Building a Registered Volume Database: an Object-Oriented Octree Program

Lynn W. Jones
Virginia Polytechnic Institute, Blacksburg, Virginia
lwjones@vt.edu

Abstract- The Dynamic Brain Project will create an interactive database on the human brain, available via the Internet. In addition to medical images and textual information, the completed project will give users access to all types of data, including video and sound. One of the primary tasks is to create a point-and-click navigation interface for the user, giving access to the data through a 3-dimensional, volumetric image generated from medical scans. The project uses Postgres, an object-oriented database management system. Data relations (tables) are spatially linked to the interface image by the x,y,z-coordinates to which they belong. These coordinates are determined by building the volume with an octree data structure and outputting the spatially registered data to a Postgres table. This paper summarizes the database schema and the octree algorithm for the project.

The Dynamic Brian Project (DBP) [3] proposes to create an interactive database of multimedia information on the human brain to be accessed over the World Wide Web. The point of entry will be a 3-dimensional, volume image of the brain. The user will have both a "point and click" and a structured query interface. With point-and-click browsing, he or she may visually navigate through regions of the brain, whereas a structured query will search for index terms relating to brain structure name, function, connectivity, pathology and the like. Either access method might generate a new volume image showing the requested information, as well as retrieve data of varied formats: 2- and 3-dimensional images; text; video and sound clips; spreadsheet and statistical data; or links to any of these types residing on other Internet servers. The DBP will employ Web tools such as Java, HTML (Hyper Text Markup Language) and VRML (Virtual Reality Modeling Language).

Principle Investigators of the DBP are Dr. Terry Huntsberger of the Computer Science Department at the University of South Carolina and Dr. James Augustine of the School of Medicine at the University of South Carolina. Other contributors include Dr. Caroline Eastman and Dr. John Rose. Through the Computing Research Association Distributed Mentorship Project, I was able to work with Dr. Eastman on the DBP for eleven weeks in the summer of 1996. Much of my research involved the initial data organization to create a volumetric database. This paper discusses database schema for the project and the octree implementation which generates the spatially-registered, object-oriented table of information on the human brain.

Our view of the brain is that of an inherently hierarchical structure. Each general area of the brain is composed of parts with increasingly specific names, characteristics, and functions. Most people are familiar with terms such as frontal lobe, temporal lobe, etc. Each of the lobes is a construct of its lobules, which are comprised of their gyri, and so forth. It is expected that a user would also approach the data in an hierarchical way, first browsing or requesting fairly general information and becoming more specific as he or she becomes more knowledgeable. The top level of the DBP is the initial image: a "solid 3-dimensional model of the cerebral hemispheres with the five lobes color coded and identifiable" [3]. The user selects a lobe to view in more detail, and the DBP system creates an image based on the next level in the hierarchy.

This natural, hierarchical structure and the need to support mixed-media data both suggest the use of an object-oriented database management system (DBMS), which "stores and manages objects [and] can easily handle 'unconventional' data types and highly interrelated data" [5]. Preliminary efforts employ Postgres, an object-extended, relational DBMS, which supports SQL and also provides indexing and query optimization [8]. The complete DBP database will include class objects and relationships which model the brain's structural hierarchy. Class declarations based on brain structure and function [1] as proposed by Dr. Eastman are shown in Figure 1. The REFERENCE class will record metadata or identification of medical images. Links to further information may also be provided here or in additional classes. Querying these tables of objects could generate a new image or might present mixed formats of related information to the user.

The volume image displayed by the DBP will be generated from MRI (magnetic resonance imaging), CT

(computerized tomography) and PET (positron emissions test) scans. To provide an interactive, navigable interface, the class objects representing structures and functions will be registered to their physical locations within the volume image. A VOLUME object class will be created to hold this information. Its definition is shown in Figure 2. MRI-, PET-, and CT-value data members record color values for each object. Level refers to the level of resolution in the information hierarchy. The scans will be mapped with structure and function names which are also recorded in each VOLUME object. Because a region of the brain will belong to overlapping classifications, from general to specific, these may be entered in list form or may be coded to indicate a particular grouping of structures and functions. Data member Corners is a set of eight vertices of a cube in x,y,z-space. The collection of VOLUME objects will become a Postgres table indexed by location coordinates and by structure and function names. To create the image of a particular brain structure, the query will search the VOLUME table for the requested key(s) and return a list of corners and color values. A separate algorithm will format the points for rendering in VRML. A user's click in the image will query the table for the object at that position in the volume. The retrieved structure names and functionality can be used in an SQL join to obtain information in the other Postgres tables.

Once the database schema has been determined, we look at the task of entering the data. Information for relations shown in Figure 1 may be added using regular Postgres commands and SQL statements. For the VOLUME relation in Figure 2, however, some of the data is not yet known. We have medical scan data as a starting point. Each cross-sectional scan produces a 2-D array of values, which determine display color at each x-y coordinate of the image. Structures and functionality are not clearly delineated in the scan images, so these must be manually identified and mapped on each slice. A very basic mapping was made for this preliminary research. Recent advances in scan technology permitting simultaneous PET and MRI scans [7] may in the future allow automation of the mapping process

VOLUME
{
MRI_Value
PET_Value
CT_Value
Level
Is_Structure
Has_Function
Corners
}

Fig. 2. Class definition for object-oriented octree nodes to be output to the Postgres object relation.

using pattern recognition techniques. To assign Corners data to each object, we must construct a 3-D volume from the cross sections. The octree data structure was chosen to represent this volume, as it supports efficient, hierarchical organization and access of spatial data [2]. From the octree, we will also determine the value for Level for each object.

An octree, as the name implies, is a tree structure in which each internal node has exactly eight children. It recursively decomposes space into cubic volumes [6], as shown in Figure 3. An internal node in the tree represents the eight sub-cubes, or child nodes, contained within. The extent of decomposition corresponds to the level of resolution in the image and to the level of hierarchy or containership in the data. In many implementations, a nodes location in the volume is determined from its location in the octree. Using a database management system, however, the DBP will not maintain the octree but rather use it to create registered VOLUME objects (the cubes represented by each octree node) to import into a Postgres table. For this reason, we will explicitly store the location of each object by recording the vertices of the cube it describes.

We start with 64 MRI scans, each a 64x64 array of unsigned characters (bytes) with a range in value from 0 to 255. The PET and CT scans were not yet available when this work was done. Metadata headers are stripped from each MRI scan and the files concatenated into one file. The most general structure and function names were identified

Fig. 1. DBP class definitions for related Postgres tables.

STRUCTURE	CONNECTION	FUNCTION	PATHOLOGY	REFERENCE
{	{	{	{	{
Name	Name	Name	Name	Bibliography
Part_Of	Source	Structure	Structure	Structure
Next_To	Destination	Subfunction	Function	Function
Connection	Type	Pathology	Reference	Pathology
Function	Afferent	Reference	}	Species
Reference	Efferent	}		Methodology
}	Projection			NumberOfSubjects
	Reference			Findings
	}			}

on each scan, creating two additional files of unsigned character values. The octree program first reads the values from the three files into a single 64x64x64 array. The volume array is recursively subdivided into octants, first 32x32x32, then 16x16x16, and so forth. At each division, if the octant size remains greater than 1x1x1, an internal octree node is created, that octant is again subdivided, and a smaller octant is given to each of the node's eight children. When the octant size reaches 1x1x1, a leaf node is allocated. Scan, structure and function values are read from the array, and the node records its coordinates and its depth in the tree. As the recursive calls return, each internal node sets its data from its children's data (see Figure 4). An internal node's MRI value is the average of the eight child values; the structure and function names are the union of those contained in the children. Parent node corners are also set: corner number 1 of the parent is the first child's corner number 1; corner 2 is the second child's corner 2, etc. The parent level is one less than the child's level. The initial call to the octree-building function returns a pointer to the root, a node which represents the entire volume at very low resolution.

At this point we have an octree of the complete sets of data, and we begin to take advantage of the octree structure by "pruning" out the similar nodes which can be sufficiently represented by their parents. The pruning function visits each parent of leaf nodes, deciding whether the eight children are similar enough to be represented by the parent alone. The MRI values are tested to determine if each child's value is within a certain range of the parent value. The structure and function identifications are compared for equality. If the children have the same identifications and all eight contain scan values falling within the given range, they are removed from the tree. The pruning operation could have been done in the recursive building function; however, in order to observe its effects, it was written as a separate routine called only on parents of leaf nodes and not propagating upward through the tree. More than four calls of the prune function had no further effect on the size of tree; i.e., no node collapsed more than four levels. As the structure and function identifications become more detailed and the nodes more dissimilar, we will need to preserve the deepest nodes for maximum resolution, and two or three calls of the pruning function should suffice. After pruning, a level-traversal of the octree is made and the contents of each node are written to a file. These are then imported into a Postgres relation (table) of objects, and Postgres manages persistence and access. Recording each node's coordinates in the volume and depth in the tree allows us to manipulate the Postgres table in a fashion comparable to tree traversals.

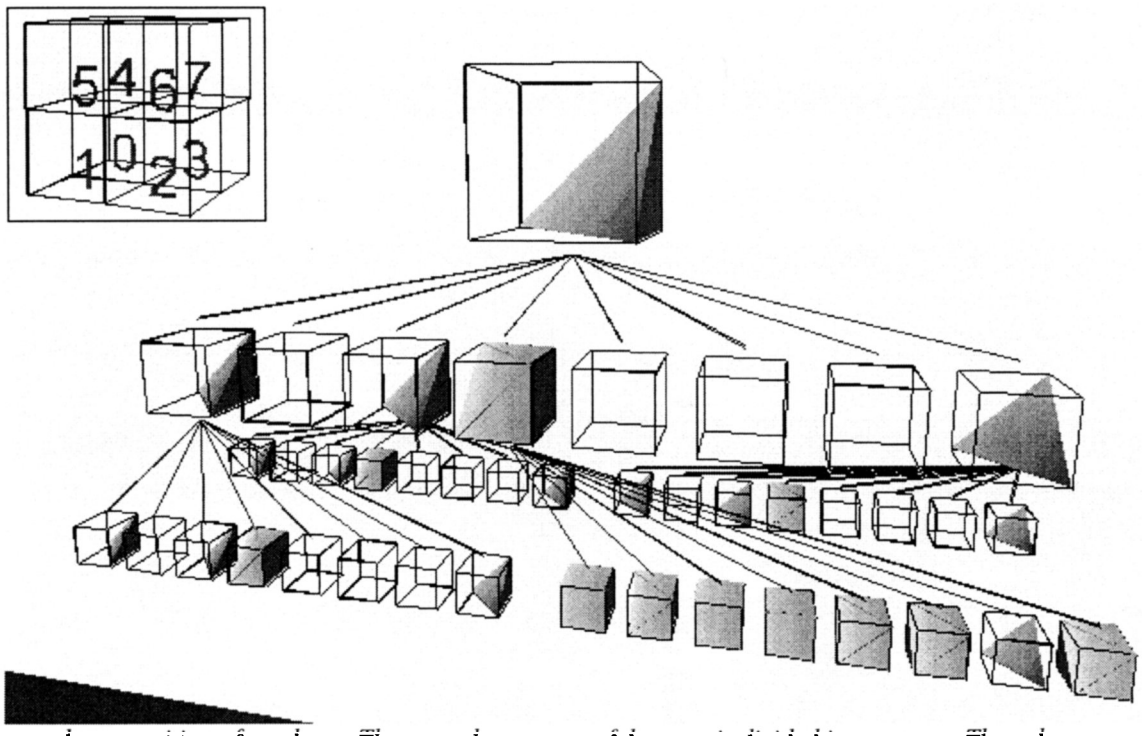

Fig. 3. Octree decomposition of a volume. The top cube, or root of the tree, is divided into octants. The volumes contained in each octant are represented by the child nodes, 0-7. Four of the children are empty and will not be divided further. Partially filled cubes are again decomposed. Cube 3's children include three full nodes, children 0, 2, and 3, which will also not be divided. Decomposition of partially filled cubes continues until a stopping measure such as tree depth, pixel-level resolution, or other threshold is reached.

The pruned octree is efficient because it effectively stores the same information as the full tree but in fewer nodes. Figure 4 uses empty and full cubes to illustrate how decomposition is stopped when a node satisfactorily represents its entire volume. Reducing the number of nodes in the tree reduces the size of the VOLUME table, the number of accesses needed to read or search the file, and the amount of data the DBP must transmit over the Internet. We take advantage of the octree's hierarchical organization as well. Traversing an octree by levels results in "successive refinements that increase the resolution of the object's details" [4]. Presumably, at some point in the traversal, the amount of detail is sufficient to present the desired information, and we are satisfied with the level of resolution rendered. Retrieval from the tree benefits in two ways. When an approximation or general information is requested, only nodes close to the root of the tree need be visited. On an indexed search, the decision to follow a path from parent to child eliminates a substantial portion of the tree from further searching [2]. We maintain this property in the Postgres table by using Level as a key field, retrieving only those objects at a particular level of resolution.

The octree node relation establishes a spatial, or resolution, hierarchy for rendering an image of the volume data it represents. The algorithm imposes constraints on the input, in that the x, y, and z dimensions must be equal (a perfect cube) and that they must be a power of two. The DBP will ultimately use scans sized 256x256; however, there are only 64 of these files. A preprocessing module will generate, possibly by interpolation, three additional cross-sections between each scan (another option is to pad the file with null or background values which will be pruned from the tree anyway). The program has been tested on a 64-cubed data set. Using the MRI scans and the sparse structure and function identifications described, pruning reduces the number of nodes by nearly 53%, from 299,593 nodes to 139,929 nodes. It is necessary to test on large data, but because of the number of nodes it is difficult to verify correctness of the output until the volume image can be rendered.

The algorithm itself solves the problem of generating and spatially registering data to a volume. As the project develops, test conditions for pruning nodes will be verified and decisions will be made on how to effectively overlay new indexing information on the existing nodes. Much remains to be discovered about the brain. As new structures and functions are mapped, the octree program can be used to update the database volume. Provided new files are registered to the original scans, the program can build a registered volume from any combination of input files. Newly registered objects can be output complete with Corners and Level data in the same way as the initial

Parent Node

Parent Node

MRI_VAL	LEVEL	IS_STRUCTURE	HAS_FUNCTION	CORNERS
121	3	lobe_A	motor_skills, speech, language	0,0,0; 2,0,0; 2,2,0; 0,2,0 0,0,2; 2,0,2; 2,2,2; 0,2,2

Child Nodes

#	MRI_VAL	LEVEL	IS_STRUCTURE	HAS_FUNCTION	CORNERS
0	120	4	lobe_A	motor_skills	0,0,0; 1,0,0; 1,1,0; 0,1,0 0,0,1; 1,0,1; 1,1,1; 0,1,1
1	121	4	lobe_A	motor_skills	1,0,0; 2,0,0; 2,1,0; 1,1,0 1,0,1; 2,0,1; 2,1,1; 1,1,1
2	118	4	lobe_A	motor_skills	1,1,0; 2,1,0; 2,2,0; 1,2,0 1,1,1; 2,1,1; 2,2,1; 1,2,1
3	119	4	lobe_A	motor_skills	0,1,0; 1,1,0; 1,2,0; 0,2,0 0,1,1; 1,1,1; 1,2,1; 0,2,1
4	120	4	lobe_A	motor_skills	0,0,1; 1,0,1; 1,1,1; 0,1,1 0,0,2; 1,0,2; 1,1,2; 0,1,2
5	122	4	lobe_A	speech	1,0,1; 2,0,1; 2,1,1; 1,1,1 1,0,2; 2,0,2; 2,1,2; 1,1,2
6	123	4	lobe_A	speech	1,1,1; 2,1,1; 2,2,1; 1,2,1 1,1,2; 2,1,2; 2,2,2; 1,2,2
7	123	4	lobe_A	language	0,1,1; 1,1,1; 1,2,1; 0,2,1 0,1,2; 1,1,2; 1,2,2; 0,2,2

Fig. 4. Example of data in a parent node and its eight children. These child nodes would not be pruned from the octree because, though the MRI_Values are within a close range of the average (parent) value and all eight represent the same brain structure, they have different function identifications.

objects. The updated information can then be inserted into the correct VOLUME objects using Postgres commands. The program may be easily modified to read additional files, and it is likely that the VOLUME objects will be completely rebuilt periodically as researchers discover finer details about the brain.

Programming efficiency was not a primary concern at this stage of the project. The octree program is used only to create the volume and will not influence interactive performance of the Dynamic Brain Project database. One performance issue to be explored, however, involves the storage of points in each VOLUME object. An alternative is to store a single corner rather than the eight Corner coordinates. The cube's size may be determined by its level, and from this we may calculate the other seven corners. This is a substantial savings in file size, and if the data is to be transformed for VRML on the user's processor (using Java), it would also require the transfer of much less data. Improvement really depends on how the VRML transformation is done and on the average number of nodes used for each image.

References:

[1] Augustine, James R. Circuitry and functional aspects of the insular lobe in primates including humans. In review. 1996.

[2] Carlbom, Ingrid, Indranil Chakravarty, David Vanderschel. A hierarchical data structure for representing the spatial decomposition of 3-d objects. IEEE Computer Graphics and Applications, Vol.5, April 1985, pp. 24-31.

[3] Huntsberger, Terry and James Augustine. Dynamic brain project (proposal). Columbia, SC: University of South Carolina, 1996. Submitted to National Science Foundation.

[4] Kunii, T.L., T. Satoh, K. Yamaguchi. Generation of topological boundary representation from octree encoding. IEEE Computer Graphics and Applications, Vol. 5, March 1985, pp. 29-38.

[5] Loomis, Mary E.S. Object Databases: The Essentials. Reading, Massachusetts: Addison-Wesley Publishing Company. 1995.

[6] Samet, Hanan. Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS. Reading, Massachusetts: Addison-Wesley Publishing Company. 1990.

[7] Service, Robert F. New dynamic duo: PET, MRI, joined for the first time. Science, Vol. 272 (June 7, 1996), p. 1423.

[8] Stonebraker, M. and G. Kemnitz. The Postgres next-generation database management system. Communications of the ACM, 34(10): 78-92. October, 1991.