An Interpolated Volume Model for Databases

Tianqiu Wang[1], Simone Santini[2], and Amarnath Gupta[3]

[1] Department of Computer Science and Engineering, University of California San Diego 9500 Gilman Drive, La Jolla, CA 92093, USA
tiwang@cs.ucsd.edu,
[2] Bio-Informatics Research Network, Department of Neuroscience, University of California San Diego 9500 Gilman Drive, La Jolla, CA 92093, USA
ssantini@ncmir.ucsd.edu,
[3] San Diego Supercomputer Center, University of California San Diego 9500 Gilman Drive, La Jolla, CA 92093, USA
gupta@sdsc.edu

Abstract. In this paper we present a volume data model amenable to querying volumes in databases. Unlike most existing volume models, which are directed towards specific applications (notably, volume displaying) and are therefore directed towards executing with the greatest possible efficiency a small set of operations, we are interested in defining a *volume algebra* through which generic query conditions can be composed.
We argue that a general volume model should have two characteristics: (1) it should consider volumes as first class data types that can be returned as results of queries, and (2) it should model volumes as continua, hiding the discrete nature of the measurements inside the representation.

1 Introduction

Data representing volumes are common in many fields such as magnetic resonance imaging [1] and computed tomography in medicine [2], seismic data analysis in the oil and gas industry [3], and various areas of biology, chemistry, and nuclear physics. All these data have certain characteristics in common: they represent continua of points with (continuously varying) quantifiable properties at every point, and they are acquired in the form of a discrete, finite sample of measurement taken over the continuum where the properties are defined.

Considerable research efforts have been directed at modeling volume data, the majority of which have considered visualization problems [4, 5]. In [6], a variety of methods is presented to represent volumes, and some related research topics are discussed.

There appears to be considerably less work available on models and data types for volumes intended as objects of computation. In particular, there isn't much work that we are aware of on data models for storing volumes in databases and querying them.

The issues in this case are quite different from those that are faced by the visualization researchers. While in visualization one must face the problem of performing a relatively small number of operations on one volume at a time (and consequently, of finding the correct data structures to perform those relatively few operations with the

greatest efficiency), in databases, our problem is to deal with large sets of volumes and to create a generic *volume algebra* that can support many types of queries.

A good data model should represent a volume as first class data type with reasonable accuracy, provide means to create, manipulate, and query volume data intuitively, while preserving the unique characteristics of a volume. Users should be freed from dealing with lower level primitives so that they don't have to understand the detailed of how the volume is represented at the physical level.

We define a *volume* as a function $V : D \rightarrow S$ from a three-dimensional compact domain D to a property space S. It is the continuity of D that makes discrete models inaccurate, since these models fail to represent D as a continuum.

In order to exemplify the potential problems that a discrete model can cause, let us consider 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 and its properties can be represented as a curve in the Cartesian plane, as in Figure 1. The discrete representation of the volume is constituted by the

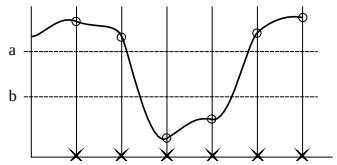

Fig. 1. A simple "one dimensional" volume.

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

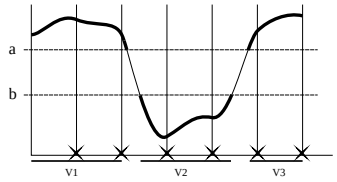

Fig. 2. Results of the query on the one dimensional volume example.

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 homothopy number (which happens, for instance, when the "hole" of a torus is filled).

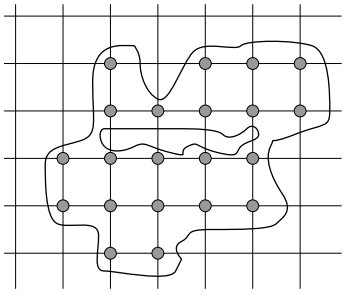

Fig. 3. Topological defect (the disappearing of a hole) consequent to the discrete representation of a two-dimensional volume.

One obvious way of obtaining a continuous model is by interpolating the measurements, a solution that is proposed by several researchers for representing infinite relation in database [7, 8].

It is very important to capture the continuous nature of volumes because in many cases a user needs to know the value of a property at a point where no measurement is available. As another example of possible problems caused by discrete models, consider a discrete volume created by measuring a property v in a portion of space. We have a query asking to return the sub-volume for which $v \geq 0$. Note that the boundary of this volume is composed of points for which $v = 0$ but, in general, for no point on the grid it will be $v = 0$ (theoretically, this is true with probability 1). Using changes in the sign of v and with a suitable continuity hypothesis, we can individuate the points on the grid adjacent to the "real" boundary of the required volume. The problem is that, with a discrete model, we have no way of representing such a boundary, and the best we can do is to return the set of points for which $v > 0$: the condition that holds true for the points on the boundary is not what one would expect from the query.

To make things worse, let us now take te result of the previous query, and let ϵ be the smallest value of v among all the points. We ask now for the sub-volume such that $v \leq \epsilon/2$. The query will obviously return the empty set, that is, the empty volume. But, of course, for every volume for which there is at least a point $v > \epsilon$, the condition

$$v \geq 0 \wedge v \leq \frac{\epsilon}{2}$$

should return a non-empty volume.

We propose a model in which volumes can be defined on continua, and an algebra for this data model. In addition to the modeling advantages mentioned above, modeling a volume as a continuum provides the right framework to solve certain issues that are

hard to deal with in discrete models, since they require changing the underlying representation. For example, in our model it is easy to join volumes even if the underlying grids of properties values do not coincide (e.g. one is more dense than the other).

2 Related Work

Regular grids of point measurements are a fairly common representation of volume data [9]; it is the easiest way to model a volume in the common case in which the property values are on a regular grid to begin with [10]. An alternative representation is to consider the volume as a collection of finite volume, regular elements called "voxels." Each element has an associated value. In this case, unless additional hypotheses are made, the measurements are not associated with any point inside the element, but only with the whole element.

A regular, parallelepipedal grid can be stored in a three-dimensional array. There have been several proposals of array data models for databases, and several array languages have been defined such as the *Array Manipulation Language* (AML) and the *Array Query Language* (AQL). The AML [11] is a declarative language for optimizing the non-relational sub-expressions on the array data type (ie. the one appears in the select clause) in a relational query. As the name Array Manipulation Language suggests, the AML is mainly designed to do array manipulations and transformations based on index patterns and sub-arrays. Its focus is not on selection and retrieval based on properties of the values stored in the array.

Unlike the AML, AQL [12] also supports queries. The AQL is a high level comprehension-style language based on nested relational calculus for arrays which is an extension to the nested relational calculus. The user has the possibility to define new functions and register them as AQL primitives. The uniqueness of the AQL is the point of view that arrays are functions rather than collection types.

In general, most existing array based models do not have a rich set of query support based on content or property value of volumes. An exception is the model from [9], where some value selection primitives are provided.

In modern databases, volume data can be stored as spatial data [13]. Spatial data types have good support for geometric operations, but they don't allow operations on properties.

Interpolation has been proposed as a possibility for representing continuous functions because it can represent any point in a continuum using a finite representation [7, 8]. While in [7], interpolated and sampled data are modeled differently at the logical level, leaving to the user the task of managing the interpolation, in [8], it is suggested that samples and interpolation functions should be hidden from the logical level, and that only the continuous model should be visible in the abstract data type. While an equal treatment of interpolated and sampled data is applicable in volume data, from the point of view of our application, 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. The model in [8] suffers, from our point of view, from two drawbacks. First, while the continuum (which is there considered as an infi-

nite 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 [8] doesn't include the explicit representation of the boundaries of a bounded continuum, so that the topology problems outlined previously would not disappear.

A similar continuous model for volume visualization has been proposed [14]. Volume is modeled as a scalar filed $F(p)$ which offers representations that defines data for every point p in E^3. It also uses a discrete physical representation consists of a set of sampled points, their topological relationships, and an interpolation function. The recommended algebra used for this model is Constructive Volume Geometry [15]. However, this algebra framework is designed for modeling complex spatial objects for visualization. It is not based on a continuous domain, and its geometric operations are designed to rearrange volumes in a scene.

The problem we are considering in this paper is not solved by any of the models mentioned above. We are interested in a framework to perform general computation on large sets of volumes, and at the same time, to use these operations to identify volumes that satisfies certain requirements either based on geometry or content. From this perspective, it is obvious not only that databases have an important role to play, but also that we must develop a conceptual model of volume data broad enough to support general query operations. It is the goal of this paper to introduce such a model.

3 The volume model

In this section, we introduce the abstract model that we are going to use for our volume algebra, as well as some relevant associated concepts.

All the definitions are relative to the three-dimensional Euclidean space, since this is the case in which we are interested. In particular, a *point* is a triple $(x, y, z) \in \mathbb{R}^3$. Many of the definitions below, and many of the properties of the model, however, generalize to higher dimensional spaces.

Definition 1. *A volume* domain *(or, simply, a* domain*) is a compact, closed subset of* $\mathbb{R}^3$. *The measure of the domain* D *is*

$$m(D) = \int_V dxdydz$$

We give this definition essentially to establish our terminology. In many cases, confusion may arise because the word "volume" is used to denote quite different things. For instance, both the entities that we have called domain and its measure are occasionally referred to as volumes. Since to us, a volume is yet another concept, it is worth risking being pedantic now to avoid confusion later.

Definition 2. *A volume is a continuous function* $V : D \rightarrow S$, *where* D *is a domain, and* S *is a* property space *that we will assumed endowed with the structure of a vector space and such that all the components of* S *are named. That is,* S *is represented as* $S = \{N_1 : T_1, \ldots, N_n : T_n\}$, *where* N_i *are the names of the content measurement and* T_i *are their data types.*

The property space S is specific to each volume, and it goes without saying that two volumes $V_1 : D \to S_1$ and $V_2 : D \to S_2$ 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 D and will be used mostly to "cut" pieces from other volumes.

Note that in our model, we take what one might call a "functional" view of a volume, that is, our primary objects are functions from a domain to a measurement space, rather than compact sets in $\mathbb{R}^3$. We argue that this view models more accurately the way in which volumetric data are used in practice.

4 An algebra for volume data

In this section, we introduce our volume algebra. Due to our view of a volume as a function, our algebra is a hybrid between a function algebra (which operates on the function types $D \to S$), an algebra for sets in space (which operates on the domains D), and a subset of relational algebra (which operates on the property spaces S).

A few boolean topological primitives need to be defined so that they can be used in volume queries. They are all defined based on the basic topological primitive **inside**: $inside(p, V)$ is true if the point p is contained in the domain D of the volume V.

disjoint: $disjoint(V_1, V_2) \iff \forall p, inside(p, V_1) \Rightarrow \neg inside(p, V_2)$.
overlap: $overlap(V_1, V_2) \iff \exists p, inside(p, V_1) \wedge inside(p, V_2)$.
enclose: $encloses(V_1, V_2) \iff \forall p, inside(p, V_2) \Rightarrow inside(p, V_1)$.

4.1 Operators

Our operators take either one or two volumes as operands. The common results types of these operations are volumes, sets of volumes, and scalar measurement values.

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 source volume, determining its homothopy number, and so on, but we will not consider them in this paper. Also, although we will not prove it in this paper, our model has the following completeness property: assume that a "volume table" is created in a relational database, with columns for x, y, and z, and for all the properties of the volume. Also, assume that we restrict the operations on this table to those which return a similar volume representation (for instance, we don't allow to project out y and z, as this would not produce a volume). Then any operation that can be expressed using relational operators and operators on the data types that constitute the columns can be executed on our volume model using our algebra.

The rest of this section is dedicated to an analysis of the operations in Table 1.

Affine. The *affine* operator is used to rotate, transform, scale, and shear a volume. It takes a volume and an affine transformation matrix and returns a new volume with

Name	Use	Description
`affine`	$V_{result} = \texttt{affine}(A, V_{src})$	Applies an affine transform to a volume
`proj`	$V_{result} = \texttt{proj}(V_{src}, [N_1, \ldots, N_n])$	Projects out columns in the property space.
`intrs`	$R = \texttt{intrs}(V_{src1}, V_{src2}, op)$	Apply op to the intersection
`cut`	$R = \texttt{cut}(V_{src}, Cg)$	Selects from a volume based on the geometric condition Cg.
`sel`	$R = \texttt{sel}(V_{src}, Cp)$	Selects from a volume based on the property condition Cp.
`val`	$v = \texttt{val}(p, V_{src})$	Value of a volume at point p.
`vol`	$m = \texttt{vol}(V_{src})$	Measurement of a volume's size.

Table 1. Operations of the volume algebra

the domain obtaining by applying the matrix $A \in \mathbb{R}^{4\times4}$ to the domain of the argument volume. The general form is:

$$V_{result} = \text{affine}(A, V_{src})$$

where V_{src} is a volume and A is a 4×4 affine transformation matrix [16].

Projection. A volume is designed to have multiple properties or multiple measurements at each point. A projection on the volume measurement space is provided to select only a subset of these properties. The projection operator takes a volume and one or more names of measurements of that volume, and returns a volume with the same domain but now with only the projected measurements. The general form is:

$$V_r = \text{proj}(V, \{N_1, \ldots, N_n\}) \tag{1}$$

where V is a volume, and $N_1, \ldots, N_n$ are names of components in the measurement space of V. If $V : D \to S$, where $S = \{N_1 : T_1, \ldots, N_n : T_n, N_{n+1} : T_{n+1}, \ldots, N_p : T_p\}$, then $V_r : D \to S'$, where $S' = \{N_1 : T_1, \ldots, N_n : T_n\}$, that is V_r has the same domain as the V but with the property space restricted to the components named in the projection.

Intersection. The intersection operator take the intersection of two source volumes and then apply a operation. The general form is:

$$R = \text{intrs}(V_1, V_2, \text{op}) \tag{2}$$

where V_1, V_2 are volumes, op is an operator on property spaces. If $V_1 : D_1 \to S_1$, $V_2 : D_2 \to S_2$, and op : $S_1 \times S_2 \to S_r$, then

$$\begin{array}{c} R = \{V_1, \ldots, V_q\} \\ V_i : D_i \to S_r \\ \bigcup D_i = D_1 \cap D_2 \end{array} \tag{3}$$

that is, the domain of the set of volumes R is given by the intersection of the domains of V_1 and V_2, and the property space is obtained by applying the operator op pair-wise

to elements of S_1 and S_2 corresponding to points in V_1 and V_2. Note that our definition of volume requires its domain to be connected. Since the intersection of two connected domains is not necessarily connected, this operation returns a set of volumes rather than a single volume. The operators that our algebra supported are element-wise addition ("+"), element-wise subtraction ("-"), element-wise multiplication ("*"), Cartesian product ($\times$), and the null function (nil : $\alpha \times \beta \rightarrow$ unit, for all data types α and β). Note that intersection is commutative (resp. associative) if and only if the operator op is commutative (associative). This operation can also be used as a way to select part of a volume by intersect with a mask volume.

Cut. The cut operator changes the domain of a volume by putting restrictions on the coordinates (x, y, z) of the points, expressed as a set of conditions that the coordinates are required to satisfy. Because in our definition, points in a volume must be connected, the result of *cut* could be more than one volume, therefore, *cut* returns a set of volumes.

$$R = \text{cut}(V, C) \tag{4}$$

where V is a volume, and C is a set of predicate on the point coordinates. If $V : D \rightarrow S$, $C = \{c_1, \ldots, c_n\}$, then

$$\begin{gathered} R = \{V_1, \ldots, V_q\} \\ V_i : D_i \rightarrow S \\ \bigcup D_i : \{p | \forall i, i \leq n \Rightarrow c_i(p)\} \end{gathered} \tag{5}$$

The predicates c_i often takes the form of polynomial equalities or inequalities: $c_i \equiv \mathbb{P}(x, y, z)$, which describes portions of space delimitated by polynomial surfaces. In this case, the operation corresponds to "cutting" (hence its name) the source volume with the given polynomial surface.

Selection. The select operator takes a volume and select a portion of it based on properties. Just like in **Intersection** and **Cut**, this also changes the domain, and may lead to several disjointed volumes. The general form is:

$$R = \text{sel}(V, c) \tag{6}$$

where V is a volumes, and C contains predicates based on the properties of the volume. If $V : D \rightarrow S$, then

$$\begin{gathered} R = \{V_1, \ldots, V_q\} \\ V_i : D_i \rightarrow S \\ \forall p \; \exists i : \text{inside}(p, V_i) \Rightarrow c(V_i(p)) \end{gathered} \tag{7}$$

The essential difference between **Cut** and **Selection** is that **Cut** picks portions of the source volume based on the conditions on the domain, while **Selection** does based on conditions on the properties.

Some properties of the algebra that can be used, for instance, for query rewriting, are the following:

- $\text{sel}(\text{cut}(V, C_1), C_2) = \text{cut}(\text{sel}(V, C_2), C_1)$;
- $\text{intrs}(\text{affine}(A, V_1), \text{affine}(A, V_2)) = \text{affine}(A, \text{intrs}(V_1, V_2))$;
- $\text{intrs}(\text{proj}(V_1, C_1), \text{proj}(V_2, C_2), \times) = \text{proj}(\text{intrs}(V_1, V_2), C_1 \cup C_2)$;

– intrs(proj(V_1, C), proj(V_2, C), op) = proj(intrs(V_1, V_2), C) (if V_1 and V_2 have the same schema);
– cut(proj(V, C_1), C_2) = proj(cut(V, C_2), C_1);
– sel(proj(V, C_1), C_2) = proj(sel(V, C_2), C_1) (if none of the names in C_1 appears in the conditions C_2);

as well as some obvious extensions (for example, "affine" commutes with both selection and cut).

5 Representation

Our abstract data model is compatible with a number of finite representations: the only requirement is that the representation allows the definition of a suitable interpolation function. This is true, in general, for all representations that considers point measurements, while doesn't hold for "voxel" models, in which each measurement is associated with a finite volume, unless 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 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. We will use the expression "grid point" to refer to a point on the sampled grid. Whenever there is no danger of confusion with the points of the domain as defined in the previous section, we will simply call them "points".

The discrete representation is transformed into a function defined on a continuum with the introduction of an *interpolation function*, which allows us to determine the property values of the volume between sampled points. 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 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

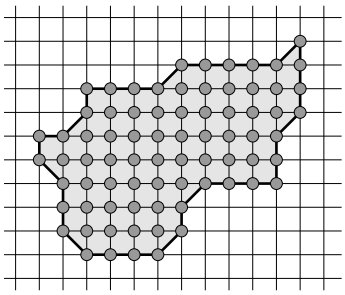

Fig. 4. Boundaries of a volume at creation time.

topological error with respect to the real data introduced by this representation would fall below the measurement precision, and would be undetectable.

The grid of points is stored in a three-dimensional array *PS* of dimension $N_1 \times N_2 \times N_3$. The *physical address* of a grid point p is the triple of non-negative integer $L = (i, j, k)$.

The translation between the physical address L of a point and its coordinates in the domain is done by keeping an affine transform matrix **A** alongside the volume. The transformation between domain coordinates (x, y, z) and physical address is given by:

$$\begin{bmatrix} i \\ j \\ k \\ 1 \end{bmatrix} = \mathbf{A} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{8}$$

(rounding to the greatest integer smaller than the computed value is implicit here).

Note that with this representation, the algebra operator *affine* doesn't need to operate on the point array: all it has to do is to change the matrix **A**. Other components of the representation are connected to the problem of representing the boundary of the domain, about which we will make a little digression.

When a volume is obtained by cutting pieces of another volume, for example with a selection operation, the boundaries of the new volume will not in general be aligned with the grid. In order to return a more accurate volume and—more importantly—to avoid the topological defects mentioned in the introduction, we allow the boundary of the volume to be displaced with respect to the data grid. We do this 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.

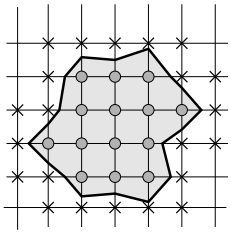

Fig. 5. Boundaries of a volume dispalced with respect to the grid.

In volumes, the specification of the boundary is a bit more complicated. First, the boundary itself is a piecewise linear surface rather than a 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 which are illustrated in Figure 6. This boundary surface can be constructed in many ways (see, for example, the Marching Cubes algorithm [17]). If we classify the values at each corner of the boundary cell as either being below or above the boundary condition, there are 256 possible configurations. If we account for symmetries, there are only 14 unique configurations. For each one of these configurations, we need to store the points at which the boundary surface

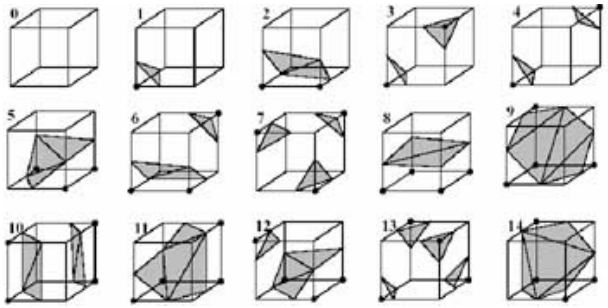

Fig. 6. Boundaries of three-dimensional volume.

intersects the edges of the cube. Once the various possibilities have been accounted for, 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.

This model of boundary entails an additional complication, also illustrated in Figure 5. In order to extend the interpolated function to the geometric boundary of the domain, it is necessary to retain a number of points outside the volume so that every boundary cell have a full set of eight measurements on which the interpolation can be built. These points are called *phantom points* and are kept as part of the sampled grid with an indication that they do not belong to the volume.

6 Query examples

In this section, we will present some examples of queries on our volume model. We will consider examples from quantum chemistry and biology, which are the domains to which we are currently applying our volume model.

6.1 Zero-flux surface extraction

Background. The electrons of an atom are distributed around it in the attractive filed exerted by the nucleus. Although we often represent an atom as a sphere, its shape is actually determined by the electron density $\rho(p)$ which describes the manner in which the electronic charge is distributed in space. Certain physical properties of the molecular configuration are derived not from the electron density itself, but from those properties of the vector field generated by the gradient of the electron density. Starting at any point, one determines the gradient of $\rho(p)$, $\nabla\rho$. This is a vector that points in the direction of maximum increase in the density. A gradient vector map generated in this manner is illustrated in the upper diagram of Figure 8 for the same plane of the ethene molecule shown in Figure 7. In quantum mechanics, an atom can be defined as a region of space bounded by the so-called *zero flux surface*. The zero-flux surface is the locus of points for where the gradient of electron density equals to 0, and this inter-atomic surface separates the atoms in a molecule.

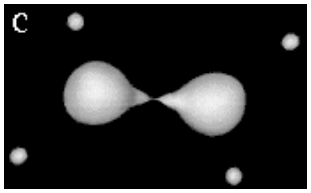

Fig. 7. spatial distribution of the electron density in the plane containing the two carbon and four hydrogen nuclei of the ethene molecule.

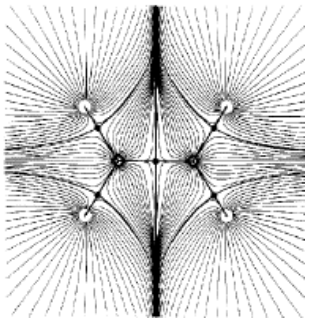

Fig. 8. A display of the trajectories that terminate at the nuclei.

The problem. Let's assume that out volume V_e contains measurements of the electron density. If we need to find the zero-flux surfaces, the first step is to create a gradient volume. Since we are interested finding where gradient equals 0, we can use the modulo of the gradient (which is a scalar field) in lieu of the complete gradient.

The sequence of volume algebra operators that determines our surface is as follows.

1. Using an affine operator to create a volume V_{exp} which is the V_e shifted by one on the positive *x* direction.
2. Using intersection to produce a gradient volume V_{gxp} which is result of subtracting V_{exp} from V_e. This gradient volume contains the complete gradient volume's positive *x* direction component.
3. Repeat step 1 and 2, except now create a gradient volume containing the gradient's negative *x* direction component.
4. Using intersection again to produce a gradient volume V_{gx} which is result of subtracting V_{gxp} from V_{gxn}. This gradient volume contains the complete gradient volume's *x* direction component.
5. Continue to produce the *y* and *z* gradient volumes.
6. Now we intersects V_{gx} and V_{gy} using cartesian product, then intersects again with V_{gz} using cartesian product. The result of these intersects is V_g.

This is synthesized in the following query condition:

$$V_g = \text{intrs}(\text{intrs}(\text{intrs}(V_{gxp}, V_{gxn}, -), \text{intrs}(V_{gyp}, V_{gyn}, -), \times), \text{intrs}(V_{gzp}, V_{gzn}, -), \times) \tag{9}$$

Now we can search for zero-flux surfaces from V_g. Since zero flux surfaces are the same as the boundaries of the volumes containing gradient $\geq$0, we can use the following query operation:

$$\text{sel}(V_g, (V_{gx} \geq 0 \wedge V_{gy} \geq 0 \wedge V_{gz} \geq 0)) \tag{10}$$

6.2 Brain neuron density comparison

Background. These days, volume data are quite common in the neurophysiological research and clinical practice because of the popularity of apparatuses such as magnetic imaging devices that produce volumetric "maps" of certain characteristics of the brain. Typical problem in research and clinical practice alike involving tracking certain features of the brain over time, since abnormal changes are believed to be connected to the insurgence of certain diseases of neural origin such as Alzheimer's disease.

The problem. A problem that can be regard as a good exemplification of the kind of queries that neuroscientists do is the following. Assume that for a group of patients, we have a number of neuron density measurements taken at a certain distance in time. Let's say that, for each patient, we have two measurements taken 10 years apart. The measurements are in a form of a volumetric model. For the sake of exposition, let's also assume we are only interested in the hippocampus. A legitimate scientific question is to analyze the region of the hippocampus in which the neuron density is abnormal (let us say that, for the purpose of this example, "abnormal" means no greater than a certain value α). A query might then be to select all the patients for which the ratio of the overlap of the abnormal regions in the two scans with the abnormal region in the new scan is less than or equal β. If a patient's condition is worse, this ration is small.

In order to solve the problem, we need to first use **cut** to pick the hippocampus from each brain:

$$V_h = \text{cut}(V_b, c_g) \tag{11}$$

(An alternative is to **intersect** with a mask volume which defines the hippocampus region: $V_h = intr(V_b, V_m, \text{nil})$.)

Now, we can use **intersection** to express the query condition. To find the desired overlap, we need to first select the part of hippocampus from each brain where the neuron density is no greater than α, then find and measure their intersection.

$$\begin{aligned} V_{n1} &= \text{sel}(V_{h1}, \text{density} < \alpha) \\ V_{n2} &= \text{sel}(V_{h2}, \text{density} < \alpha) \\ f(V_1, V_2) &= \frac{\text{vol}(intr(V_1, V_2, \text{nil}))}{\text{vol}(V_2)} \end{aligned} \tag{12}$$

(remember that $m(V)$ is the measure of a volume). Now the final query can be constructed as:

SELECT	*
FROM	patient
WHERE	diagnosis="alzheimer"
AND	$f(V_{n1}, V_{n2}) \leq \beta$

7 Conclusions

In this paper we have introduced a volume data model amenable to database use, and an algebra for expressing conditions on such volumes. The salient characteristics of our volume model are the retention, in the abstract data model, of the continuous nature of the domain in which volumes are defined, the fact that volumes are first class data types (and, therefore, that can be returned as results of queries), and the fact that our volume algebra is sufficiently general and amenable to operating on sets of volume (rather than on a volume at the time) to make it a suitable model for databases of volume data.

References

1. A. Pommert, M. Bomans, and K. H. Hohne, "Volume visualization in magnetic resonance angiography," *IEEE Computer Graphics and Applications*, vol. 12, no. 5, pp. 12–13, 1992.
2. D. Ebert and P. Rheingans, "Volume illustration: Non-photorealistic rendering of volume models," in *Proceedings Visualization 2000* (T. Ertl, B. Hamann, and A. Varshney, eds.), pp. 195–202, 2000.
3. L. A. Lima and R. Bastos, "Seismic data volume rendering," Tech. Rep. TR98-004, 23, 1998.
4. C. S. Gitlin and C. R. Johnson, "Techniques for visualizing 3D unstructured meshes," Tech. Rep. UUCS-94-018, 1994.
5. V. Ranjan and A. Fournier, "Volume models for volumetric data," Tech. Rep. TR-93-50, 30, 1993.
6. G. Nielson, "Volume modelling," 1999.
7. L. Neugebauer, "Optimization and evaluation of database queries including embedded interpolation procedures," in *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data, Denver, Colorado, May 29-31, 1991* (J. Clifford and R. King, eds.), pp. 118–127, ACM Press, 1991.
8. S. Grumbach, P. Rigaux, and L. Segoufin, "Manipulating interpolated data is easier than you thought," in *The VLDB Journal*, pp. 156–165, 2000.
9. B. S. Lee, R. R. Snapp, L. Chen, and I.-Y. Song, "Modeling and querying scientific simulation mesh data," Tech. Rep. CS-02-7, February 2002.
10. T. T. Elvins, "A survey of algorithms for volume visualization," *Computer Graphics*, vol. 26, no. 3, pp. 194–201, 1992.
11. A. P. Marathe and K. Salem, "A language for manipulating arrays," in *The VLDB Journal*, pp. 46–55, 1997.
12. L. Libkin, R. Machlin, and L. Wong, "A query language for multidimensional arrays: design, implementation, and optimization techniques," pp. 228–239, 1996.
13. R. H. Gting, "Spatial database systems."
14. M. Chen, A. Winter, D. Rodgman, and S. Treavett, "Enriching volume modelling with scalar fields," 2002.
15. M. Chen and J. V. Tucker, "Constructive volume geometry," *Computer Graphics Forum*, vol. 19, no. 4, pp. 281–293, 2000.
16. J. D. Foley, A. V. Dam, and S. K. Feiner, *Introduction to Computer Graphics*. Addison-Wesley Pub Co, 1 ed., August 1993.
17. W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *Proceedings of the 1987 ACM Conference on Computer Graphics (SIGGRAPH'87), Anaheim, USA, pp. 163-169, July 1987*, pp. 163–169, ACM Press, 1987.