile the SQL queries, and some round-off error.

Our single-study queries fall into the following classes:

- A "simple" query, "show a full PET study", which provides a reference point for comparing more selective queries. A "flat file" system that ships the whole VOLUME to the visualization module would have similar disk I/O and network measures as this full-study query.
- Spatial queries, such as "show the data from a PET study inside a rectangular-solid with corners (30,30,30) and (100,100,100)", which demonstrate I/O and time savings throughout the system for brain structures (e.g., ntal and ntal1) or simple geometric objects compared to the times for the full-study query.
- Attribute queries, such as "show the data from a PET study within the intensity range 224-255", which demonstrate similar savings for more complicated REGIONs.

TABLE 3. Full-system run-time measurements for single-study queries. All times are in seconds. The numbers in bold are independent real time components of the totals in the last column.

	Query: *display data from a particular PET study in...*	Query Result Size		Starburst / MedicalServer			Network		DX Execution Time			Other Time (calculated, real)	Total Execution Time (real)
		H-runs	Voxels	LFM Disk I/Os (4KB)	Total Execution Time (cpu)	Total Execution Time (real)	IPC Messages	Answer Time (real)	Import-Volume (cpu)	Import-Volume (real)	Rendering+ (real)		
Simple	**Q1: entire study**	1	2097152	513	0.18	**3.4**	2103	**24.8**	10.44	**10.7**	**27**	**3.1**	**69**
Spatial	**Q2: 71x71x71 rectangular solid**	5252	357911	450	0.45	**3.5**	372	**4.4**	3.19	**3.2**	**13**	**3.9**	**28**
Spatial	**Q3: ntal**	1088	16016	29	0.14	**0.6**	22	**0.5**	0.15	**0.2**	**10**	**3.7**	**15**
Spatial	**Q4: ntal1**	14364	162628	265	0.35	**2.5**	195	**2.3**	1.44	**1.5**	**14**	**3.7**	**24**
Attribute	**Q5: band 224-255**	508	2383	32	0.13	**0.7**	7	**0.4**	0.10	**0.1**	**12**	**3.8**	**17**
Mixed	**Q6: band 224-255 in ntal1**	150	683	72	0.32	**1.0**	4	**0.4**	0.06	**0.1**	**10**	**4.5**	**16**

- Mixed queries, such as “show the data from a PET study inside ntal1 within the intensity range 224-255”, which demonstrate the ability to filter data even more finely through spatial intersection computations while yielding further time savings. Notice that query Q6, which computes the intersection of queries Q4 and Q5, requires much fewer I/Os than Q4 and Q5 combined, and less overall execution time than either Q4 or Q5.

6.3: Multi-study queries

Table 4 shows the Starburst activity from our multi-study run-time experiments. These queries are all variations of “compute the REGION in which each study’s intensity values are consistently in a particular intensity band.” Such queries require the database to compute an n-way spatial intersection. We used different REGION encoding methods, to measure their relative performance. Specifically, we used z- and h-runs with the “naive” scheme, as well as octants. We found h-runs to be superior, as expected.

TABLE 4. Run-time measurements for Starburst multiple-study queries. All times are in seconds.

Query: ***compute the REGION in which all 5 PET studies consistently have intensities in the range 128-159***	**Starburst**		
	LFM Disk I/Os (4K Pages)	**Total Execution Time**	
Encoding Method		**(cpu)**	**(real)**
h-runs, naive	446	1.02	5.7
z-runs, naive	593	1.26	7.3
octants (z order)	664	1.49	8.1

6.4: Results from the performance experiments

From these measurements we draw the following conclusions:

- The database component of the system is I/O bound since the real times far exceed the cpu times. This implies that the computational cost of managing REGIONs and performing spatial operations on them is low.
- By comparing the full-study query Q1 to the others, we can see that it is crucial to reduce the data traffic: bytes read from the disk, shipped through the network and imported for visualization. Without spatial processing support, the response time would always be comparable to the full-study time (69 seconds for Q1, versus 15-28 seconds for the others). In short, *early filtering pays off*.
- The early filtering will be even more beneficial in multiple-study queries, such as “display the voxel-wise average intensity inside ntal for these 1,000 PET studies”. In such queries, the database need only read the relevant disk pages of each study, compute the averages, and return the average values to DX. The reduction in data traffic will be linear in the number of studies involved.

7: Conclusions and future work

We have described the design and implementation of QBISM, a prototype system for querying and visualizing 3D medical images. We believe that such a system should be built on top of an extensible DBMS engine, appropriately extended to handle spatial data types, and combined with a high-quality visualization tool as the user interface. The challenges in the project were to define and implement operators and types that enable medical researchers to ask ad-hoc queries over numerous 3-d patient studies, and to provide fast responses despite the large space requirements of even a single study.

The primary contributions of this work are:

- The articulation and identification of the database requirements for supporting medical research into functional and structural brain mapping.
- The development of a logical database design, including the introduction of data types VOLUME and REGION and the implementation of operations on them within an extensible DBMS.
- The study of physical database design alternatives, including the detailed analysis of representation and compression methods for the REGION data type, the proposal to use runs along the Hilbert curve suitably compressed, and formulas for predicting space requirements.
- The performance results from our prototype, which show that the database component of the system is I/O bound and that reducing data traffic through compact representations and early filtering significantly improves performance.

Future directions for this work include:

- Spatial indexing and query optimization techniques for efficiently locating spatial objects in large populations of studies [23].
- The integration of data mining [1] and hypothesis testing techniques to support investigative queries like “find PET study intensity patterns that are associated with any neurological condition in any subpopulation”.
- The determination of image feature vectors and the study of multi-dimensional indexing methods [3, 10, 17] for them to enable similarity searching in queries

like “find all the PET studies of 40-year old females with intensities inside the cerebellum similar to Ms. Smith’s latest PET study”.

We should note that creating a practically useful brain mapping environment also requires the integration of facilities for measurement, statistical analysis and general image processing of the data. Finally, we want to mention that we have recently re-implemented our prototype using the ObjectStore OODBMS from Object Design in place of Starburst.

Acknowledgments

We’d like to thank Walid Aref and Brian Scassellati for helping with the implementation and design; Felipe Cabrera, George Lapis, Toby Lehman, Bruce Lindsay, Guy Lohman, and Hamid Pirahesh for guiding us to use Starburst effectively; the UCLA LONI Lab staff for providing and helping to interpret the human brain data; and Peter Schwarz for providing a formatting template for this paper. Furthermore, Christos Faloutsos, who contributed to this work at the IBM Almaden Research Center while on sabbatical from the University of Maryland at College Park, would like to thank SRC and the National Science Foundation (IRI-8958546) for their support as well as Empress Software Inc. and Thinking Machines Inc. for matching funds.

Bibliography

[1] R. Agrawal, T. Imielinski, A. Swami, “Mining Association Rules between Sets of Items in Massive Databases”, *ACM SIGMOD*, May 1993.

[2] M. Arya, W. Cody, C. Faloutsos, J. Richardson and A. Toga, ``QBISM: Extending a DBMS to Support 3D Medical Images'', Proceedings of the 10th International Conference on Data Engineering, IEEE Computer Society Press, February 1994, pp. 314-325. Also available as IBM Research Report RJ 9480, IBM Almaden Research Center, August 1993.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider and B. Seeger, “The R*-tree: An Efficient and Robust Access Method Points and Rectangles”, *ACM SIGMOD*, pp. 322-331, Atlantic City, NJ, May 23-25, 1990.

[4] T. Bially, “Space-Filling Curves: Their Generation and Their to Bandwidth Reduction”, IEEE Trans. on Information Theory, IT-15, 6, pp. 658-664, Nov. 1969.

[5] T.A. DeFanti, M.D. Brown, and B.H. McCormick, “Visualization: Expanding Scientific and Engineering Research Opportunities”, IEEE Computer, 22, 8, pp. 12-25, Aug. 1989.

[6] IBM AIX Visualization Data Explorer/6000 Programmer’s Reference, Second Edition, Publication No. SC38-0497-1, 1992.

[7] IBM AIX Visualization Data Explorer/6000 User’s Guide, Second Edition, Publication No. SC38-0496-1, 1992.

[8] P. Elias, “Universal Codeword Sets and Representations of Integers”, *IEEE Trans. on Information Theory*, IT-21, pp.194-203, 1975.

[9] C. Faloutsos and S. Roseman, “Fractals for Secondary Key Retrieval” *Eighth ACM SIGACT-SIGMOD-SIGART Symposium Principles of Database Systems (PODS)*, Philadelphia, PA, pp. 247-252, March 29-31, 1989. (also available as UMIACS-TR-89-47 and CS-TR-2242)

[10] M. Freeston, “The BANG File: A New Kind of Grid File”, *Proc. of ACM SIGMOD*, pp. 260-269, San Francisco, CA, May 27-29, 1987.

[11] H. Fuchs, M. Levoy, and S.M. Pizer, “Interactive Visualization of 3D Medical Data”, *IEEE Computer*, 22, 8, pp. 46-51, Aug. 1989.

[12] R.G. Gallager and D.C. Van Voorhis, “Optimal Source Codes for Geometrically Distributed Integer Alphabets” *IEEE Trans. on Information Theory*, IT-21, pp. 228-230, March 1975.

[13] I. Gargantini, “An Effective Way to Represent Quadtrees”, *Comm. of ACM (CACM)*, 25, 12, pp. 905-910, Dec. 1982.

[14] S.W. Golomb, “Run Length Encodings”, *IEEE Trans. on Information Theory*, IT-12, pp. 399-401, July 1966.

[15] J. Helman and L. Hesselink, “Representation and Display of Vector Field Topology in Fluid Flow Data Sets”, *IEEE Computer*, 22, 8, pp. 27-36, Aug. 1989.

[16] H.V. Jagadish, “Linear Clustering of Objects with Multiple Attributes” *ACM SIGMOD Conf.*, pp. 332-342, Atlantic City, NJ, May 23-25, 1990.

[17] H.V. Jagadish, “A Retrieval Technique for Similar Shapes”, Proc. ACM SIGMOD Conf., pp. 208-217, Denver, Colorado, May 29-31, 1991.

[18] T.J. Lehman and B. Lindsay, “The Starburst Long Field Manager,” *VLDB Conf. Proc.*, Amsterdam, Aug., 1989 pp. 375-383.

[19] A. Desai Narasimhalu and S. Christodoulakis, “Multimedia Information Systems: The Unfolding of a Reality” *IEEE Computer*, 24, 10, pp. 6-8, Oct. 1991.

[20] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, and P. Yanker, “The QBIC Project: Querying Images By Content Using Color, Texture, and Shape”, *SPIE 1993 International Symposium on Electronic Imaging*: Science &Technology, Conference 1908, Storage and Retrieval for Image and Video Databases, February 1993.

[21] J. Orenstein, “Spatial Query Processing in an Object-Oriented Database System” *Proc. ACM SIGMOD*, pp. 326-336, Washington D.C., May 1986.

[22] J. Orenstein and F. Manola, “PROBE:Spatial Data Modeling and Query Processing in an Image Database Application” *IEEE Trans. on Software Engineering*, 14, 5, pp. 611-629, May 1988.

[23] J. Orenstein, “Redundancy in Spatial Databases” *Proc. of ACM SIGMOD conf.*, Portland, Oregon, May 1989.

[24] C.A. Pelizzari, G.T.Y. Chen, D.R. Spelbring, R.R. Weichselbaum and C.T. Chen, “Accurate three-dimensional registration of CT, PET and/or MR images of the brain”, *J. Comput. Assisted Tomogr.*, 13, 1989, pp. 20-26.

[25] P. Reilly, “Data Visualization in Archeology”, *IBM Systems Journal*, 28, 4, pp. 569-579, 1989.

[26] H. Samet, "*Applications of Spatial Data Structures Graphics, Image Processing and GIS*", Addison-Wesley, 1990.

[27] P. Schwarz, W. Chang, J.C. Freytag, G. Lohman, J. McPherson, C. Mohan, and H. Pirahesh, "Extensibility in the Starburst Database System," *Proc. 1986 Int'l Workshop on Object-Oriented Database Systems*, Pacific Grove, September 1986, pp. 85-92.

[28] D.G. Severance, "A Practitioner's Guide to Data Base Compression", *Information Systems*, 8, 1, pp. 51-62, 1983.

[29] J. Talairach and P. Tournoux, "Co-planar stereotactic atlas of the human brain", Thieme, Stuttgart, 1988.

[30] A.W. Toga, P.K. Banerjee, and E.M. Santori, "Warping 3D models for interbrain comparisons", *Neurosc. Abs.*, 16, 1990, pp. 247

[31] A.W. Toga, P. Banerjee, and B.A. Payne, "Brain warping and averaging", *Int. Symp. on Cereb. Blood Flow and Metab.*, Miami, FL 1991.

[32] A. W. Toga, "A digital three-dimensional atlas of structure/function relationships", *J. Chem. Neuroanat.*, 4(5):313-318.

[33] A.W.K. Wong, R.K. Taira, and H.K. Huang, "Digital Archive Center: Implementation for a Radiology Department", *AJR* 159, pp. 1101-1105, November 1992.