University of Alberta

Efficient Query Support for a Brain Tumour Database

by

Jonathan Levesque

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Edmonton, Alberta
Fall 2007

Your file Votre référence
ISBN: 978-0-494-33294-8
Our file Notre référence
ISBN: 978-0-494-33294-8

To my parents Rejean & Denise and my sister Tiffanie, all hard-working people who have helped me become the person I am today.

Abstract

This thesis details the implementation of a brain tumour imaging database. The database server necessity is two-fold: it must support researchers seeking to determine the growth patterns and other properties exhibited by brain tumours, and must support physicians requiring data to formulate treatment plans as well. This project is undertaken as part of the Brain Tumour Analysis Project, whose members will be the first to benefit from the database server, gaining easy access to data for theory validation. The system presented follows the client-server model and provides secure encrypted transmission of data between the two ends to maintain medical record security. An easy-to-use front-end client has been created to abstract away any implementation details and allow a physician (the target user) to intuitively find data applicable to their treatment plan. A new data structure, called the Volume Distribution Tree, for the efficient processing of Jaccard queries is described.

Acknowledgments

I would like to thank my supervisor Dr. Jörg Sander for being available to discuss my ideas and review my work. As the oncologist in the Brain Tumour Analysis Project, Dr. Albert Murtha of the Cross Cancer Institute has provided very valuable insight and ideas to my research. Thanks also to Bret Hoehn of the AICML for showing me how to obtain and process raw data on the BTAP disk space, as well as the BTAP members in general for their feedback during meetings.

Table of Contents:

List of Tables:

List of Figures:

Chapter 1: Introduction

1.1 Motivation

The general location of a brain tumour is discernable on an MRI scan, but the true extents of the cancerous cells are undeterminable using current imaging technology [6]. Cancerous cells not showing up as part of the tumour are called occult cells and must be destroyed to avoid re-growth of the tumour after treatment. Physicians must therefore treat (kill the included cells using conformal radiation) a larger volume than that apparent with imaging in order to properly mitigate the risk of relapse. The accepted rule of thumb is to take a 2-centimetre margin around the volume segmented from an MRI scan.

The problem with this strategy is that tumours do not tend to grow spherically and thus the treatment is likely to destroy healthy cells contributing to the patient's functionality and quality of life. Occult cells having grown beyond the treatment margin also present a high risk of relapse. Based on evidence from many years of radiation treatments, physicians allow limited radiation doses to patients, measured in gray (Gy), a measure of radiation energy absorption per unit mass (1 Gy = 1 J/kg). This limit implies that there is a budget of radiation treatment available and that savings in one area that we can determine does not need treatment can correspond to a higher allowance of treatment of a higher risk area.

A major goal of the Brain Tumour Analysis Project (BTAP) at the University of Alberta is to predict the growth pattern of tumours such that the treatment focuses on high-probability cancer-containing areas and ignores areas where it is almost impossible for cancerous cells to have infiltrated. A physician having noted a few instances of tumours in a certain location and none of which grew beyond a certain membrane or in a certain direction may predict the same to occur with a new patient, but may not be confident in that assumption due to the limited cases upon which to base the assumption. The work detailed in this thesis aims to augment such estimates using a large collection of data, rendering more statistically significant predictions.

1.2 Problem Definition and Scope

An oncologist planning a patient's treatment will sometimes edit the common 2-cm margin based on prior experience. These edits are generally cuts to the treatment edges due to impassable barriers like bones and the Falx line. Having a human sort through thousands of MRI studies to find similar cases is unfeasible and unmanageable. A database is the natural choice for the storage and management of large datasets, but the tables will have to be well laid-out for efficient retrieval. The oncologist will generally be looking for cases similar to the one in question in order to get a sense of how this tumour is likely to grow and where the best chance of finding occult cancer cells are. Finding similar data necessitates similarity measures conducive to the retrieval of what expert oncologists would agree is relevant. Once the similarity measures are implemented, a reasonably efficient manner of indexing the data with respect to that measure must be defined and implemented to avoid linearly scanning the whole database to resolve each query.

Researchers coming up with theories regarding the growth of tumours will also want data to back up or refute their theories. Postulating a theory involving a drug or therapy would take several years to complete, and even then would have a small dataset due to the shortage of volunteers, ethics approvals and other obstacles. There are, however, theories that can be validated using existing data, such as tumours not being able to grow through certain regions or preferentially growing through certain classes of brain matter. With some data, say the data from a few years at a hospital, a small amount of results can provide support for a theory. With a very large amount of data, collected from hospitals all over the world, researchers can place much more confidence in their results. Thus the long-term goal for this database is for medical research centers worldwide to contribute brain imaging of tumour-stricken patients via internet communication. These research centers would also make use of the database to aid in their cancer research. The short-term goal is to use and contribute to the system locally within the Cross Cancer Institute and the University of Alberta.

Predicting the growth of tumours is beyond the scope of the current database project. This project is limited to setting up a database to house the data and provide the necessary facilities to access relevant data without any SQL knowledge and to do

so remotely without compromising patient privacy. This will lay the groundwork for further research by the BTAP and act as its common repository for data, avoiding copies in multiple locations. New data will be able to be seamlessly integrated into the query system without the user having to update any software. New members to the group can be given Matlab and Java connection modules to be abstract away from the data storage, enabling them to focus on their scope of work. The work presented here does not focus on improving a specific technique such as modelling tumour growth; the goal is instead to create a solid software base to provide others with an invaluable tool for their research. Now a researcher modelling tumour growth has access to a queryable data repository and can focus on inventing and testing models rather than spending time finding data and on system implementation details.

1.3 Introduction to the Technical Terms in this Document

This document uses some acronyms and scientific terms, so this subsection will give a little background on these. A glossary is also available on page 86 as a quick reference when reading the text.

1.3.1 Brain Tumour Anatomy

Cancerous tumours growing in the brain are called gliomas and have several common hallmarks. The live tumour cells have a metabolic rate much higher than normal cells; the outer part of the tumour uses up too much of the available oxygen and nutrients, causing cells deeper into the tumour to die. This portion of the tumour is called the necrotic core as it is a cluster of dead cells. The tumour cells' very large division rate causes it to increase in size, putting pressure on the rest of the brain as the space available is confined by the skull. This causes swelling of the brain tissue, called edema, around the tumour.

1.3.2 Brain Imaging Techniques

When suspected of having any abnormalities, a patient's brain is generally imaged using MRI (Magnetic Resonance Imaging) and sometimes MRS (Magnetic Resonance Spectroscopy), PET (Positron Emission Tomography) and DTI (Diffusion Tensor Imaging). All of these are non-invasive means for a physician to see parts of the patient's brain anatomy and chemistry to help with diagnosis and treatment.

Although the future plan for the database involves storing all of these types of images and more, the current system only deals with MRI images. MRI really only shows the macroscopic brain anatomy as opposed to the more refined DTI which traces nerve tracts and MRS which shows the distribution of brain chemicals. This however generally suits the purpose of finding tumours as these show up (for the most part) in MRI.

An MRI machine houses a large electromagnet capable of creating a very strong (usually 1.5 Teslas or greater) and uniform magnetic field. The imaged patients are primarily composed of water and thus, there are vast quantities of hydrogen protons inside their tissues. Each hydrogen proton spins about its own axis, and this spinning proton creates a small electric field and thereby produces a perpendicular magnetic field. While the orientation of these small magnetic fields is usually random, once in the bore of an activated MRI machine's magnet the rotation axes line up with the magnetic field. When a radio frequency (RF) pulse at a precisely calculated frequency is then delivered, this forces the hydrogen protons to precess about the magnetic field axis. When the RF pulse is stopped, the precession of the protons gradually decays as the protons once again align their spins with the magnetic field. During this decay, the protons move to a lower energy state and in doing so emit energy in the form of an RF signal which can be detected and measured. The rate of decay is dependent upon the relative chemical composition of the tissue imaged. This information can be used to render a depiction of the tissues imaged.

There are 3 modalities in which MRI images are taken, T1-weighed, T1C-weighed, and T2-weighed. T1 images show water as dark and lipids brightly, making white matter appear white, grey matter appear in various intensities of gray and brain fluids to appear black. T2 images show the opposite response, with lipids appearing dark and fluids showing up brightly. Bones are dark in both cases; although there appears to be a skull outline to both sets of images, this bright outline is actually the skin covering the skull containing both lipids and water. T1C imaging is just T1 imaging performed after the patient has been injected with the contrast agent gadolinium. This agent causes only the tumour to have an increased intensity, greatly helping tumour segmentation.

1.4 Outline

A database of brain imaging and patient data is being assembled both to aid clinical treatment planning and for research use. This document outlines the status of the database work as it stands, the current similarity queries supported by the system, and the client software provided to the clinicians, as well as the security permitting client access without compromising sensitive data.

The primary contributions of this project are:

- Simple segmentation procedures.
- New similarity measures:
    - Depth Jaccard
    - Ray Trend
    - MidSim
    - Eigen Decomposition
    - Elongation and Flatness
    - Growth Direction
- A new tree structure to more efficiently support Jaccard-type queries.
- A working set of database tables and a secure client-server implementation including client software that is easy-to-use by physicians.
- A means of easily testing tumour growth hypotheses on a large quantity of data.
- A remote, encrypted query system that is extensible by non-expert computer users with only Matlab programming knowledge.

The remainder of this document is organized as follows:

- Chapter 2 summarizes and comments upon literature related to this work.
- Chapter 3 discusses simplified segmentation approaches devised to help the influx of data.

- Chapter 4 goes over the numerous similarity measures used in data retrieval as well as the methods used to efficiently support queries employing these similarity measures.
- Chapter 5 gives a detailed explanation of the Volume Distribution Tree.
- Chapter 6 gives a high-level overview of the system architecture followed by a focus on each major component.
- Chapter 7 presents experimental results.
- Chapter 8 discusses future work ideas.
- Chapter 9 summarizes the work performed.

Chapter 2: Related Work

The system detailed in this thesis encompasses technology fitting into several research fields – tumour segmentation, object similarity measures, and spatial database indexing. Other work that can be compared and contrasted with ours is the topic of this chapter. Commentary regarding the works' relation to and influence on this project is provided.

2.1 Tumour Segmentation

2.1.1 Automatic Brain Tumour Segmentation

The segmentation algorithm currently used by the BTAP group is discussed in the paper "Segmenting Brain Tumors using Alignment-Based Features" [25] and as part of a pipeline including intensity standardization and registration in the Master's thesis related to the aforementioned paper [24]. Although the segmentation ideas presented in Chapter 3 do not supersede this work as the group's standard segmentation algorithm, they are presented in this thesis as simple ideas that could be used in a registration framework that avoids warping the original data. Here we summarize Schmidt's work as it not only contrasts with the segmentation ideas presented here but actually serves as the accepted method of segmentation, providing the tumour labels currently stored in the database.

Schmidt *et al.* employ machine learning to perform automatic brain tumour segmentation. The feature sets used for the machine learning task are alignment-based features. The four features extracted from the training images were the brain mask (B set), the a priori tissue type probabilities (P set), average intensities (A set), and left-to-right symmetry (S set). The features were all extracted at multiple resolutions to include neighbourhood information along with voxel-level (localized) information. The segmentations provided on the test cases after training are compared with human expert-drawn labels using the Jaccard measure, with a higher score denoting a better automatic segmentation as it corresponds well with the human expert labelling, which is as good a ground truth as is available. All combinations of

the four feature sets were tested to see which would provide the best segmentations, and these experiments found the multi-scale PAS features to provide the best score, tied with the multi-scale BPAS features. The PAS features would therefore be declared the best option as adding the B feature to this set did not help the final result. In the current incarnation of the automatic segmentation program described in Schmidt's thesis, the alignment-based, or coordinate-based, features are supplemented by image-based features, registration-based features, and feature-based features. Image-based features are extractions of brain structures, textures, the intensities of pixels and their neighbourhoods, and image histograms. Registration-based features include using information from prior segmentations of the same patient and warping fields calculated during registration with a template. Feature-based features involve using multiple resolutions, gaining neighbourhood information, or using a subset of other features as a feature.

2.2 Object Similarity Measures

2.2.1 The Jaccard Coefficient

A popular similarity measure is the Jaccard coefficient [12]. To obtain the similarity between two sets, the number of intersecting elements is divided by the number of total unique elements in the two sets combined. Disjoint objects have no intersection and thus score a 0 on the Jaccard measure. Any intersection raises the Jaccard score. For two identical objects, both the intersection and union are equal to either object's volume, yielding a maximal similarity measure of 1. The Jaccard coefficient is within the range [0,1] with increasing values denoting higher similarity.

2.2.2 Shape Histograms

From the centerpoint of an object, the object is split into sectors and shells, with the volumes in each division being inserted in a histogram used to describe the shape [2]. The center could be defined in many ways, but the object's center of mass is used to locate the center in [2]. Sectors are pie-shaped divisions like dividing a pie into 12 or 16 sections, whereas shells are divisions made at specified radii (see Figure 2.1). Combining these two puts a 2D grid resembling a spider web over an image of the object.

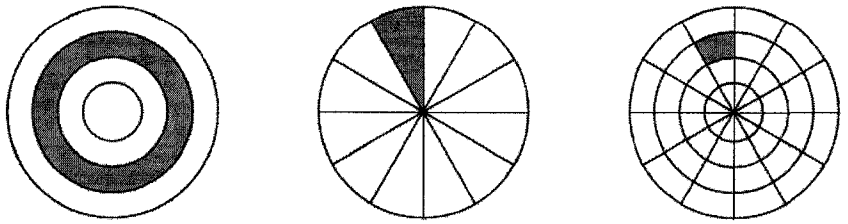


Figure 2.1: Shape Histogram Shells and Sectors [2]

This grid can be extended to 3 dimensions by splitting into sectors all 360 degrees around one axis, again splitting into sectors for 180 degrees around an orthogonal axis, and having 3D shells which are now splits along a spherical shell of a given radius. This is a discretized version of continuous spherical coordinates, with one set of divisions along the 360° azimuth range, one set along the 180° zenith range, and finally one set along the linearly outward radius range. The volume occupied by the object in each grid cell is entered as one element of a feature vector, leading this vector to be of length (sectors)(shells)^2/2.

The distance from other vectors is found using quadratic form distance instead of Euclidean distance, in order to take into account that the last sector around a circle is actually very close to the first. The form of this distance measure is $d_A^{\,2}(x,y) = (x-y)A(x-y)^T$, which would be Euclidean in the case where the weighing matrix A is the identity matrix, or would be weighed Euclidean if it is diagonal with the diagonal values being the weights. By having nonzero values outside the diagonal, these weigh in a bin's neighbours along with itself. The weight matrix used and suggested by the Shape Histogram creators is the matrix A with elements $a_{ij} = e^{-\sigma \cdot d(i,j)}$ with σ between 1.0 and 10.0 and d being the distance between bins (for example with 16 divisions, sectors 1 and 16 are neighbours and thus have a distance of one between them). A higher σ is closer to the identity matrix I and therefore closer to Euclidean distance. We actually implement the shape histogram idea with 3 shells, 16 sectors and σ=1.0 as one of our similarity measures for similar tumour retrieval.

2.2.3 Other Similarity Measures

Although we only implement the Jaccard measure and Shape Histograms in this work, many other object similarity (and distance) measures exist in the literature. In [10], the similarity between solid objects is computed by comparing their Reeb graphs' R-node lengths and areas. In [5], objects are recursively decomposed into smaller components, yielding a tree where the leaves are the smallest decomposition done by the algorithm and a parent is the union of its children before comparing these components' Reeb graphs. Oriented bounding boxes, which are bounding boxes aligned with the principal axes of an object, are organized into OBB (Oriented Bounding Box) trees in [13]. A distance measure can be obtained using two objects' OBB trees by computing the sum of the Euclidean distances between the (x,y,z) positions indexed by the nodes. In [1], a model is meshed and then one node is put into a first group, its neighbours are put into a second group, and so forth with each group forming a conical 'wavefront'. The Cone-Curvatures' Euclidean distance is taken as a distance metric separating the objects. In [19], a solid object is decomposed into positive and negative polyhedrons which make up the original object by addition. The distance measure is taken as the norm of 7 distances: mean value and standard deviation of gaps between matched vertices projected on a sphere around the object, distance between matched components' centroids, difference in volumes, and difference in angle between the x, y, and z directions of the components. 3D models are posed from 42 different viewpoints in [20]. These each lead to a 'depth image' which is then converted to a polar coordinate image (each pixel (x,y) is instead plotted in its (r,θ) coordinate), which is then Fourier transformed. This creates a 2D matrix of values for each image, and the distance metric is the sum of differences between matrices.

2.3 Spatial Database Indexing

2.3.1 The R-Tree and its Variants

The R-Tree is an index structure originally meant for low-dimensional spatial data; generally 2-4 dimensions [9]. The objects in a database are amalgamated in a hierarchical manner. That is, objects that are close together are grouped together into

one rectangular area encompassing them all; this is the combined objects' minimum bounding rectangle. A larger rectangular area R2 groups together several smaller rectangles R1a, R1b... to make a more generalized node acting as the parent to these R1's (see Figure 2.2). Continuing this strategy, the root node of an R-Tree contains the whole space of objects in the dataset, whereas each of its children's rectangles only contains part of the set of objects. Note however that rectangles on the same level are not necessarily disjoint; they may overlap and thus one extended object may be partly covered by more than one rectangle.

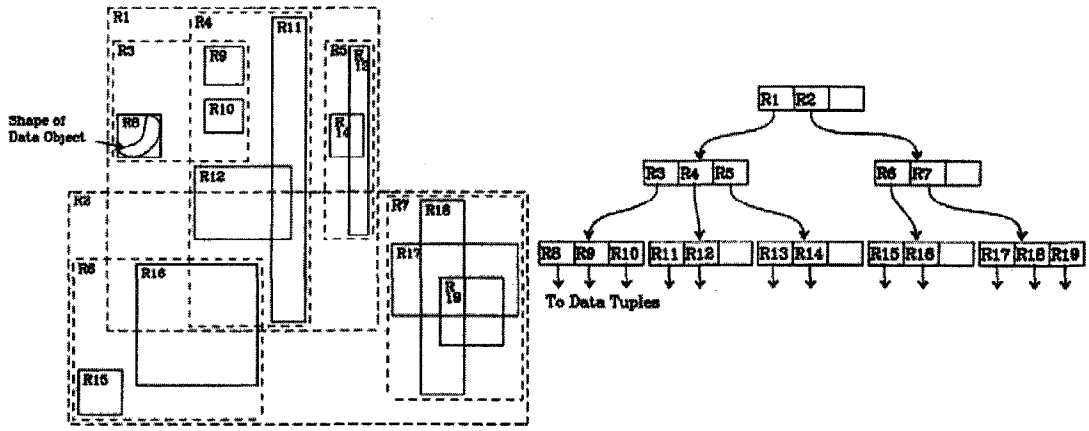


Figure 2.2: R-Tree: Object Groupings and Corresponding Tree [9]

A query using the R-Tree would typically be a two-dimensional window along with a desired relationship, such as that the query window touches the objects returned or that it entirely contains them. A query could also be a small window or point to be contained within an object in the database. At each node expanded, only the children whose MBRs meet the query criteria need be examined further, with the rest being pruned.

A variation of the R-Tree is the R+-Tree [26], where overlap between nodes is disallowed. Due to this property, when descending an R+-Tree with a point query no more than one path needs to be followed (whereas in a regular R-Tree, the point may be covered by several sibling nodes' rectangles). This makes R+-Trees advantageous for point queries despite the added complexity inherent in keeping the rectangles disjoint while still indexing all objects in a dataset.

Another variant is the R*-Tree, which is effectively an R-Tree with a revised node splitting strategy [3]. During the insertion of a new object into any R-Tree variant,

the node housing the new object may overflow (have more children than allowable by design). When this overflow occurs, the strategy taken by the R*-Tree is to re-insert selected rectangles in the overfull group in order to optimize the placement of rectangles to get better performance out of the tree. The authors identify the primary criteria for optimizing the R-Tree structure: minimizing the dead space within a rectangle, minimizing the dead space between rectangles, minimizing the perimeter of a rectangle, and maximize the number of children for each node, although this last goal competes with the first three. Although in construction of an R-Tree these parameters could be optimized, in a dynamic update situation the R-Tree tends to lose some performance due to suboptimal organization and this is where the modifications in the R*-Tree help.

Although the R-Tree can technically handle data of higher than 2 dimensions using hyper-rectangles, its performance rapidly degrades with increasing data dimensionality due to the hyper-rectangles naturally intersecting each other by large amounts due to the 'curse of dimensionality'. An early R-Tree modification to help with multidimensional indexing issues was the Similarity Search Tree (SS-Tree) [29]. The SS-Tree directory nodes contain children clustered within ellipsoids rather than hyper-rectangles. Rather than nodes spanning a range of values in each dimension like R-Trees, the SS-Tree nodes hold objects within a set threshold distance from its representative point. This would make spherical regions for a Euclidean distance metric but the authors instead use a weighed-Euclidean scheme resulting in ellipsoidal regions having principal axes that are aligned with the space's dimensional axes.

To further improve the ability of databases to index higher-dimensional data (above about 5 dimensions), the X-Tree (eXtended node tree) was created by extending the R-Tree [4]. The X-tree permits 'supernodes' to form, meaning that overfull nodes are allowed to remain that way in order to avoid splits that would cause large overlaps. In the X-Tree, splits are only allowed if they cause no more than a preset amount of overlap. The authors prove that an overlap-free split (for point data) is possible if and only if there exists at least one dimension that no MBR spans. This is because if all dimensions have at least one spanning MBR each, for

each dimension an MBR other than the spanning one has some length in that dimension, overlapping the spanning one. A split-history tree is maintained to keep track of the splits, as shown in Figure 2.3.

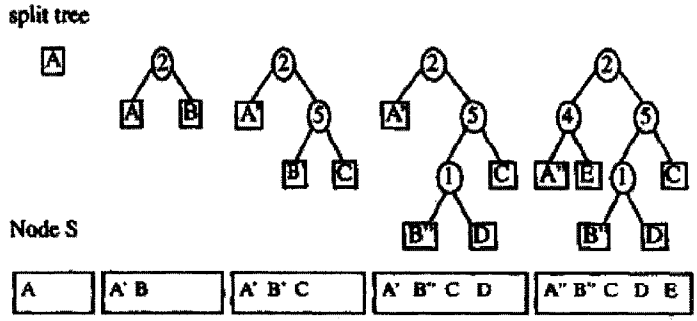


Figure 2.3: Split History Tree Example [4]

Examining this tree we instantly know that C is the result of splits along dimensions 2 and 5. Now when we want to split set S {A" B" C D E} into subsets S1 & S2, it would be advantageous for them to have as many common splits as possible. Logically then, the best split is the overlap-free one into the subtrees rooted by nodes 4 (containing {A" E}) and 5 (containing {B" C D}). The anti-overlapping measures are what allow the X-Tree to extend to higher dimensions without suffering immense time increases.

2.3.2 The GSS Tree

One main contribution of this thesis is in the spatial database indexing field, using the Volume Distribution Tree to efficiently access 3D objects not resembling any geometric primitives. A similar application to this one is tracking the shape of the hippocampus in medical imaging. The spatial indexing demonstrated by Keim is the Geometric Similarity Search (GSS) tree, grouping together closely located objects in a tree using the objects' minimum included volume (MIV) and maximum surrounding volume (MSV) [14]. Figure 2.4 displays the MIV and MSV concepts.

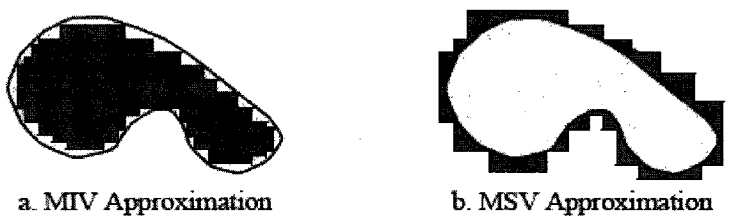


Figure 2.4: MIV and MSV Example [14]

The GSS tree uses hierarchical approximations, with the directory nodes deeper down the tree being closer approximations of the objects below them; parent nodes are lower resolution amalgamations of their children. Many algorithms could hierarchically determine the MIV & MSV as do the two instances given in the paper, the Cuboid and Octree GSS trees. The Cuboid version keeps finding the largest rectangular box available to add to the MIV and subtract from the MSV in order to give a closer approximation to the exact volume indexed, while the Octree version recursively deems any octants of the space containing some object volume as full and others empty. An example of the Cuboid GSS tree is provided in Figure 2.5.

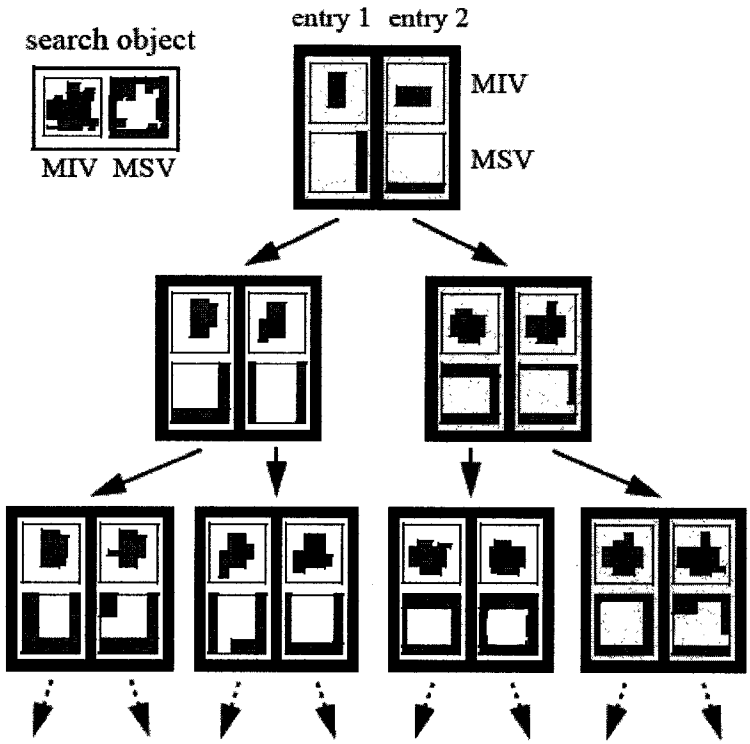


Figure 2.5: Cuboid GSS Tree Example [14]

We can see in this example the increasingly accurate geometric representations when descending the tree and that the construction and insertion try to keep similar representations grouped together for effective pruning.

The strategy carried over to the Volume Distribution Tree is to represent a group of volumes with a small set of numbers and to set reasonably tight mathematical bounds on the attainable similarity score of a group based on those numbers.

2.3.3 *The P-Tree*

The P-Tree (Peano Count Tree) is effectively a Quad Tree [8] with counts of the 1's in its quadrant included in each node; the 1's being the binary high pixels in a black and white Boolean image[7]. Each node either has 0 or 4 children – if the node's quadrant contains some white voxels and some black ones it is recursively divided into its quadrants, giving 4 children and otherwise the node has 0 children as it is a uniform color and thus no more information is required to explain it. Any node's value is the number of white pixels in its quadrant, so white pixels are summed for each leaf node, and then the directory node values are simply the sum of their children's values. A P-Tree then contains all of the information necessary for reconstructing the image. An example of a P-Tree is shown in Figure 2.6, with the left box containing an array of ones and zeros which can be the representation of a black and white image, and the right box containing the corresponding P-Tree with the left-to-right node order being the Z-Order of the picture (NW,NE,SW,SE).

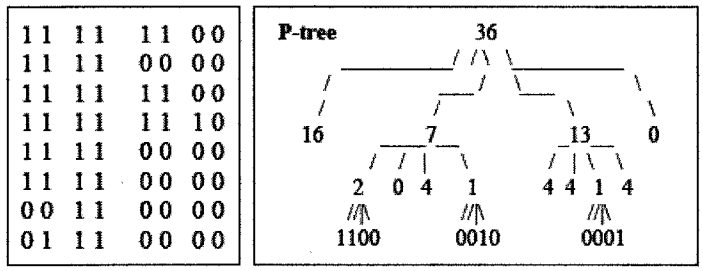


Figure 2.6: P-Tree Example [7]

In contrast to our Volume Distribution Tree which is global, one P-Tree is created for each image. The Volume Distribution Tree employs the idea of denoting the number of white voxels in each section, but only using a forced 1-level representation with a user-defined grid rather than variable levels with quadrants. That is, whereas a P-Tree would recursively decompose a single object into volume octants, the Volume Distribution Tree is an index structure where the leaves are single-level volume distributions. There is no recursion in the VD-Tree's volume distributions and the partitioning is defined according to the application rather than always being octants. By avoiding variable levels, the Volume Distribution Tree allows easier comparisons between objects, sacrificing accuracy however, and allows objects to be listed in nodes which can be grouped together to form a global tree indexing all objects in a

database. To clarify, by taking a P-Tree, expanding its divisions from octants to an application-defined scheme, and then taking the values from its first level below the root we have one VD-Tree leaf node. The VD-Tree structure is built to index these leaf nodes.

2.3.4 The RI-Tree

The Relational Interval (RI) Tree was designed for range overlap queries [16]. That is, given an interval, to be able to return all intersecting stored intervals. This is achieved with a tree where nodes contain sorted lists L & U being the lower and upper bounds respectively, of the 1D intervals represented by that node. Instead of stored intervals being held exclusively in leaf nodes, they are each held in the highest node where the stored interval entirely contains the node's range. This node is deemed the interval's 'fork node'. A query starts by determining the query range's fork node and then travelling down to its parent from the root of the tree. Since the descent is to one level above the fork node (recall that the fork node is the highest node entirely contained within the query range), each node explored has necessarily had an extension to the left or right of the query range. For the left ones, their upper range can be examined and if not meeting the query's lower range, they can be pruned, as can nodes right of the range whose lower bounds are also to the right of the range. The intervals directly stored within the fork node can be reported since they definitely intersect the query range. The search continues below the fork node first to the left, where if that node's lower bound is above the query's lower bound all results contained in that node's subtree can be reported, and otherwise the node must be linearly scanned for intersecting intervals. Processing for the right side of the fork node is analogous. Part of a tree and its processing during a query is shown in Figure 2.7 with the labels 'lower' and 'upper' denoting the query range.

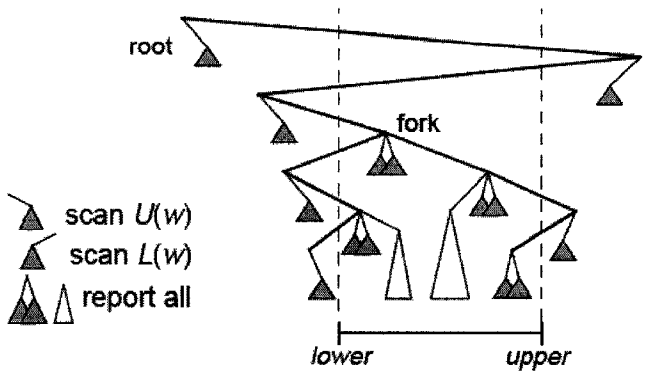


Figure 2.7: RI-Tree Query Processing [16]

A method of indexing 3D spatial data using this structure is necessary to make it relevant to our work, but [17] does just that by encoding 2D and 3D objects into 1D by means of space-filling curves. The objects are turned into multiple intervals in the space filling curve, and these intervals are indexed by the RI-Tree.

A further extension to permit more efficient use of the RI-Tree is proposed in [15]. The main problem with the decomposition of objects into intervals is the shear amount of intervals necessary to compose an object. This huge amount of intervals causes poor retrieval performance. In an effort to reduce this burden [15] uses gray intervals, which are intervals covering many black intervals (the ones directly extracted from the space-filling curve). One gray interval being able to represent many black intervals reduces the burden of storing and finding so many intervals, at a cost of lower accuracy representation requiring more post-processing. The authors create the GroupInt algorithm to take advantage of the gray intervals. The representation of data in that algorithm is hierarchical, with the top level using only one big gray interval to encode an object, and at the next level splitting this interval along the largest gap between black intervals. This strategy continues by giving a better approximation to the object in each level down, allowing a search to stop after only a few levels by determining that there is already no potential for this object to be relevant to the query.

Chapter 3: Segmentation

The database and surrounding system described in this work assume a working segmentation technique providing an enhancing vs. non-enhancing label matrix for database entry. Segmentation's purpose, whether done manually by an expert or automatically by a computer program, is to segment the region that enhances on the T1C imaging, as this corresponds to the visible edema region housing the tumour. Any good segmentation technique, consistently used across studies in the database, is valid and can be used as a replaceable module without changing the database work. The accuracy of similar tumour retrieval from the database is however tied to the segmentation quality and thus improving the segmentation is advantageous to the database project.

3.1 Motivation

Ideally, medical imaging should be automatically segmented into enhancing and non-enhancing regions. Clinicians simply do not have the time to segment every MRI image, and there has been shown to be significant variation in segmentation not only between clinicians but between the same clinician at different times [18]. Image segmentations are stored in the database and then are used as a fundamental part of the query process, and as such must be precise.

The current pre-processing of MRI imaging includes automatic segmentation, but its time-consumption, inaccuracy and mistakes have prompted investigation into new methods. Since the idea of the database is to supply data for processing, the data itself should be clean and free from harmful pre-processing. Currently the raw imaging in DICOM format is converted to a series of PNG images via a pipeline performing spatial registration and intensity standardization [24]. Although this pipeline provides a common intensity and coordinate system, it locally warps the data. In order to place a brain image in a standardized position, the system minimizes the difference from the template brain Colin27 [11]. In doing so the system however

performs local deformations, changing the shape of the tumour and therein distorting any morphological study. An effort has thus been made to circumvent the pipeline, replacing it with an affine registration based on placing the Falx line on the center plane and new segmentation procedures whose internals are transparent and easily understood.

3.2 Projection onto Eigenbrain Space

Principal components analysis (PCA) was suggested for the compression of images to a set of weights, a concept known as eigenfaces due to the mapping of face images into a space defined by the eigenvectors of a training set of faces [28]. An idea derived from this paper is that a face vastly different from the training set or having a new feature (glasses, beard if these were not in the training set) would likely be poorly reconstructed, and thus casting an unhealthy brain into a space created via only normal brains would yield errors at a much higher rate in the tumour region. The plot of absolute reconstruction error could then be a basis for automatic segmentation. Since the segmentation of the error plot is well defined as getting a region best encompassing the large values, it should be an easier problem to solve than the original segmentation problem where we need to segment an area that is somehow different from the rest.

Due to the nature of our studies and collaborations, our brain imaging database consists only of brains with tumours rather than normal, healthy brains. Since a set of normal brains is required to form a healthy image basis, the set was created from the right halves of exclusively left-side tumour patients and left halves from exclusively right-side tumour patients. With this trick we can avoid the cardinality of the set of tumour images from greatly surpassing the cardinality of the set of normals.

PCA is used to create a basis in v dimensions, where v is the number of voxels used in each original image or 3D matrix describing a normal brain. By reshaping each brain image into a 1 x v vector and stacking all of these together, we get an n x v matrix (n being the number of training brains) containing the whole training set information. With PCA we replace this matrix by its eigenvectors, forming an orthonormal basis that is still of size n x v. To eliminate high-frequency noise as well

as obtain better compression, we only use the eigenvectors corresponding to the highest 40 eigenvalues (this value selected after looking at the rapid drop in eigenvalues, as these represent the variance). See Figure 3.1 for the eigenvalue trend.

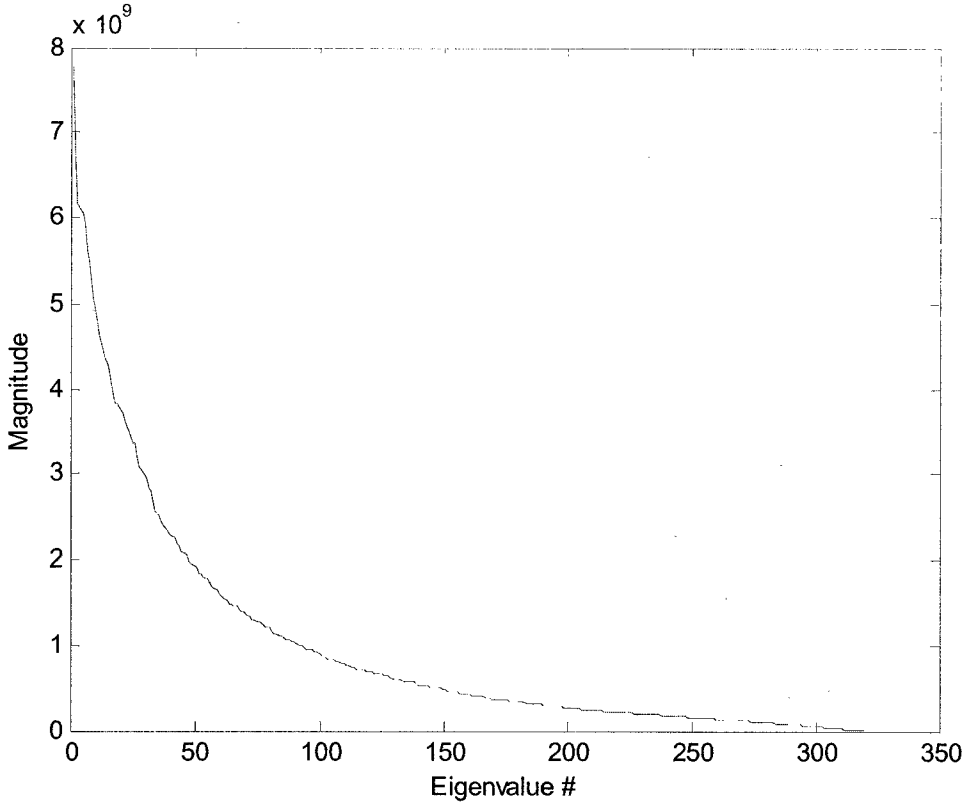


Figure 3.1: Eigenvalues of the brain space, sorted in descending order

The eigenbrains each represent some intensity component of the brain such that any brain can be expressed (with lossy compression) by a set of 40 weights meaning the brain can be said to be $w_1 \cdot \text{eigenbrain}_1 + w_2 \cdot \text{eigenbrain}_2 + \ldots + w_{40} \cdot \text{eigenbrain}_{40}$ where the w_i's are the weights. Our 1st and 9th eigenbrains' 40th slice are displayed in Figure 3.2 as an example of what the different eigenvectors can focus on. Projection of unhealthy brains onto the normal brain space somewhat localized reconstruction error to the tumour, but this was marred by much noise.

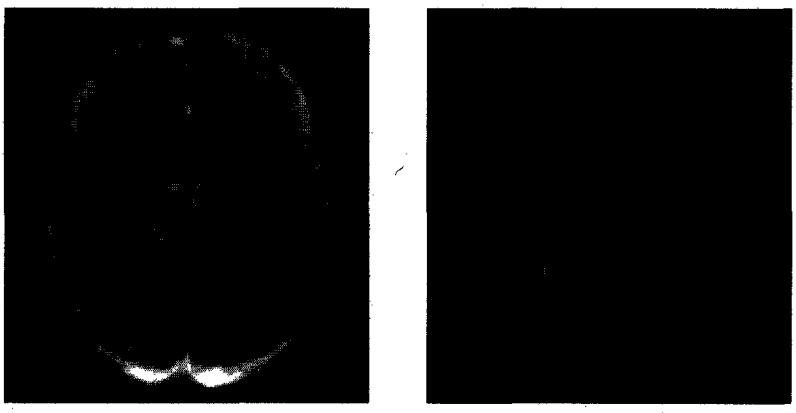


Figure 3.2: Eigenbrain Examples

One suggestion that has been implemented here to reduce this noise is to use a robust fitting technique which iteratively reconstructs the brain, with each iteration putting lower weight on high-error voxels. The idea is that the normal part of the unhealthy brain is the part that should be well-reconstructed without interference from trying too hard to fit the bad parts. This raised the ratio of average intensity inside the tumour to that outside the tumour, indicating more focus on highlighting the tumour compared with other artefacts.

To see an example, refer to Figure 3.3, which shows the original brain to the far left, as segmented by our current segmentation program, next the map of the reconstruction error, clearly highest in the tumour region, next is the reconstruction error thresholded to provide a clear segmentation, and to the right the post-processed segmentation overlaid onto the brain image.

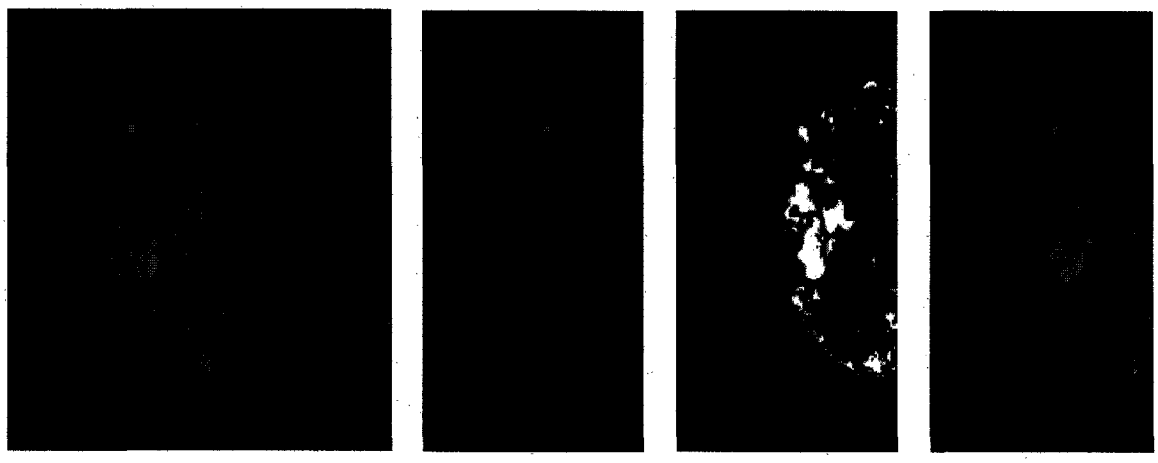


Figure 3.3: Segmentation Using Eigenbrains

3.3 Finding High Z-Scores

An even simpler means of locating the tumour is to find out which voxels are far out of tolerance compared with the normal set. The mean and standard deviation of every voxel for normal brains is kept in two 3D matrices, keeping a low memory footprint. Any brain to be segmented has its z-score map computed by dividing the absolute difference between this brain and the mean for the set by the set's standard deviation. High z-scores indicate variation from the mean that is much higher than usual, and therefore abnormal. This map can therefore be used as a basis for segmentation in the same way as the eigenbrain projection. Note that due to the lack of a large enough set of normals, the training set was split into a left half using patients with a right-side only tumour and a right half in the same manner, as explained in Section 3.2.

This procedure can be done with T1 and T1C images compared to the normal statistics for T1's, as well as T2 images compared to normal T2 values. Combining the results of these two filtered out much of the noise since the tumour is found in approximately in the same place with either modality while the noisy bits did not match up very often. The combination was done by segmenting only the voxels whose average Z-Score (in T1C and T2) was above a set threshold.

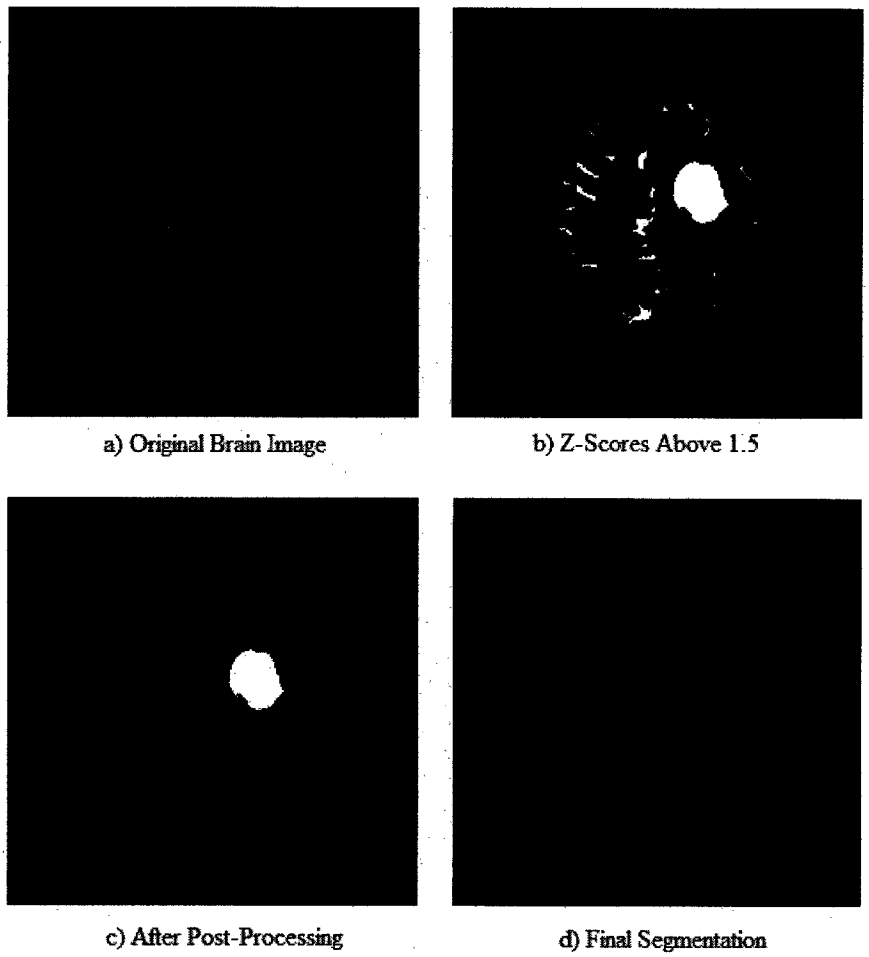


Figure 3.4: Z-Score Segmentation Process

What is left of the noise is filtered out by keeping only the deepest white section, since erroneous segmentations generally appear as tiny blobs and larger but thin shells. Matlab's bwlabel function labels all disjoint white segments with different integers, allowing easy separation of all segments such that they are alone in their own matrices. The depth of each of these segments is then taken as its maximal Euclidean distance transform value. This distance transform is a morphological operation where each white voxel is replaced by a real number equal to its distance to the closest black voxel, and is performed by Matlab's bwdist function. Figure 3.4 helps explain the procedure of segmenting by Z-Scores; in (a) we see one slice of a T1C-weighed brain image to be segmented and in (b) the location of the voxels being 1.5 standard deviations away from the norm. In this figure we can see that the thresholding of Z-Scores to above 1.5 mostly focuses on the tumour region. The

smaller regions identified are later filtered out by post-processing as shown in (c). The final segmentation is shown in (d) with the red outline denoting the segmentation extents overlaid onto the original brain image. The reader may note from Figure 3.4 (b) that the segment with the largest volume could more simply be selected than the one with the largest depth. The selection by depth however avoids selecting long thin membranes instead of the tumour. A lot of noise, joined with a thin but large area of high z-scores can make up a larger contiguous volume than a small tumour, leading to obviously poor results.

Chapter 4: Similarity Measures

4.1 The Notion of Similarity

What it means for two objects to be 'similar' is subjective and as such no concrete true value of similarity is available for comparison with the presented similarity measures' outputs. In this case we wish to predict a tumour's growth based upon similar tumours in the past and through experience with the query system presented, physicians can pick the similarity measures whose results turn out to be most conducive to this purpose.

Several similarity measures will be defined in this section, each of which describes a subset of the following aspects of similarity: location, shape, volume, extents, texture, and orientation. Efficiency-increasing implementations will also be discussed here.

Note that the imaging data has been registered to a template in order to maintain a consistent coordinate system between brains. Each brain image is a 258 x 258 x 88 voxel matrix (due to the template brain being of this size) with each voxel containing an 8-bit greyscale value.

4.2 Supported Similarity Measures

4.2.1 The Jaccard Measure

The classic similarity between two sets of voxels in a common space is the Jaccard measure, which is the intersection of the sets divided by their union volume [12]. A simple but very effective optimization for this calculation is obtained by noticing that the union is simply the sum of the two volumes minus the intersection. Since the volumes remain constant between queries, each tumour's volume is stored, ready for quick access during queries. Thus only the intersection must be computed, but this still requires accessing the tumour image from disk, which makes a linear scan too costly. To save on I/O and computational costs, instead of storing each segmentation in a 258 x 258 x 88 matrix they are stored in smaller matrices the size of their

Minimum Bounding Rectangle (MBR). Thus if a segmentation only resides within 100-120 in the x-axis, 145-185 in the y-axis and 40-51 in the z-axis, its MBR field will list [100 120 145 185 40 51] and the segmentation matrix will be of size 21 x 41 x 12; the rest is known to be full of 0's (empty). Despite the gains from this storage strategy, the linear scan is still too slow so a specialized data structure is applicable here. A tree structure has been created to quickly filter out the distant tumours and provide a short list of candidates to check thoroughly. This structure, the Volume Distribution Tree, is the topic of the 5th chapter of this thesis.

4.2.2 Depth Jaccard

Although the Jaccard measure captures the intersection of two bodies, it ignores the differences in volume distribution between the bodies. For example, two bodies only differing in that one of them has a thin protrusion from its main mass may get a lower score than bodies who intersect a lot but whose core parts are not that close.

To capture the depth of a tumour, we use the Euclidean distance transform (3D version of [21]), which approximately indicates how far a tumour voxel is from the outside of the tumour. Figure 4.1's upper set of images shows an example tumour shape in 3D on the left and a 2D slice of this shape on the right, along with their analogous distance transform images on the lower set of images. Note that the 2D picture of the distance transform has uneven contours since it is only a slice of the 3D distance transform, not a distance transform of the 2D slice.

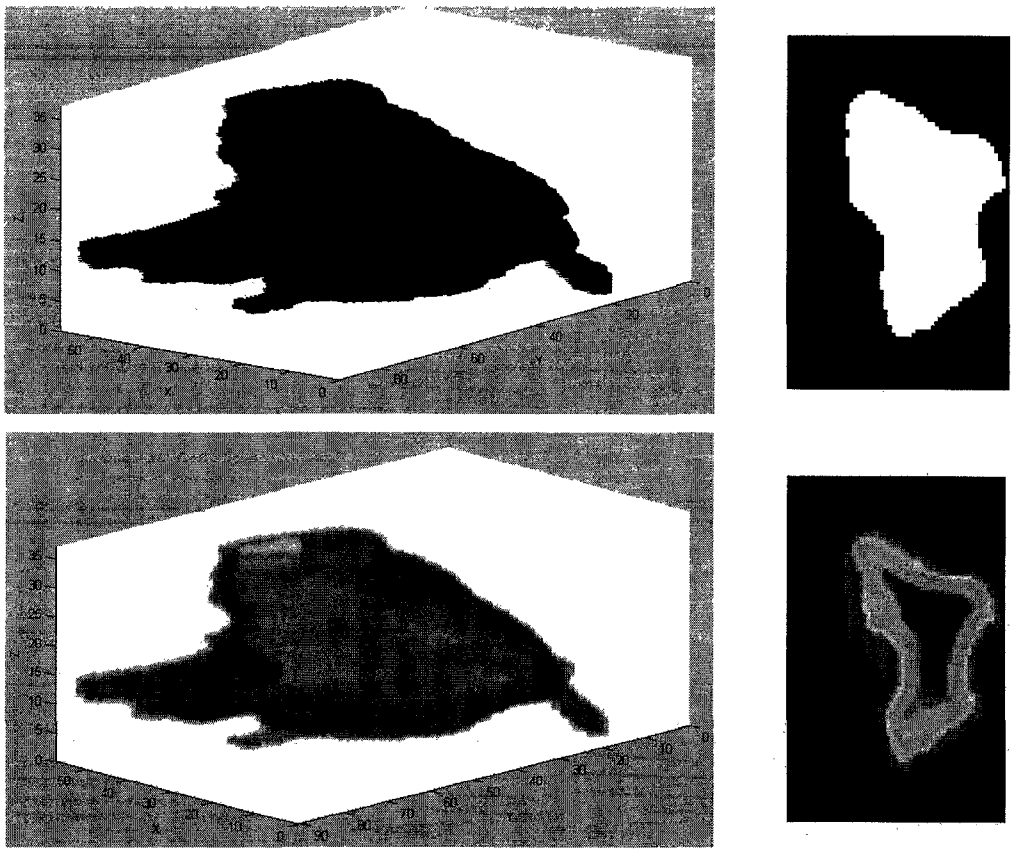


Figure 4.1: Euclidean Distance Transform of a Tumour in 2D and 3D

To capture the degree to which the objects' cores are intersecting, for each voxel that intersects instead of adding 1 to the intersection count as the Jaccard measure does we add $(1-|d_1-d_2|)$ where d_1 and d_2 indicate the normalized distance transform values for the voxels from tumour 1 and tumour 2 respectively. That is, after the Euclidean distance transform is computed for a segmentation it is normalized, meaning that all values are divided by the maximal distance transform value obtained. What is added to the intersection sum is one minus the difference in normalized depths of the intersecting voxels. For example a point near the core of tumour 1 may have a depth of 0.9 while its intersecting voxel is near the edge of tumour 2 with a depth of 0.2, in this case their intersection would be counted as $(1-|0.9-0.2|)=0.3$ instead of 1.

This method of counting intersection means that tumours whose cores are at each other's edges but still intersect at 30% will be given a much lower than 30% score. For example consider one shape intersecting with two others, shown in Figure 4.2.

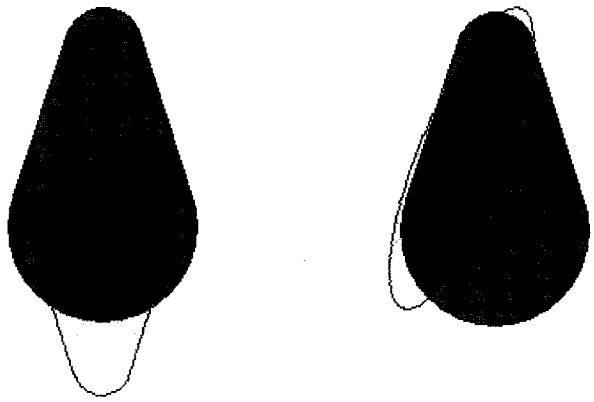


Figure 4.2: Depth Jaccard Intersection Example

Here compare the left-side case where the two objects share the same core and have similar extensions in different directions to the intersection shown in the right-hand case where there is a sizeable intersection between somewhat dissimilar shapes. Now instead of purely finding the intersection each intersecting voxel is weighed down according to the difference in depth. The result of this modification is that the right-side example's score will suffer far more than the left-side example due to the left one's lining up of the cores. This would be a desirable modification for a user wishing to find tumours growing from the same structure of the brain, or wanting to ignore small fringes on the edges of tumours in favour of the bulk of each tumour.

Since each intersecting point adds one or less to the intersection and we count the union exactly like the regular Jaccard case, the Depth Jaccard scores are always equal to or lower than the corresponding Jaccard scores. Therefore we can use the VD-Tree (detailed in Chapter 5) created to support the Jaccard measure to also support the Depth Jaccard variant since any branch excluded due to not meeting the Jaccard quota can definitely be excluded from meeting the same Depth Jaccard quota since the latter cannot be larger than the former.

4.2.3 Finding the Center Point of a Tumour

The next few similarity measures require a definition of the center point of a tumour. Here we discuss three central measures, each of which can be used to define the center point (thus each similarity measure requiring a center point naturally has three variants). The easiest to compute is to simply take the center of the tumour segmentation's minimum bounding rectangle (MBR). This point can however be

heavily swayed by a long appendage to a tumour and may not actually reside inside the tumour outline (although in general for good segmentations it does). The second means of picking the center is to pick the center of volume, which is the point at which half of the volume resides on each side of any plane that includes this point. The third method used (and the one used in the current database system) is to pick the core as defined by the Euclidean distance transform. With the distance transform applied, only the set of maximum values is kept, with the center of this volume being deemed the center point. This method has the advantage that it is not only definitely within the tumour volume, but is actually at the deepest point. Thus this point indicates the middle of the largest mass and is unaffected by protrusions at the edge of the tumour. Figure 4.3 shows an example tumour slice and its center as defined by the three methods discussed above, along with the distance transform map on the left showing why the red point on the right was picked as the core point.

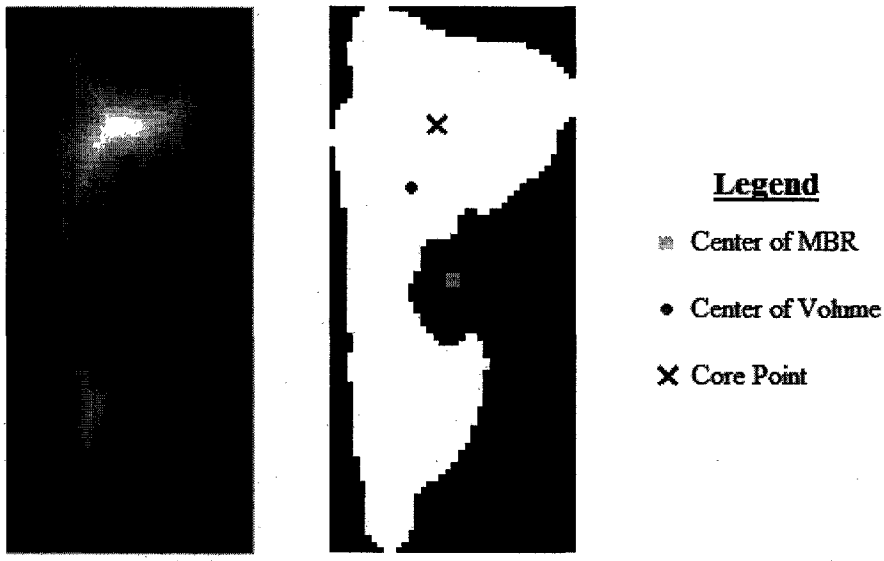


Figure 4.3: Center Point Measures

4.2.4 Ray Trend

The similarity measures thus far only take into account the Boolean tumour segmentation; i.e. whether a voxel is or is not cancerous. To take into account the texture of the tumour, the Ray Trend measure uses the actual greyscale values of the MRI image which are located within the segmented region. From the centerpoint of the tumour, we take a vector in each of the 26 directions (there are 26 combinations of (right, left, center) for 3 dimensions other than (center, center, center) which is the

origin). Each vector comprises the greyscale values along the straight line path from the center to the tumour edge in the specified direction. This serves as a set of feature vectors, whose Euclidean distance to other tumours' rays is taken and summed up into a distance. The smaller the distance, the more similar the two tumours are said to be. This is an extension to Vranic and Saupe's use of the extents of a shape from its center in the sense that it uses values defining the interior texture of the tumour as well as a simplification in the sense that we use only 26 standard directions rather than defining continuous functions [23].

4.2.5 Shape Histogram

This technique developed by Ankerst *et al.* is described in Section 2.2.1. It was implemented here since it provides an innovative means of describing the shape of a tumour, which is important in finding similar tumours. In our work the center of the tumour was found using the core point as opposed to the original work, which used the center of mass as the center of the object.

4.2.6 MidSim

To create a similarity measure targeting the shape of the tumour, we start by finding the center of volume. Using this center as the origin, the three standard planes partition the volume into 8. A recursive application of this to each partition a preset number of times yields a group of 3D points describing the shape of the tumour. These points can be normalized with respect to the extents in each dimension to provide invariance to the size of the tumour. The sum of the Euclidean distance between the points describing two tumours is used as the distance between the tumours for queries.

4.2.7 Eigen Decomposition

As described in Section 3.2, the important information can be extracted from a brain image by projecting it into a well-made standard space. By using the voxels where the brain can actually be present (i.e. excluding the huge always black section outside the head), we get a much smaller representation of the brain image (1,397,314 voxels instead of 5,857,632). This makes it possible to perform an eigen decomposition of the whole dataset. Thinking of each imaging session as a long vector, we can find the eigenvectors making up a new basis set for these vectors. Although the new basis has

the same dimensionality as the original set, many of the dimensions contain very little variability compared with others and thus can be discarded with very little effect on the information carried. After plotting the eigenvalues, it was decided that the 40 rows corresponding to the 40 highest eigenvalues would be kept since after 40, the eigenvalues are significantly lower than the first few eigenvalues. That is, the 40 dimensions of the dataset with the most variability would be kept.

Each MRI image is then projected onto this space, being compressed into a 40-length vector representing the image's weights in the 40 most important directions according to PCA. The distance between two images' weight vectors can be used as a distance metric to assess their similarity with respect to the group.

4.2.8 Elongation, Flatness and Sphericity

As an easy-to-understand pure shape descriptor, we approximate the aspect ratio of the tumour. This is invariant to the tumour location, volume and orientation. The actual quantities calculated are the eigenvalues corresponding to the principal components of the tumour. Ratios close to 1 indicate a somewhat spherical tumour while larger ratios indicate a more elongated ellipsoid-like shape. Note that the eigenvalue indicates the variance of all of the voxels comprising the tumour in a particular direction. Although the ratio of eigenvalues is not proportional to the aspect ratio, a larger eigenvalue ratio generally corresponds to a larger aspect ratio. The ratios maintained in the database are that of the first to the second eigenvalues (Elong), indicating elongation and the second to the third eigenvalues (Flat), indicating flatness. Since some tumours have quite concave faces, dividing the Elong ratio by the tumour depth is a good idea to minimize this nonlinear effect.

Another scalar value kept in the database is the sphericity Ψ, calculated as $\Psi = \frac{\pi^{\frac{1}{3}}(6V)^{\frac{2}{3}}}{A}$, where A is the object's surface area and V its volume [27]. This value lies in the range (0,1], with 1 indicating a perfect sphere and low values indicating an elongated or highly folded structure.

4.2.9 Growth Direction

Since a main goal of the BTAP is to predict tumour growth, an important piece of information to find the growth pattern of tumours similar to the one we wish to

predict. Given a current patient with a brain tumour it would be advantageous to find similar tumours in the database with the later stages of their growth also documented in the database. The further growth of past tumour cases in other patients may guide physicians in predicting the growth of a current tumour whose future cannot be seen. How do we find similar growth patterns efficiently though? One way is to index growth between two visits by a vector pointing out from the core of the tumour in the direction of greatest growth.

The natural choice for picking the primary direction of data is PCA. By subtracting the older tumour from the newer one, a 'shell' representing the growth is left and this can be run through principal components analysis. An uneven growth all around a spherical tumour should align the principal axis in the direction of the largest growth. We may however be looking at a growth shell that is empty on one side instead of a full volume and thus PCA may not be the best choice. For a 2D example, a round tumour with a purely right-side growth should be pointed to by a vector in the positive x-direction but may actually have its main principal axis in the y-direction (see Figure 4.4).

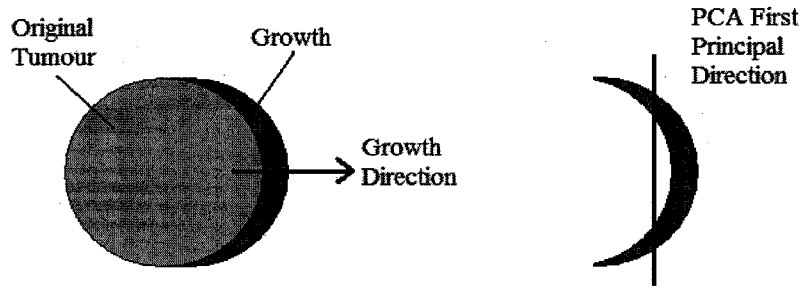


Figure 4.4: Problem With Finding Growth Direction By PCA

The alternative used is the vector sum of the growth voxels' normalized vectors originating at the core. Each voxel appearing in the later timestamp that does not in the earlier timestamp is tagged as a growth voxel. These are all indexed by their vector from the first timestamp's tumour core. The set of vectors are projected onto a unit sphere around the tumour core to avoid skewing of the resultant by further points since we are only concerned with the direction of growth, not how far away from the core it happens. The resultant of all these vectors is converted to spherical coordinates (ρ,ϕ,θ). The radius ρ has a [0,1] range with a value close to 1 denoting

very directed growth with altitude ϕ above the x-y plane and rotation θ around the z-axis, while a radius of 0 denotes equal and opposite growth on each side of the tumour.

4.3 Other Relevant Information Queries

Many standard pieces of data are put into the database and may well be required by someone wanting to study only the subset of the population where a certain property applies or is within a defined range. One such example is the sex of the patient – this is stored as a column and may be used as a predicate when querying the database. The only other stored patient data is their birth date; no other patient information is available after anonymization due to ethical guidelines preserving anonymity of the patient. The image time is stored, as well as the MRI machine used since specifications such as magnet strength can be important. Stored quantities derived from imaging are the Gross Tumour Volume (GTV) in units of voxels, and the center point of the tumour as defined in three ways: by the center of its volume, by the center of its Minimum Bounding Rectangle, and by the center of the peak of its D26-distance transform. The extents of the tumour in its 3 primary directions are listed since physicians often discuss tumour size by these measures in addition to the GTV. The box created by these extents can also be compared with the other ones in the database by a routine giving the Jaccard score between two boxes aligned and centered together. The tumour contrast enhancement is listed in a column and therefore a user can easily query for enhancements in a certain range. The enhancement number is defined here as the mean intensity growth within the segmented tumour region between the T1 image and the T1C image taken on the same day. A proposed additional column is whether or not the tumour enhancement has a sharp contour, or perhaps a measure of the sharpness of the enhancement's contour. Other data that may be added to the database include the survival (time from when a study was done until the patient passed away), types and dates of treatments, and tumour grade.

4.3.1 Growth Through Regions

Physicians often want to know if tumours can cross a certain barrier or membrane. We provide two simple means of asking that query of the database. The simplest being for the user to provide 2 points, a and b, to find the tumours that grew from a to b. This is simply achieved by finding all tumours intersecting (a & ¬b) and finding their further timestamps who intersect b. Since the interface provides drawing facilities, the user can easily pick two points to query for tumours which have grown from the first to the second point. When the user chooses 'Growing A to B' from the similarity measure drop-down list, two boxes for entering coordinates appear, but the user can just pick the points while looking at brain images by clicking on the desired spots.

4.4 Comparison of the Similarity Measures

How do the similarity measures complement each other? Table 4.4.1 gives a succinct rundown of the aspects of similarity each similarity measure (including its variants) takes into account. The blank spots therefore denote aspects with respect to which a particular similarity measure is invariant.

Similarity Measure	Takes Into Account This Aspect of Similarity					
	Location	Shape	Volume	Extents	Texture	Orientation
Jaccard	Y	Y	Y	Y		Y
Jaccard (Centered by Volume)		Y	Y	Y		Y
Jaccard (Centered by MBR)		Y	Y	Y		Y
Jaccard (Centered by Core)		Y	Y	Y		Y
Depth Jaccard	Y	Y	Y	Y		Y
Depth Jaccard (Centered by Volume)		Y	Y	Y		Y
Depth Jaccard (Centered by MBR)		Y	Y	Y		Y
Depth Jaccard (Centered by Core)		Y	Y	Y		Y
Ray Trend (Appended with 0's)		Y	Y	Y	Y	Y
Ray Trend (Appended with Out of Tumor Values)		Y	Y	Y	Y	Y
Ray Trend (Vector Stretched)		Y	Y		Y	Y
Shape Histogram		Y	Y	Y		Y
Shape Histogram (Sectors Only)		Y	Y			Y
Shape Histogram (Rings Only)		Y	Y	Y		
3D Shape Histogram (Bin, GrayValue)		Y	Y	Y	Y	Y
Distance Between Centers (By Center of Volume)	Y					
Distance Between Centers (By MBR Center)	Y					
Distance Between Centers (By Core)	Y					
Volume Difference			Y			
MidSim (Recursive Division of Volume into 8)		Y				Y

Table 4.4.1: Similarity Measure Dependence Comparison

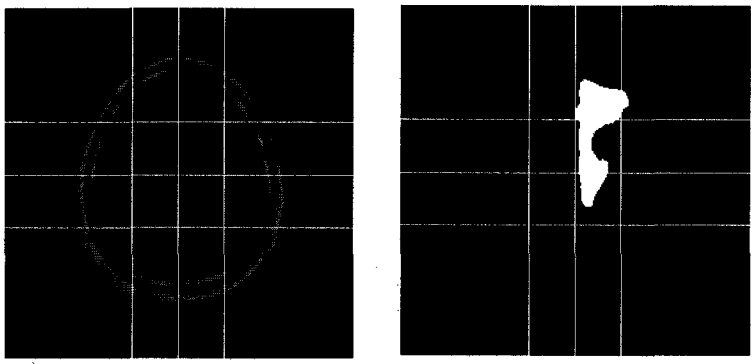
The above table can be used as a quick reference, summarizing the aspects of similarity integral to each similarity measure. Ideally, we would like each similarity measure to represent only a single aspect of similarity such that there is no cross-coupling between measures when used together. This perfect decomposition is not easily achieved and instead Table 4.4.1 represents the actual coupling involved in weighing together the results of several similarity measures. Since the similarity measures shown naturally cross-couple shape and volume, we differentiate these using the volume as a number and the MidSim measure which is independent of the volume as a quantity.

Chapter 5: The Volume Distribution Tree

Since queries using the Jaccard coefficient as a similarity measure (see Section 4.2.1) are so frequently used to retrieve similar tumours, an efficient implementation of this query was in order. Linearly searching the whole database is very inefficient and slow due to the large size of the matrices to be intersected. Organizing the tumour minimum bounding rectangles (MBR's) into an R-Tree allows a query script to efficiently target objects in the database with MBR's intersecting the query object's MBR but this unfortunately does a poor job of narrowing down the list of possible candidates to be exactly checked. The Volume Distribution Tree, or VD-Tree, is our solution to this problem.

5.1 Volume Distribution Concept

The base idea involves dividing the space into n-by-n-by-n blocks and computing their filled-in volumes since this can provide information bounding the maximum possible Jaccard measure while taking up little space. Figure 5.1 shows the division of the volume into a 4-by-4 grid (for a 2D example). In this work the matrix of numbers is flattened into a vector for storage, and this vector is what will be referred to by "volume distribution".



$$volDiv = \begin{bmatrix} 0 & 0 & 822 & 55 \\ 0 & 0 & 605 & 0 \\ 0 & 0 & 337 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 5.1: Division of Volume for the Volume Distribution Tree

At Left: A standard brain image showing that the grid is concentrated toward the middle since the head only covers that part.

Middle: A graphical 2D example of a 4x4 division of a tumour segmentation.

At Right: The matrix showing the volume (in number of pixels) of the tumour in each division.

Note that the divisions are set up closer to the middle instead of evenly distributed – this is meant to somewhat equalize the space in each section. The two lines beside the middle are each placed at 35% of the maximum distance from the middle to the outside of the brain since this is the half-volume point in an ellipsoid modelling the head. Note that the maximum extents of a head are located in a lower slice than that shown in Figure 5.1, so despite the lines shown being 35% of the distance to the true extents they appear too be about 50% of the way to the edge since at this height, the head is narrower than its maximal width. Note that the grid cells can be set up to be anywhere so long as all volumes in the database are split in the same way, making this structure easily adaptable to other applications. A guideline to set up a grid for a specific application is to make each cell approximately equally likely to contain part of an object. In other words, for a grid of N total cells, each cell should be sized to hold an expected value of 1/N of a random object's voxels. In this way we spread the information over all the cells, avoiding any cells whose value is always zero (which would provide no information gain for that cell).

The volume distribution concept is analogous to the feature vector of volumes employed in shape histograms as both divide the volume of an object into cells and use it as a vector conducive to their respective purposes (see Section 2.2.2) [2]. The

volume distribution in the shape histograms is used to compare shapes using a quasi-Euclidean distance metric to get dissimilarity between object shapes. In contrast, the VD-Tree does not use any distance metric between distributions to obtain similarity or distance. Rather, the VD-Tree employs volume distributions to compute bounds on the intersection and union of spatial objects, and uses this in a hierarchical manner (a tree of volume distributions) to efficiently index objects for similarity search employing the Jaccard measure. To achieve this end, the volume distributions for the VD-Tree must be taken on a grid consistent to the whole dataset rather than each object's center. The partitioning method employed in creating shape histograms could certainly be used as the grid in creating a VD-Tree, although the grid's center would have to always be at the center of the full space. For the brain tumour similarity search application, the spherical shells would also have to be modified to be ellipsoidal to take the brain shape into account; since the brain is not spherical, there would be many cells that hardly contain any of the volume.

5.2 Volume Distribution Tree Properties

The VD-Tree is a tree designed to index objects with spatial extents for Jaccard measure queries. The VD-Tree query processing strategy guarantees 100% recall as it can never exclude a valid result when pruning branches. Recall is a measure denoting the percentage of the relevant documents in a database that are actually retrieved by query processing.

Leaf nodes index exactly one study each (a single patient visit) and thus contain the patient number, study number along with their own node number (to be pointed to by other nodes), their parent's node number and the volume distribution vector for the tumour segmentation done for the study indexed. A directory node contains its own node number as well as its parent's and children's and two vectors to indicate volume distribution. A directory node roots a subtree and the directory contains element-wise maximum and minimum vectors of the volume distributions in this subtree. Directory nodes index a group of 2 to M children (either leaves or other directories), where M is the node capacity defined by the administrator prior to tree construction (in this work we use M=6). Using a large value for M would cause the max / min

bounds to be too wide leading to following many branches during a query, and also increasing the cost of a split due to the number of pair-wise distances computed increasing exponentially with M. Small M values would cause the tree to be very deep as well as requiring frequent splitting of nodes.

All leaves appear on the same level of the tree. As will be shown, each node (leaf or directory) permits the calculation of an upper bound on the Jaccard score for the subtree it roots. This upper bound allows branches to be pruned during queries, saving computational and IO effort.

The tree structure is created bottom-up either by bulk-loading or by insertions, trying to cluster similar volume distributions together. The volume distribution vectors are the leaves of the tree. A set of vectors clustered together are aggregated into a node carrying the minimum and maximum volume for each cell, rendering directory nodes roughly twice as large for storage as leaves. This process is recurred and higher-level directory nodes stay the same size as lower-level ones since they still simply carry the maximum and minimum bounds for any node below them. A miniature example of a VD-Tree is displayed in Figure 5.2, and it will be used as a running example later in this chapter.

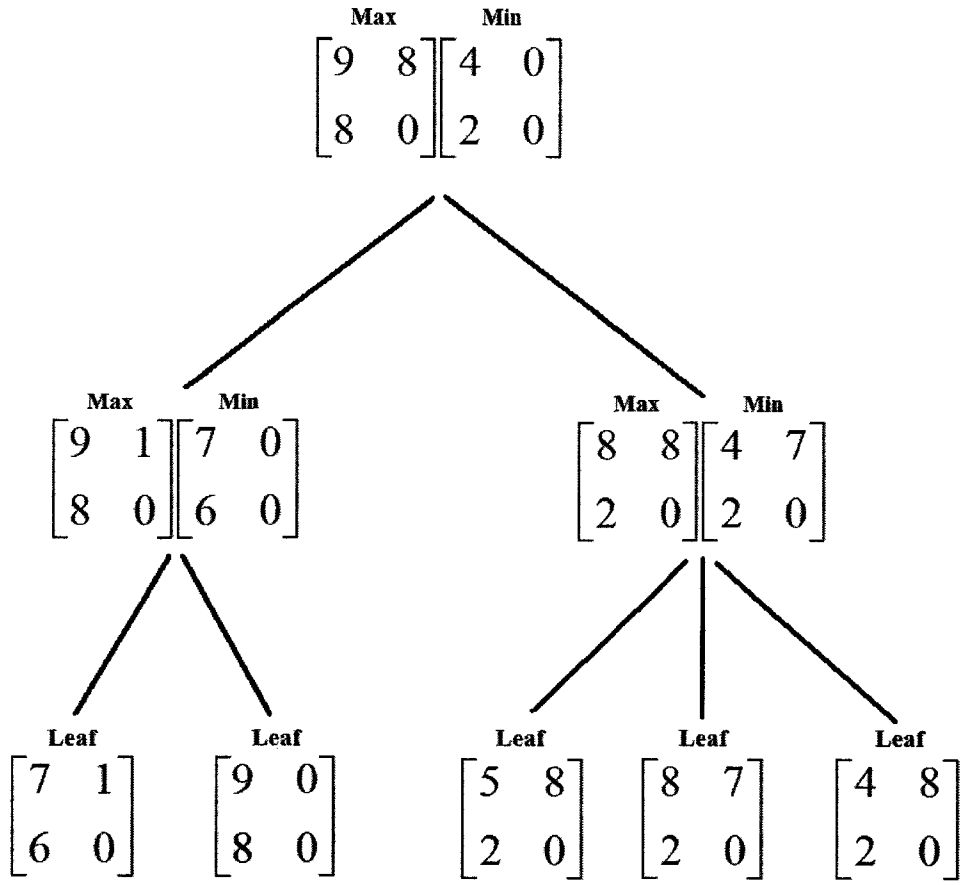


Figure 5.2: Sample Volume Distribution Tree

5.3 Establishing Jaccard Score Bounds

Following from Figure 5.1, the 258 x 258 picture has now been compressed to 16 numbers so how does the loss of information affect the Jaccard measure? We never want to exclude a result within the query range (i.e. we demand 100% recall), so the maximum possible Jaccard value (given only the volume distribution) is computed and the node being examined can be discarded if this best case scenario cannot meet the query threshold. The maximal Jaccard measure occurs when the intersection is maximized, thereby also minimizing the union. The maximum intersection within each cell occurs when the smaller volume is entirely included in the larger and thus the actual objects' intersection within each cell is upper bounded by the smaller volume. The union cannot sink below the larger volume's value since any of the smaller volume sticking out from the larger could only add to the union, and therefore the actual objects' union within each cell is lower bounded by the larger volume.

Equations 1-3 formalize these notions for the whole group of cells (in a vectorized way).

Let Q be the query object and X a leaf node's indexed object.

Let $V_Q = (q_1, \ldots, q_n)$ be the query object's volume distribution.

Let $V_X = (x_1, \ldots, x_n)$ be a leaf node.

Then,

An upper bound for the intersection is: $$|Q \cap X| \leq \sum_i^n \min(q_i, x_i) \quad (1)$$

A lower bound for the union is: $$|Q \cup X| \geq \sum_i^n \max(q_i, x_i) \quad (2)$$

Hence an upper bound on the Jaccard score between Q and X is:

$$Jaccard(Q, X) \leq \frac{\sum_i^n \min(q_i, x_i)}{\sum_i^n \max(q_i, x_i)} \quad (3)$$

What happens when we want to obtain an upper bound for the Jaccard score of any node that is a child of a given directory node? To calculate the upper bound on intersection we can use the vector of maximum volumes in place of the exact set of volumes available in a leaf node. For example if the child node volumes are [3 6 5 2] and [7 5 1 3] the matrix of maximums would be [7 6 5 3] as this is the matrix where each element indicates the maximum value, for that position of the matrix, in any of its children's distributions. Matrices representing minimums and maximums will be used for pruning branches during descent of the VD-Tree. Using the matrix of maximum values does not allow underestimation of the true maximum Jaccard score as we show in Theorem 5.1.

Lemma 5.1: The upper bound on intersection calculated using the vector of maximum volume distribution values cannot be lower than that calculated using the volume distribution values of any node under it.

Proof:

From (1), $|Q \cap X| \leq \sum_i^n \min(q_i, x_i)$

Let $D_{mx} = (d_{mx,1}, \ldots, d_{mx,n})$ be directory D's upper bound.

$D_{mx} \geq$ any of the leaves X in its subtree.

Since a minimum value cannot decrease by increasing the numbers considered, we know that $\min(q_i, d_{mx,i}) \geq \min(q_i, x_i)$.

Since the sum of a vector cannot decrease by increasing its components, we can see that: $\sum_i^n \min(q_i, d_{mx,i}) \geq \sum_i^n \min(q_i, x_i)$.

Therefore $\sum_i^n \min(q_i, d_{mx,i}) \geq |Q \cap X|, \forall X \in subtree(D)$; the maximum intersection calculated using the maximum of the range of a directory node is equal to or greater than the maximum intersection calculated using any node below this directory node.

Since there can be no underestimation, the upper bound calculated using the vector of maximums is definitely the highest intersection possible between the query node and any child (no matter how many levels below) of the directory node D. By the same logic using the vector of minimums in place of the node volume distribution in Equation 2's union lower bound calculation cannot cause overestimation of the union, thereby keeping a conservative estimate of the union lower bound.

Lemma 5.2: The lower bound on union calculated using the vector of minimum volume distribution values cannot be higher than that calculated using the volume distribution values of any node under it.

Proof:

From (2), $|Q \cup X| \geq \sum_i^n \max(q_i, x_i)$

Let $D_{mn} = (d_{mn,1}, \ldots, d_{mn,n})$ be the directory lower bound.

$D_{mn} \leq$ any of the leaves X in its subtree.

Since a maximum value cannot increase by decreasing the numbers considered, we know that $\max(q_i, d_{mn,i}) \leq \max(q_i, x_i)$.

Since the sum of a vector cannot increase by decreasing its components, we can see that: $\sum_{i}^{n} \max(q_i, d_{mn,i}) \leq \sum_{i}^{n} \max(q_i, x_i)$.

Therefore $\sum_{i}^{n} \max(q_i, d_{mn,i}) \leq |Q \cup X|, \forall X \in subtree(D)$; the minimum union calculated using the minimum of the range of a directory node is less than or equal to the minimum union calculated using any node below this directory node.

Theorem 5.1: The Jaccard coefficient between an object Q and a directory node D obtained using the maximum volume distribution values to calculate intersection and the minimum vales to calculate union cannot be exceeded by any node in the subtree rooted by D.

Proof:

Recall (3): $Jaccard(Q, X) \leq \dfrac{\sum_{i}^{n} \min(q_i, x_i)}{\sum_{i}^{n} \max(q_i, x_i)}$

Per Lemmas 5.1 and 5.2, the intersection is the highest possible for the subtree rooted by D when the directory's maximum vector is used in place of any of the actual volume distributions and the union is the lowest possible for any node in this subtree when the directory's minimum vector is used in place of any of the actual volume distributions.

$$\max_{X \in subtree(D)} (Jaccard(Q, X)) = \max_{X \in subtree(D)} \left(\frac{\sum_{i}^{n} \min(q_i, x_i)}{\sum_{i}^{n} \max(q_i, x_i)} \right) \leq \frac{\sum_{i}^{n} \min(q_i, d_{mx,i})}{\sum_{i}^{n} \max(q_i, d_{mn,i})}$$

Obtaining a maximum Jaccard score below the query threshold is therefore justification to eliminate all children of the tested node from contention without sacrificing the 100% recall criterion. Equations 4-6 formally define the upper bound on Jaccard score for any node in the subtree when examining a directory node.

Let Q be the query object and X be any object indexed within the subtree of a directory node D.

Let $V_Q = (q_1, \ldots, q_n)$ be the query object's volume distribution.

Let $D_{mn} = (d_{mn,1}, \ldots, d_{mn,n})$ be a directory node's volume distribution lower bound and $D_{mx} = (d_{mx,1}, \ldots, d_{mx,n})$ be that node's volume distribution upper bound.

Then,

An upper bound for the intersection is:

$$|Q \cap X|_{X \in subtree(D)} \leq \sum_{i}^{n} \min(q_i, d_{mx,i}) \tag{4}$$

A lower bound for the union is:

$$|Q \cap X|_{X \in subtree(D)} \geq \sum_{i}^{n} \max(q_i, d_{mn,i}) \tag{5}$$

Hence an upper bound on the Jaccard score between Q and D is:

$$\underset{\forall X \in subtree(D)}{Jaccard}(Q, X) \leq \frac{\sum_{i}^{n} \min(q_i, d_{mx,i})}{\sum_{i}^{n} \max(q_i, d_{mn,i})} \tag{6}$$

Theorem 5.1 justifies the use of the bound provided by Equations 4-6 for pruning branches. Note that these equations generalize Equations 1-3 since the element-wise maximum and minimum set for only one distribution are identical to the distribution itself. Equations 4-6 merely substitute the maximums for the exact volumes in the intersection calculation and the minimums instead of the exact volumes in the union calculation. We therefore can use Equations 4-6 to test both directory nodes and leaf nodes if we see the leaf node maximums and minimums as one and the same, the actual volume distribution for that leaf. From a bound-checking perspective, a leaf node is just a directory node encompassing the subtree it roots, which for a leaf node is only itself.

5.4 Range Query Processing

This subsection examines the process of performing a range query with the VD-Tree. A range query is one where the desired return set is the set of objects in the database

having a value within a specified range for a specified field. Instead of a field, the result of an algorithm using any database object as an argument can be used. This is the case here, where the user desires to know which tumour segmentations in the database have a Jaccard score in the range [x,1] ($0 \leq x \leq 1$) with the query tumour.

Equations 1 through 6 are the basis for querying using the VD-Tree structure. Starting at the root, the volume distribution representation of the query is tested for maximum Jaccard score using Equation 1 for leaf nodes and Equation 2 at the directory nodes. Directory nodes getting at least the query threshold for a score are expanded into their children; others are discarded as even their guaranteed upper bound is not high enough to be accepted by the query. Leaf nodes passing the test denote a single study that must be examined to determine the exact Jaccard score. So the end of the crawl down the tree yields a shortlist of studies to be examined further, saving considerable time over checking each study linearly. The algorithm yielding this shortlist is displayed in Figure 5.3.

```
Algorithm VectorNode GetCandidateList
Input:   Query object's volume distribution Q, minimum Jaccard score required for
         an object to be returned (Threshold)
Output:  List of candidates to check exactly by performing matrix-wise intersections

Queue[1] = pointer to root node of VD-Tree
While queue not empty
   X = extract head of the queue
   intMax = min(Q,X.max)
   unionMin = max(Q,X.min)
   JaccardMax  = intMax / unionMin
   If JaccardMax > Threshold
      If X.children == directories
         Append X.children to queue
      Else
         Append X.children to leafQueue
      End If
   End If
End While

For Y = each member of the leaf queue
   intMax = min(Q,Y.volDist)
   unionMin = max(Q,Y.volDist)
   JaccardMax  = intMax / unionMin
   If JaccardMax > Threshold
      Append X.study to studies
   End If
End For

Return studies
```

Figure 5.3: Candidate List Algorithm

An example should help clarify matters. Say we have 2D binary images and to make things even simpler we make the volume distribution only a 2 x 2 grid.

If we have the following 3 volume distributions:

$$C1 = \begin{bmatrix} 5 & 8 \\ 2 & 0 \end{bmatrix}, C2 = \begin{bmatrix} 8 & 7 \\ 2 & 0 \end{bmatrix}, C3 = \begin{bmatrix} 4 & 8 \\ 2 & 0 \end{bmatrix}$$

all stored under a directory node DN, this directory would have the following properties (obtained by maximizing and minimizing all indices individually):

$$DN_{mx} = \begin{bmatrix} 8 & 8 \\ 2 & 0 \end{bmatrix}, DN_{mn} = \begin{bmatrix} 4 & 7 \\ 2 & 0 \end{bmatrix}$$

Now that the grouping of nodes is obvious, a query example is in order. Say we have the following query object and tree root (note that directory nodes are displayed here with the maximums on the left and minimums on the right), with threshold 0.6:

Query: $\begin{bmatrix} 5 & 0 \\ 10 & 1 \end{bmatrix}$, Root: $\begin{bmatrix} 9 & 8 \\ 8 & 0 \end{bmatrix}\begin{bmatrix} 4 & 0 \\ 2 & 0 \end{bmatrix}$

The maximum intersection and minimum union are calculated as:

$$\sum_{i}^{n} \min(q_i, d_{mx,i}) = \sum_{elements} \min\left(\begin{bmatrix} 5 & 0 \\ 10 & 1 \end{bmatrix}, \begin{bmatrix} 9 & 8 \\ 8 & 0 \end{bmatrix}\right)$$

$$\sum_{i}^{n} \min(q_i, d_{mx,i}) = \sum_{elements} \begin{bmatrix} 5 & 0 \\ 8 & 0 \end{bmatrix}$$

$$\sum_{i}^{n} \min(q_i, d_{mx,i}) = 13$$

$$\sum_{i}^{n} \max(q_i, d_{mn,i}) = \sum_{elements} \max\left(\begin{bmatrix} 5 & 0 \\ 10 & 1 \end{bmatrix}, \begin{bmatrix} 4 & 0 \\ 2 & 0 \end{bmatrix}\right)$$

$$\sum_{i}^{n} \max(q_i, d_{mn,i}) = \sum_{elements} \begin{bmatrix} 5 & 0 \\ 10 & 1 \end{bmatrix}$$

$$\sum_{i}^{n} \max(q_i, d_{mn,i}) = 16$$

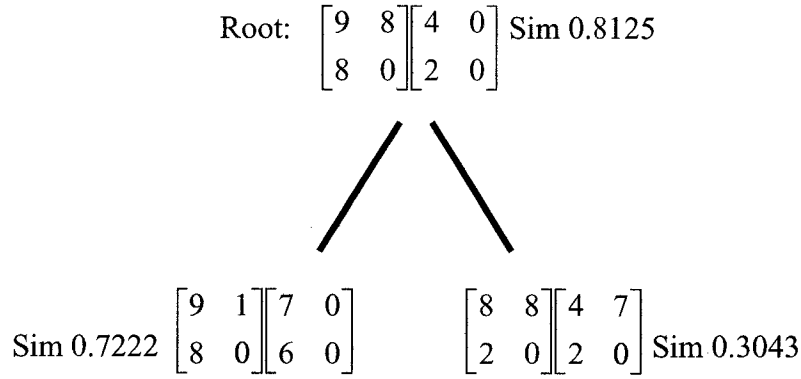
The maximum Jaccard score we can attain in the subtree below this node is therefore:

$$\underset{\forall X \in subtree(D)}{Jaccard}(Q, X) \leq \frac{\sum_{i}^{n} \min(q_i, d_{mx,i})}{\sum_{i}^{n} \max(q_i, d_{mn,i})}$$

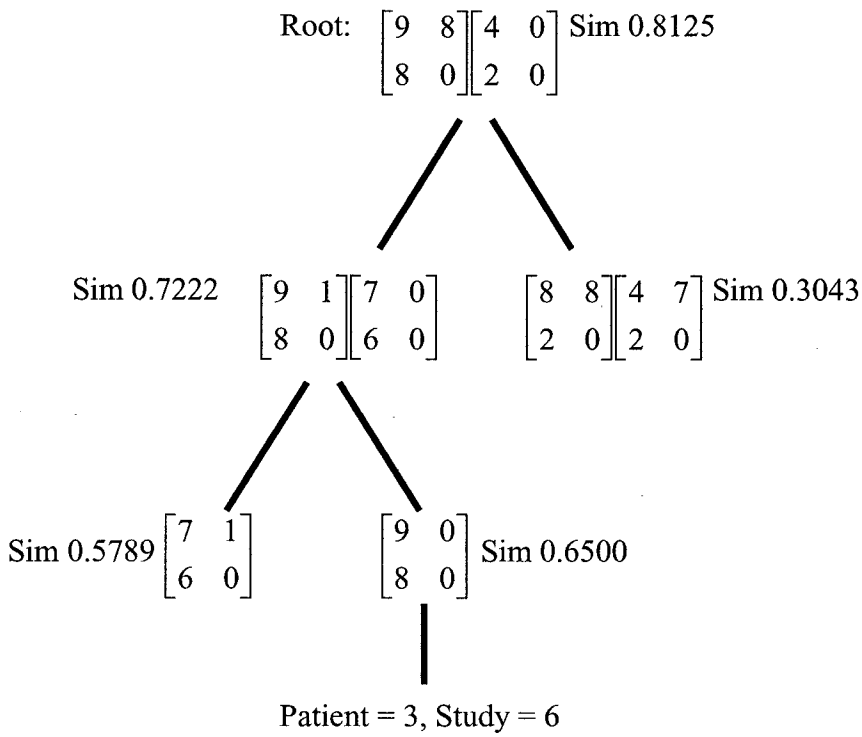
$$\underset{\forall X \in subtree(D)}{Jaccard}(Q, X) \leq \frac{13}{16}$$

$$\underset{\forall X \in subtree(D)}{Jaccard}(Q, X) \leq 0.8125$$

Since this upper bound is above the query threshold we must expand this node and perform the same procedure on its children:



For both children we calculate the upper bound on the Jaccard similarity measure but this time the right branch can be pruned since the upper bound does not meet the query threshold. We continue with only the left branch, revealing leaf nodes:



We get the upper bound on a leaf using its only volume distribution in both the intersection and union calculations per Equation 1 and obtain an unsatisfactory result (below threshold) for the left node and a satisfactory one for the right node. We therefore want to know which imaging is referred to by the right node. The patient and study numbers are contained within the node as fields in the LeafNodes table (see Section 6.2). Each leaf node showing an upper bound that is higher than the query

threshold is put into a list to be checked exactly after the tree has been processed. The advantage of the tree is that the list to be checked is typically only a small fraction of the whole list that a linear scan must check. As we show in the results section, even an R-Tree cannot help shorten the list nearly as well as the VD-Tree.

It is worth noting that a k-nearest neighbour search is possible with the VD-Tree. Due to the provable upper bounds given at any node in the tree, a best-first approach would be a good strategy to obtain the nearest neighbour or the few nearest neighbours if we keep the k^{th} best value as our bound in the best-first search. Best-first search is an algorithm where the best-looking path (the one with the highest upper bound) is followed to get a potential nearest neighbour, and other paths are only followed is their upper bound lies above the closest neighbour found to date. See Section 5.5.1 for one example of best-first search.

As a side note to this indexing for the Jaccard measure, the Volume Distribution Tree could be used to index any measure exclusively requiring some combination of volumes and the maximum or minimum of either intersection or union. Previously we showed how to obtain the minimum union and maximum intersection, but their counterparts can also be quickly computed from stored values. The maximum union and the minimum intersection occur when the volumes are as disjoint as possible; that is when they fill up different portions of the volume in each cube, intersecting only when no empty space is left. Equations 7 and 8 are the mathematical formulation of this idea.

$$|Q \cap X| \geq \sum_{i}^{n} \max(q_i + x_i - fullvolumes_i, 0) \tag{7}$$

$$|Q \cap X| \leq \sum_{i}^{n} \min(q_i + x_i, fullvolumes_i) \tag{8}$$

Each of the grid sections in the volume division has a constant volume up to which it could be filled, and this is what is listed in the fullvolumes variable above. For example if one of the grid cells is of dimension 5 x 5 x 5 it has a volume of 125 voxels so if we have 2 tumours with volumes 70 and 80 we still cannot have a maximum union of 150 voxels in this cell as there is no more than 125 voxels available to be filled. The fullvolumes variable thus is a vector of length equal to the number of cells in the grid, with each element being equal to the volume capacity of

the cell it denotes. Note that the cell volume is not necessarily equal in all divisions – in our case we unevenly divided the brain to get a more meaningful set of values, but in any case the set of full volumes is constant and can be stored as one small constant vector.

To set up a tree using the lower bound on intersection and the upper bound on union by using volume ranges like the VD-Tree directory nodes do, the minimum intersection formulation requires the use of the minimum volumes to avoid overestimation of the lower bound on intersection. Technically the set of minimum volumes could all belong to a single object, so using any values above the minimums would only heighten the computed lower bound, excluding this small-volume object despite that it may be a valid result. This potential exclusion of a valid result must never be allowed to happen. By the same logic, the maximum union requires the use of the large end of the volume range to avoid underestimation of the upper bound. Equations 9 and 10 summarize these two points.

$$\left|Q \cap X\right|_{X \in subtree(D)} \geq \sum_{i}^{n} \max(q_i + d_{mn,i} - fullvolumes_i, 0) \tag{9}$$

$$\left|Q \cap X\right|_{X \in subtree(D)} \leq \sum_{i}^{n} \min(q_i + d_{mx,i}, fullvolumes_i) \tag{10}$$

It is also worth noting that the Volume Distribution Tree structure is trivially extensible to any number of dimensions; a 4 x 4 x 4 grid is used in this work but could easily be a 3 x 3 x 3 x 3 x 3 grid of hypercubes if desired for another application.

5.5 Construction, Deletion and Insertion

A VD-Tree can be constructed by bulk-loading, insertion or bulk-loading followed by insertion. For bulk-loading, the all-pairs Manhattan distances between the volume distributions are taken, and in a loop the lowest distance between nodes where at least one is unassigned to a group is chosen. If one of the two nodes are unassigned, it is assigned to the other one's group and if they are both unassigned, they form a new group together. The upper triangular all-pairs distance matrix is updated to show infinite distances from all nodes to any node already in a full group. Once the loop has completed, the groups of leaves are completed. Each group is assigned a parent, a

new directory node whose maximum and minimum volume distribution vectors are then filled in using the group members' volume distributions. These new directory nodes are then clustered in the same manner, but since there are necessarily less of these nodes than there were in the previous clustering, there will be less groups. Each of these groups is assigned a parent directory node as before. This process continues until only one group can be formed. This singular group's parent becomes the root of the tree. The bulk-loading algorithm is shown in pseudo code in Figure 5.4.

Algorithm BulkLoad
Input: Matrix VD where each row is one volume distribution, and the maximum nodes to be put into each group, GroupSize.
Output: Grouping hierarchy and directory node bounds.

```
//Group the distributions we have together
numObjects = number of rows in VD
[G[1],Mx[1],Mn[1]] = GroupSimilars(VD,GroupSize)
//Now cluster the dir. nodes representing the lower-level groups
i = 1;
While G[i] has more than 1 element
   i = i + 1
   MxMn = concatenate Mx[i-1] with Mn(i -1) (i.e. [Mx Mn])
   [G[i], Mx[i],Mn[i]] = GroupSimilars(MxMn,GroupSize)
End While
//G[last] is the root' N children,
// G[last-1] will have groups 1 to N, with the k^th group
// representing the level above's k^th node's children
```

Algorithm GroupSimilars
Input: Matrix VD where each row is one volume distribution, and the maximum nodes to be put into each group, GroupSize.
Output: Clusters of volume distributions and their max and min bounds.

```
NumObjects = number of rows in VD
Cells = number of columns in VD
If not(leaves)
   Cells = Cells / 2
End If

For i = 1 to NumObjects
   For j = (i+1) to NumObjects
      D[i,j] = Σ |VD[row i] – VD[row j]|;
   End For
```

```
End For
The rest of D is set to Inf

While any nodes unassigned
   [i,j] = min(D)
   D[i,j] = Inf
   If i & j are unassigned
      Append group [i j] to G as a new row
   Elseif i is in group x, and j is unassigned
      Add i to group x
   Elseif i is in group x, and j is unassigned
      Add i to group x
   End If

   If group x was just filled
      D[involving members of x] = Inf
   End If
End While

NumGroups = number of groups formed
For i = 1 to NumGroups
   NumElems = elements in group i
   For j = 1 to NumElems
      If leaves
         VDmax[row j] = VD[row i]
         VDmin[row j] = VD[row i]
      Else
         VDmax[row j] = VD[1st half of row i]
         VDmin[row j] = VD[2nd half of row i]
      End If
   End For
   For k = 1 to Cells
      Mx[i,k] = max(VDmax[column k])
      Mn[i,k] = min(VDmin[column k])
   End For
End For

Return G, Mx, Mn
```

Figure 5.4: Bulk Loading Algorithm

When deleting a node, delete its parent's pointer to this node and re-calculate the parent's volume distribution. Propagate this volume distribution recalculation up the tree until it does not change or the root has been found. If the node deletion has caused its group to be reduced to only a single node, that node is also deleted and then

re-inserted into the tree. In this case its parent is no longer useful and thus it is also deleted; the node and its parent are deleted and then the node is re-inserted.

Insertion begins with a nearest-neighbour search as we would like to insert the new node below the same directory as its nearest neighbour. The node is inserted into its nearest neighbour's group as a child of this neighbour's parent, which can cause overflow (in this work we allow a maximum of 6 children per parent but this can be adjusted). In an overflow situation, the parent node is split with the children being re-grouped into two not necessarily equal-sized groups. The insertion algorithm, including the nearest-neighbour algorithm and the node-splitting algorithm, is listed in Figure 5.5.

5.5.1 Finding the Most Similar Volume Distribution

The Manhattan / city block distance metric is used to determine the nearest neighbour instead of the Euclidean metric. The Euclidean distance metric emphasizes uneven distance distributions such as a large difference in one dimension over many small distances spread over many dimensions, for example a distance of 4 spread over 4 dimensions has a Euclidean distance of 2 or half of the score of the case where the difference of 4 is in a single dimension. For insertion into the tree we want the least stretch to the directory max/min volume bounds regardless of the spread of these stretches.

To find the most similar volume distribution, a Roussopoulos-type nearest neighbour algorithm is used since it quickly finds the nearest neighbour, is easy to implement, and needs no global variables [22]. We only search nodes where it is possible to find a neighbour closer to the inserted node than the current best one found. We thus start with an effectively infinite bound such that any node is a potential candidate, but we quickly update this. The search starts off by looking at the root's children and calculating the new node's lowest possible distance to the subtrees rooted by each. The lowest distance from a new node A to any node in a subtree rooted by node D is the sum of the distances of A's vector outside the bounds of D's max and min vectors. That is, for A's volume distribution V_A and D's maximum and minimum volume distributions $\{D_{mx}, D_{mn}\}$, the distance from A to D is taken as the minimum Manhattan distance between multidimensional point V_A and

multidimensional bounding box $\{D_{mx}, D_{mn}\}$. For example with a volume distribution of [2 6 4], A cannot be any closer than a distance of 3 from any child of D with max [7 5 6] and min [4 3 3] since A's first element is at least 2 lower than any child of D and its second element at least 1 above any child of D.

With the best distances to all of the root's children calculated we expand the best one. This process is recursive, meaning that the first few expansions will be the root, then its 'best' node, then that one's 'best' node and so forth until a leaf node is found. With leaves we do not need to set bounds on the best possible distance, we directly calculate it and return that value back up the chain of recursive calls. This proceeds as follows: when a 'best leaf' value is received by the program instance checking that leaf's parent and its siblings, the instance updates its bound to reflect the new 'best node' and now will not search any nodes that don't have a possibility of yielding a lower distance. This instance now sends new recursive calls to check any still-promising nodes on the same level and once done returns the best node bound up to its caller, who then continues with the same process. In the end, the original instance of the nearest neighbour function called returns the best node and its distance from the inserted node.

5.5.2 Overfilled Nodes

When we add the node to a group of children, it may cause that group to become too large and thus we have to split it. If not, the new node is added as a leaf, its nearest neighbour's parent adds the new node number to its child list and update the parent's min and max bounds, propagating this update up the tree. The overfilled node situation is handled by splitting the parent node into two. Note that this splitting process can be recursively carried out up the tree if necessary.

To maintain the performance of the tree, we need to form two close-knit groups as the child sets of the two parents after the split. To this end, we start by getting the pair-wise distances between each pair of nodes which for a capacity of 6 nodes i.e. 7 nodes at overfill we need to compute (7 choose 2) = 21 distance combinations. The furthest two should definitely be in different groups and thus are chosen to be the first members of each group. Next, the closest node to either group is added to its closest group and the other group gets its closest node as a second member. Each group now

has two members and this is the minimum allowed for a group per this procedure and this tree implementation. The groups' maximum and minimum bounds are now computed. The nodes that are still unassigned to any group will now be assigned in a loop. In each iteration of this loop, the amount by which each node would stretch the current bounds of each group is computed. The node causing the lowest amount of stretch for either group is added to this best group. For example, adding a node with volume distribution [2 6 4] to group 1 with max [5 5 6] and min [3 4 3] would stretch the group's max to [5 6 6] and its min to [2 4 3], i.e. a stretch of 1+1+0 = 2. If the stretch to group 2 is larger than 2 and no other node could be inserted into either group without stretching them by less than 2, this node would be added to group 1. The max and min bounds of the group with the new member are updated where necessary and the next iteration is set to add another node. The iteration stops once all nodes have been placed in a group. Note that the groups can be quite different in size, with the minimum of two nodes in each group being the only size constraint.

Through recursive splits we may increase the height of the tree (if splitting the root), but can never cause leaves to become on different levels. Each split maintains the two parents at the same level as the single one before the split and retains the children at the level below these.

Algorithm NodeInsertion
Input: One study s and its volume distribution VD.
Output: A database row is written.

```
Node NN = NearestNeighbour()
Int ID = Next Unused Value in LeafNodes table
Insert into database (LeafNodes):
   (Node = ID
   Study = s
   Parent = NN.parent
   VolDist = VD)
AddChild(True, ID, NN.parent, VD, null)
```

Algorithm NearestNeighbour
Input: A volume distribution VD (single or [max min] concatenated), the number of cells, the subtree root and the current nearest neighbour acting as a bound.

```
Output:        The volume distribution's nearest neighbour in the subtree rooted by
               Root, and this nearest neighbour's Manhattan distance from the given volume
               distribution.

Best = Bound
BestNode = null

If Children Are Leaves
  For each child C
    //Calculate how far it is from the node's distribution
    D(C) = 0;
    For i = 1 to Cells
      D(C) = D(C) + |VD(i) – C.VolDist(i)|
    End For
  End For
  Return Root.children(index of lowest D)
Else
  For each child C
    //Calculate how much it is out of the node's bounds
    D(C) = 0;
    For i = 1 to Cells
      If VD(i) > C.Mx(i)
        D(C) = D(C) + VD(i) – C.Mx(i)
      Elseif VD(i) < C.Mn(i)
        D(C) = D(C) + C.Mn(i) - VD(i)
      End If
    End For
  End For

  D = Sort(D) in descending order
  For each child C in ascending order of distance D
    If D(C) >= Best
      //Stop looking down this branch
      Return [BestNode, Best]
    End If

    [Candidate, Dist] = NearestNeighbour(VD, Cells, C, Best)
    If Dist < Best
      Best = Dist
      BestNode = Candidate
    End If
  End For
  Return [BestNode, Best]
End If
```

Algorithm AddChild
Input: A node to be added to the tree, its new parent node, whether or not the child is a leaf node, the child's volume distribution (single or range).
Output: A child is added to a directory node.

```
Append child to Parent's children vector
NumChildren = Cardinality of Parent's children
If NumChildren > MaxChildrenPerNode
   int D(,) = Upper triangular matrix where D(i,j) = manhattan dist
      between Child(i) & Child (j)
   [i,j] = argmax(D(D<Inf))
   Set Child(i) as Group1's only member
   Set Child(j) as Group2's only member
   k = Child with smallest distance to either Child(i) or Child(j)
   If Child(k) closer to Child(i) than to Child(j)
      Append Child(k) to Group1
      Append to Group2 the unassigned node nearest to Child(j)
   Else
      Append Child(k) to Group2
      Append to Group1 the unassigned node nearest to Child(i)
   End If
   Calculate element-wise Max/Min bounds for the 2 groups
   While there are still unassigned nodes
      stretch1 = manhattan distance from group1's bounds to each
         remaining node
      stretch2 = manhattan distance from group2's bounds to each
         remaining node
      Append node with min stretch to either group to that group g
      Update group g's Mx & Mn bounds
   End While

   Node NewNode = new node housing group 2
   //Append NewNode to Parent's children
   addChild(False, NewNode, Parent.parent, Group2.Mx,
      Group2.Mn)
End If
Update Parent's Mx & Mn Bounds
```

Figure 5.5: Node Insertion Algorithm

5.6 Implementation-Specific Details

For the query of brain tumours in a database, we have implemented the Volume Distribution Tree on top of a Relational Database Management System (RDBMS).

The tree structure is actually two simple database tables with PL/SQL functions controlling the querying and management of data within the tables.

The two tables are LeafNodes and DirectoryNodes, keeping track of leaves and directories respectively. As remarked earlier, the min and max volume range for a leaf node are equal, and actually by using Equation 2, we can compare a query with a node without needing to know what type of node it is if we redundantly store leaves' volume distributions twice as a max and min. In this case, only one table would be required instead of two, but we opted for the two-table design to avoid redundant storage and null values for properties that one kind carries and the other does not.

Each LeafNodes row houses a node ID, a parent ID, the represented (patient, study) pair, and the vector of volumes of the cells. A DirectoryNodes row also contains a node number and a parent number, but there are two vectors for the volumes – a maximum and a minimum volume division for its children, as well as a child pointer vector (has the node numbers of the children). The directory nodes have positive ID's and the leaves have negative ID's to make it easy to discern which kind of node a pointer is referring to.

A query proceeds down the tree using SQL select statements to pick up nodes, where each node is a row from one of the two tables. A queue of nodes to be examined is kept, starting out with only the root in it. Each time around a loop, the head of the queue is extracted and used as the node to expand for the duration of that loop. This node is used to compute the upper bound for the Jaccard score between the set of objects it represents and the query object. If the upper bound meets or exceeds the query threshold its children are either added to the back of the queue if they are directory nodes or to the leaves queue if they are leaves. The loop can now begin again, ending when it checks for a new node and instead finds an empty queue. Once this has occurred, we simply have a list of leaves to check. First the leaves are checked for their upper bound on Jaccard score to see if there is any reason to load their actual matrices. The ones with an upper bound meeting or exceeding the query threshold are exactly checked by having their binary tumour label matrices retrieved from the database and checking for intersection of the intersecting part of their minimum bounding rectangles. Thus, once a query has made its way down a tree, we

have a shortlist of real imaging studies to be closely examined and then possibly sent to the user.

Chapter 6: System Architecture

6.1 Overall System Component Connection

Before going into how each component works, we shall look from high above at the communication between these parts. Figure 6.1 displays the interconnections between the primary components of the system as well as the components' locations. The server machine runs Oracle 10g and Matlab R2007a with the Image Processing and Database toolboxes on Red Hat RHAS4.

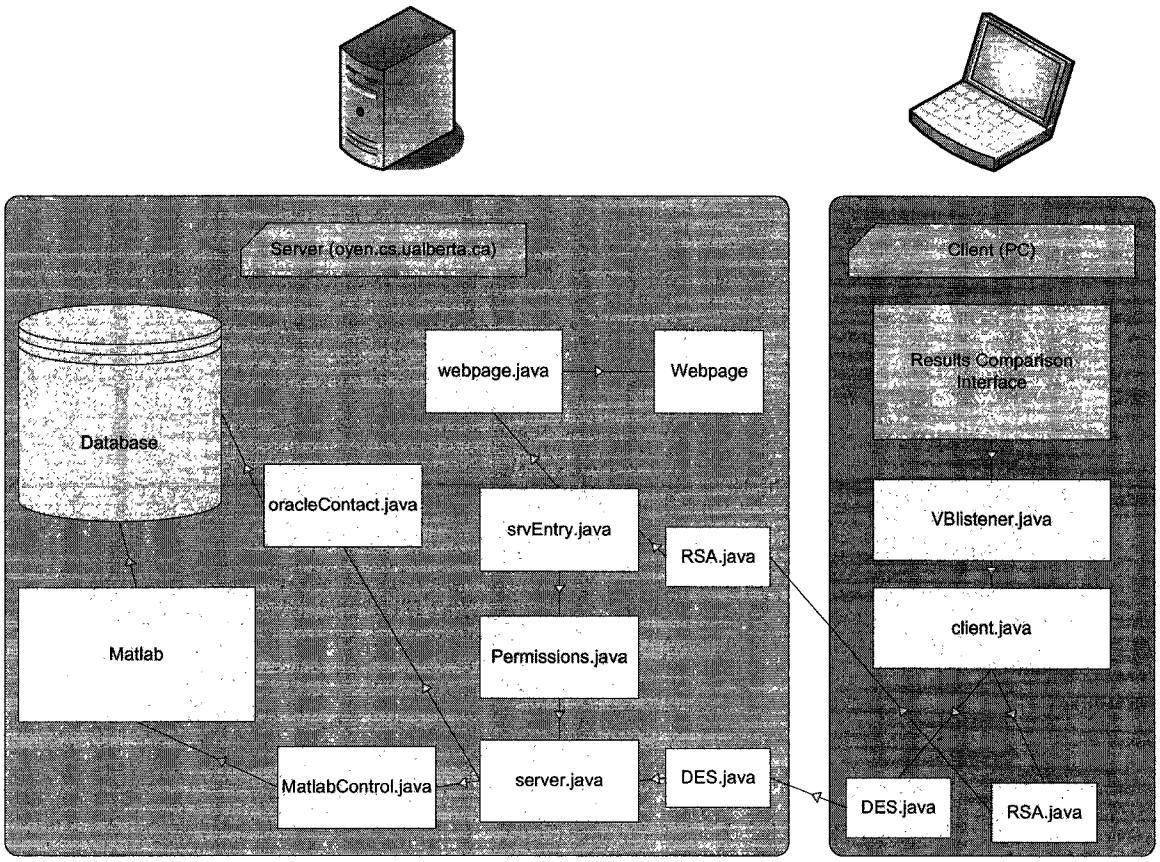


Figure 6.1: High-Level System Architecture

The front-end is a Windows form providing an easy-to-use interface to the user. The form has been implemented in Visual Basic.NET to facilitate development and rapid addition of new user-desired features. As we will explore shortly, the front-end communicates with java components by passing strings over a port and thus this

front-end could later be modularly replaced with a program written in another language or even a web interface.

The user logs into the server, via the front-end, to create a session. During this session, the user poses queries and views the results until deciding to terminate the session. A session will be defined here as the entirety of interaction between the client and server to provide the user with query results and visualizations, from login to closing the client and connections.

When a user wants to use the Results Comparison tool, they can simply double-click on it as is the norm (after installation of course). At the start of the session (during launch of the interface window), the Results Comparison Interface starts the VBlistener class (stored inside javaclient.jar as its main class) and connects to it via a local port. The VBlistener class' purpose is to listen to a local port for instructions from the VB interface to pass to send along to the server, as well as returning strings from the server to the interface. VBlistener instantiates a single Client class when starting up and this establishes a connection to the server's srvEntry class via the RSA class. The srvEntry class provides a single point of entry to the server and sets up the actual session with the Server class upon checking the client's credentials; i.e. srvEntry is the receptionist and security checkpoint for the server. The RSA class merely encapsulates the RSA encryption and decryption along with send and receive capabilities. The client's job is to handle the communication with the server such that after authentication and delegation of a port by srvEntry the VB program can effectively communicate with server.java, which can locally access both Matlab and the database. If srvEntry approves the user, the first available port is allocated to this user – client.java is notified and the user information is listed along with the port in the singleton permissions.java class. If there are no immediately available ports, the user is placed in queue until there is room. The server's opening webpage is a simple list updated by srvEntry to show the queue such that the user gets a sense of how long the wait might be.

The number of users that can simultaneously use the system is limited to the number of Matlab licences on the server, as each user requires a unique instance of Matlab to have their own variable set in a workspace. Each instance is running one

instance of the server.java class listening to one specific port. Using the open-source Octave as a replacement for Matlab would only be possible with open-source equivalents to the database toolbox and the image processing toolbox, and even then the number of open instances would have to be limited to maintain acceptable performance from the server. An alternative, now that Matlab is becoming multithreaded, may be to have one instance of Matlab run all the commands with each user's variable names being appended with their own hash code.

In any case once the user's information has been entered into Permissions and the client has been notified, the client disconnects from srvEntry and establishes a new connection with server.java, this time via DES.java. The connection details are examined in-depth in Section 6.5. From the connection to the Server class until the user disconnects, the specific instance of Matlab (and server.java) connected to are considered checked-out and only usable by this user, therefore not accepting requests from anyone else. Upon starting communication with a user, the first order of business is to set up the database connection objects in Matlab. These are the Matlab's database toolbox ODBC-JDBC connection as well as oracleBlob and plsqlRunner, which handle requests for BLOBs, user-defined types such as vectors of integers, PL/SQL functions, and any other functionality where Java was more suitable for implementation than Matlab. Now that the user interface can securely communicate with server.java, the user interface can echo the user's commands to this java class, who in turn relays the commands to either oracleContact.java or MatlabControl.java if the command is one of the listed allowable commands and is in the right form. The oracleContact module is just an API for easy access to the database, in this case usually to obtain a specific image from the database in png form to be shown to the user. Similarly, the MatlabControl module (created by Kamin Whitehouse while a PhD student at the University of California at Berkeley) is just an API to allow server.java to call Matlab functions as though it were typing into Matlab's command line. Matlab typically obtains some data from the database through its own ODBC-JDBC module and then performs manipulations on it before returning results to MatlabControl. The information can be passed back along the

same chain of components as far as need be. The current list of permitted Matlab calls is:

- PutIntoCell / PutIntoMat (amalgamates several variables into a cell array or matrix, checking that each input is a workspace variable to ensure that no dangerous commands could be issued using this name change as a circumvention method)
- DBlogin (logs this Matlab instance into the main database)
- query (runs a database query to find similar tumours to a specified instance, which must be previously put into the workspace using another command)
- notInRes (replaces the query results list with the list of studies *not* retrieved by the query; this is used to fulfill the NOT operator in Boolean queries)
- multiQueryDB (runs a few queries, with the end result being the application of the user-supplied Boolean operations to the set of queries; for example a user may create a drawing and ask for all tumours intersecting it at least 30% but not touching the left side of the brain)
- intersectionMapPostQueryDB (uses the query results to make a 3D matrix where each voxel's value is the number of tumours intersecting there)
- volumePcts (finds the percentage of the aggregated tumour volume that is taken up by 1 or more tumours, 2 or more, 3 or more... such that the colormap below the aggregated picture can show what percent of the volume is taken up by x number of results or more)
- writeAggregatesFromDB (retrieves the T1-weighed modality imaging from the database as a 3D matrix, overlays the aggregation of tumours (from intersectionMapPostQueryDB) onto it and writes the slices as png images to the filesystem for pickup by the user interface)
- createOverlaySet (amalgamation of intersectionMapPostQueryDB, volumePcts, writeAggregatesFromDB)
- makeColorChart (creates the correct image to display the mapping between color and represented value)

- resultsetTouchingXYZ (finds all tumours in the current result set that touch the voxel (x,y,z)
- removeRow (removes the specified result from the set)
- makeStruct (amalgamates tumour information into a Matlab 'struct' for processing by scripts such as query)
- amendMat (adds points to a polygon describing the user-drawn query region)

Also note that each expected number is extracted by parseInt or parseFloat in Java to avoid any user-defined strings from getting through and issuing illegal commands. Note that the workspace variables are preserved during a session, so having x=4; at one point means the variable x is still 4 later unless overwritten by another user function. Because of this workspace, operations can be defined using string variables in the VB program thereby avoiding the need to transfer large matrices back and forth through a network.

At the end of each session, Matlab's workspace variables (other than the server java object) are automatically cleared such that users cannot feel effects from the last user nor spy on them. The permissions class is given back permissions to allocate that Matlab instance to another user; i.e. the server class loses the user's public key and the port number re-enters the srvEntry class' allocation queue.

An administrator must start the Matlab instances and the single srvEntry for the system to be ready to accept client requests. An easy way to perform these tasks and keep them running after disconnecting from them is to use VNC (Virtual Network Computing). With the vncserver process started, a remote VNC client just watches a screen output rather than connecting like SSH, so after graphically starting the required processes the VNC client can be closed without affecting the server. The srvEntry process is started simply by entering "java srvEntry" at the command line while each instance of Matlab needs to have "srv = server(port);" entered into it after start-up – each with a different port number selected from the list inside of srvEntry.java. The simplified Matlab interface available with the nojvm option cannot be used since creating Java objects, which requires an active JVM (Java Virtual Machine), is necessary for the operation of the command-receiving subsystem

as well as the database query subsystem. Trying to start the server with the UNIX nohup option did not work as Matlab would not keep listening for commands upon closing up the SSH terminal. VNC allows the administrator to graphically use Matlab as well as later return to view and modify the current state of affairs. Thus the recommendation here is to use a VNC terminal, but any means of starting and checking the Matlab instances that is convenient to the administrator will allow the system to work and be transparent to the end-user.

6.2 Database Subsystem

The backbone of the system is the database, wherein lies all the data to be viewed via the front-end. The database must not only be a large repository for imaging and patient data but must also be organized in such a way to provide efficient access to the data and be extensible. The database chosen was Oracle 10g due to its features and its large user base. More users would mean more accessible support for issues in setup and maintenance via web forums.

Patient data and their imaging are stored in one main table, where each row corresponds to one study, i.e. one day that the patient came in to be imaged. The table has the (patient #, study #) pair as its primary key although the RowID also provides a unique key for the table. The other columns are:

- Tone001 – 088 (88 slices of T1 images stored as BLOBs)
- ToneC001 – 088 (88 slices of T1C images stored as BLOBs)
- Ttwo001 – 088 (88 slices of T2 images stored as BLOBs)
- Seg001 – 088 (88 slices of segmented images stored as BLOBs)
- Outlined001 – 088 (T1 images with red-outlined segmentations as BLOBs)
- T1 (T1 image as a BLOB holding a 3D matrix of 8-bit integers)
- T1C (T1C image as a BLOB holding a 3D matrix of 8-bit integers)
- T2 (T2 image as a BLOB holding a 3D matrix of 8-bit integers)
- Mat (segmented image matrix, but only the section within the minimum bounding rectangle is stored here)

- MBR (the segmentation's minimum bounding rectangle as a vector)
- Vol (segmented tumour volume in voxels)
- DMap (distance transform map in 3Dmatrix form)
- Depth (maximum value of the tumour segmentation's distance transform)
- Core (core point; the point with maximal distance transform value)
- Mids (2Dmatrix where each row is a point in the recursive mid splitting – see Section 4.2.6)
- OctVol (2x2x2 division of volume for the VD-Tree, as a vector)
- VolDiv4 (4x4x4 division of volume for the VD-Tree, as a vector)
- Date (day the imaging took place)
- Wts (eigenweights found by eigen decomposition)
- ShapeHist (shape histogram values flattened to a vector)
- Elong (ratio of the 1^{st} to the 2^{nd} eigenvalues from PCA of the segmentation)
- Flat (ratio of the 2^{nd} to the 3^{rd} eigenvalues from PCA of the segmentation)
- Sphericity (Sphericity value of the binary tumour segmentation)
- Enhancement (average intensity level increase from T1 to T1C within the segmented region)
- Diam1/Diam2/Diam3 (the diameter of the tumour along its 3 principal directions)

Note that some of the columns contain redundant information to improve performance. Tumour imaging is kept in picture form for viewing with any image display application, as well as in 3D matrices meant for Matlab. Matlab technically could pull up each slice of an MRI image and construct a 3D matrix and could conversely take the matrix and write each slice's image to disk but both of these would be very computationally wasteful. The red-outlined tumour images are produced by a Matlab function using the T1 image and the binary segmentation, but the result would be the same every time the outlined image is created for a study so why not just perform the manipulation offline and store the resulting pictures for viewing by the user interface? Although this takes up more disk space, hard drives

are now inexpensive to upgrade and the server won't have to recreate the images on demand every time a user wants it (which will likely be quite often). Note that a 'metadata' table exists in the database to include an explanation of each field of each table created. A user can therefore query by table and/or column to get textual descriptions of the fields selected.

Trees supporting queries are implemented in PL/SQL, a procedural language version of the standard database Structured Query Language (SQL). These scripts are entered into the database and generally use their own tables for support. For example, the Volume Distribution Tree implementation described in Section 5.6 keeps all of its nodes as rows in one of two tables: DirectoryNodes and LeafNodes. The directories and leaves could technically be stored in one table together, but in that case would leave plenty of null values in the columns unused by that data type.

6.3 Matlab-Based Subsystem

The mathematical manipulations performed on the data are almost entirely done by Matlab. Matlab has been proven a wonderful mathematical tool, providing a high-level language while remaining efficient for vector operations (i.e. so long as operations on a large group of numbers are called on a vector / matrix containing these numbers as opposed to writing a nested loop construct). The Matlab language not only allows for rapid development of programs but greatly decreases the difficulties in code maintenance. Due to the constant turnover of graduate students graduating and moving on, the code must be maintained and improved upon by people that have perhaps not even worked with the last person doing so.

A large percentage of the code for this project is in Matlab, and much of the Java and Visual Basic code don't need to change to add functionality to the system. For example, the encryption and inter-machine communication system can stay as a static package while changing which results are pulled up and what similarity measures are implemented. Coding new similarity measures can be done exclusively in Matlab with the database abstracted away by some Matlab functions created for this project. Matlab functions have been implemented to allow retrieval of numbers, strings, and even vectors and matrices whether stored as user-defined array types of BLOBs

(Binary Large OBjects) with no pre-requisite database knowledge. Thus, even though the Volume Distribution Tree query and insertion are implemented in PL/SQL, they could have been implemented exclusively in Matlab only possible sacrificing some of the performance and nothing else. A Matlab routine could just as well query the DirectoryNodes and LeafNodes tables, compute the maximum possible intersection, query the next children and at the end load the actual matrices from the database and perform Boolean comparisons. Since we have created Matlab functions to create and drop tables, add and delete rows and columns, and utilities to get the list of tables and their schemas including field descriptions and hints, there is generally no need for most users to even login to the SQL prompt. Since the scientific community outside computing science is generally much more comfortable with Matlab programming than c++ or Java, this is very attractive for the multi-disciplinary research aspect of the BTAP group. All of the PL/SQL functions are implemented as such for tight integration leading to efficiency, but could more easily be programmed in Matlab.

The Matlab functions created can be put into a few categories; data conversion, pre-processing of raw data to ready it for entry into the database, interface to the database, query support, visualization tools, and utility programs to perform often-used sets of commands much like macros.

Although the functions created for this project are far too many to list, a specific one whose strategy should be discussed is multiQueryDB. This is the main function handling one or more query criteria joined together. For example, how do we handle a user asking for all tumours having a shape histogram within a set Euclidean distance from a specified one, along with at least a 0.3 Jaccard score with the query object and having a volume of less than 15000 voxels? The naïve strategy would be to execute the queries in the order given and to intersect the results. In that case, we would linearly scan the whole database for shape histogram vectors within the threshold from the specified one (since large-dimensional indexing suffers from the curse of dimensionality), and then have the VD-Tree find the few Jaccard results, a simple SQL query could find the volumes above 15000, and then we intersect the results. It may be noted that the later intersection is wasteful and that the volume query could

ask for volume > 15000 AND within the current list of candidates. The main inefficiency however here is linearly scanning the whole database for vectors since there is a better way. The strategy is to always start with the tree(s) to efficiently find a short list of results. Using trees anytime later would be wasteful since the trees would have a hard time using the information provided by the other queries' results. Now that we have reduced the set of return candidates, we further reduce this set by applying the standard database query criteria such as number relations and string matching, in this example's case the volume query. Finally the most inefficient queries are run only on the set of return candidates, in the example this means the shape histogram query. This query will be run by linearly scanning only the (presumably short) list of candidates instead of the full database.

6.4 Front-End Client

A large portion of the expected user base for the tumour data repository are oncologists, as this will help them with treatment planning. The front-end must therefore allow them to pull up relevant data with ease and without any computer / database training. Database queries as well as handshaking with the server and encryption must be hidden from the user.

The client designed is a Windows multiple-document interface (MDI), created in Visual Basic.NET. Users can either choose an example from the database to use as a query object, upload a new segmentation, or even freehand draw their query. A query form can be started simply by clicking Run > Query. An example of this form is shown in Figure 6.2.

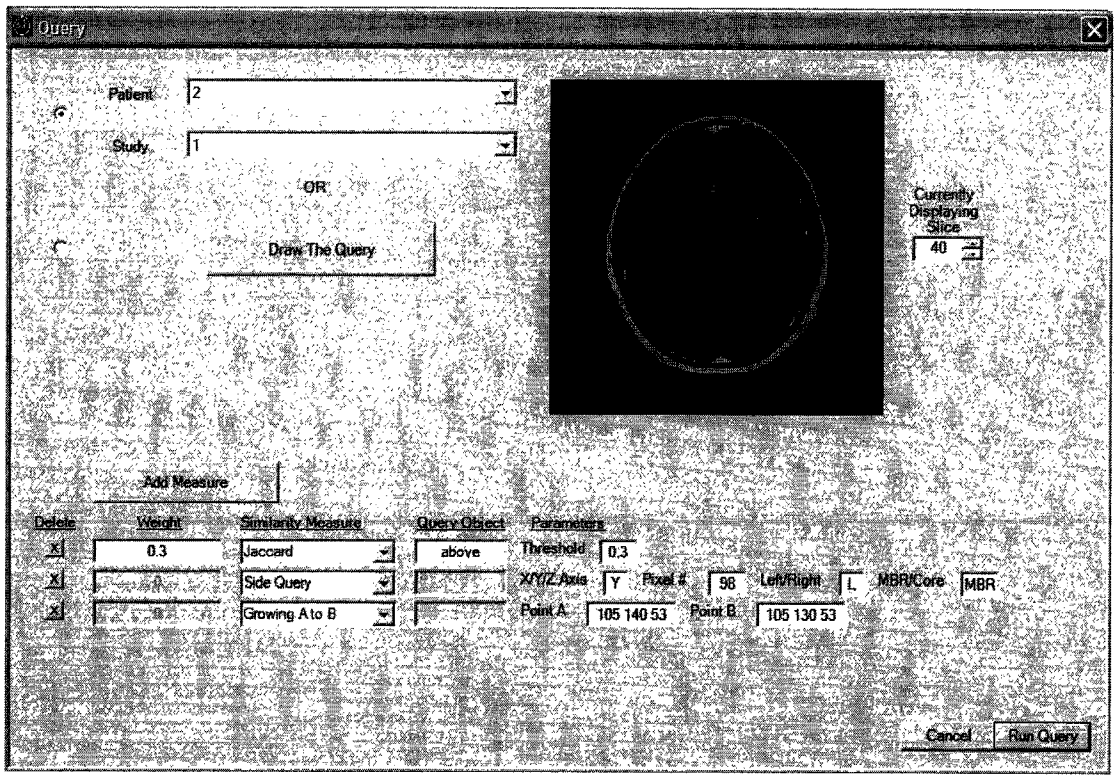


Figure 6.2: Query Form

The top half contains a choice of query objects. Most (but not all) queries necessitate a query object, which is either a segmented brain imaging session or a user-supplied drawing indicating the region of interest. The patient and study lists are obtained from the database, and the list of studies is always narrowed to only the available ones for the chosen patient. A user may well want to draw a region, wondering what tumours are present in that region – by clicking on the 'Draw the Query' button that becomes an option, popping up the 'Draw Your Own Query' form shown in Figure 6.3.

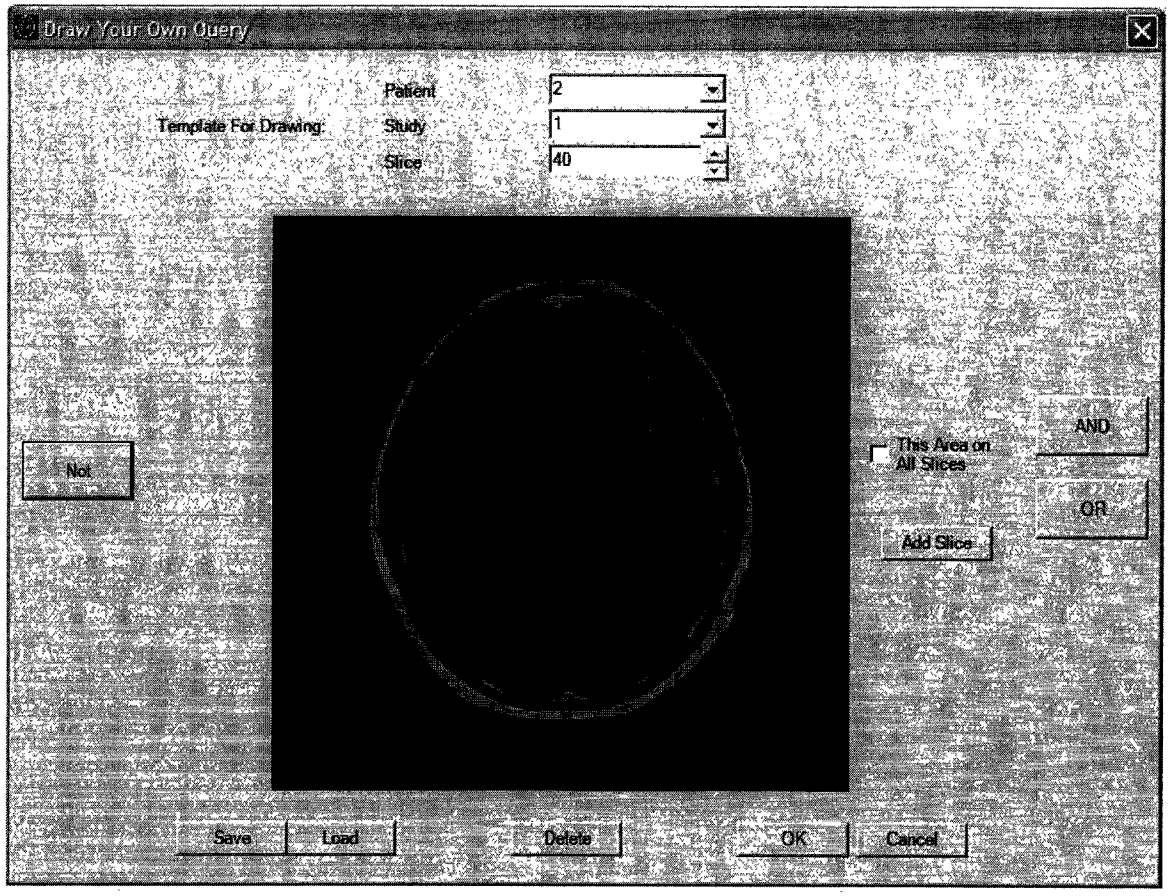


Figure 6.3: Draw-Your-Own-Query Form

Here the user can draw upon any brain image in the database, with the brain in the background being but a template such that users can orient themselves to draw in the right place. Once one slice is drawn, the user may hit the 'Add Slice' button to keep going, creating a 3D object with which to query. The 'This Area on All Slices' checkbox shortens the drawing time to find say all tumours in the medial region, by just copying the same drawing to all slices, analogously to the process of turning a circle into a cylinder. The drawing can even have regions desired and regions not desired, which can be combined using the Boolean operators AND, OR, and NOT; for example the drawing in Figure 6.3 is asking for all tumours intersecting the red area and not intersecting the purple area. The drawings can be loaded from or saved to files for later use (which is useful for reproducible results). Note that the OK button can be hit after any [positive] number of operations (slices, Boolean...) to return to the query form with the object created to date as the query object, while Cancel returns to the query form as if nothing had happened.

The bottom half of the query form serves to actually pose the query, now that we have a query object. Each time the 'Add Measure' button is pressed, a row for imposing a similarity measure appears below the rest. Any row can be removed by pressing its delete button (with an X in it) on its left side. For each row, choose a similarity measure from the list and enter the parameters desired for that measure. While many measures will simply restrict the results by discarding those that do not meet the required threshold, some return a degree of similarity ranging from 0 (not similar at all) to 1 (exactly the same). The measures returning degrees of similarity will have activated boxes for entering a weight – this is in order to rank the results with different importance placed on different queries. The weights are always internally normalized, so there is no need for the user to ensure that they add up to 1; rather it is recommended to keep one of the weights at 1 and to change the others – for example setting the next one to 2 to reflect it being twice as important as the first. The 'Query Object' box is activated for all similarity measures requiring an object – it can be set to a user-defined number or set of numbers, but is generally set to "above" to use the query object set in the top of the form.

Once returned, query results are displayed within the MDI, as small windows containing a thumbnail of slice 40 of this result's T1 image and the patient and study numbers are displayed in the title bar along with the similarity score. The results are ordered with respect to similarity from left to right in rows from most to least similar. Figure 6.4 displays a typical set of results returned from a query.

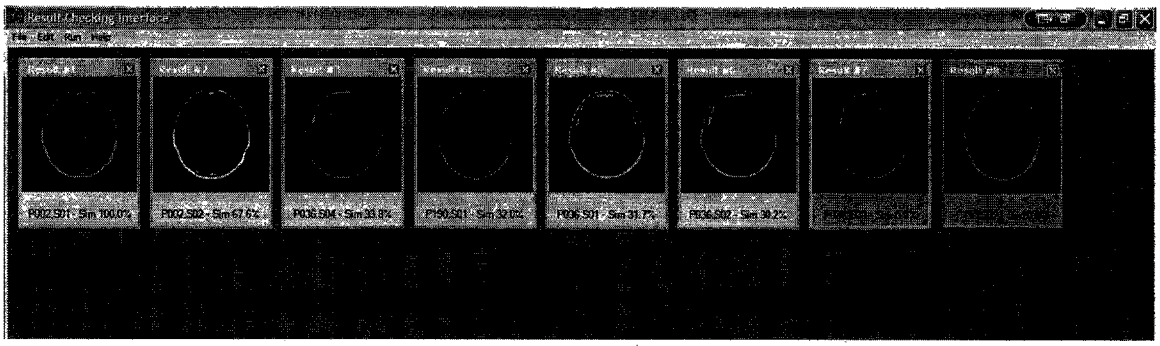


Figure 6.4: Typical Display of Query Results

Note that the last few results are in orange-coloured boxes – this indicates that they are not true results of the query in the sense that they do not meet all of the query criteria (*e.g.* do not meet a specified similarity threshold) but instead are present as

the future timestamps of at least one of the retrieved results. The idea behind this is that the user wants to study the growth of tumours and thus needs to see what later happened with the relevant tumours. For example take a tumour that meets a 30% Jaccard score with the query but later grows in another direction and thus no longer meets that 30% threshold (see Figure 6.5) – in this case the user would still likely want to see the latter case to know that that sort of growth pattern can happen to the tumours retrieved.

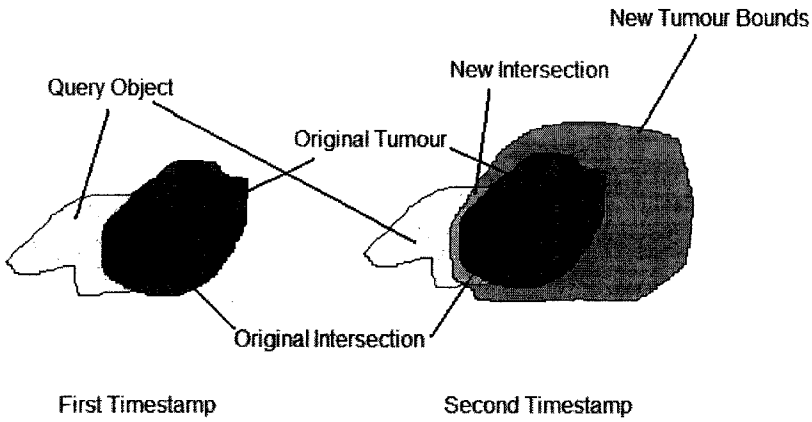
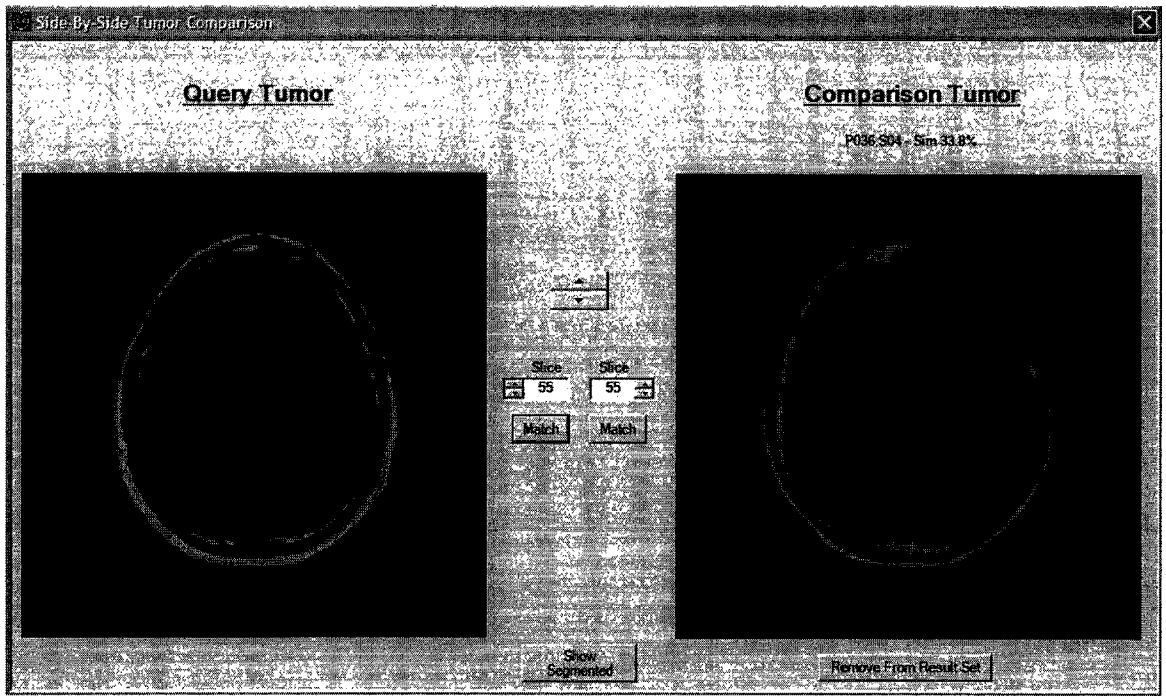
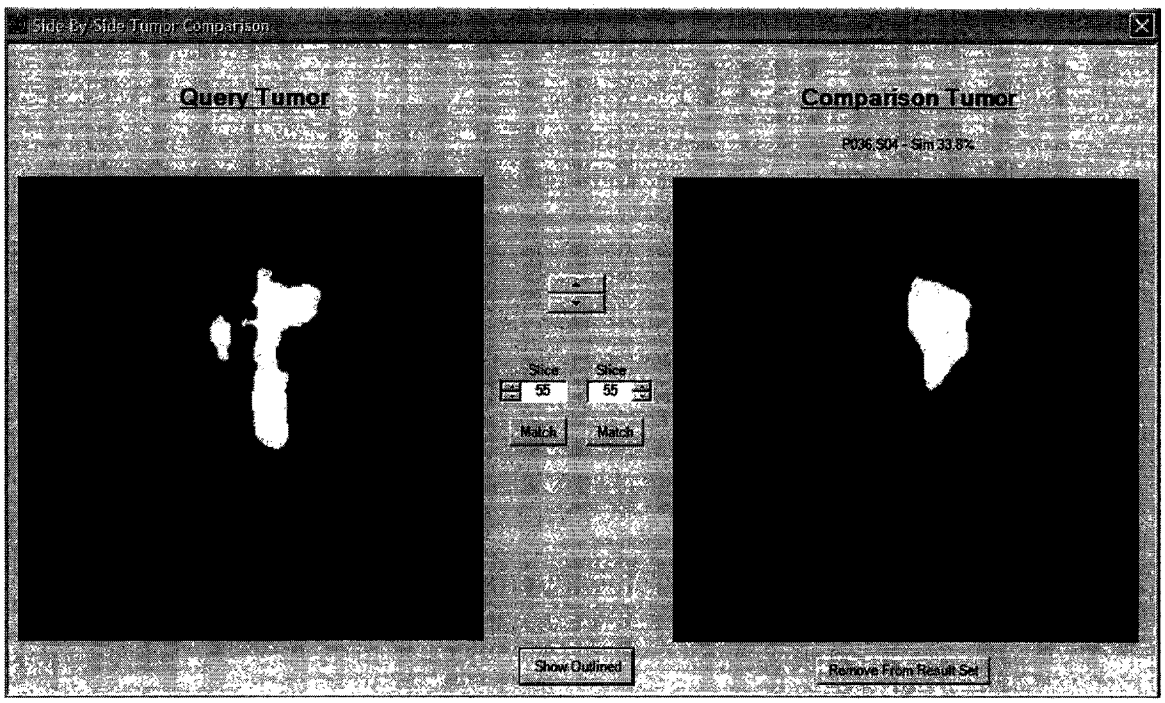


Figure 6.5: Future Timestamp Justification Example

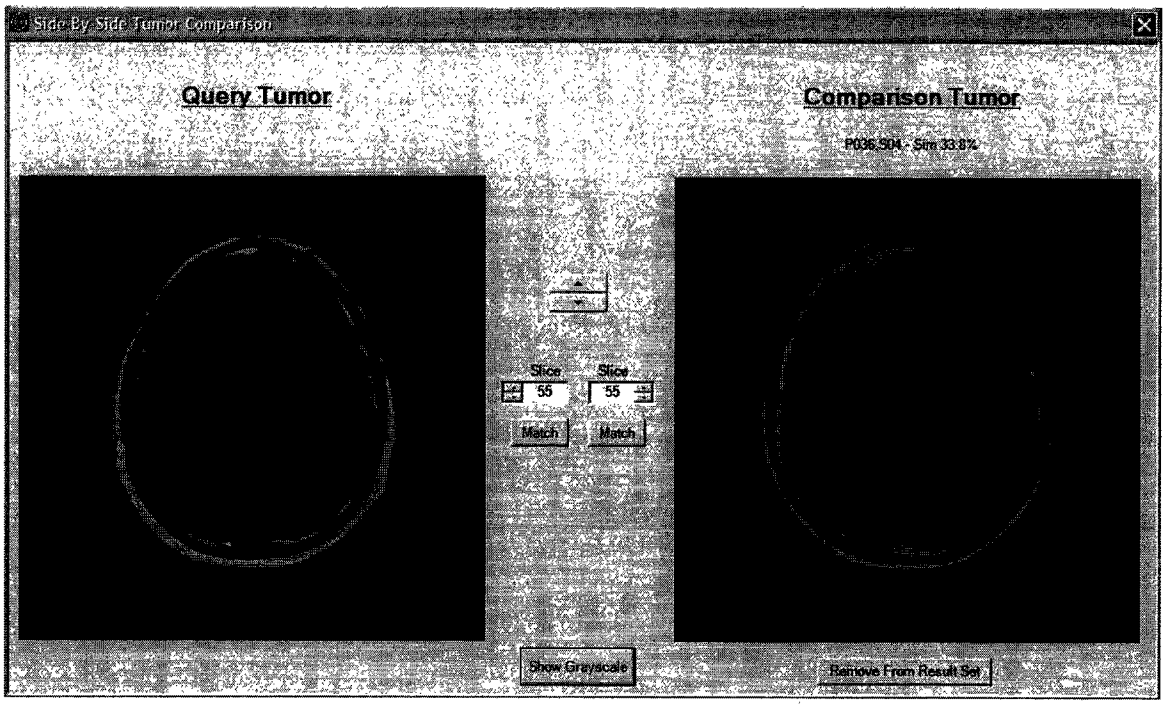
Double-clicking on any window brings up that result in a larger window comparing full-size images of the query on the left and the chosen result on the right. The two images' type can be toggled between three types: pure T1, segmentation, and overlay (T1 with a red outline representing the segmentation). These three forms are displayed in the screenshots of Figure 6.6.



A: T1 Images



B: Segmented Images



C: Outlined Images

Figure 6.6: Side-By-Side Result Comparison

The 'Side-By-Side' form permits easy scrolling through the slices to visually assess the similarity of the particular result as well as the quality of its segmentation. The user may at their own discretion press a button to discard this study from the result set, so as not to be distracted by it nor have it be shown in any aggregate statistics. A great opportunity for improving the query system would be to collect data regarding which results were excluded by physicians and to use machine learning to gradually optimize the retrieval of relevant studies.

When faced with a group of results, instead of looking at them all individually, it may be advantageous to see an aggregate picture of the set (this feature was requested by an oncologist in our group). By choosing the Aggregate option from the Run menu, a form similar to the single result comparison form is opened. It shows the query image on the left, but the right now has this query image overlaid with coloured regions much like heat on a weather map. The red areas are those where all tumours in the dataset intersect whereas much of the brain should not be overlaid with any color as areas untouched by color contain no tumours in the dataset. A screenshot of this form is displayed in Figure 6.7.

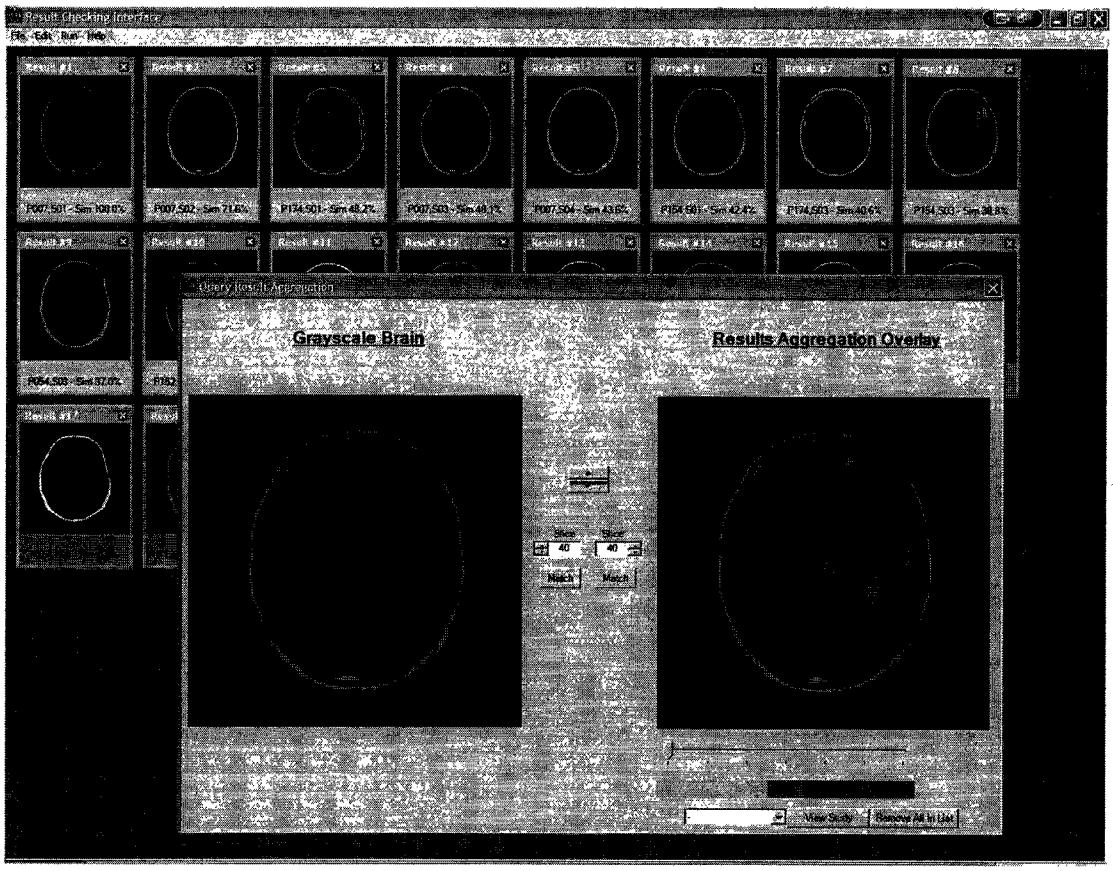


Figure 6.7: Results Aggregation Form

The mapping between color and number of intersecting tumours is displayed in a box below the aggregation picture. This color map has a slider bar underneath, which is used to clip off results below a chosen threshold. This can be useful to focus attention on a narrower region when there are a large number of results. Clicking on any pixel in the aggregate image populates a list with the tumours which include that pixel. Not only does this inform the user, but it can be used to immediately bring up the single result comparison form of a chosen member. This renders the exclusion of a small number of non-belonging results very easy. That is, some results that although technically meeting the query criteria didn't turn out to be relevant to the user's questions (in the same way as not every Google search result is relevant to you when posing a question) can be quickly and easily excluded from the dataset.

6.5 Secured Internet Connectivity

Care must be taken to ensure that the brain tumour database is not compromised. It must be secured such that an attacker cannot view sensitive data and cannot delete or modify the data in any way. Ethics approvals allowing us the use of brain imaging data prohibits these images from being shown to the public to ensure patient confidentiality. Deletion of data would be an obvious problem due to needing to restore the data from backup, and modification of data is even more threatening as it may not be noticed and can sway query results.

A human-reviewed list of allowed users is maintained on the BTAP server, with each user being associated with a password. Obviously the connection between the client and this server must be encrypted prior to password transmission until the connection is terminated to prevent both decryption of packets by snoopers and snooping of the password (which subsequently allows the snooper to log into the system as a standard user). All communication between the client and server is done by writing byte arrays to sockets. So any communication is converted to a byte array, encrypted, and then sent to a pre-specified port to be picked up by another component.

Sending information back and forth can most efficiently be done using symmetric-key cryptography. The Data Encryption Standard (DES) encryption algorithm implemented in Java was chosen to provide an easy-to-use and efficient interface for writing and reading streams of bytes that would travel encrypted between two components. By piping the output of a Java DataInputStream into the input of a CipherInputStream, the programmer can transparently write to a stream as if it were not encrypted.

Symmetric key algorithms have the obvious weakness that both parties require the same key, so after the server generates this key (one generated per session), it is wrapped by RSA encryption and sent to the client. Since RSA is an asymmetric key algorithm, the client is able to decrypt any byte arrays that the server has encrypted with the client's public key. RSA could be used for the entire session if time was not important but encrypting and decrypting a 1MB picture takes on the order of minutes

even for state of the art hardware on both the server and client side, whereas this takes only part of a second using DES.

The full procedure is as follows:

1. Both client and server only possess their own set of RSA keys (1 public, 1 private) as files in a private folder.
2. Client connects to server's main port, sending its public key in the clear as its first message.
3. If the server finds no error opening this key, the server replies with its public key in the clear. The client receiving this knows that the server is now expecting its password, and thus reacts by encrypting the password with the server's public key and sending it. The string sent is of the form "ip address//username//password".
4. The server checks the password and if correct, sends back (encrypted with the client's public key) a string indicating which port to connect with for this session. The server entry class also writes to the local static permissions class for said port to only accept connections from this same "ip//user//pass" combination.
5. Upon receiving this number, the client disconnects from the server entry point to connect to the port specified, sending its "ip//user//pass" combination encrypted with the server's public key as a first message.
6. The server class handling this session checks the password and sends an encrypted pass or fail message to the client.
7. If passing the password test (which should be the case if steps 1-4 have been followed), the client generates a DES key, wraps it using the server's public key and sends it over.
8. The server unwraps the DES key and sets up a cipher stream to communicate with the client. All further communication is performed over this DES stream.

Chapter 7: Experimental Results

In this section we analyze the performance of the record retrieval done by the Matlab and PL/SQL scripts. All tests were run on the database server oyen.cs.ualberta.ca, having an Intel Xeon 5130 (dual-core 2.0GHz CPU, 1333MHz FSB, Core® microarchitecture, 4MB shared cache), 8GB of DDR2-667 ECC memory, hardware Raid-0, running Red Hat RHAS4 with Oracle 10g EE. For experiments, each of the 320 tumour segmentations in the database was used one-by-one as the query object. The grid used for the volume distribution was a 4 x 4 x 4 grid, covering the voxel ranges:

x = 1-84, 85-123, 124-162, 163-258
y = 1-95, 96-129, 130-163, 164-258
z = 1-27, 28-42, 43-57, 58-88.

For a given query object, the Jaccard score threshold was taken as one of the following values: 0.01, 0.1, 0.2, and 0.3 with the reason being that lower thresholds result in higher numbers of returned results. For each query object and Jaccard score threshold combination, we performed 20 runs of the query, interleaving the two query methods being compared in order to minimize caching effects. Each of the 20 runtimes for each method was then averaged to give a mean time for an object and dataset percentage combination. The runtime for a given percentage of the dataset was then taken as the mean runtime for all queries which returned that percentage of the dataset.

We first quantify the speedup gained by using the Volume Distribution Tree as compared to a linear scan of the database which checks the intersection of minimum bounding rectangles before checking the actual intersection of the tumour volumes. In this way the linear scan discards large portions of the MBR and often prevents a check of the actual intersection (in cases where the MBRs don't intersect) to be effectively as efficient a linear scan as possible. Figure 7.1 shows the time to return the query as a function of the portion of the database returned, comparing the Volume Distribution Tree to a linear scan of the dataset.

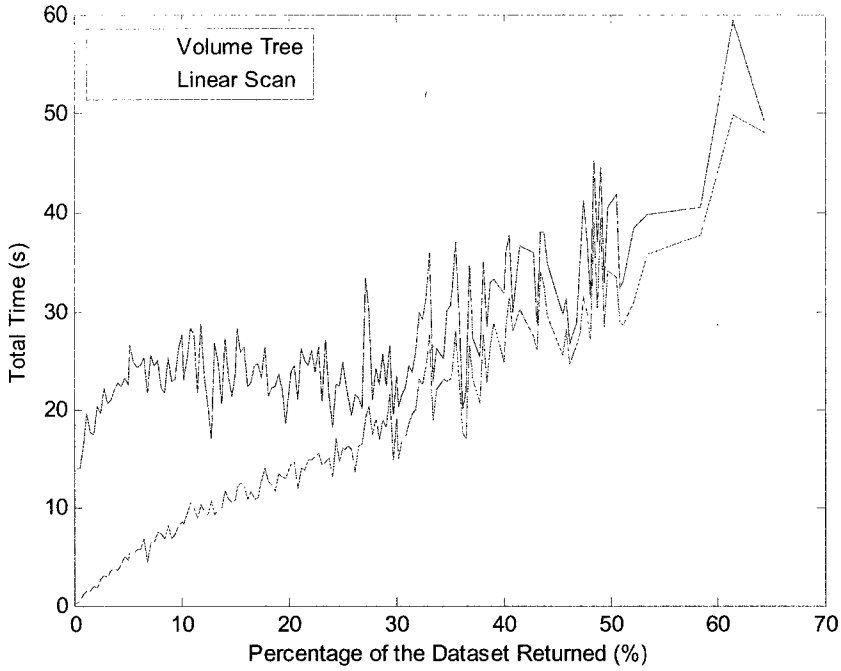


Figure 7.1: VD-Tree Speedup with Increasing Fraction of the Database Returned

As the fraction of the dataset returned becomes higher, the narrowing down done by the tree becomes less significant, and for the case where the full dataset is returned, no pruning is possible, so any work done by the tree is pure overhead above the subsequent sequential scan. The increasing trend in the sequential scan plot shows this decrease in effectiveness of the Volume Distribution Tree as a large portion of the database is retrieved. Different query objects are what causes the variation in the percentage of the dataset returned, and intuitively the MBR's of the objects returning more query results are likely to have a higher average intersection with the MBR's of the tumours in the database. This is the reason for the linear scan graph having an increasing trend despite checking a constant amount of objects. The Volume Distribution Tree shows a clear advantage until at least 25% of the database is returned, and additionally outperforms a linear scan at every point tested (tests up to 64% of the database returned).

Oracle Spatial has a built-in R-Tree in 2 to 4 dimensions, so it can quickly sort through the 3D MBR's to come up with a shortlist to check exactly. As we can see in Figure 7.2, MBR intersection is a rather poor filter when compared with volume

division. It is due to the tighter filtering done by the Volume Distribution Tree that this tree can easily outperform even an efficiently-implemented R-Tree.

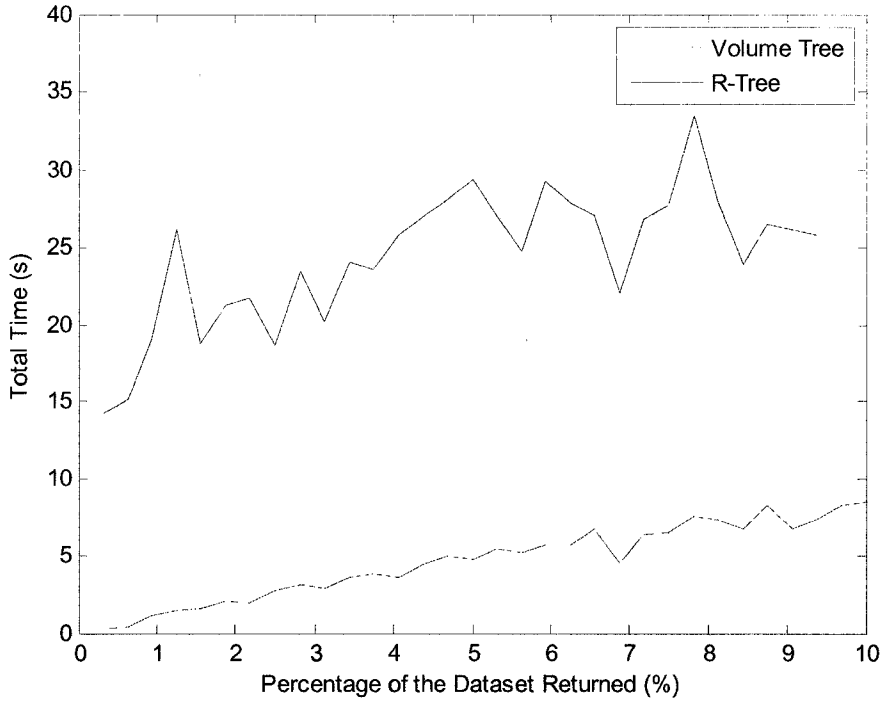


Figure 7.2: Volume Distribution Tree vs. 3D R-Tree

By using each tumour image in the database as a query object with a Jaccard score threshold of 0.3, the mean number of results is 10.8. The Volume Distribution Tree gives us an average short list length of 19.5 to check while the R-Tree only narrows things down to 177.6 on average. Since it is the exact checking that takes much more time than the trees, trees that can provide the least amount of useless items to check while still being efficient perform best. The Volume Distribution Tree running time was only 16.5% more than the R-Tree's on average (0.0965s vs. 0.0876s), and with an average total query time of 22.8s with the R-Tree, the exact checking time clearly dominates the running time of the tree itself, rendering the Volume Distribution Tree easily faster than the R-Tree for this problem.

The resolution of the Volume Distribution Tree (i.e. changing it from 4 x 4 x 4 to n x n x n) can be optimized experimentally, mitigating the trade-off between tree processing length and exact checking length but this would be valid only for a given data table length and number of retrieved results. The number of records will

continue to increase as new patients are imaged which will increase the tree processing time, and user queries have result sets with wildly different cardinalities.

Chapter 8: Future Work

The database will keep expanding as new requirements are placed upon it as well as with the influx of new data. Users will be the greatest source of new feature requests as they will find things they would like to do that the system does not allow or is too difficult to understand at which point the software should be modified to meet the users' needs. This section will detail possible routes for improvement as seen by the author.

A major area for future work is the machine learning of relevant results based upon which results are discarded by the oncologists using the system. The users of this system will be highly-qualified individuals whose opinion of what tumours are relevantly similar is very valuable to our research. The system as built enables collection of this data while improving the user experience rather than impeding them.

Recall the Z-Score segmentation technique from Section 3.3: a mean value matrix and a standard deviation matrix computed offline give for each voxel an expected value and a standard deviation. Outliers, i.e. voxels falling abnormally far away from the mean, are labelled to create an easy-to-segment map generally highlighting the tumour region. The segmentation by Z-score suffers from noise in terms that an area of bright voxels being slightly moved from the usual similar area (e.g. a normal brain fold being slightly off from its standard location) can cause an outline of high Z-scores. To combat this, for each voxel the surrounding voxels' expected values should be considered rather than just the voxel in question. A multi-resolution strategy would also be in order for dealing with small displacements between images.

As for an implementation detail that needs work, the crash recovery needs to be improved to keep the system responsive when widely available. A client-side crash or disconnection should result in the instance of Matlab being used on the server to free itself from the client connection after a reasonable reconnection period. Currently a crash will generally cause a disconnection, cutting off the current user and causing them to re-start at the back of the line. Even worse, occasionally

disconnections are unregistered by the server, keeping Matlab locked in to the disconnected user. Upon reconnecting, the user a new Matlab instance while the old one stays unused until reset by an administrator.

An extension to the Volume Distribution Tree would be for each node to contain multi-resolution volume distributions. For example when obtaining the upper bound for Jaccard score from a node and the query object, instead of only calculating this using the 64-length vector we do now we could first calculate the bound with an 8-length vector made by just using the middle planes to separate the space into octants. When at a node we would check the Jaccard upper bound using the coarse grid and if above threshold then check it again with the fine grid and if still above threshold then explore the branch. The idea is that sometimes we can prune branches with just the coarse resolution and not even have to use the finer one, saving on computation. An optimization should be done to find the optimal set of resolutions for the dataset we use and again when much new data comes in, to check if the optimal set is heavily or lightly changing.

We have explored here some avenues for improvement that have not yet been implemented for shortage of time and/or uncertainty about the usefulness of the improvements until the use patterns can be analyzed.

Summary

This document has outlined the data accessibility work performed for the Brain Tumour Analysis Project. The traditional Jaccard measure describing similarity is now only one of a set of similarity measures designed to provide more flexibility to the user, especially in terms of choosing the applicable aspects of similarity. Data structures have been implemented in the database where necessary to speed-up the query process.

A computing science department server, *oyen.cs.ualberta.ca*, now houses both Matlab and the database, as well as the middleware necessary to communicate with the custom client software on the user end. The questions the user asks the interface are transformed into short encrypted strings and sent to the server. The server decrypts questions from allowed clients and sends these to the database, generally via Matlab. Processed and encrypted results are sent back to the client for the user's viewing and analysis.

Functionality can be added to the system using almost exclusively Matlab programming, yielding an easily extensible system for non-computer scientists. This makes working on this project more attractive and feasible for medical students wishing to get into research.

Finding the optimal treatment envelope for brain tumours is far from solved, but the groundwork laid out in this thesis is a good base for research leading to this.

Glossary

- **BLOB:** Binary Large OBject; long binary array loaded into a database. This array can be anything, so the retriever must have a way of knowing the type of the data to make it useful. For example, matrices and pictures can be stored in this way.
- **DES:** Data Encryption Standard.
- **Distance Transform:** Image processing technique for binary images that assigns each Boolean high pixel a value indicating its distance to the nearest Boolean low pixel.
- **Edema:** Abnormally large concentration of fluid around the tumour; swelling.
- **GTV:** Gross Tumour Volume. The tumour visible in MRI images, including the necrotic core and edema.
- **JVM:** Java Virtual Machine.
- **MRI:** Magnetic Resonance Imaging.
- **Pixel:** Elementary point in a picture, usually displayed as a single-coloured square.
- **RSA:** Popular public-key encryption algorithm invented by Rivest, Shamir, and Adleman.
- **SQL:** Structured Query Language, the standard syntax for using a database management system.
- **SSH:** Secure Shell; UNIX program allowing one machine to control a UNIX machine via a virtual terminal.
- **Study:** A set of images of a patient's brain, all taken in one session.
- **T1:** MRI image modality showing fat locations (such as white matter and gray matter) more brightly than non-fat locations.
- **T1C:** T1 image taken after patient is injected with a contrast agent (gadolinium) to more clearly differentiate the tumour from the rest of the brain.

- **T2:** MRI image modality highlighting water-filled locations (such as the ventricles) more brightly than fat locations.
- **VB:** The Visual Basic.NET programming language a high-level compiled language, executable on systems running the .NET framework.
- **VD-Tree:** Volume Distribution Tree, a structure to index solid objects, described in this document.
- **VNC:** Virtual Network Computing; allows a user to remotely view and affect another machine's screen output. As opposed to SSH terminal sessions where each SSH session is a new UNIX session analogously to logging in to a machine, each VNC session just connects to an already running UNIX session analogously to turning on a computer monitor.
- **Volume Distribution:** Used in the VD-Tree. It is the vector or matrix whose elements are equal to a segmentation's volume in the element's corresponding grid cell.
- **Voxel:** 3-dimensional version of a pixel.

Bibliography

[1] A. Adan, and M. Adan. *A Flexible Similarity Measure for 3D Shapes Recognition.* IEEE Transactions on Pattern Analysis and Machine Intelligence. Volume 26, No 11, p1507-1520, 2004.

[2] M. Ankherst, G. Kastenmuller, H.P. Kriegel, and T. Seidl. *3D Shape Histograms for Similarity Search and Classification in Spatial Databases.* Advances in Spatial Databases, 6th International Symposium (SSD'99). Volume 1651, p207-228, 1999.

[3] N. Beckmann, H. P. Kriegel, R. Schneider, B. Seeger. *The R*-tree: an efficient and robust access method for points and rectangles.* Proceedings of the 1990 ACM SIGMOD international conference on Management of data, p322-331, Atlantic City, New Jersey, May 23-26, 1990.

[4] S. Berchtold, D.A. Keim, and H.P. Kriegel. *The X-tree: An Index Structure for High-Dimensional Data.* Proceedings of the 22nd International Conference on Very Large Databases (VLDB 96), p28-39, Bombay, India, 1996.

[5] D. Bespalov, A. Shokoufandeh, W.C. Regli, and W. Sun. *Scale-Space Representation of 3D Models and Topological Matching.* SM '03, Seattle, WA. p208-215, 2003.

[6] Brain Tumour Analysis Project (BTAP) website (Online). BTAP, http://www.cs.ualberta.ca/~btgp/.

[7] Q. Ding, M. Khan, A. Roy, and W. Perrizo. *The P-Tree Algebra.* Proceedings of the 2002 Symposium on Applied Computing, Madrid, Spain. p426-431, 2002.

[8] R. A. Finkel and J. L. Bentley. Quad Trees: A Data Structure For Retrieval on Composite Keys. Acta Informatica. Vol. 4, No. 1, p1-9. 1974.

[9] A. Guttman. *R-Trees: A Dynamic Index Structure for Spatial Searching.* Proceedings of the 1984 ACM SIGMOD International Conference of Management of Data. Boston, MA. P47-57, 1984.

[10] M. Hilaga, Y. Shinagawa, T. Kohmura, and T.L. Kunii. *Topology Matching for Fully Automatic Similarity Estimation of 3D Shapes.* ACM SIGGRAPH, Los Angelas, CA. Vol.3, p203-212, 2001.

[11] C. Holmes, R. Hoge, L. Collins, R. Woods, A. Toga, and A. Evans. *Enhancement of MR Images Using Registration for Signal Averaging.* Journal of Computer Assisted Tomography. Vol. 22(2), p324-333. 1998.

[12] P. Jaccard. Bulletin de la Société Vaudoise des Sciences Naturelles. Vol. 37, p241-272.

[13] K. Kaku, Y. Okada, and K. Niijima. *Similarity Measure Based on OBBtree for 3D Model Search*. Proceedings of the International Conference on Computer Graphics (CGIV '04). 2004.

[14] D. A. Keim. *Efficient Geometry-based Similarity Search of 3D Spatial Databases*. ACM SIGMOD, p419-430, 1999.

[15] H. P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. *Effective Decompositioning of Complex Spatial Objects into Intervals*, Proceedings of the IASTED International Conference on Databases and Applications (DBA 2004), Innsbruck, Austria, 2004.

[16] H.P. Kriegel, M. Potke, and T. Seidl. *Managing Intervals Efficiently in Object-Relational Databases*. Proc. 26th VLDB Int. Conf. on Very Large Data Bases (VLDB'00), Cairo, Egypt, p407-418, 2000.

[17] H.P. Kriegel, M. Potke, and T. Seidl. *Interval Sequences: An Object-Relational Approach to Manage Spatial Data*, Proceedings of the 7th International Symposium on Spatial and Temporal Databases (SSTD'01), Redondo Beach, CA, in: Lecture Notes in Computer Science, Springer, Vol. 2121, p481-501, 2001.

[18] G. Mazzara, R. Vlthuizen, R. Pearlman, J. Greenberg, and H. Wagner. *Brain Tumor Target Volume Determination for Radiation Treatment Planning Through Automated MRI Segmentation*. International Journal of Radiation Oncology • Biology • Physics. Vol. 59(1), p300-312, 2004.

[19] S. Mukai, S. Furukawa, and M. Kuroda. *An Algorithm for Deciding Similarities of 3D Objects*. SM '02. p367-375, 2002.

[20] R. Ohbuchi, M. Nazakawa, and T. Takei. *Retrieving 3D Shapes Based On Their Appearance*. MIR '03. p39-46, 2003.

[21] A. Rosenfeld and J. L. Pfaltz. *Distance Functions on Digital Pictures*. Pattern Recognition. Vol. 1, p33-61, 1968.

[22] N. Roussopoulos, S. Kelley, and F. Vincent. *Nearest Neighbor Queries*. Proceedings of ACM Sigmod. P71-79. 1995.

[23] D. Saupe, and D. V. Vranic. *3D Model Retrieval With Spherical Harmonics and Moments*. DAGM '01, p392-397, 2001.

[24] M. Schmidt. Automatic Brain Tumor Segmentation. Master's Thesis. University of Alberta, 2005.

[25] M. Schmidt, I. Levner, R. Greiner, A. Murtha, and A. Bistritz. *Segmenting Brain Tumors Using Alignment-Based Features.* The Fourth International Conference on Machine Learning and Applications. Los Angeles, CA. Dec. 2005.

[26] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. *The R+-Tree: A Dynamic Index for Multi-Dimensional Objects.* Proceedings of the 13th International Conference on Very Large Databases (VLDB '87). p507-518, 1987.

[27] Sphericity (Online). Wikipedia, http://en.wikipedia.org/wiki/Sphericity.

[28] M. Turk, and A. Pentland. *Eigenfaces for Recognition.* Journal of Cognitive Neuroscience, Vol. 3, No. 1, p71-86, Winter 1991.

[29] D. A. White, and R. Jain. *Similarity Indexing with the SS-Tree.* Proceedings of the Twelfth International Conference on Data Engineering. p516-523, 1996.