SPATIAL DATBASE FEASIBILITY FOR FACIAL CHARACTERIZATION USING FUZZY LOGIC QUERIES

James L. Mastros

Directed Research, Dr. Lorraine Parker

VCU School of Engineering, Computer Science

Table of Contents

1 - Introduction

In the past thirty years, storage of information has become increasingly vital to many areas of modern human life. Financial institutions and government agencies, to name a few examples, rely heavily on databases for their daily operations. It is from this need to store all types of information that there has grown specialized database management systems to administer specific types of data.

In many cases, data of one type such as financial data may have domain specific requirements or operations that aid in the storage and retrieval of that data type. An example of financial data that requires additional retrieval and update capabilities are bank transactions. Bank transactions require the ability to rollback account changes in the event that an update to an account doesn't fully complete. In the event that a user tries to transfer money from a savings account into checking, there must be measures in place to prevent the money being lost if the transfer fails and doesn't update the checking with the new balance.

One data type that has shown exceptional promise and growth over the past thirty years, especially for use with geographic information services, is spatial data. Spatial data refers to information describing "the interrelationship between objects in space" [1] and can be multidimensional. A classic example of data containing spatial relationships can be seen from a simple question involving driving directions. Imagine a person driving a car and asking where the nearest gas station is within a five mile radius relative to their position [2]. Or if they want to know several driving routes past an upcoming traffic jam that will still take them to their destination.

Those are just a few examples where an ordinary person could take advantage of spatial databases. But who else would use them? In *Spatial Databases: A tour*, written by Shekhar and

Chawala, the authors describe three classes of spatial database users. There are business, scientific, and web users. Business users utilize spatial data to "augment other forms of information while deciding about marketing campaigns, distribution centers, and retail locations" [2]. Scientific users "specialize in the study of the environment, natural resources, and geography" [2]. Web users want to focus their results to provide greater meaning to them.

Although web users, using services such as MapQuest® and Google Earth®, have been able to harvest additional capabilities in Spatial Database Management Systems (SDMS). There is also another class of users. These users are classified as criminal justice or law enforcement users. Users interested in the cataloging and mining of human facial characteristics for purposes of identification or analysis. An agency such as the *Federal Bureau of Investigation* might want to take a victim's description of a suspect's face and run that query through a spatial database to return potential matches of criminals with those facial features. Human facial features such as the distance between eyes or retina patterns have long been researched for identification purposes; a field commonly referred to as biometrics. However, the purpose of this paper is not to pick out a facial feature solely for identification purposes, but to select several facial objects and their relationship to each other for the purpose of facial characterization. In particular, to use these relationships and combine them with the work currently been done by Virginia Commonwealth University's Database Research Group.

The Database Research Group, headed by Dr. Lorraine Parker, is looking into translating fuzzy queries regarding facial characteristics into SQL statements that can search a spatial database system. Fuzzy queries are questions with non-deterministic words such as long, short, or broad. They are non-deterministic because it is unclear what constitutes a 'short' nose. The definition of short in one part of the world is completely different than another part of the world.

Therefore, the database group has been researching into translating queries containing these fuzzy terms such as “Give me all the suspects with a short nose and broad chin” into SQL statements. In order to accomplish this, community learning will be utilized to derive meanings for short and long before they are translated into queries. However, imagine for a moment the potential uses and benefits of this effort. A suspect’s facial data may be searched on a database containing millions of criminals and top results returned as an image helping a victim identify them faster.

This paper attempts to explore if spatial database management systems are capable of being used to store facial characterization data for the purpose of searching with fuzzy queries. And if so, what points, features, and objects of the human face should be captured? As far as is known, this paper will be the first to look into characterizing the human face using spatial databases. There has been significant research into accurate identification of a particular face, but none into selecting out a range of faces with features displaying a wanted spatial relationship such as “those suspects with a broad chin”.

Chapter 2 - Background

Traditional relational databases have long been and still are the primary method of storing and retrieving data. However, as explained earlier, specific types of data warrant specialized managements systems to efficiently handle data of that type. But what in particular, does a spatial database packages do that traditional relational packages do not?

In order to answer this, it is necessary to show how spatial data is different from other data. First, spatial data tends to be “more complex compared with traditional business data” [2]. As a result, the standard indexing methods are not suited to derive 2D and 3D relationships quickly or efficiently and often result in excess computation. For example, take a query “List all students located in zip code 23294”. A traditional relational database will have no retrieval problems and utilize a standard index to quickly find the results. However, if a professor asks “List all students twenty miles from the main campus” the DBMS will struggle. This is due to the distance relationship that must be derived before a search can be narrowed. In order to complete this query, the DBMS must take each zip code and translate its position in longitude and latitude and compare against the position of the main campus [2]. Second, “old database constructs are not adequate for handling” [2] spatial data. Meaning, the methods of indexing and representing spatial objects are inadequate. Third, storage requirements for spatial data are also more complex; one “low-resolution satellite picture of the United States can take as much as 30Mb” [2]. Therefore, how does a spatial DBMS handle indexing of data that is larger and more complex than traditional data?

In most relational database systems, the B-tree is utilized for indexing. Indexing is similar to an index used to find a specific page in a book and can greatly decrease the time needed to process a query [2]. However, due to the nature of spatial data, B-trees are not enough and in

some cases can result in the loss of spatial neighbor information. For example, consider a two dimensional matrix of size 4 by 4 with integer values inside each location. At any position, there will be a certain subset of values that are the chosen position's neighbors. If ordering is used as a part of B-tree conditions for indexing, the neighbors might change and result in the loss of previous neighbors at those locations [2]. Therefore, B-trees were modified into R-trees to more effectively handle multidimensional objects. An R-tree is a "height-balanced tree which is the natural extension of the B-tree for k-dimensions" [2]. It "approximates each geometry with the smallest single rectangle that encloses the geometry" [11], also known as the minimum bounding rectangle (MBR). In other words, spatial objects are "divided into cells by a regular grid" [2] then the cells it intersects with are recorded as a square object. Square objects are simpler representations and thus faster in searching and used as the spatial key. R-trees can also capture spatial neighbor information in the branches of the R-tree. See figure 1.1 for an example showing that matrix position with 2 has neighbors 4,6, and 8.

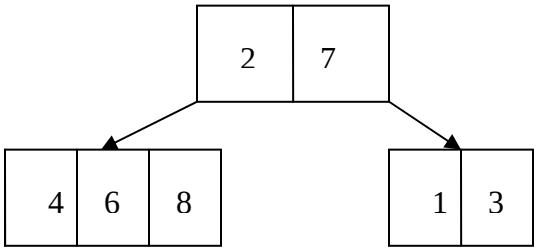

Figure 1.1

Generally, spatial indexing is accomplished in two ways. The first is through data structures designed to hold spatial objects [2]. The second is to transform spatial objects into one-dimension so they can be used with the default one-dimensional index used with B-trees [2]; often implemented with the Z-order or Hilbert curve algorithm to reduce processing time of spatial queries. In addition, spatial indexes often utilize 'buckets' to organize objects. A bucket

or container will have those objects that fall into specified regions or categories and are used for faster retrieval of data.

Spatial data is usually represented or described through geometric shapes. To be more precise, one dimensional spatial object types include points, lines, linestrings, and curves. Points, for example, can represent buildings or other locations while lines can be used for state boundaries, railroads, or power lines. Two dimensional shapes include polygons and circles. City outlines or districts are good examples of polygons [11].

The built-in functions involving these shapes can be broken up into two groups: topological and nontopological [2]. Topological refers to characteristics that involve relationships between objects that are not affected by transformations. For example, if you draw two squares on a balloon and then stretch the balloon, the squares still connect. Nontopological examples are perimeter, area, and length. If you stretch the balloon, these properties will be affected [2]. Examples of topological functions include *endpoint*, *touches*, and *contains*. *Endpoint* will return the point at the end of an arc and *touches* returns what shapes are touched by a shape. *Contains* returns those shapes enclosed within another shape. Nontopological functions include those that compute *Euclidean distance* or *area*.

In addition, most spatial packages allow collections of shapes. The advantage of providing collections is that additional geometric relationships involving set theory can be used such as intersection and difference. An example of such a query would be, “Give me all the states in which interstate 95 transverses” or “What counties border Henrico county in Virginia?” An intersect function would return true for all objects that intersect its space.

Currently, there are several spatial database management systems (SDMS) available. Commercial packages include Oracle, Sybase, and SQL Server. Oracle’s flagship product,

Enterprise Database Server 10g, has very robust spatial extensions. Autometric, a division of Boeing, which has developed *Spatial Query Server*, has as an add-in for Sybase databases. As far as open-source options, MySQL is the most popular open source database package and includes support for two dimensional geometric shapes.

In addition, each of the above packages conforms to Open Geographic Information System (OGIS) standards. OGIS is an "international consortium of more than 250 companies, agencies, and universities participating in the development of publicly available conceptual solutions that can be useful with all kinds of applications that manage spatial data. The Open GIS Consortium (OGC) maintains a Web site at http://www.opengis.org/" [10].

The two predominant formats for representing spatial data in the above SDMS are Well-Known Text (WKT) and Well-Known Binary (WKB). The WKT format is designed to use ASCII values to represent geometric shapes. For example, representing a Point in WKT uses *Point(x, y)*, similar to object constructs in Object Oriented Languages such as Java. A collection of shapes would be represented as *GEOMETRYCOLLECTION (POINT(4 35), LINESTRING(13 13, 40 40))* [10]. The WKB format utilizes binary streams of data to represent geometric shapes. It is formatted as follows: "one-byte unsigned integers, four-byte unsigned integers, and eight-byte double-precision numbers (IEEE 754 format)" [10]. A WKB example that would translate from hexadecimal into binary and represents a POINT object is *0100010000011000000000D02C000000000000E05F* [10].

At the time of writing, the latest version of MySQL is 5.0 and it does not support more than two dimensional (2D) spatial objects. Oracle supports 2D, 3D, and 4D objects. An example of a 4D object would be a tesseract or series of nested hypercubes [12] (See Figure 1.2).

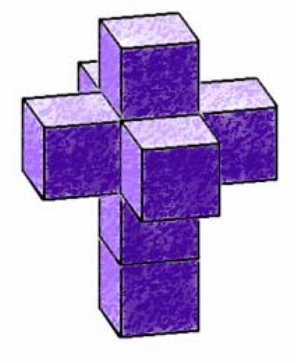

Figure 1.2 Tesseract - “4-dimensional analogue of the cube” [13]

Image taken from http://en.wikipedia.org/wiki/Tesseract

Sybase, which can handle both 2D and 3D data, takes a unique approach and categorizes its shapes into three categories. The first category contains shapes with a “predefined number of components” such as point or rectangle. The second category consists of a “variable number of components” such as a collection of shapes. And the last category, consisting of lines and polygons, are those with a “fixed number of components and defined by the user a time of creation” [23]

According to MySQL’s reference manual, MySQL 5.0 uses two methods for optimizing spatial searches used in constructing spatial indexes. The first is to search for all points within a desired region and the second is to search for all objects that are contained within a region. MySQL uses “R-Trees with quadratic splitting to index spatial columns” [4]. Oracle also uses ‘Fast R-trees and quadtree indexing”[11]. Sybase utilizes the above methods, but emphasis clustering techniques for improved efficiency. When using a clustered index, “the rows of the table are stored in clustered index order with respect to the spatial index, thereby minimizing the number of page reads for range searches” [23].

Chapter 3 - Project

The task still remaining is to represent the human face as a collection of geometric shapes using one of the SDMS discussed earlier. Shapes, that would not only capture all information needed for the proposed fuzzy queries used by Virginia Commonwealth's database research group, but other potential queries. The SDMS chosen to test the feasibility of representing facial objects was MySQL 5.0, because it is open source, easy to use, and supports spatial objects. It is important to note that this research is not concerned with how the points are to be automatically selected from an image, but purely with what points should be selected. This eliminates the need to delve into the classic 'black box' that deals with automated point capture from images.

The method for selecting the points consisted of first analyzing the queries being asked of the database. These included queries about the nose such as "Give me all suspects with a long nose". It also included queries that referenced the shape and separation of the chin, eyes, head and forehead. Once familiar with these features, the task was to represent them as individual or collections of geometric shapes. As mentioned earlier, the shapes available in MySQL ranged from lines and polygons to circles or ellipses. However, with MySQL 5.0 spatial objects were limited to a smaller set of 2D geometry.

Therefore, each eye was represented as a collection of shapes consisting of two lines and a polygon. The lines pass through the horizontal and vertical midline of the eye. This will give both the height and horizontal length of the eye. See Figure 3.1

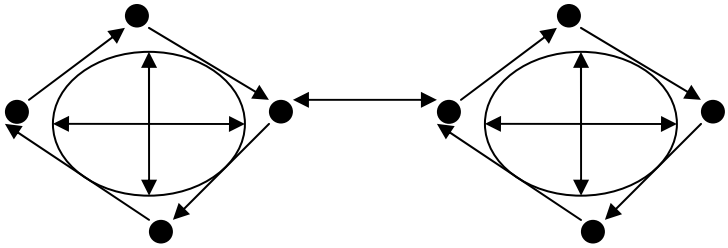

Figure 3.1 Eyes

The polygon stored is made up of the four included points used from the above midlines. It is possible to eliminate the two separately stored lines and used built-in functions to extract the internal line information from the polygon. However, for initial feasibility testing it was stored twice. The points on both sides of each eye where chosen to measure orbital distance or the width of the eyes. The vertical line provides the height of the eyes, while the polygon yields the area. The SQL syntax for the left eye would be as follows:

GeometryCollection (Line (Point, Point), Line(Point, Point), Polygon(Line, Line, Line, Line)).

As will be shown further in this chapter, knowing the width, height, and area of the eyes is all that is needed for comparison. See figure 3.2 for an example.

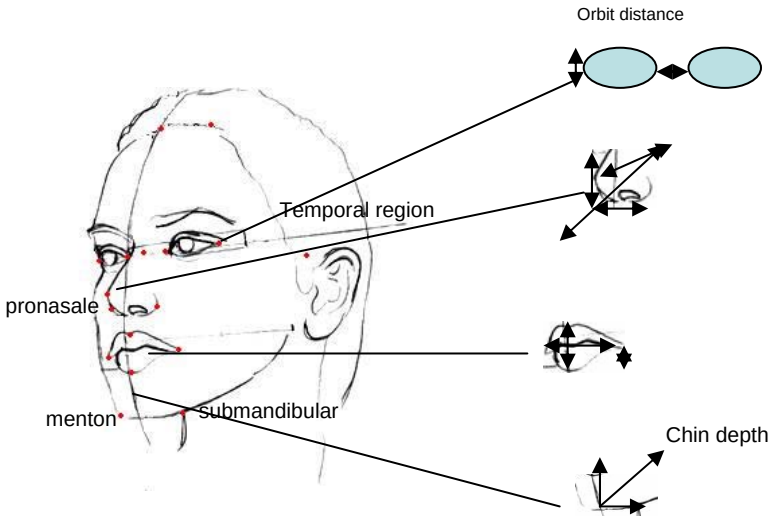

Figure 3.2 – Image taken from [16]

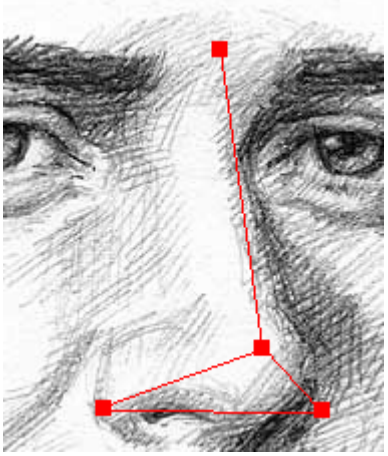

Figure 3.3 Nose - Image from [17]

The nose was also represented with two lines and a polygon. In this case, the polygon contained three points and formed a triangle which was referenced against a point located at the midpoint of spherical human skull. This provided a measurement of depth as well as the other dimensions of the nose. In addition, one line running horizontal through the nasal cavity and another line running from the midpoint between the eyes to the tip of the nose was stored. These lines provided valuable information when deriving nose length.

The mouth consisted of two lines and a square, while the chin was represented with one point and a triangle (See Figure 3.4).

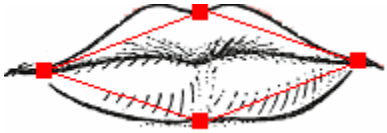

Figure 3.4 Image from [18]

Depth information could be derived if referenced against the center point of the skull. Overall, there was a total of 22 points chosen from the face that would effectively capture the information needed. The figure 3.5 shows each point chosen and the geometric shapes formed.

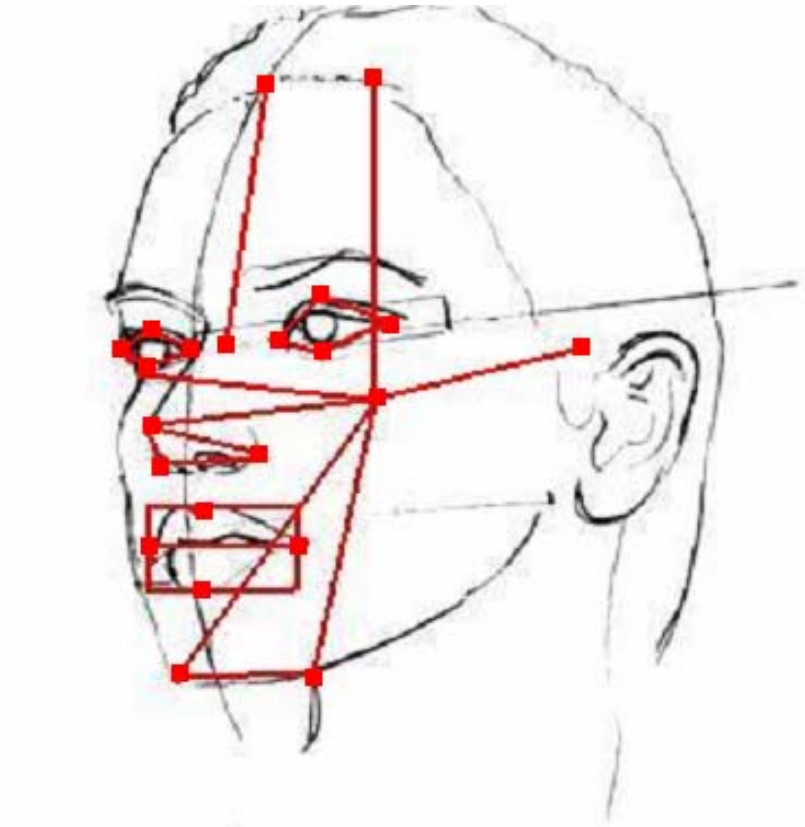

Figure 3.5 Image from [16]

The RELVARS for the objects made up by the chosen 22 points are shown in Table 3.1.

Schema: Face	
Table: Eyes	
Field	Type
leftEye	GeometryCollection(Line, Line, Polygon)
rightEye	GeometryCollection(Line, Line, Polygon)
midPoint	Point(x,y)
SSN*	Varchar (9)
Table: Nose	
Tip	Polygon(Line, Line, Line)
Alvar	Linestring(Point(x,y), Point(x,y))
Peak**	Linestring(Point(x,y), Point(x,y))
SSN*	Varchar (9)
Table: Mouth	
vermillionHorizontal	Linestring(Point(x,y), Point(x,y))
vermillionVertical	Linestring(Point(x,y), Point(x,y))
Lips	Polygon(Line, Line, Line, Line)
SSN*	Varchar (9)
Table: Chin	
Mandiblepeak	Point(x,y)
Peak	Polygon(Line, Line, Line)
SSN*	Varchar (9)
Table: Forehead	
Frontal	Linestring(Point(x,y), Point(x,y))
Vertical	Linestring(Point(x,y), Point(x,y))
Horizontal	Linestring(Point(x,y), Point(x,y))
Widowspeak	Point(x,y)
SSN*	Varchar (9)
* primary key, **spatial index – a spatially derived feature such as nose length is chosen as an index.	

Table 3.1

In the search for previous research in this area, only one paper appeared to be directly related. *GIS as a tool for facial recognition* [14], proposed using geographic imaging systems for purposes of facial recognition. The authors identified 10 points on the face that could be captured and used with a computer aided design system to show the 'unique map' the connected points formed (see Figure 3.5). This face map would ultimately be used similar to how fingerprints are used to identify a person and have future relevance in facial recognition. A visual basic program analyzed x and y coordinates, angle, distance data of the 'map' and returned those faces with the best match.

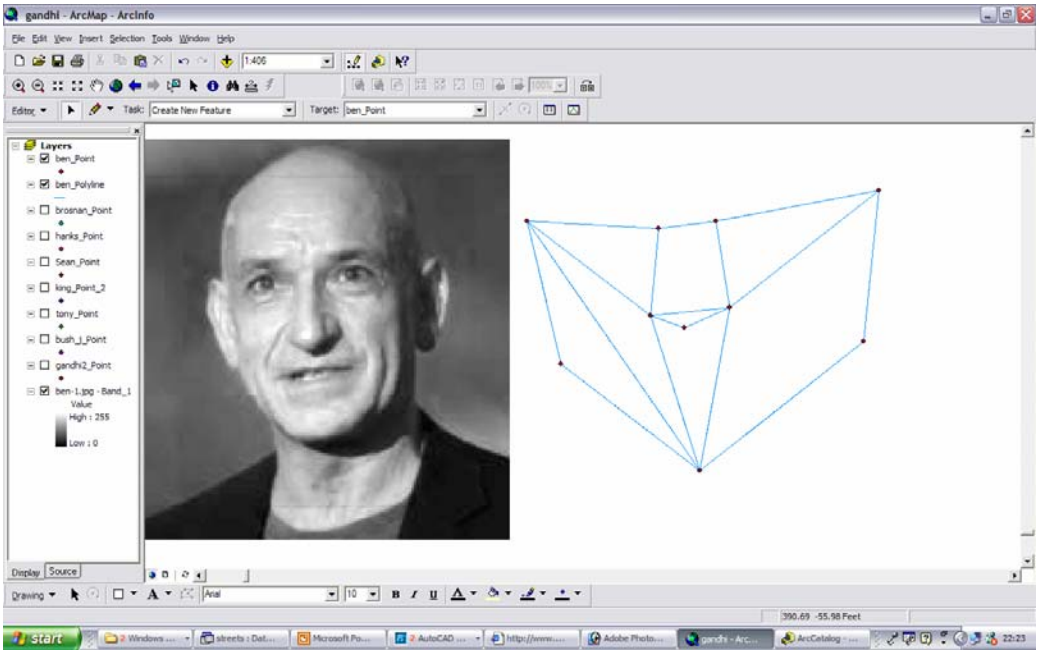

Figure 3.5 Image from [14]

Although closely related, this research takes that idea further by dividing the human face into collections of geometric shapes and implementing them with the special capabilities of a SDMS for the purposes of facial characterization, not solely for identification.

The concern now was the number of points chosen to represent the face. Was 22 points more than needed? Was there a way to reduce any redundancy in the points chosen? The strategy for answering this question can be found in studying the face itself. The human face, much like other pieces of nature, exhibits the property of symmetry. Symmetry can be defined as “exact correspondence of form and constituent configuration on opposite sides of a dividing line or plane or about a center or an axis” [15]. If one were to draw a line down the center of the face, would symmetry allow the points located on the other side to be derived and thus eliminate the need to store them in the database?

Oddly enough, the search for point elimination began with studies linking symmetry to human perception of beauty; the more symmetrical the face, the more beautiful [4]. If the symmetry present in the human face is consistent, then mirrored data points could be eliminated. In *Anatomy of A Beautiful Face & Smile*, the authors describe proportions of the human face in great detail, for example, noting that the “width of an eye should be three-tenths that of the face as measured at eye level” [4]. However, due to the variation in nature, the symmetry present in the face is inconsistent. This limits the amount of deduction that can be used to guess the mirrored point’s position. It is important to note, that even though data redundancy is not improved, symmetry could be used to check for data point outliers. Meaning, even if the mirrored point is not exactly where it should be, a user can use an approximated position to gauge whether the data point position is beyond a chosen threshold and incorrect. Thus taking advantage of the symmetry property differently and utilizing it with error correction.

4 – MySQL Implementation

The process of setting up MySQL started with the generation of test data for 10 individuals. The schema ‘face’ consist of 5 tables; one table for nose, eyes, mouth, chin, and forehead. The MySQL script used for schema creation is provided in Appendix A. In addition, spatial and non-spatial indexes were added for each table. The non-spatial index was the primary key of SSN, while the spatial index varied per facial object. The nose, for example, is indexed on the line used to derive nose length. Several queries were run on the example table (see Table 3.4) to test feasibility.

AsText(g)	ssn	AsText(alvar)	AsText(peak)
POINT(0 0)	333333333	LINESTRING(7 2,3 2)	LINESTRING(7 1,3 2)
POINT(0 0)	444444444	LINESTRING(7.2 2.2,3 2.1)	LINESTRING(6.4 1.4,3 2)
POINT(0 0)	555555555	LINESTRING(7.7 2.3,3.2 2.4)	LINESTRING(6.6 1.3,3 2)
POINT(0 0)	666666666	LINESTRING(7.8 2.5,3.3 2.4)	LINESTRING(6.2 1.1,3 2)
POINT(0 0)	777777777	LINESTRING(7.2 2.9,3.6 2.7)	LINESTRING(6.2 1.3,3 2)
POINT(0 0)	888888888	LINESTRING(7.4 2.3,3.7 2.7)	LINESTRING(6.6 1.9,3 2)
POINT(0 0)	999999999	LINESTRING(7.5 2.5,3.3 2.9)	LINESTRING(6.1 1.9,3 2)
POINT(0 0)	222222222	LINESTRING(7.1 2.2,3.8 2.9)	LINESTRING(6.9 1.7,3 2)
POINT(0 0)	111111111	LINESTRING(7.1 2.2,3.2 2.2)	LINESTRING(6.4 1.6,3 2)
POINT(0 0)	000000000	LINESTRING(7.1 2.3,3 2.2)	LINESTRING(6.1 1.1,3 2)

Table 3.4

The first example,

SELECT ssn from nose where GLength(peak)>3.0;

utilizes the *GLength()* method that returns the length of a line. This query is important because it ultimately returns the SSN’s of individuals with noses of length greater than 3. It also fits nicely into the fuzzy query scheme because a community perceived decimal weight ranging from 0 to 1 can be easily mapped to values of spatial features such as nose length. For example, if the

community repeatedly determines that a displayed picture of a person has a long nose, then the weight in the ‘strong’ category will increase. This weight can be mapped two ways, either by first determining nose length and picking a weight, or by mapping a given weight to a specific length range.

The next example shows the SQL syntax for the query “Give me the suspect with the biggest mouth”. Notice that the term ‘big’ does not indicate a width or height. For this example, the area of the mouth would best describe the size of the mouth. However, to verify the answer returned is correct, the values in the lips polygon must be known. In order to see a textual representation of a shape, the method AsText() is used.

SQL: Select AsText(lips) from mouth; *Select Area(lips) from mouth;*

Result:

POLYGON((7 2.1,3 2.1,3 2,7 2.1))	0.2
POLYGON((7.2 2.2,3 2.1,3 2,7.2 2.2))	0.21
POLYGON((7.7 2.3,3.2 2.4,3 2,7.7 2.3))	0.91
POLYGON((7.8 2.5,3.3 2.4,3 2,7.8 2.5))	0.885
POLYGON((7.2 2.9,3.6 2.7,3 2,7.2 2.9))	1.2
POLYGON((7.4 2.3,3.7 2.7,3 2,7.4 2.3))	1.435
POLYGON((7.5 2.5,3.3 2.9,3 2,7.5 2.5))	1.95
POLYGON((7.1 2.2,3.8 2.9,3 2,7.1 2.2))	1.765
POLYGON((7.1 2.2,3.2 2.2,3 2,7.1 2.2))	0.39
POLYGON((7.1 2.3,3 2.2,3 2,7.1 2.3))	0.41

The query: *Select Max(Area(lips)) from mouth;* will then return *1.95* and is correct.

Chapter 5 – Conclusion

Representing the human face as a collection of geometric shapes using spatial databases is feasible. SDMS possess the ability to determine spatial relationships efficiently and quickly and would be well suited for facial characterization using fuzzy queries. This research will build nicely into Virginia Commonwealth University's research into how community learned spatial relationship values are used to search within SDMS.

Although the targeted use of this research is with government or law enforcement agencies, government agencies are not the only institutions that can benefit from the ability to search on facial characteristics. Facial analysis is also done by plastic surgeons to gauge how successful their operations progressed [3]. A patient's facial data could be recorded before an operation and then used for analysis afterward. Spatial queries would return values used to compare distances or degrees of symmetry of facial objects. There might be also potential for use in other research studies involving beauty or behavior. Studies involved in analyzing how facial symmetry affects human perception of beauty or how nasal area or mouth width affects human behavior.

Chapter 6 – Future Work

However, this research still has many significant issues yet to be addressed. One such issue or 'black box' keeping this research from real-world implementation is how the automatic capture of the chosen points will be accomplished. This is a classic problem that has been one of the greatest challenges in image processing used in facial recognition. How do you program a computer to identify human eyes from an image with high accuracy and precision? And how do you select the exact points needed for analysis? Is there existing research such as *GIS As a Tool*

For Facial Recognition, which has point extraction capabilities? These questions need to be solved before large databases containing facial characteristics can be compiled and the true power of facial queries known.

Spatial database management systems also have room for improvement. Many packages, for example, only support a limited number of functions for each geometric shape. Additional mathematical methods would increase the analytical power used for facial analysis. MySQL 5.0 could be improved to allow representation of shapes greater than 2 dimensions, as Oracle does. A potential area of future research would be to compare facial objects represented in 2D verses 3D and to determine which is better suited for facial analysis. In particular, it needs to be determined if SDMS offer improved performance over non-SDMS in this application. Database packages should also be compared against each other to determine the most efficient and capable system.

References

[1] Crouse, B., and Pandrinath, A., (2004). *Spatial Databases: A Directed Research Project*. Virginia Commonwealth University.

[2] Shekhar, S., and Chawla, S., (2003). *Spatial Databases: A Tour*, Prentice Hall.

[3] Scognamillo, R., Rhodes, G., Morrone, C., (2003). A feature-based model of symmetry detection. The Royal Society. Volume 270, 1727-1733.

[4] Patnik, V., Rajan, S., Sanju, B. (2003). *Anatomy of A Beautiful Face & Smile*. J. Anat. Soc. India 51(1) 3-5.

[5] Web. The Rhinoplasty Center, Anatomy of the Nose. http://www.therhinoplastycenter.com/index.html. (2005.09.14).

[6] Web. Oracle 10g Documentation. *Spatial Databases*. http://www.oracle.com/technology/products/spatial/pdf/qt.pdf (2005.09.14)

[7] Zhang, et al., (2004). *Comparison Between Geometry-Based and Gabor-Wavelets-Based Facial Expression Recognition Using Multi-Layer Perception*. Third IEEE International Conference on Automatic Face and Gesture Recognition, April 14-16 1998, Nara Japan, IEEE Computer Society, pp. 454-459.

[8] Web. *Learn to draw, Introduction to drawing people*. http://www.learn-to-draw.com/drawing-people/08-eyeline.htm (4/5/2005)

[9] Web. *3D Faces and Face proportions*. http://www.sierravista.wuhsd.k12.ca.us/basicart/faces.htm

[10] OpenGIS Consortium and Feature Specification. www.opengis.org (2004.11.08).

[11] Web. *Oracle Spatial Users Guide and Reference*. http://www.oracle.com/technology/products/spatial/index.html

[12] Web. *Wikipedia, the free encyclopedia*. http://en.wikipedia.org/wiki/Fourth_dimension

[13] Web. *Wikipedia, the free encyclopedia*. http://en.wikipedia.org/wiki/Tesseract

[14] Lachhwani, V. and Kim, Eok. *GIS as a Tool for Facial Recognition*. www.uwm.edu/Dept/GIS/flyers/vikascompetition04.doc

[15] Web. Dictionary. www.dictionary.com

[16] Web. *3 Faces and Face Proportion* http://www.sierravista.wuhsd.k12.ca.us/basicart/faces.htm

[17] Web. *Drawing the Portrait*. http://www.portrait-artist.org/face/

[18] Web. Human Lips. http://www.lipaugmentation.com/lipauggraphics/myologyfibers_Orbicularisoris.gif

[19] Date, C.J. (2004). *Spatial Databases*. An Introduction to Database Systems. 8th ed. Page x-y.

[20] Web. MySQL 5.0 manual (2005) http://www.browardphp.com/mysql_manual_en/manual_Introduction.html

[21] Lee, Kuo, Hsu et al. (2004). *3-D Face recognition system based on feature analysis and support vector machine*, IEEE.

[22] Balasuriya, L. and Kodikara, N., (2001). *Frontal View Human Face Detection and Recognition*.

[23] Web. Spatial Query Server 4.3 Manual. http://sismissionsystems.boeing.com/support/downloads/sqs34212SunSyb12_0.tar

Appendix A - SQL Scripts

Nose table

```
CREATE TABLE face.nose (ssn VARCHAR(9)) TYPE = MYISAM;
ALTER TABLE nose ADD tip POLYGON;
ALTER TABLE nose ADD alvar LINESTRING;
ALTER TABLE nose ADD peak LINESTRING;
ALTER TABLE `face`.`nose` MODIFY COLUMN `ssn` VARCHAR(9) NOT NULL,
ADD PRIMARY KEY(`ssn`);
INSERT INTO face.nose VALUES ('333333333', GeomFromText('POLYGON((7.0 2.1,3.0 2.1,7.0 2.1))'), GeomFromText('LINESTRING(7.0 2.1,3.0 2.1)'), GeomFromText('LINESTRING(6.1 1.0,3 2)'));
INSERT INTO face.nose VALUES ('444444444', GeomFromText('POLYGON((7.2 2.2,3.0 2.1,7.2 2.2))'), GeomFromText('LINESTRING(7.2 2.2,3.0 2.1)'), GeomFromText('LINESTRING(6.4 1.4,3 2)'));
INSERT INTO face.nose VALUES ('555555555', GeomFromText('POLYGON((7.7 2.3,3.2 2.4,7.7 2.3))'), GeomFromText('LINESTRING(7.7 2.3,3.2 2.4)'), GeomFromText('LINESTRING(6.6 1.3,3 2)'));
INSERT INTO face.nose VALUES ('666666666', GeomFromText('POLYGON((7.8 2.5,3.3 2.4,7.8 2.5))'), GeomFromText('LINESTRING(7.8 2.5,3.3 2.4)'), GeomFromText('LINESTRING(6.2 1.1,3 2)'));
INSERT INTO face.nose VALUES ('777777777', GeomFromText('POLYGON((7.2 2.9,3.6 2.7,7.2 2.9))'), GeomFromText('LINESTRING(7.2 2.9,3.6 2.7)'), GeomFromText('LINESTRING(6.2 1.3,3 2)'));
INSERT INTO face.nose VALUES ('888888888', GeomFromText('POLYGON((7.4 2.3,3.7 2.7,7.4 2.3))'), GeomFromText('LINESTRING(7.4 2.3,3.7 2.7)'), GeomFromText('LINESTRING(6.6 1.9,3 2)'));
INSERT INTO face.nose VALUES ('999999999', GeomFromText('POLYGON((7.5 2.5,3.3 2.9,7.5 2.5))'), GeomFromText('LINESTRING(7.5 2.5,3.3 2.9)'), GeomFromText('LINESTRING(6.1 1.9,3 2)'));
INSERT INTO face.nose VALUES ('222222222', GeomFromText('POLYGON((7.1 2.2,3.8 2.9,7.1 2.2))'), GeomFromText('LINESTRING(7.1 2.2,3.8 2.9)'), GeomFromText('LINESTRING(6.9 1.7,3 2)'));
INSERT INTO face.nose VALUES ('111111111', GeomFromText('POLYGON((7.1 2.2,3.2 2.2,7.1 2.2))'), GeomFromText('LINESTRING(7.1 2.2,3.2 2.2)'), GeomFromText('LINESTRING(6.4 1.6,3 2)'));
INSERT INTO face.nose VALUES ('000000000', GeomFromText('POLYGON((7.1 2.3,3.0 2.2,7.1 2.3))'), GeomFromText('LINESTRING(7.1 2.3,3.0 2.2)'), GeomFromText('LINESTRING(6.1 1.1,3 2)'));
ALTER TABLE `face`.`nose` MODIFY COLUMN `peak` LINESTRING NOT NULL,
ADD SPATIAL `Index_2`(`peak`);
```

Eyes table

```
CREATE TABLE face.eyes (ssn VARCHAR(9)) TYPE = MYISAM;
ALTER TABLE eyes ADD lefteye GEOMETRYCOLLECTION;
ALTER TABLE eyes ADD righteye GEOMETRYCOLLECTION;
ALTER TABLE eyes ADD midpoint POINT;
ALTER TABLE `face`.`eyes` MODIFY COLUMN `ssn` VARCHAR(9) NOT NULL,
ADD PRIMARY KEY(`ssn`);
INSERT INTO face.eyes VALUES ('000000000', GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'),
GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'), GeomFromText('POINT(1 1)'));
INSERT INTO face.eyes VALUES ('111111111', GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'),
GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'), GeomFromText('POINT(1 1)'));
INSERT INTO face.eyes VALUES ('222222222', GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'),
GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'), GeomFromText('POINT(1 1)'));
INSERT INTO face.eyes VALUES ('333333333', GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'),
GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'), GeomFromText('POINT(1 1)'));
INSERT INTO face.eyes VALUES ('444444444', GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'),
GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'), GeomFromText('POINT(1 1)'));
INSERT INTO face.eyes VALUES ('555555555', GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'),
GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'), GeomFromText('POINT(1 1)'));
INSERT INTO face.eyes VALUES ('666666666', GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'),
GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'), GeomFromText('POINT(1 1)'));
INSERT INTO face.eyes VALUES ('777777777', GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'),
GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'), GeomFromText('POINT(1 1)'));
INSERT INTO face.eyes VALUES ('888888888', GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'),
GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'), GeomFromText('POINT(1 1)'));
INSERT INTO face.eyes VALUES ('999999999', GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'),
GeomFromText('GEOMETRYCOLLECTION(LINESTRING(7.0 2.1,3.0 2.1),LINESTRING(7.0 2.1,3.0 2.1),POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5)))'), GeomFromText('POINT(1 1)'));
```

Appendix A - SQL Scripts

Mouth table

```
CREATE TABLE face.mouth (ssn VARCHAR(9)) TYPE = MYISAM;
ALTER TABLE mouth ADD lips POLYGON;
ALTER TABLE mouth ADD vermillionHorizontal LINESTRING;
ALTER TABLE mouth ADD vermillionVertical LINESTRING;
ALTER TABLE `face`.`mouth` MODIFY COLUMN `ssn` VARCHAR(9) NOT NULL,
ADD PRIMARY KEY(`ssn`);
INSERT INTO face.mouth VALUES ('333333333', GeomFromText('POLYGON((7.0 2.1,3.0 2.1,3 2, 4 5))'), GeomFromText('LINESTRING(7.0 2.1,3.0 2.1)'), GeomFromText('LINESTRING(6.1 1.0,3 2)'));
INSERT INTO face.mouth VALUES ('444444444', GeomFromText('POLYGON((7.2 2.2,3.0 2.1,3 2, 4 5))'), GeomFromText('LINESTRING(7.2 2.2,3.0 2.1)'), GeomFromText('LINESTRING(6.4 1.4,3 2)'));
INSERT INTO face.mouth VALUES ('555555555', GeomFromText('POLYGON((7.7 2.3,3.2 2.4,3 2, 4 5))'), GeomFromText('LINESTRING(7.7 2.3,3.2 2.4)'), GeomFromText('LINESTRING(6.6 1.3,3 2)'));
INSERT INTO face.mouth VALUES ('666666666', GeomFromText('POLYGON((7.8 2.5,3.3 2.4,3 2, 4 5))'), GeomFromText('LINESTRING(7.8 2.5,3.3 2.4)'), GeomFromText('LINESTRING(6.2 1.1,3 2)'));
INSERT INTO face.mouth VALUES ('777777777', GeomFromText('POLYGON((7.2 2.9,3.6 2.7,3 2, 4 5))'), GeomFromText('LINESTRING(7.2 2.9,3.6 2.7)'), GeomFromText('LINESTRING(6.2 1.3,3 2)'));
INSERT INTO face.mouth VALUES ('888888888', GeomFromText('POLYGON((7.4 2.3,3.7 2.7,3 2, 4 5))'), GeomFromText('LINESTRING(7.4 2.3,3.7 2.7)'), GeomFromText('LINESTRING(6.6 1.9,3 2)'));
INSERT INTO face.mouth VALUES ('999999999', GeomFromText('POLYGON((7.5 2.5,3.3 2.9,3 2, 4 5))'), GeomFromText('LINESTRING(7.5 2.5,3.3 2.9)'), GeomFromText('LINESTRING(6.1 1.9,3 2)'));
INSERT INTO face.mouth VALUES ('222222222', GeomFromText('POLYGON((7.1 2.2,3.8 2.9,3 2, 4 5))'), GeomFromText('LINESTRING(7.1 2.2,3.8 2.9)'), GeomFromText('LINESTRING(6.9 1.7,3 2)'));
INSERT INTO face.mouth VALUES ('111111111', GeomFromText('POLYGON((7.1 2.2,3.2 2.2,3 2, 4 5))'), GeomFromText('LINESTRING(7.1 2.2,3.2 2.2)'), GeomFromText('LINESTRING(6.4 1.6,3 2)'));
INSERT INTO face.mouth VALUES ('000000000', GeomFromText('POLYGON((7.1 2.3,3.0 2.2,3 2, 4 5))'), GeomFromText('LINESTRING(7.1 2.3,3.0 2.2)'), GeomFromText('LINESTRING(6.1 1.1,3 2)'));
ALTER TABLE `face`.`mouth` MODIFY COLUMN `vermillionHorizontal` LINESTRING NOT NULL,
ADD SPATIAL `Index_3`(`vermillionHorizontal`);
```

Chin table

```
CREATE TABLE face.chin (ssn VARCHAR(9)) TYPE = MYISAM;
ALTER TABLE chin ADD peak2 POLYGON;
ALTER TABLE chin ADD Mandiblepeak2 POINT;
ALTER TABLE `face`.`chin` MODIFY COLUMN `ssn` VARCHAR(9) NOT NULL,
ADD PRIMARY KEY(`ssn`);
INSERT INTO face.chin VALUES ('333333333', GeomFromText('POLYGON((7.0 2.1,3.0 2.1,7.0 2.1))'), GeomFromText('POINT(1 1)'));
INSERT INTO face.chin VALUES ('444444444', GeomFromText('POLYGON((7.2 2.2,3.0 2.1,7.2 2.2))'), GeomFromText('POINT(1 2)'));
INSERT INTO face.chin VALUES ('555555555', GeomFromText('POLYGON((7.7 2.3,3.2 2.4,7.7 2.3))'), GeomFromText('POINT(1 3)'));
INSERT INTO face.chin VALUES ('666666666', GeomFromText('POLYGON((7.8 2.5,3.3 2.4,7.8 2.5))'), GeomFromText('POINT(1 4)'));
INSERT INTO face.chin VALUES ('777777777', GeomFromText('POLYGON((7.2 2.9,3.6 2.7,7.2 2.9))'), GeomFromText('POINT(1 5)'));
INSERT INTO face.chin VALUES ('888888888', GeomFromText('POLYGON((7.4 2.3,3.7 2.7,7.4 2.3))'), GeomFromText('POINT(1 6)'));
INSERT INTO face.chin VALUES ('999999999', GeomFromText('POLYGON((7.5 2.5,3.3 2.9,7.5 2.5))'), GeomFromText('POINT(1 7)'));
INSERT INTO face.chin VALUES ('222222222', GeomFromText('POLYGON((7.1 2.2,3.8 2.9,7.1 2.2))'), GeomFromText('POINT(1 8)'));
INSERT INTO face.chin VALUES ('111111111', GeomFromText('POLYGON((7.1 2.2,3.2 2.2,7.1 2.2))'), GeomFromText('POINT(1 9)'));
INSERT INTO face.chin VALUES ('000000000', GeomFromText('POLYGON((7.1 2.3,3.0 2.2,7.1 2.3))'), GeomFromText('POINT(1 0)'));
ALTER TABLE `face`.`chin` MODIFY COLUMN `peak2` POLYGON NOT NULL,
ADD SPATIAL `Index_4`(`peak2`);
```

Appendix A - SQL Scripts

Forehead table

```
CREATE TABLE face.forehead (ssn VARCHAR(9)) TYPE = MYISAM;
ALTER TABLE forehead ADD widowspeak POINT;
ALTER TABLE forehead ADD frontal LINESTRING;
ALTER TABLE forehead ADD vertical LINESTRING;
ALTER TABLE forehead ADD horizontal LINESTRING;
ALTER TABLE `face`.`forehead` MODIFY COLUMN `ssn` VARCHAR(9) NOT NULL,
ADD PRIMARY KEY(`ssn`);
INSERT INTO face.forehead VALUES ('333333333', GeomFromText('POINT(1 1)'), GeomFromText('LINESTRING(7.0 2.1,3.0 2.1)'), GeomFromText('LINESTRING(6.1 1.0,3 2)'), GeomFromText('LINESTRING(6.1
1.0,3 2)'));
INSERT INTO face.forehead VALUES ('444444444', GeomFromText('POINT(1 1)'), GeomFromText('LINESTRING(7.2 2.2,3.0 2.1)'), GeomFromText('LINESTRING(6.4 1.4,3 2)'), GeomFromText('LINESTRING(6.1
1.0,3 2)'));
INSERT INTO face.forehead VALUES ('555555555', GeomFromText('POINT(1 1)'), GeomFromText('LINESTRING(7.7 2.3,3.2 2.4)'), GeomFromText('LINESTRING(6.6 1.3,3 2)'), GeomFromText('LINESTRING(6.1
1.0,3 2)'));
INSERT INTO face.forehead VALUES ('666666666', GeomFromText('POINT(1 1)'), GeomFromText('LINESTRING(7.8 2.5,3.3 2.4)'), GeomFromText('LINESTRING(6.2 1.1,3 2)'), GeomFromText('LINESTRING(6.1
1.0,3 2)'));
INSERT INTO face.forehead VALUES ('777777777', GeomFromText('POINT(1 1)'), GeomFromText('LINESTRING(7.2 2.9,3.6 2.7)'), GeomFromText('LINESTRING(6.2 1.3,3 2)'), GeomFromText('LINESTRING(6.1
1.0,3 2)'));
INSERT INTO face.forehead VALUES ('888888888', GeomFromText('POINT(1 1)'), GeomFromText('LINESTRING(7.4 2.3,3.7 2.7)'), GeomFromText('LINESTRING(6.6 1.9,3 2)'), GeomFromText('LINESTRING(6.1
1.0,3 2)'));
INSERT INTO face.forehead VALUES ('999999999', GeomFromText('POINT(1 1)'), GeomFromText('LINESTRING(7.5 2.5,3.3 2.9)'), GeomFromText('LINESTRING(6.1 1.9,3 2)'), GeomFromText('LINESTRING(6.1
1.0,3 2)'));
INSERT INTO face.forehead VALUES ('222222222', GeomFromText('POINT(1 1)'), GeomFromText('LINESTRING(7.1 2.2,3.8 2.9)'), GeomFromText('LINESTRING(6.9 1.7,3 2)'), GeomFromText('LINESTRING(6.1
1.0,3 2)'));
INSERT INTO face.forehead VALUES ('111111111', GeomFromText('POINT(1 1)'), GeomFromText('LINESTRING(7.1 2.2,3.2 2.2)'), GeomFromText('LINESTRING(6.4 1.6,3 2)'), GeomFromText('LINESTRING(6.1
1.0,3 2)'));
INSERT INTO face.forehead VALUES ('000000000', GeomFromText('POINT(1 1)'), GeomFromText('LINESTRING(7.1 2.3,3.0 2.2)'), GeomFromText('LINESTRING(6.1 1.1,3 2)'), GeomFromText('LINESTRING(6.1
1.0,3 2)'));
ALTER TABLE `face`.`forehead` MODIFY COLUMN `horizontal` LINESTRING NOT NULL,
ADD SPATIAL `Index_4`(`horizontal`);
```