An Interpolated Volume Data Model

Tianqiu Wang
Department of Computer Science and Engineering
tiwang@cs.ucsd.edu

Simone Santini
National Center for Biomedical Research
ssantini@ncmir.ucsd.edu

Amarnath Gupta
San Diego Supercomputer Center
gupta@sdsc.edu

University of California San Diego

Abstract

1 Introduction

Representing volume data is an important task in many fields, from medicine [1] to physics and geology. Volumes are generated by collecting discrete measurements over a finite region of space, and this collection process leads naturally to two observations: first, what is usually called a *volume* is in reality a function $f : V \to M$ from a volume V to a measurement space M; second, the volume—which is commonly understood to be a continuum—is in reality represented as a discrete (finite, in fact) set of samples.

Most volume data models carry the discreteness of the measurements all the way to the level of the abstract data type. Many a model, for instance, consider a volume as a rectangular arrangement of cubic elements called "voxels" that is, essentially, as a three-dimensional array [2]. From the point of view of storing volume data into a database and querying them, this solution has the obvious advantage of relying on a data type (the array) that is already available in commercial databases and for which a sizable literature exists on issues like their effects on query optimization [3, 4]. We argue, however, that a discrete data type is not the best way to model a continuum such as a volume at an abstract level: the finiteness of the sample should be confined to the internal representation, while the abstract data model should be continuous.

Consider, as a simple example, a "one dimensional volume" that is, an interval on a line. Assume that the measurements available for this volume consist of a single real value v, so that the volume can be represented as a curve in the cartesian plane, as in Figure 1. The discrete representation of the volume is constituted of the points marked by crosses. Assume now that the following query is posed:

> *return all the sub-volumes for which it is* $v < b$ *or* $v > a$

where a and b are suitable constants. A query on the discrete set would return all the points in the data model that is, it would return a single, connected volume. In reality, by considering the volume as a continuum (which allows us to introduce the further hypothesis of continuity of the volume functions), it is clear that the query should return three separate pieces, as in Figure 2. In two or three dimensions, using the discrete model at the abstract data type level can result not only in the union of disconnected components, but in other topological defects as well. In particular, holes may disappear, as exemplified in the surface in Figure 3. In three dimension, other topological defects are possible, such as an incorrect fundamental group (which happens, for instance, when the "hole" of a torus is filled). Note that in many cases the precision afforded by the grid is sufficient for the application at hand (if not, presumably, there

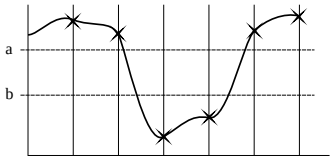

Figure 1: A simple "one dimensional" volume.

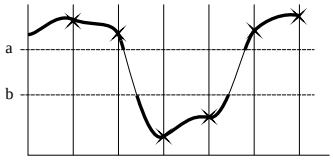

Figure 2: Results of the query on the one dimensional volume example.

**Proceedings of the 29th VLDB Conference,
Berlin, Germany, 2003**

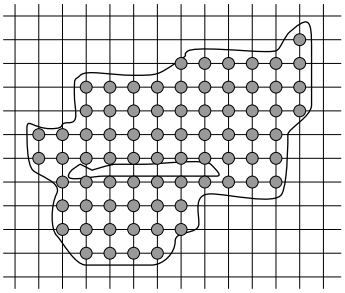

Figure 3: Topological defect (the disappearing of a hole) consequent to the discrete representation of a two-dimensional volume.

would have been a denser measurement grid to begin with), so that the error committed in placing the boundary at a location instead of another within a grid cell can be regarded as negligible; yet, the use of a discrete model can produce results that, although metrically within an acceptable precision, are topologically incorrect.

One obvious way of obtaining a continuous model is by interpolating the measurements. The use of interpolated data as an abstract data type is not new, and some principles regarding their use have begun to be established. In [5], for example, it is argued that the interpolation function and the underlying discrete data set should be kept hidden, and that only the continuous model should be visible in the abstract data type. From the point of view of our application, the model in [5] suffers from two drawbacks. First, while the continuum (which is there considered as an infinite relation) is used in the query condition, there appears to be no way to return it as the result of a query: only finite relations are returned. Second, the model in [5] doesn't include the explicit representation of the boundaries of a bounded continuum, so that the topology problems outlined previously would not disappear.

In our model, volumes are not infinite relations, but data types. This means that, at least conceptually, they are not tables, but elements that are stored in columns of tables. They are, in other words, first class values and, among other things, can be returned as results of queries. This doesn't mean, of course, that the underlying data can't be stored in tables. If this is the case, however, care must be taken to ensure that the volumes returned as query results and, potentially, exported out of the database still have access to the underlying representation in a way that is transparent to the user. This and other representation issues will be considered in the next section.

2 The volume model

In this section, we will briefly discuss the two principal aspects of our volume model, namely the abstract data type that is exported (including the algebra that manipulate it) and the representation upon which the model is based.

2.1 The abstract data type

Formally, a volume is a continuous function $f : V \rightarrow M$, where V is a three dimensional closed, compact, and bounded set, and M is a *measurement space* that we will assumed endowed with the structure of a vector space and such that all the components of M are named. That is, M is represented as $M = (N_1 : T_1, \ldots, N_n : T_n)$, where N_i are names and T_i are data types. In order to be able to interpolate, we assume that each one of the data types T_i has the structure of a linear space. The measurement space M is specific to each volume, and it goes without saying that two volumes $f_1 : V \rightarrow M_1$ and $f_2 : V \rightarrow M_2$ which share the same domain but map into different measurement spaces should be regarded as instances of two different data types. A special volume type is what we call the *mask*. Formally, a mask is a volume that maps to the data type *unit* (the "bottom" data type, with one value only). A mask is uniquely identified by its domain V and will be used mostly to "cut" pieces from other volumes.

The most important operations of the volume algebra are summarized in Table 1. Other operations are defined for determining the bounding box of a volume, returning the points in its representation, creating a volume, determining the connectivity of its fundamental group, and so on, but they are not essential for the discussion that follows.

The selection operator `sel` extracts from a volume the portions that satisfy the condition C. Since in our model volumes are always connected, the operation returns a set of connected components, each one represented as a volume, rather than a single volume. The condition C can be based on the values of the volume (being expressed in terms of the names of the measurement space M) or on the coordinates of the points, using the conventional name `$pt` to represent a volume point and `$pt.x`, `$pt.y`, `$pt.z` for its coordinates. The next operation, for instance, returns the sub-volumes of $f : V \rightarrow [meas : \mathbb{R}]$ composed of points with negative x coordinate and such that their `meas` value is at least 5:

$$f_2 = \texttt{sel}(f, \$\texttt{pt.x} \leq 0 \texttt{ and meas} \geq 5) \qquad (1)$$

Note that the conditions $\texttt{meas} > 5$ or $\$\texttt{pt.x} < 0$ would be illegal because they would not return a closed set that is, they would return a set that, according to our definition, is not a volume.

The projection operator works on the measurement space much like the synonymous relational algebra operator. The intersection (resp. union) operator acts as a set intersection (union) on the domain of its arguments and uses the operator `op` to compute the values associated to the points of the resulting domain. If $f_1 : V_1 \rightarrow M_1$, $f_2 : V_2 \rightarrow M_2$, and $\texttt{op} : M_1 \times M_2 \rightarrow M$, then

$$\texttt{intrs}(f_1, f_2, \texttt{op}) : V_1 \cap V_2 \rightarrow M. \qquad (2)$$

In the case of union, we have the additional complication that the functions f_1 and f_2 may not be defined on the whole $V_1 \cup V_2$. The missing values are replaced

Name	Use	Description
`affine`	$V_1 = \texttt{affine}(A, V)$	Applies an affine transform to a volume
`sel`	$\{V\} = \texttt{sel}(V, C)$	Selects from V based on the condition C.
`proj`	$V = \texttt{proj}(V, [C_1, \ldots, C_n])$	Projects out columns in the measurement space.
`intrs`	$V = \texttt{intrs}(V_1, V_2, op)$	Algebraic intersection
`union`	$V = \texttt{union}(V_1, V_2, op)$	Algebraic union
`inside`	$t = \texttt{inside}(p, V)$	Checks is a point belongs to the domain of a volume.
`val`	$v = \texttt{val}(p, V)$	Value of volume V at point p.

Table 1: Operations of the volume algebra

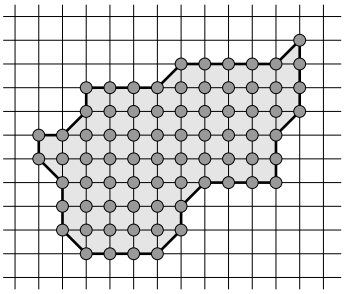

Figure 4: Boundaries of a volume at creation time.

with the conventional value "null" which, conventionally is the neutral element for each operator *op*, that is, for each x and each operator, $x$op null $= x$. This means, of course, that in the regions where f_1 is not defined, we have $\texttt{intrs}(f_1, f_2, \texttt{op})(x) = f_2(x)$ and similarly for the regions where f_2 is not defined. The common operators supported natively in our data type are addition ("+"), subtraction ("-"), multiplication by a scalar ("*") and join ($\bowtie$).

2.2 Representation

Our abstract data model is compatible with a number of finite representations: the only requirement is that the finite representation allows the definition of a suitable interpolation function. This is true, in general, for all representation that considers point measurements. It doesn't hold for "voxel" model, for which the measurement is associated with a finite volume, unles some additional assumption is made as to the location of the measurement inside the volume. Several measurement structures accommodate this model, from a regular grid of measurement points, to an irregular tetrahedral grid, to a set of disconnected points (also called a "point cloud").

In our current implementation, the measurements are arranged in a regular parallelepipedal grid. The interpolation function used is a configuration parameter determined during the installation of the system; in the following we will always make reference to the common case of a tri-linear interpolation function. When a volume is created, its boundaries are determined naturally by the grid on which the volume is defined, as exemplified, for a two-dimensional volume, in Figure 4. Any topological error with respect to the real data introduced by this representation would fall below the measurement precision, and would be undetectable.

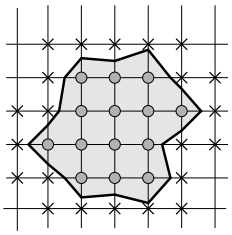

Figure 5: Boundaries of a volume dispalced with respect to the grid.

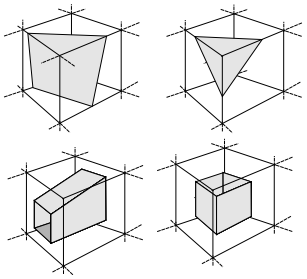

Figure 6: Boundaries of three-dimensional volume.

When a volume is obtained by cutting pieces of another volume, for example with a selection operation, the boundaries if the volume will not in general be aligned with the grid. Approximating the boundary with points on the grid would introduce the topological problems outlined in the previous section. In order to avoid these problems, we allow the boundary of the volume to be displaced with respect to the data grid by registering, for each boundary cell, the position of the boundary inside it. The resulting model is that of a piecewise linear boundary, as exemplified for a two-dimensional volume, in Figure 5. In order to extend the interpolation up to the boundary, it is necessary to keep a number of points not belonging to the volume. There *pahntom points* are represented by crosses in Figure 5.

In volumes, the specification of the boundary is a bit more complicated. First, the boundary itself is a piecewise linear surface rather than a simpler piecewise linear curve; second, the relation between a portion of the boundary and a parallelepipedal cell must take into account a larger number of possibilities, some of which are illustrated in Figure 6 Once the various possibilities have been accounted for, however, we have the representation of a continuous

piecewise bi-linear surface up to which we can interpolate the volume values, and that can be placed at arbitrary positions with respect to the grid points.

A final issue that we want to discuss briefly here arises when volumes are returned as results of queries. The "things" that are returned are volume objects and, to fix the ideas, let us say that these results are exported from the database as java objects (which is actually the case in our implementation). This volume object must carry with it its internal representation that is, the grid of of volume points and phantom points necessary for its computation.

Carrying along the representation can be a problem when the volume is represented by thousands or tens of thousands of points (a typical volume for a human brain derived from an MRI scan has between 100,000 and 500,000 points). Especially if the volume object is to be sent over a communication network, such large data set can make the communication extremely slow. All this is more unreasonable if we consider that the user (or the application) that requested the volume to begin with might not need to access all the points in the representation to carry out the computation that is needed: if all that is necessary is to call methods to, say, measure the volume (which is usually kept in a separate variable and doesn't have to be computed on the fly), keeping the representation is useless.

To avoid moving around inordinate amounts of data, we allow the volume object to have a virtual representation. That is, while the volume object travels around the system, its representation stays in the database. The various methods, instead of accessing a local representation, issue dtabase queries to access just enough of the internal representation to do their job. Clearly, every volume has the possibility to be "grounded" into a local representation by calling a suitable method. This will create a full local representation for the volume object independent of the database and is useful, in addition to the case in which the representation is manageably small, if there is the risk that during the life of the volume object the database will be updated.

3 The Demo

In this demonstration we show the basic functionality of our volume data model. We will consider examples from biology (brain MRI data) and volumes of measurement from quantuum physics. The data model is similar in the two cases, but the operations that are commonly used are quite different. In the second case, for instance, one is often interested in conditions on the behavior of local differential operators, such as zero-flow surfaces, while in the biological case one has a mix of value conditions (e.g. homogeneity) and geometric condition (e.g. conditions on curvature). We hope, with these two application fields, to highlight the generality and flexibility of our model.

The system is based on a commercial database (specifically, the Oracle 9i database), augmented with specialized functions to manipulate our volume data model. We will demonstrate various operations specifying queries both using a graphical user interface and entering them directly in SQL augmented with the volume algebra operations.

The demo testbed includes utilities to translate the volumes that are created as results to queries into standard graphic formats that can be used by volume display programs, as well as a volume display program that we will use to give a more visual demonstration of the query results.

4 Acknowledgements

The work presented in this paper was done under the auspices and with the funding of NIH project NCRR RR08 605, *Biomedical Imaging Research Network*, which the authors gratefully acknowledge.

References

[1] A. Toga and P. Thmpson, "Multimodal brain atlases," in *Medical Image Databases* (S. Wong, ed.), Kluwer Academic, 1998.

[2] J. D. F. amd Andries Van Dam and S. K. Feiner, *Introduction to Computer Graphics*. Addison-Wesley, 1993.

[3] A. P. Marathe and K. Salem, "A language for maniplating arrays," in *Proceedings of the 23rd VLDB Conference, Athens*, pp. 46–55, 1997.

[4] A. Alcantara and B. Buckles, "Supporting array types in monoid comprehensions."

[5] S. Grumbach, P. Rigaux, and L. Segoufin, "Manipulating interpolated data is easier than you thought," in *The VLDB Journal*, pp. 156–165, 2000.

La Jolla, February 2003