GeMBASE: A Geometric Mediator for Brain Analysis with Surface Ensembles

Simone Santini† Amarnath Gupta*

†National Biomedical Computing Resource *San Diego Supercomputer Center
ssantini@ncmir.ucsd.edu gupta@sdsc.edu
University of California San Diego

1 Introduction

Brain researchers, neurologists and neurosurgeons acquire 3D brain images from normal and diseased subjects with the idea to study the properties of brains, make comparisons and to measure changes caused by factors like age and disease. Over the last few years, significant efforts have been made [1, 2, 3, 4] to create software to extract the cerebral cortex (or other structures) from the brain, and reconstruct it as a 3D digital object for further analysis. While they all use different techniques to derive the digital representation of the structure, they all represent it as some form of a *surface mesh* object, often with a thickness at every node of the mesh. In this demonstration, we present GeMBASE, a prototype system that stores brain structures in a database, and allows researchers to perform their tasks in a database environment. The system borrows ideas from information mediation to answer complex queries on surface meshes.

2 The Need for Geometric Mediation

In general, a surface mesh M can be modeled as a 5-tuple $M = \{V, L_V, P, L_P, T\}$, where V is a relation of *vertex* objects containing an ID (Vid) and 3 coordinates, L_V is a relation of *labels* over vertex points, P is a relation of k vertex ids that make a *polygon*, L_P is a relation of labels over polygons, and T (for topology) is an *adjacency matrix* over polygon IDs (Pid), specifying the neighborhood of each polygon. The cortex has a very complex geometry with numerous hills, valleys and intricate foldings. This makes it difficult for humans and programs to assign labels to its vertices and polygons, and compute derived attributes (called *measurements* hereafter) such as distances between vertices along the surface. The standard workaround provided by a surface manipulation software [2, 3, 4] is to create *an ensemble of representations* for a single brain structure, each with a different simplification. The typical representations are spheres, planes (called flatmaps) and inflations. Each simplified representation makes some labeling or measurement simple, and others impossible or prohibitively expensive. For instance, the spherical representation is well suited for the efficient computation of the distance between two points, but the original (fiducial) representation the computation is very expensive. Table 1 gives a qualitative idea of the relative cost of computing measurements on different representations.

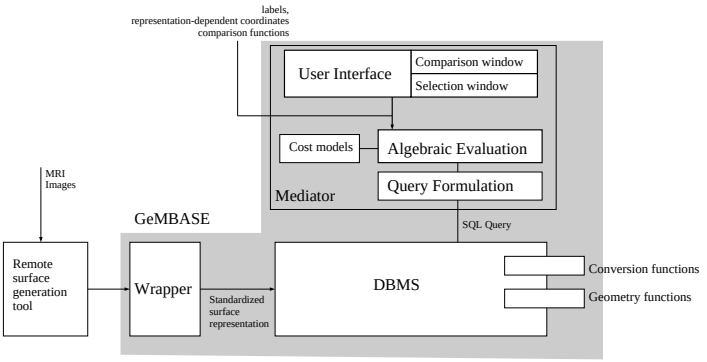

Figure 1: Architecture of the GeMBASE system

Similar to the AMOS-II system [5], GeMBASE presents to the user an *integrated complex object type* called SURFACE, such that individual data sets become instances of *surface*. At query time the system composes the user view with the integrated data type definition to determine which *concrete* representation of the surface should be used for the query.

Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002

Operation	Operations				
	Curvature	Connectivity	Distance	Area	Label Selection
Fiducial	Hi	Lo	Hi	Hi	Lo
Inflated		Lo			Lo
Spherical		Lo	Lo	Hi	Lo
Flat				Lo	Lo

Table 1: Relative costs of sample operations on the different representations

3 The GeMBASE System

Figure 1 shows the architecture of the GeMBASE system. GeMBASE uses a set of wrappers to convert from the proprietary brain surface representations produced by different surface generation software packages into a canonical form, consisting of relational tables for vertex labels, vertex coordinates, vertex-to-polygon mappings, polygon labels and topology. The relational tables are placed in a commercial RDBMS (Oracle 8i) under the control of GeMBASE. Each representation (e.g., sphere, flatmap ...) creates a different table for vertex coordinates and topology. The database has been augmented with special-purpose function libraries to perform operations like distance-based selection, neighborhood computation and conversion between coordinate systems.

Queries can be posed, and results collected, through a browser-based interface, shown in Figure 3 or through an application, shown in Figure 2 which allows the user to create selection queries that may use functions on the surface structures. In this case, the user has selected a region on the interface, and a window with high level operation is displayed. The user can choose what class of query will be made based on the selection. Additional windows will then be displayed to define the parameters of the queries.

Query rewriting in GemBASE is rule-based, and the rule set can be extended or edited remotely through a Java-based rule editor.

A limited number of join queries are allowed for making comparisons across different data sets. While textual queries can be sent from the interface, usually, the user first navigates to a data set by non-geometric attributes to identify a set of patients. The geometry query is initiated when the user brings up one (or more) surface (sphere, flatmap ...) representation into a geometric query window on the interface. The interface provides a number of buttons to perform operations such as selecting query points and labels from a region, select a range of regions inside a bounding box or a sphere, and then pass the selected regions to an "outer query" specified by filling in forms. Comparison requests are formulated by bringing up two geometry windows and choosing which regions need to be compared and then selecting the comparison function.

Internally, the syntax of a GeMBASE query is like a limited fragment of SQL where nested queries, aggregates and GROUP-BY constructs are not allowed.

Example. Suppose the schema for an Functional MRI is declared as follows:

> **create type** MYSURFACE **as surface**
> **with region labels** (stimulus_type:STRING, stimulus_amount:FLOAT, activation:STRING).

Note that **create type** statement parameterizes the surface by a flat relation for each polygon (called **region label**). Using this type definition, the actual schema is declarared as:

```
create table
    PATIENT (
        name : STRING,
        date : DATE,
        image_type : STRING,
        image_id : URL,
        cortex : SURFACE,
        cerebellum : SURFACE
    )
```

A query against this schema may look like:

```
select Point pt
from PATIENT P
where pt.label.region in
      select (q : Point).label.region
      from P.Cortex
      where dist(flatmap(screen).point(1.5, 2.1), q) < 3
```

This query retrieves all the points from regions that are at less than three mm from a given point. The expression flatmap(screen).point(200,-30) states that the input point is given from a flatmap coordinate (see Figure 2 (forcing the system to use the FLATMAP_VERTEX table). The query is parsed by the GeMBASE system and rewritten as an SQL query against the RDBMS. The important part of the rewrite is to replace the reference to the surface (i.e., cortex) by one or more concrete geometric representations. This is accomplished by the Algebraic Evaluation module of the software, under the direction of the rule base.

In this case, the response to the query is returned in a spherical representation. However, if the second query condition is replaced by

$$\text{total\_area}(R) < 300$$

the system would choose a flatmap representation instead.

It is possible to force the system to use a particular output representation by citing it explicitly in the query:

```
select Point pt as fiducial from PATIENT P
....
```

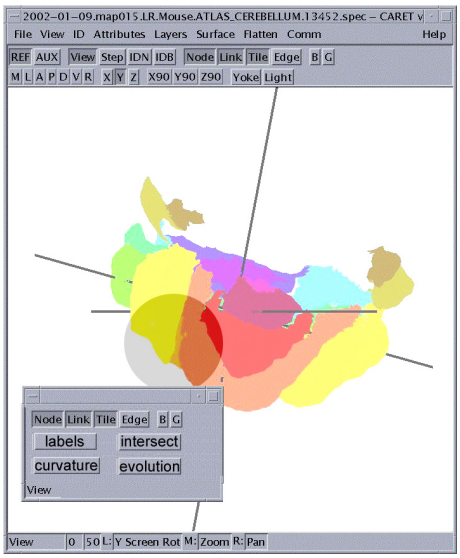

Figure 2: A user interface of the GeMBASE system

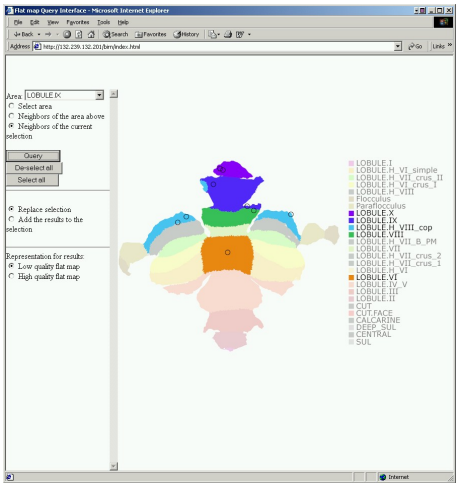

Figure 3: Web Based interface for the GemBASE system

Passing from a representation to another in our model requires a join on the point id. Given that certain operations can be executed on multiple representations, and that the input and output representations are fixed, opportunities for query optimization arise from balancing the efficiency of the required operations in certain representations with the need of minimizing the number of representation changes, that is, the number of joins. Note also that the previous query specified a geometry (fiducial) but not a topology. In this case, the default topology for the fiducial geonetry (that is, the open topology) will be used.

The query of the previous example (with the **as** clause) builds a fiducial map of all the regions that are at less than 3mm from the query point, given in the flatmap. The internal SELECT query doesn't specify in which representation should the distance be computed. However, distances can't be computed in the flatmap that constitute the input, therefore the options are either to compute distances in the fiducial map (in which case the result can be used directly to generate the output representation), or in the spherical representation (in which case the output must be converted to the fiducial representation).

More specifically, assume that the database is constituted by the following tables:

- **Label**(id : int, anatomical : int),
- **Flat** (id : int, map : SURFACE),
- **Spherical**(id : int, sphere : SURFACE),
- **Fiducial** (id : int, fid : SURFACE).

The system contains rules for rewriting queries that involve representation. The distance rule, for instance, applies to queries of the form

select c.surface.point, α **as** fiducial
from CORTEX c
where $B(dist2d(\text{c.fid.point}, \beta))$

where α and β are two unspecified parts of the query, and B is a boolean expression. This query is to be transformed into the query:

select c.surface.point, α
from FIDUCIAL c, SPHERICAL s
where $B(dist2d(\text{s.sphere.point}, \beta))$
and s.sphere.point.id == c.fid.point.id

Note that if the **as** clause is missing, and no other operations require the fiducial representation, the system will rewrite the query using the spherical representation as

select s.surface.point, α
from SPHERICAL s
where $B(dist2d(\text{s.sphere.point}, \beta))$

In general, unless the **as** clause is specified, the system will return the result using the most complex representation among those used in the rewritten query, where fiducial is more complex than spherical, and spherical is more complex than flat.

In the present example, due to the presence of the **as** keyword, the SELECT part of the query would be translated into the following:

```
select Fiducial.fid.point
from Fiducial, Label
where Fiducial.id = Label.id and
    Label.anatomical in
        select Label.anatomical
        from Label, Spherical
        where Spherical.id = Label.id and
        dist2D(Spherical.sphere.point,
                FlatToSphere(1.5, 2.1)) < 3
```

The system uses an underlying relational database (Oracle 8i), augmented with functions to compute the geometric quantities of interest on the cortex representation. The mediator is written in Java and takes care of query rewriting as well as of creating an output representation with the data retrieved from the database. The output representation is either an SVG file for display in the interface, or a file in a standardized, XML based surface format. The latter solution is used when the data are requested by a remote site which uses its own interface package.

The query rewriting portion of the mediator is written in Prolog, and is composed of three paers:

1. an SQL query parser, which takes the original query (written, in the current implementation, using a subset of SQL) and returns its syntactic tree;
2. a syntactic re-writer, which contains rules for the manipulation of the syntactic tree,
3. a query writer which takes the syntactic tree produced by the re-writer and creates the resulting SQL query which will then be send to the relational database.

At this time, the cost-based optimizer of Figure 1 is not fully implemented, so that the rewriting process is purely rule-based.

4 The Demo

The demo will show the basic functionality of the GemBASE system, using the Oracle database installed at the San Diego Supercomputer Center. The database has been extended with data types for representing surfaces, which also implement all the necessary operations. Surface objects are specialized in Spherical surface objects and in Flat surface objects.

Note that syntactc considerations alone are not sufficient for determining the correct translation of a query in one using representations. For instance, all representations contain a `dist2D` method but these methods are different semantic interpretation: on the fiducial representation, the `dist2D` method computes the distance between two points on the cortical surface; on the spherical representation `dist2D` computes the distance between two points on the surface of the sphere, obtaining a good approximation of the true distance, while in the flat map, the method `dist2D` computes the distance between two points on the flat map, which gives no indication as to the real distance between the points..

Substitutions in queries should be derived from these semantic considerations, rather than from the syntactic consideration of the availability of the function `dist2D`.

Using this data model, the demo will show examples of queries and query rewriting, as well as the effect of changing the rewriting rules on the result of the query. The query interface runs within a browser while the editor for the rules (which are written in Prolog), runs as a Java application.

5 Acknowledgments

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

References

[1] B.A. Wandell, S. Chial & B.T. Backus, *Visualization and measurement of the cortical surface*, J. Cognitive Neurosci., 12, 739-752, 2000.

[2] B. Fischl, M.I. Sereno & A.M. Dale, *Cortical surface-based analysis*, Human Brain Mapping, 8, 272-284, 1999.

[3] R. Goebel, *A fast automated method for flattening cortical surfaces*, Neuroimage, 11, 684-696, 2000.

[4] D.C. Van Essen, H.A. Drury, D. Hanlon & J. Harwell, *User's guide to SUREFIT and CARET: cortical segmentation, surface reconstruction and flattening software*, 2002.

[5] V. Josifovski & T. Risch, *Integrating Heterogeneous Overlapping Databases through Object-Oriented Transformations*. In *Proc. 25th Intl. Conf. On VLDB*, 435-446, 1999.