Query Transformation Method by Delaunay Triangulation for Multi-Source Distributed Spatial Database Systems

Jung-Rae Hwang, Ji-Hyeon Oh, and Ki-Joune Li

Spatiotemporal Database Lab.
Pusan National University, South Korea
{jrhwang,jhoh}@quantos.cs.pusan.ac.kr, lik@hyowon.cc.pusan.ac.kr

ABSTRACT

One of the difficulties in building a distributed GIS comes from the heterogeneity of spatial databases. In particular, positional mismatches between spatial databases arise due to several reasons, such as different scales, or different ground control points. They result in unreliable outputs of query processing. One simple solution is to correct positional data in spatial databases at each site, according to the most accurate one. This solution is however not practical in most of the cases where the autonomy of each database should be respected. In this paper, we propose a spatial query processing method without correcting positional data in each spatial database. Instead of correcting positional data, we dynamically transform a given query region or position onto each space where spatial objects of each site are located. Our method is based on an elastic transformation method by delaunay triangulation.

1 INTRODUCTION

In recent years, the need has arisen for accessing data from distributed and preexisting spatial database systems or GIS. In order to realize a distributed spatial database system, we should overcome several problems, one of which comes from the heterogeneity of spatial databases. The heterogeneity includes a number of issues from spatial database management systems to data models. In this paper, we focus on the inconsistency of positional data between local spatial databases.

The positional mismatches of spatial databases come from several reasons. Different organizations with their own acquisition and management policy of data may have inconsistent positional data. Different scales or accuracy are important reasons of the mismatches as well. Inconsistent positional data often comes from different surveying methods and different ground control points. And mismatches are often found when we merge two adjacent maps.

Suppose that we have two different spatial databases, one for buildings and another for roads and traffic conditions. When the databases have been independently constructed, they may include mismatching positional data. If a user gives a query, "*find the nearest object to point p*" on these two databases at the same time, the query point *p* may be differently considered by each database and the system will give incorrect answers to the query as shown by figure 1.

Since the user is normally ignorant in positional mismatches between two databases, he cannot specify different query positions for each database respectively, but the system is in charge of giving correct answer to the query regardless of the mismatches.

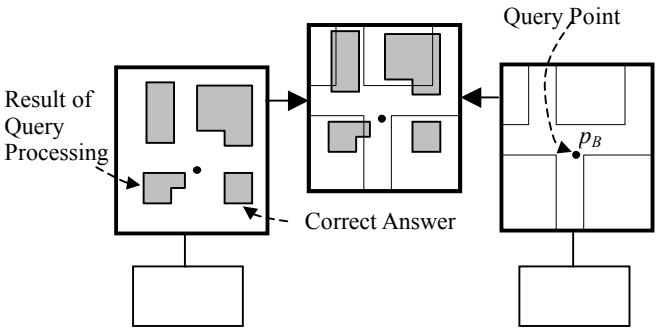

Figure 1. Positional Mismatch and Example of Query

A number of researches and efforts have been done to resolve the heterogeneity of functions, data models and schemas in spatial database systems and GIS. The standardizations by ISO TC/211[7,8] and OGC (OpenGIS Consortium)[9] are important efforts to overcome the heterogeneity. But the heterogeneity of contents in spatial databases has been paid attention by very few works.

In this paper, we propose a query processing method for distributed spatial database systems, where they have positional mismatches. In section 2, we will give a survey on related work and introduce the motivation of this paper. And in the next section, we will explain an elastic transformation method by delaunay triangulation, which is a fundamental concept of our method. We will propose our query processing method in section 4, and conclude this paper in section 5.

2 Related Work and Motivation

As other types of databases, the heterogeneity problems arise in

spatial database systems, when we integrate preexisting databases. A number of researches and efforts have been made to resolve the heterogeneity of functions and data models between spatial database systems. In this paper, we focus on the heterogeneity of positional data between local spatial databases. The heterogeneity of positional data has been known as mismatches between maps, for example mismatches between adjacent maps.

A limited number of works have been done to resolve positional mismatches between local spatial databases. Most of these researches rely on elastic transformation of maps. Servigne and Laurini [2] have given an overview of mismatching problem with some solutions. An elastic transformation with a weighted linear function has been proposed by Langlois [10]. To improve the quality of transformation, Dufour [11] has proposed to weight Langlois function by a consine function. These methods assume that we are provided with a set of corresponding homologous points from two maps. They are simple and efficient but do not preserve geometric properties such as rectangularity or parallelism.

Cho, et al. [1] have proposed a method of elastic transformation by different approach based on polygon morphing. Instead of using homologous pairs of control points, the transformation is performed by analyzing the similarity between two homologous polygons on mismatching maps. They improved this method by introducing some measure functions to evaluate the validity of transformation [3].

The methods mentioned above assume that positional data in each local database should be corrected according to a database considered as the most accurate one. In many cases, we must however respect the autonomy of databases at each site, and it is not acceptable to modify positional data of each local database. For example, boundary lines data of cadastral databases in South Korea cannot be corrected without legal procedure, even if inaccurate data would be found.

One possible solution is to dynamically transform query region or point onto each space, respectively where the spatial objects at each site are located, rather than correcting the entire positional data. Suppose that a query is given with a region as illustrated by figure 2. The query region is specified on map *B*. Then we submit the same query to database *A* with transformed query region, as shown by figure 2.

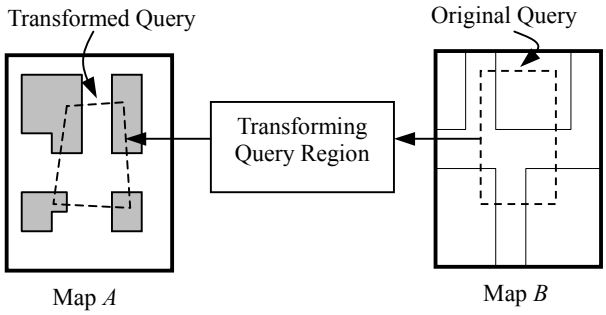

Figure 2. Transformation of Query Region

By transforming query regions or positions, we do not need to modify the positional data of each database, and therefore respect the autonomy of each site. In the subsequent sections, we will explain how to transform query regions or positions.

3 Elastic Transformation by Delaunay Triangulation

Our method is based on an elastic transformation by delaunay triangulation [6], which we will explain in this section. The elastic transformation refers to a set of operations including translation, shearing, and resizing of objects in a space without changing their topological properties. The goal of elastic transformation is to correct positional data of spatial databases with respect to the most accurate one, which we will call *reference map* in the rest of the paper.

Suppose that we have two maps, a reference map and a corresponding map. In order to perform an elastic transformation by delaunay triangulation, we need a set of control points on the reference map, and corresponding ones on the other map. Each pair of corresponding control points from the two maps indicates identical positions on each map. Then the transformation procedure consists of two steps.

At the first step, we perform delaunay triangulation with a set of control points on the reference map, and make a set of corresponding triangles with corresponding control points on the other map. And for the next step, we derive the coordinates on the reference map from every point or vertex of spatial object on the other map, using the similarity coefficients. The similarity coefficients are calculated by comparing two corresponding enclosing triangles, which are one from the reference map and another from the other map, respectively. The detail algorithm is given by algorithm 1.

Algorithm 1 : Elastic Transformation
Input V_S : the set of vertices on the reference map.
V_T : the set of vertices on the rest maps.
$G = \{(g_s, g_t) \mid g_s$ and g_t are the corresponding control points on reference map and other map, respectively$\}$
Output V_T : the set of corrected coordinates of V_T
Begin
$T_S \leftarrow$ DelaunayTriangulation(G_S);
// $G_S = \{g_s \mid g_s$ is the control point on reference map$\}$
// T_S : set of triangles on the reference map
$T_T \leftarrow$ MappingTriangles(T_S);
// T_S : set of triangles on the other map
For each $p_T \in V_T$ {
$t_T \leftarrow$ FindTriangleContaining(T_T, p_T);
// search the triangle containing p_T
$t_S \leftarrow$ MappingTriangle(t_T);
// t_S : corresponding triangle on reference map
TriangularTransformation(t_S, t_T, p_T);
// correct p_T by relationship t_s and t_T
}
End

The most essential and complicated part of this algorithm lies on the function *TriangularTransformation*(t_s, t_T, p_T). It performs a correction of a point p_T, comparing the similarity coefficients between two triangles t_S and t_T. Suppose that M_s is a reference map and M_T is a corresponding map that contains positional data to correct. And let the triangle enclosing a point $p_T(x_T, y_T)$ on M_T be $t_T = (p_{T1}, p_{T2}, p_{T3})$, where p_{T1}, p_{T2} and p_{T3} are its vertices. Then $p_T(x_T, y_T)$ is given in terms of three vertices of t_T and three transformation coefficients λ_1, λ_2, and λ_3, by

$$x_T = \lambda_1 x_{T1} + \lambda_2 x_{T2} + \lambda_3 x_{T3} \text{ -- (1)}$$
$$y_T = \lambda_1 y_{T1} + \lambda_2 y_{T2} + \lambda_3 y_{T3} \text{ -- (2)}$$

Since p_T is within the triangle, $0 \le \lambda_1, \lambda_2, \lambda_3 \le 1$ and $\lambda_1 + \lambda_2 + \lambda_3 = 1$. If $t_S = (p_{S1}, p_{S2}, p_{S3})$ is the corresponding triangle on the reference map, then the transformed coordinates p_S (x_S, y_S) of p_T onto M_S are computed as follows,

$$x_S = \lambda_1 x_{S1} + \lambda_2 x_{S2} + \lambda_3 x_{S3} \text{ -- (3)}$$
$$y_S = \lambda_1 y_{S1} + \lambda_2 y_{S2} + \lambda_3 y_{S3} \text{ -- (4)}$$

We can derive the transformation coefficients λ_1, λ_2, and λ_3 from (1), and (2) as

$$\lambda_1 = \omega \{(y_{T2} - y_{T3}) x_T + (x_{T3} - x_{T2}) y_T + x_{T2} y_{T3} - x_{T3} y_{T2}\} \text{ -- (5)}$$
$$\lambda_2 = \omega \{(y_{T3} - y_{T1}) x_T + (x_{T1} - x_{T3}) y_T + x_{T3} y_{T1} - x_{T1} y_{T3}\} \text{ -- (6)}$$
$$\lambda_3 = \omega \{(y_{T1} - y_{T2}) x_T + (x_{T2} - x_{T1}) y_T + x_{T1} y_{T2} - x_{T2} y_{T1}\} \text{ -- (7)}$$
where $\omega^{-1} = x_{T1} y_{T2} + x_{T2} y_{T3} + x_{T3} y_{T1} - (x_{T2} y_{T1} + x_{T3} y_{T2} + x_{T1} y_{T3})$

Since $((y_{T2} - y_{T3}) x_T + (x_{T3} - x_{T2}) y_T + x_{T2} y_{T3} - x_{T3} y_{T2})/2$ is the area of a piece of triangle t_S divided by point p, and $\omega^{-1}/2$ is the area of t_s, λ_1 becomes the fraction of the area of a divided triangle over the area of t_s. Consequently, we can compute the coordinates of p_S (x_S, y_S) by (3) and (4). An example of execution flow of the algorithm is illustrated by figure 3.

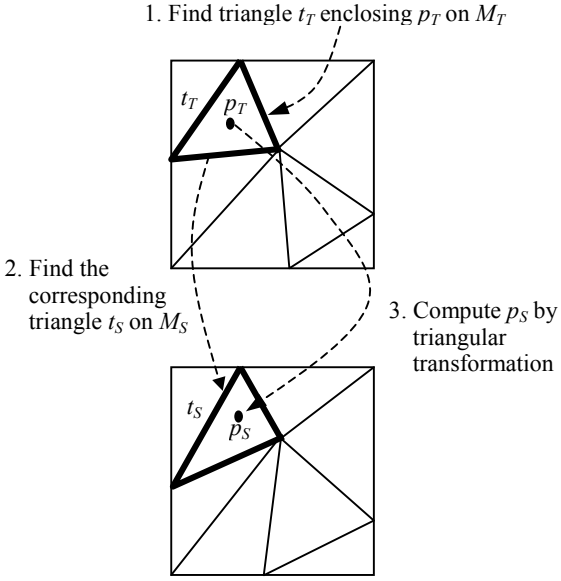

Figure 3. Example of Elastic Transformation by Delaunay Triangulation

This algorithm has several important properties. First of all, the topological properties between spatial objects on the original map are preserved on the transformed space as well. For example, if an object A contains another object B on the original map, A still contains B on the transformed space. This is a basic property of elastic transformation and we omit the proof of this property in this paper. In fact, the preservation of topology is a very important requirement for real applications. The second property of this algorithm is related to the maximum difference of positional data between the original and transformed spaces. It is explained by the following theorem.

Theorem. Maximum Difference of Elastic Transformation

Suppose that $G_S = \{g_{S1}, g_{S2}, \ldots, g_{Sn}\}$ and $G_T = \{g_{T1}, g_{T2}, \ldots, g_{Tn}\}$ are corresponding sets of control points on the reference map M_S and on a corresponding map M_T, which is to be transformed, respectively. Then

$$0 \le \text{dist}(p_T, p_S) \le \text{Max}(\text{dist}(g_{Tk}, g_{Sk}), 0 \le k \le n),$$

where p_T is a point on M_T and p_S is the transformed point onto M_S.

Since this theorem is evident, we omit the proof in this paper. This maximum difference is an important factor in determining whether we should perform an elastic transformation of a map. If the maximum difference is less than the positional accuracy of maps, we do not need elastic transformation.

4 Elastic Query Transformation by Delaunay Triangulation

In this section, we will propose a spatial query processing method that does not require any modification of local spatial databases. The main idea of our method is to transform query region or position instead of correcting local spatial databases. And we perform query processing with transformed region or points at each site. In this paper, we propose a query processing method for two types; the nearest neighbor and containment queries. Other types of spatial queries can be processed by similar ways.

When the user submits a spatial query to distributed spatial databases, he always specifies a query region or point on a map among them. For example, suppose that there are two spatial databases, one for roads and another for electric facilities. If the user wants to know the nearest road from a pole, he will specify a query point of the pole on the facility map. In this case, the query point on the facility map must be transformed onto the road map to process the submitted query. In other words, the query processing in distributed spatial databases includes the transformation of query region. The general procedure of query processing is depicted by figure 4.

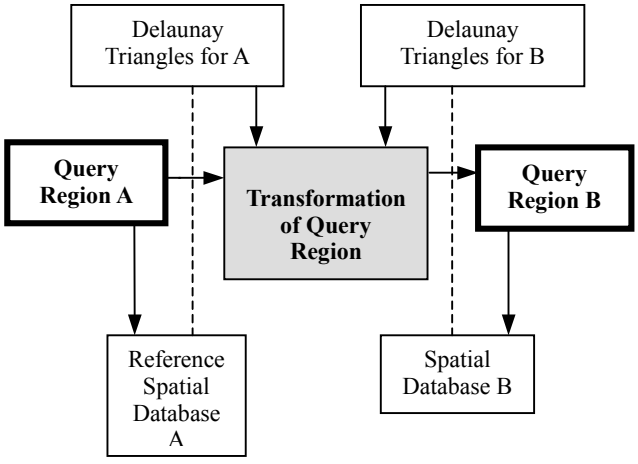

Figure 4. Transformation Procedure of Query Region

In the rest of this section, we will explain in detail how to transform query region.

4.1 Nearest Neighbor Query Processing by Elastic Transformation

To submit a nearest neighbor query, a point should be specified

on a map, which we call *query reference map*. It refers to the map on which the nearest query point is initially specified. The query point on the query reference map is to be transformed onto other maps of local databases. Each transformed point becomes the query point for nearest neighbor search on each local database and the query processing is independently performed at each site. The detailed algorithm is shown by algorithm 2.

A basic assumption of this algorithm is that we are provided with a set of pairs of the corresponding control points on the query reference map and a map of local spatial database, respectively. We have explained function *TriangularTransformation* of this algorithm in the previous section. And each nearest neighbor query is to be processed at each site by its own query processing method [5]. The performance of this algorithm is mainly determined by delaunay triangulation and local query processing time. Since the local query processing for nearest neighbor search is out of the scope of this paper, we do not discuss on its performance.

Algorithm 2: Nearest Neighbor Search Processing by Elastic Transformation

Input S_S : spatial database of the query reference map
S_T : a local database
$G = \{ (g_s, g_t) \mid g_s, g_t$: corresponding control points on S_S and $S_T \}$
p_S : query point on S_S
Output R_S , R_T : results of query processing on S_S, S_T

Begin
$T_S \leftarrow$ DelaunayTriangulation(G_S);
// $G_S = \{g_s \mid g_s$: set of control points on $S_S \}$
$T_T \leftarrow$ MappingTriangles(T_S);
// T_T: set of corresponding triangles on M_T
$t_S \leftarrow$ FindTriangleContaining(T_S, p_S);
// find the triangle t_S containing p_S
$t_T \leftarrow$ MappingTriangle(t_S);
// find the triangle on M_T corresponding with t_S
$p_T \leftarrow$ TriangularTransformation(t_s, t_T, p_S);
// derive the coordinates of p_T from t_s, t_T and p_T
$R_S \leftarrow$ PerformQuery("Nearest Neighbour", S_S, p_S);
$R_T \leftarrow$ PerformQuery("Nearest Neighbour", S_T, p_T);
End

The time complexity delaunay triangulation is known as O($n$log$n$) [6], where n is the number of control points. But its processing time is not important since n is normally a small number. If n is a relatively large number, we can pre-compute delaunay triangulation and prepare the corresponding pairs of triangles. And this algorithm is provided with the pairs of corresponding triangles instead of the corresponding control points. This approach improves the performance of the algorithm by removing dynamic recalculation of delaunay triangles per each query. Figure 5 illustrates an example of query and its result.

4.2 Range Query Processing by Elastic Transformation

The query processing procedure for range query is similar but a little more complicated than the nearest neighbor query. The reason is that a range query is given with a region and the transformation of a region is more complicated than that of a point. An example of transformation of a region is depicted by figure 6.

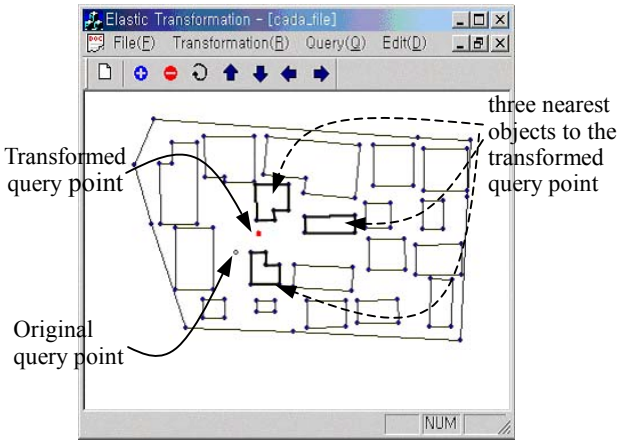

Figure 5. Result of 3-Nearest Neighbor Query

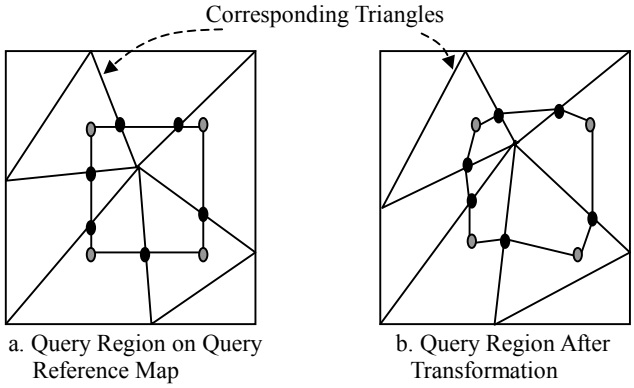

Figure 6. An Example of Range Query Processing

The gray and black points in figure 6a indicate the vertices of query region and intersecting points with edges of delaunay triangles respectively. Since the transformation parameters are differently applied to each triangle, the intersecting points should be considered for the transformation. The algorithm for range query processing with query region transformation is give by algorithm 3. This algorithm includes the transformation procedure of a region, which is also based on the triangular transformation.

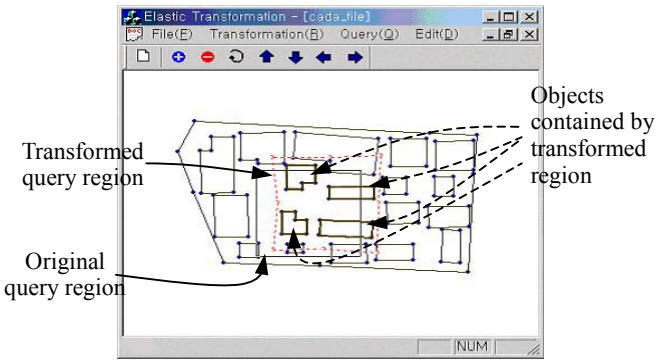

Figure 7. Result of Containment Query

An example of query processing result is shown by figure 7. The rectangle with straight lines depicts the original query

region and the dotted lines indicate its transformed region. The polygons with bold lines of the figure represent the buildings contained by the transformed query region.

The computation time for transformation in this algorithm is almost negligible. And the performance of the algorithm is nearly dominated by delaunay triangulation and local query processing time [4].

Algorithm 3: **Range Query Processing**
Input S_S : spatial database of the query reference map
S_T : other local database
G = { (g_s, g_t) | g_s, g_t: corresponding control points on S_S and S_T }
$a_S = (p_{S1}, p_{S2}, p_{S3}, \ldots, p_{Sn})$: query region given on S_S
Output R_S , R_T : results of query processing on S_S, S_T
Begin
$T_S \leftarrow$ DelaunayTriangulation(G_S);
// $G_S = \{g_s \mid g_s$: set of control points on S_S }
$T_T \leftarrow$ MappingTriangles(T_S);
// T_T: set of corresponding triangles on M_T
$b_S \leftarrow$ MakeNewPolygonWithIntersectingPoints (a_S ,T_S);
// make a new polygon with the vertices of a_S and intersecting points
For each vertex $q_{Sk} \in b_S$ {
$t_S \leftarrow$ FindTriangleContaining(T_S, q_{Sk});
// find the triangle t_S containing q_{Sk}
$t_T \leftarrow$ MappingTriangle(t_S);
// find the triangle on M_T corresponding with t_S
$q_{Tk} \leftarrow$ ElasticTransformation(t_s, t_T, q_{Sk});
}
$a_T \leftarrow (q_{T1}, q_{T2}, q_{T3}, \ldots, q_{Tm})$;
$R_S \leftarrow$ PerformQuery("Contain", S_S, a_S);
$R_T \leftarrow$ PerformQuery("Contain", S_T, a_T);
End

4.3 Correctness and Comparison

To prove the correctness of the algorithms presented in the previous sections, we should know the correct answer of given query. Unfortunately, there is no way to distinguish incorrect databases from the correct ones, and to find the correct answer. Instead of proving the correctness of the algorithms, we will intuitively explain that the query answers obtained by the map transformation and the query transformation are identical.

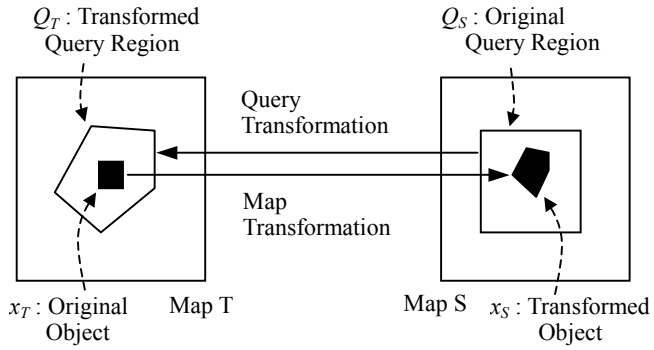

Figure 6. Equivalence between Map Transformation and Query Transformation

The equivalence of query results can be explained by topological invariant of elastic transformation. The map transformation method by delaunay triangulation is composed of triangular transformation presented in section 3. It means that topological properties on a map are preserved if each triangular transformation preserves them. And any topology on a map is maintained during triangular transformation. The detail proof for containment query is given in the appendix of the paper.

5 Conclusion

When we integrate several spatial databases of distributed environment, which have been built independently, positional mismatches become a crucial problem in processing spatial queries. A possible solution is to correct the positional data in all spatial databases according to the most accurate one. This approach however does not respect the autonomy of local systems and it is not practical in many cases.

In this paper, we proposed a spatial query processing method without correcting positional data in local spatial databases. Instead of correcting positional data, we dynamically transform a given query region or position onto each space where spatial objects of each site are located. Our method is based on an elastic transformation by delaunay triangulation. It gives the identical result with the map transformation method, which transforms positional data in spatial databases to correct mismatches. Table 1 summarizes a comparison between the map and query transformation methods.

	Map Transformation	Query Transformation
Outputs	Identical	
Autonomy	Not Respected	Respected
Coherence Maintenance Cost	Expensive (Periodic Transformation)	Inexpensive
Implementation	Easy	Difficult

Table 1. Comparison between Map Transformation and Query Transformation

A basic assumption of our method is that we are provided with a set of pairs of the corresponding control points selected from each local database. In order to implement our method in a distributed environment, the control points should be stored and managed in an efficient way by each local site or a middleware. The next step of the research should include an operational design of our method for distributed environments.

6 References

[1] M. G. Cho, K. J. Li and H. G. Cho, "A Rubber Sheeting Method With Polygon Morpging", Proc. Spatial Data Handling, pp.7A 31-42, 1996

[2] S. Servigne, and R. Laurini, "Updating Geographic Databases Using Multi-Source Information", Proc. ACM-GIS, pp.119-126, 1995

[3] M. G.. Cho and H. G.. Cho, "Resolving Mismatches and Measure Functions for Evaluating its Validity", Proc. Spatial Data Handling, pp.55-65, 2000

[4] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree, An Efficient and Robust Access Method for Points and Rectangles", Proc. ACM SIGMOD, pp.322-331, 1990

[5] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest Neighbor Queries", Proc. ACM SIGMOD, pp.71-79, 1995

[6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry*, Springer-Verlag, 1997

[7] ISO/TC211/WG2 Report N298 ISO WD15046-20, *Geographic information-Spatial Operators*, 1996

[8] ISO/TC211/WG2 Report, N289, *Geospatial Data Models and Operators*, 1996

[9] OpenGIS Consortium, *The OpenGIS Abstract Specification*, version 4., 1999

[10] P. Langlois, *Une transformation élastique du plan basée sur un modèle d'interaction spatiale, Applications à la géomatique*. Technical Paper in French, MTG, University of Rouen, 1994

[11] P. Dufour, *Les bases de données géographiques fédérées : continuitiés géométriques et topologiques*, Mémoire de DEA in French, INSA de Lyon, June 1995

Appendix

Lemma 1

When p is within a triangle $t(p_1, p_2, p_3)$, the transformed point p' from p is also in the transformed triangle $t'(p'_1, p'_2, p'_3)$, as shown by figure 7.

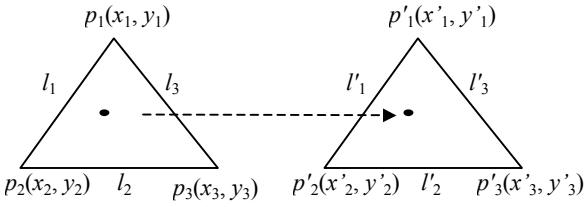

Figure 7

Proof

According to equations (1) and (2), the transformed position p' is computed by

$$p' = \lambda_1 p'_1 + \lambda_2 p'_2 + \lambda_3 p'_3, \text{ where } \lambda_1 + \lambda_2 + \lambda_3 = 1.$$

Since $0 \le \lambda_1, \lambda_2, \lambda_3 \le 1$, we see that p' is within the triangle $T'(p'_1, p'_2, p'_3)$. ▪

Lemma 2

If p is on the right side of a polyline pl $(v_1, v_2, \ldots, v_k)$ passing through a triangle t, its transformed point p' is on the same side of the transformed polyline pl' as shown by figure 8.

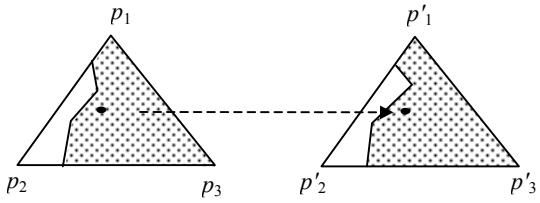

Figure 8

Proof

Let $l_i(v_i, v_{i+1})$ be a line segment of pl and l'_i be its transformed line segment. Then by lemma 1, we see that l'_i is in the transformed triangle t' from t if l_i is in t. And let v_i and v_{i+1} be (x_a, y_a) and (x_a, y_b) respectively. Suppose that $p\,(x, y)$ is on the lower side of l_i. Then

$$(x - x_a)(y_b - y_a) - (y - y_a)(x_b - x_a) > 0 \text{ -- (8)}$$

Let the ending points of l'_i be (x'_a, y'_a) and (x'_b, y'_b) and the transformed point p' be (x', y'). Then we derive the corresponding left-hand of inequality (8) for the transformed line segment and point as follows according to equations (3-7) of section 3,

$$(x' - x'_a)(y'_b - y'_a) - (y' - y'_a)(x'_b - x'_a)$$
$$= 4\,\{(x - x_a)(y_b - y_a) - (y - y_a)(x_b - x_a)\}\, A(t)\, A(t')$$

where $A(t)$ is the area of triangle t.

Since $A(t)\, A(t') > 0$ and $\{(x - x_a)(y_b - y_a) - (y - y_a)(x_b - x_a)\} > 0$ by inequality (8), we conclude that

$$(x' - x'_a)(y'_b - y'_a) - (y' - y'_a)(x'_b - x'_a) > 0$$

It means that p' is on the lower side of l'_i. Thus p' is on the same side of the transformed every line segments belonging to the polyline pl'. Consequently, p' is on the same side of pl. ▪

Theorem

If a polygon m is contained by query region R, then the transformed polygon m' is also contained by the transformed query region R'.

Proof

Polygon m and query region R are divided by a set of triangles as shown by figure 9.

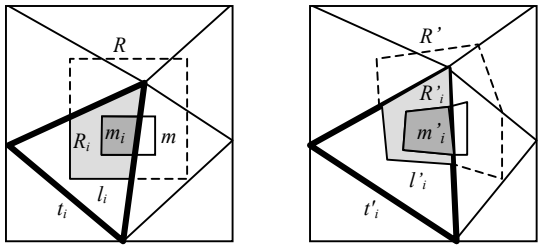

Figure 9

For any piece m_i $(= m \cap t_i)$ of m in triangle t_i contained by R_i $(= R \cap t_i)$, we should prove that the transformed piece m'_i is also contained by the transformed piece of R'_i as illustrated by figure 9. If a point in m_i is on the right (or left) side of l_i, which is a clipped query boundary line, we see that this point is also in the triangle t'_i and on the same side of l'_i as figure 9 by lemma 1 and lemma 2. It means that every point in m'_i is contained by R'_i. Therefore, m'_i is contained by R'_i. ▪