# An Algorithm for Geo-Spatial Objects Adjustment

by

Ismail Wadembere and Patrick Ogao
*{College of Computing and Information Science, Makerere University Kampala, Uganda}*
*[Corresponding author contacts: Tel: 0772472922 email: wadembere@gmail.com]*

**Abstract**

GIS practitioners always integrate geo-spatial data from different sources using map-overlay operations in order to make decisions and solve queries that deal with multiple layers. But they are always faced with openings and overlaps among objects that form features in thematically same datasets resulting into slivers (unwanted small objects) and danglings (unwanted intersections, polylines, and end points). Most GIS users end up running clean and build algorithms that just remove unwanted objects but not achieving perfect merging originally intended with main reason being lack of geometrical object based algorithm that can be used to update and adjust spatial objects to eliminate discrepancies caused by geometry differences between features of thematically similar data. We present an algorithm for manipulating geometries of geo-spatial objects that make up datasets basing on spatial points as the simplest geometrical primitive. Our approach makes it possible to represent high level spatial geometrical shapes using points and allows for more comprehensive handling of all shapes still maintaining the simplicity of working with the point primitive. Representation of spatial geometry shapes comprising of spatial points, polylines, and polygons using spatial point primitive geometry is achieved by scanning all the spatial dataset, observing the different geometry shapes and topology that exist, then representation all shapes using points and primary attribute in form of text. The algorithm carries out shape transformation of objects so that all openings and overlaps between objects making up dataset are eliminated so that geo-processing, modeling, and analysis utilize the properly integrated datasets. The algorithm can be applied in spatial data management like geometrical alignment and sharing environments like spatial data infrastructure.

**Keywords:** spatial points, geometry representation, and object adjustment

## 1 Introduction

Geo-spatial datasets especially those representing earth's surface and land uses are constantly changing, so is the need to continuously update them, which is always done by different individuals and organizations concerned with certain aspects and locations. However, storing layers that make-up those datasets separately makes it difficult to directly solve topological queries relating to features that belong to different layers [Oosterom, 1994 and Martinez et al., 2009]. Therefore, GIS practitioners integrate geo-spatial data from different sources using map-overlay and merging operations as it is difficult for an individual to have all data needed for different tasks.

As the integration takes place, there are shortcomings in terms openings and overlaps as even a simple geometric overlay operation of datasets of the same geographical extent, same thematic datasets, or adjacent location may cause features to intersect resulting into slivers (unwanted small objects) and danglings (unwanted intersections, polylines, and end points). The main reason being geometric discrepancies between features' boundaries [Sester, et al, 2007] due to cartographic (size and shape) and spatial (location) variation of features as a result of data capture and management.

We present an algorithm for manipulating geometries of geo-spatial objects basing on spatial points as the simplest geometrical primitive that can be used to update and adjust spatial objects that form features to eliminate discrepancies caused by geometry differences between features of thematically similar data.

Our approach makes it possible to represent high level spatial geometries using points and allows for more comprehensive handling of all shapes at the same time maintaining the simplicity of working with the point primitive. Representation of geometries comprising of spatial points, polylines, and polygons using spatial point primitive in text format is achieved by scanning all the dataset, observing the different geometries, topology, and attributes that exist. Then the algorithm carries out transformations so that all openings and overlaps are eliminated before geo-processing, modelling, and analyses.

## 1.1 Geometry Adjustment Problem

To put the geometry adjustment problem into perspective, we took a small part of a street in a city suburb having the cadastral plots on which residential houses are built that share boundary with cyclist path next to flower gardens for beautifying the city on edge of tarmac road as shown on Figure 1-A.

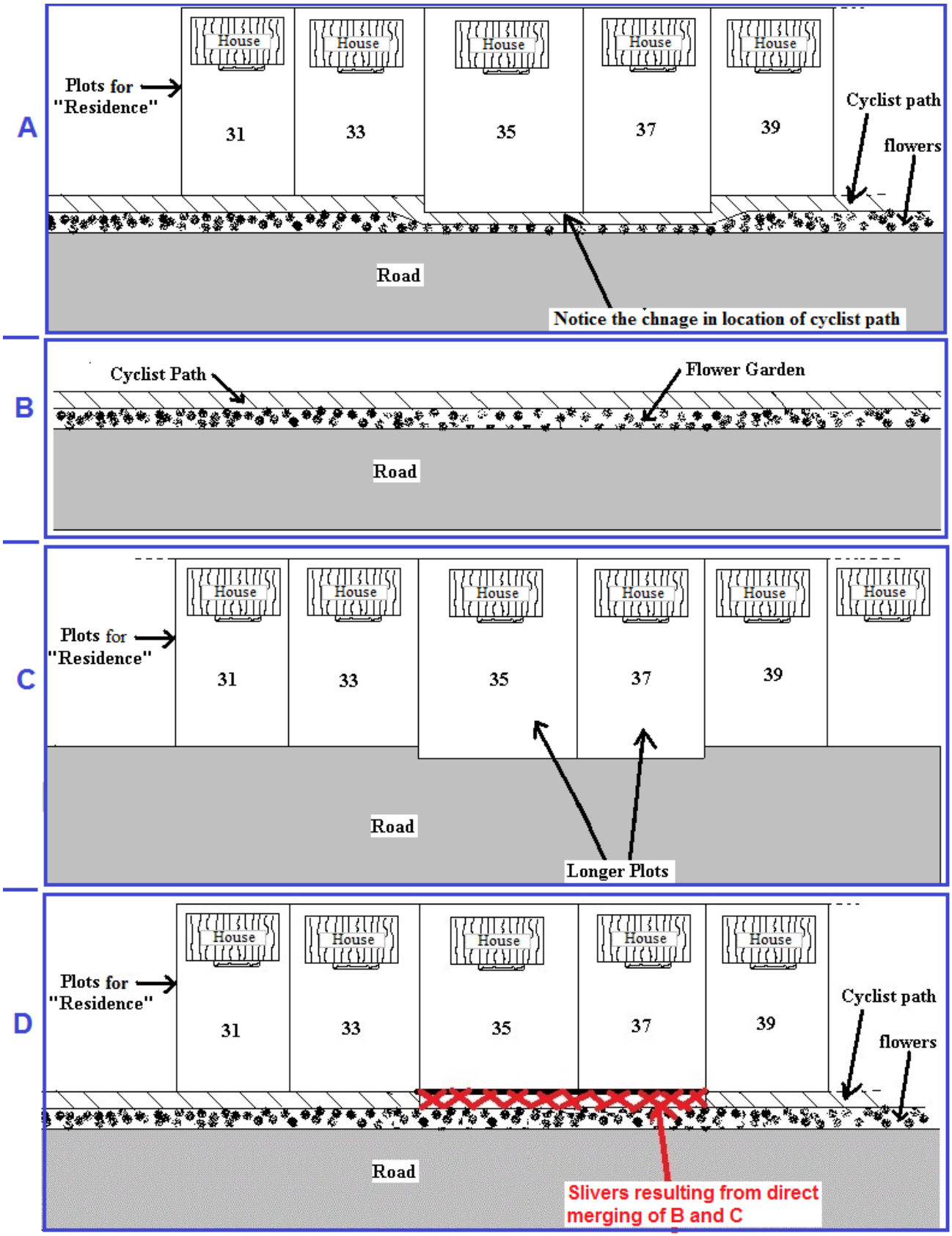


**Figure 1: Geometry adjustment problem**

It can be observed that plots 35 and 37 are longer than the plots 31, 33 and 39 making the cyclist path to be pushed towards the road along plot 35 and 37 that reduces on the width of the flower garden at that location, but it does not affect the road.

Investigating how thematic GIS layers that combine to give situation on Figure 1-A are stored, we get Figure 1-B having the straight road and Figure 1-C having plots. Analyzing Figure 1A-C, leads us to conclude that when we merge features on Figure 1-B with those on Figure 1-C, it does not give us what is actually on ground as represented on Figure 1-A, but a result where the cyclist path hits and intersects with plots 35 and 37 to create slivers as shown on Figure 1-D. Therefore, if the two layers are to be merged properly, the road layer has to be adjusted at the location of plots 35 and 37 to match with what is on cadastral layer to avoid slivers as they do not have meaning and may mislead analysts to think that new features have been added. Sometimes the GIS clean functions are used to eliminate the slivers, but that creates holes and resultant geodatabase do not depict the actual situation on ground. Thus, the need for *geometry object based algorithm* that can update and adjust localized individual geometries' discrepancies in geo-spatial dataset.

## 2 Related Work

Geo-spatial data is captured using different reference systems and geodetic datum like WGS-84 (World Geodetic System-1984) coordinate system used by GPS, Arc 1960 ellipsoid used for most cadastral work, and plane coordinate system used in GIS. These different datasets most of time have to be used together and managed under one system, this led to matching algorithms finding their ways into GIS to address various gaps :-

a) need to integrate scattered datasets for new applications to improve accuracy and reduce the costs of collecting data [Moosavi and Alesheikh, 2008]
b) facilitate multi-representation of objects so that geodatabases can consist of several geometrical elements with varying scales that improves automatic cartographic generalization [Bayer, 2008]
c) to present geodatabases uniformly that calls for adjustments to be done on datasets [ESRI, 2004)
d) to enable sharing spatial data [Liu et al., 2007]
e) for integration into one dataset that reduces on usage complication that may be difficult even when datasets are related to each other [Najar et al., 2006]

Different operations and many algorithms exist for carrying out clipping and finding intersection between two datasets. Some focus on merging similar geometric objects [Moosavi and Alesheikh, 2008] including exchange of attributes or for homogenizing geometry, address semantics heterogeneity of spatial datasets, improvement of quality in case one dataset is captured to a higher quality, multi representation based on the matching of two datasets using Unified Modeling language (UML), selecting matching objects using Structured Query Language (SQL); but non handles individual objects and part of object.

For polygon overlays, many algorithms are explained by Martinez, et al. [2009] and Liu, et al. [2007] that can work on different polygons like convex, rectangle, and concave polygons with some requiring complex and specific data structures and others computing Boolean operations on polygons that help to determine the intersection of segments. Although these operations determine the spatial coincidence (if any) of two data layers, they don't handle object's primitive in updating geospatial datasets.

Algorithms that handle points include the plane-sweep algorithms [Oosterom, 1994] that first sort the points of line-segments based on their x-coordinates which mean it does not employ an equal role for x and y values needed make better use of the "local processing principle" by taking both dimensions at the same time. Although that happens in the uniform grid algorithm, it does not perform well with non-

uniform distribution and he presents the R-tree based map-overlay algorithm that tries to overcome the problems of the existing algorithms. However, it does handle the partial object geometry adjustment.

For situations that call for moving objects using set requirements that work as constraints in the process of displacement like fixing one object and moving others. This can be achieved using the optimization technique of Least Squares Adjustment (LSA) basing on the neighborhood derivation from a Delaunay-Triangulation [Sester, 2000]. LSA using Jacobean matrix of the derivations of the functions maybe good for determining the unknowns in big datasets [Sester, 2000], but does not offer solution for specific part of object to be adjusted as it gives a global solution, where a displacement of one object occurs in accordance with all its surrounding objects. However, LSA is important in our algorithm as change of length of the objects' sides gives an indication for their deformation and the same holds for the change of the internal angles that help to decide which sides or segments to adjust. Also, LSA gives a measure for the absolute positional accuracy using change in the coordinates [Sester, 2000].

For the point-based object matching presented by Moosavi and Alesheikh [2008] of finding the best point to do the corresponding matching using weighted average between length and azimuth differences for control point parameter and target point and it involves normalization equation for distance parameter. This is good and can be employed in selecting the objects to use as reference or adjustment objects in our algorithm. But the gap is that the selected point is used to affect the whole layer, while our aim is for individual points.

# 3 Development of Spatial Object based Algorithm

## 3.1 Geo-spatial Data Structure

The spatial object based geometry adjustment algorithm is hinged on manipulation of geo-spatial data structure (see Figure 2 showing Butaleja district, Uganda) that comprises of three main components the geometry, the location, and the attributes as detailed in Table 1

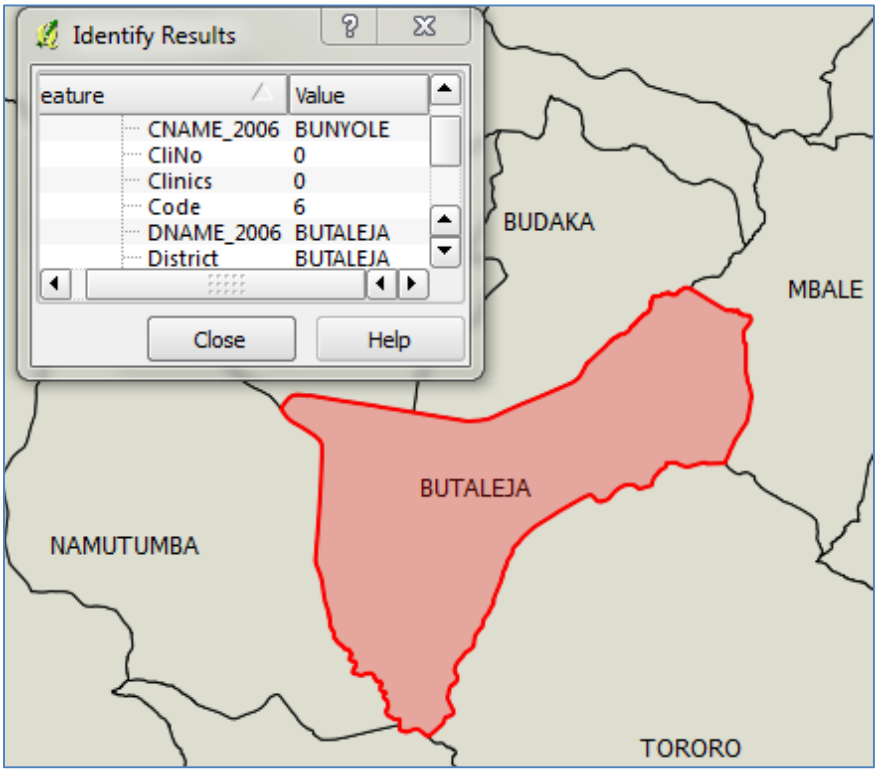

Figure 2: Geometry Shape of Butaleja District, Uganda

**Table 1: Geo-spatial Data Structure details**

| # | Field | Value | Min | Max |
|---|---|---|---|---|
| **1** | Geometry | Polygon | | |
| **2** | BoundingBox | [5.7910e+05, 2.8453e+05; 6.0896e+05, 3.1421e+05] | 2.8453e+05 | 6.0896e+05 |
| **3** | X | <1x12 double> | | |
| **4** | Y | <1x12 double> | | |
| **5** | District_Name | Butaleja | | |
| **6** | County_Name | Bunyole | | |

From table,

a) Row 1 has the **geometry** indicated in the column *"field"* as "Geometry" that can be of type point, line or polygon as shown in column "*value*". For Butaleja district, it is polygon.
b) Rows 2, 3, and 4 make up the **location**. Row 2 has the BoundingBox that indicates the location (area) extent of the data layer. Rows 3 and 4 are X-Y vectors that hold x-y coordinates of the Vertices that define the shape of objects.
c) Rows 5 and 6 plus rest of rows downwards make up the **attributes** that describe the meaning of structure and what is represented in the dataset. One of attributes is the Primary Attribute (PA) that defines main purpose or meaning of the shape of objects in the data structure. In this example, it is the district name "Butaleja" in the column "value" on fifth row.

It can be observed that the second component defines the shape of the feature and the location of the feature on/below earth's surface or in space. The algorithm manipulates the values in the second component (x-y matrices) of the data structure and bounding box that handle location of the dataset plus information on how to reproduce them. The properties that include name, metadata, value description, and type (Figure 3) are not affected. That means the structure's geometry type, topological and attributes are maintained.

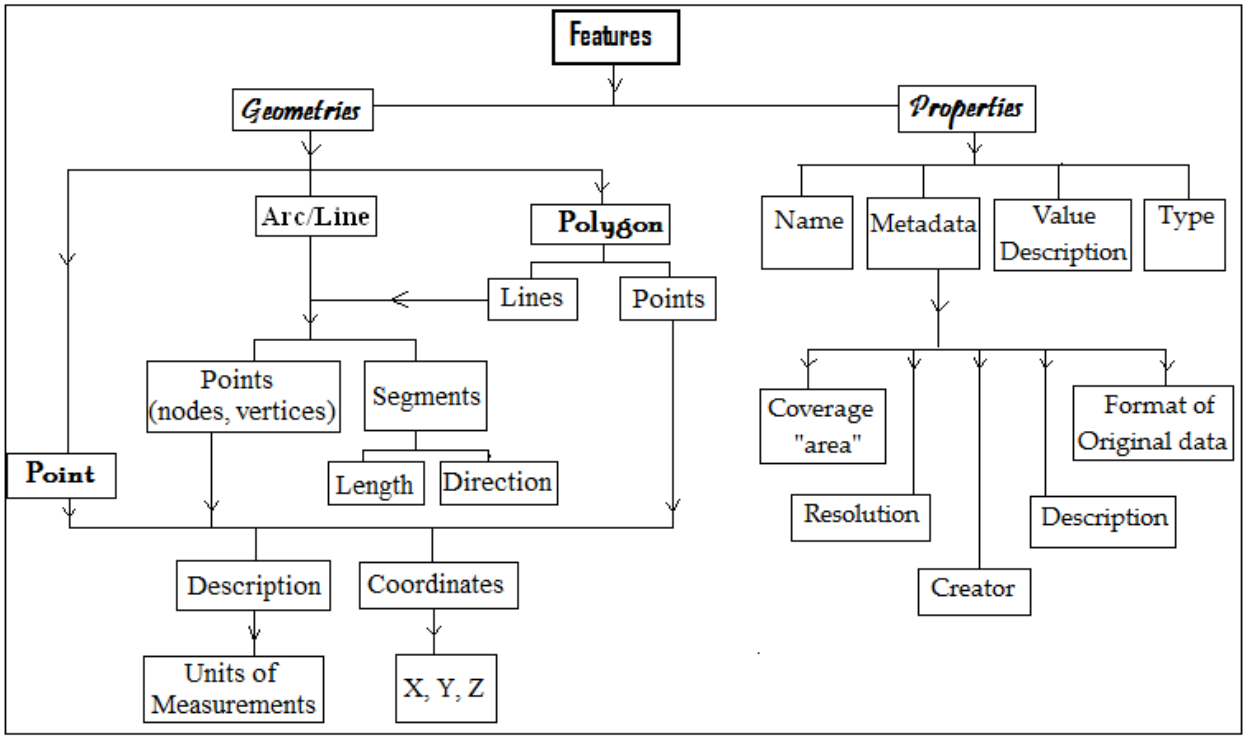


**Figure 3: Geometry represented using points**

On the figure, we start from the feature composed of geometries and their properties. The properties (name, metadata, value description, and type) of objects are not acted on by the algorithm. For geometries, the polygons are first represented by points and line-segments; the polylines are represented by points and line-segments. The line-segments are further

represented using points. At this moment, it is only the points existing as primitives plus information on how to reproduce them and it is the points that are manipulated by the algorithm as they are represented using the coordinates that form the second component of data structure as explained in Table 1

### 3.2 Algorithm Steps

The algorithm is based on individual points that form the shape of each object and it does not act on a whole layer at once. The points, vertices, and nodes of the objects or sections of objects are transformed in such a way that it changes the shape and location of objects that adjusts the dataset. This is accomplished using the steps of algorithm below as represented on the flow chart on Figure 4:

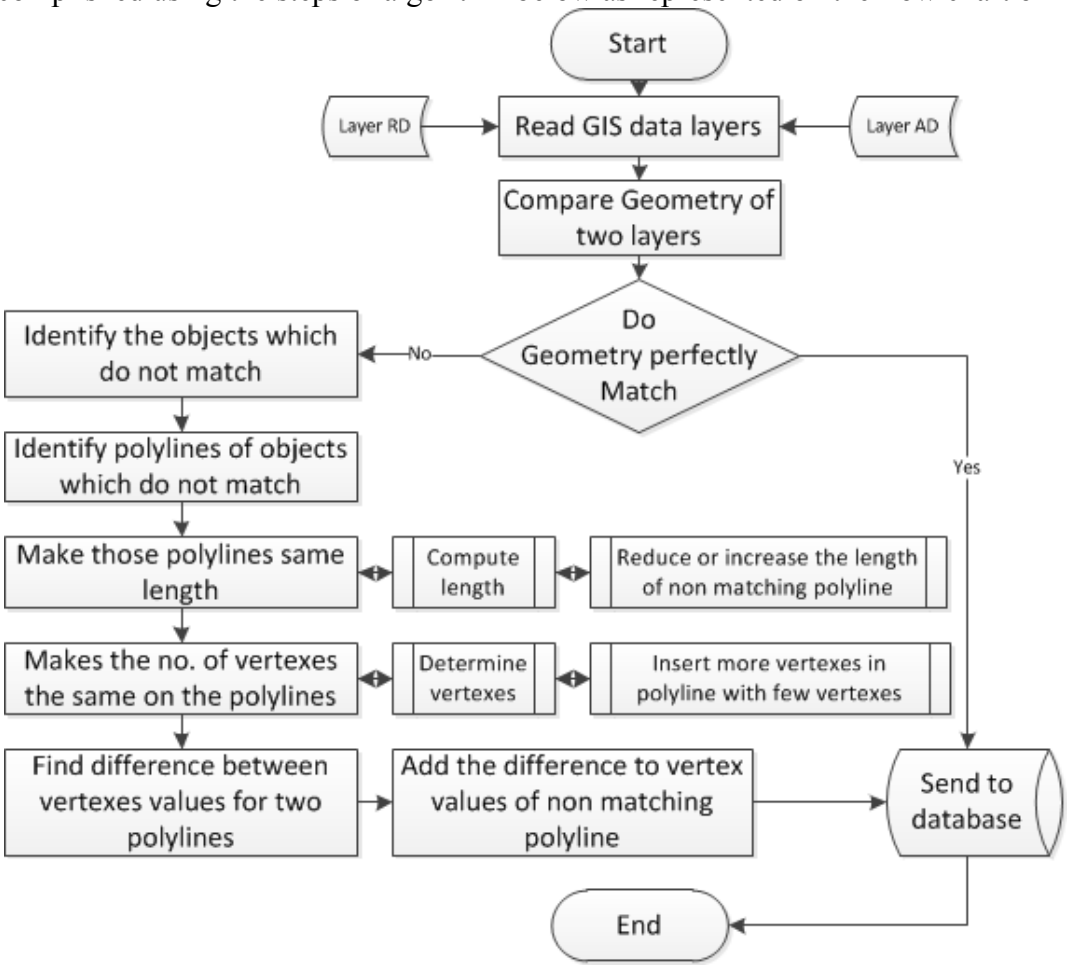


**Figure 4: Algorithm Flow chart**

From the figure, the steps of algorithm are

a) Read the datasets - adjustment dataset (AD) and reference dataset (RD) to be merged.
b) Identify adjustment object (AO) from adjustment dataset (AD) to be updated and corresponding reference object (RO) from reference dataset (RD) that will provide the adjustment values.
c) Identify polyline on AO (say QP) to be adjusted and corresponding polyline on RO (say RT) that will be used to compute adjustment parameters.
d) Compute length of the polylines (QP and RT).
e) Make the polylines (QP and RT) the same length
f) Make the number of vertices on the polylines (QP and RT) the same.
g) Read vertices' values from the polylines (QP and RT).
h) Compute the vertex value differences (dx and dy) between the corresponding vertices on the polylines (QP and RT).

i) Compute new vertices' values (Xu, Yu) for AO by adding the difference (dx and dy) to the vertices' values of RO.
j) Replace the vertices' values of polyline QP with the computed values (Xu, Yu)
k) Write the adjusted dataset (AD) on disk

### 3.2.1 Reading the datasets, identifying, and computing length

For user selected datasets, the algorithm starts by reading the datasets say adjustment dataset (AD) with various objects: AO, b, c, d, e, and f and another dataset - reference dataset (RD) having objects RO, $b^1$, $c^1$, $d^1$, $e^1$, and $f^1$. In this example the AD has adjustment object (AO) whose geometry has to be updated using values to be got from reference object (RO) on reference dataset (RD) (Figure 5A).

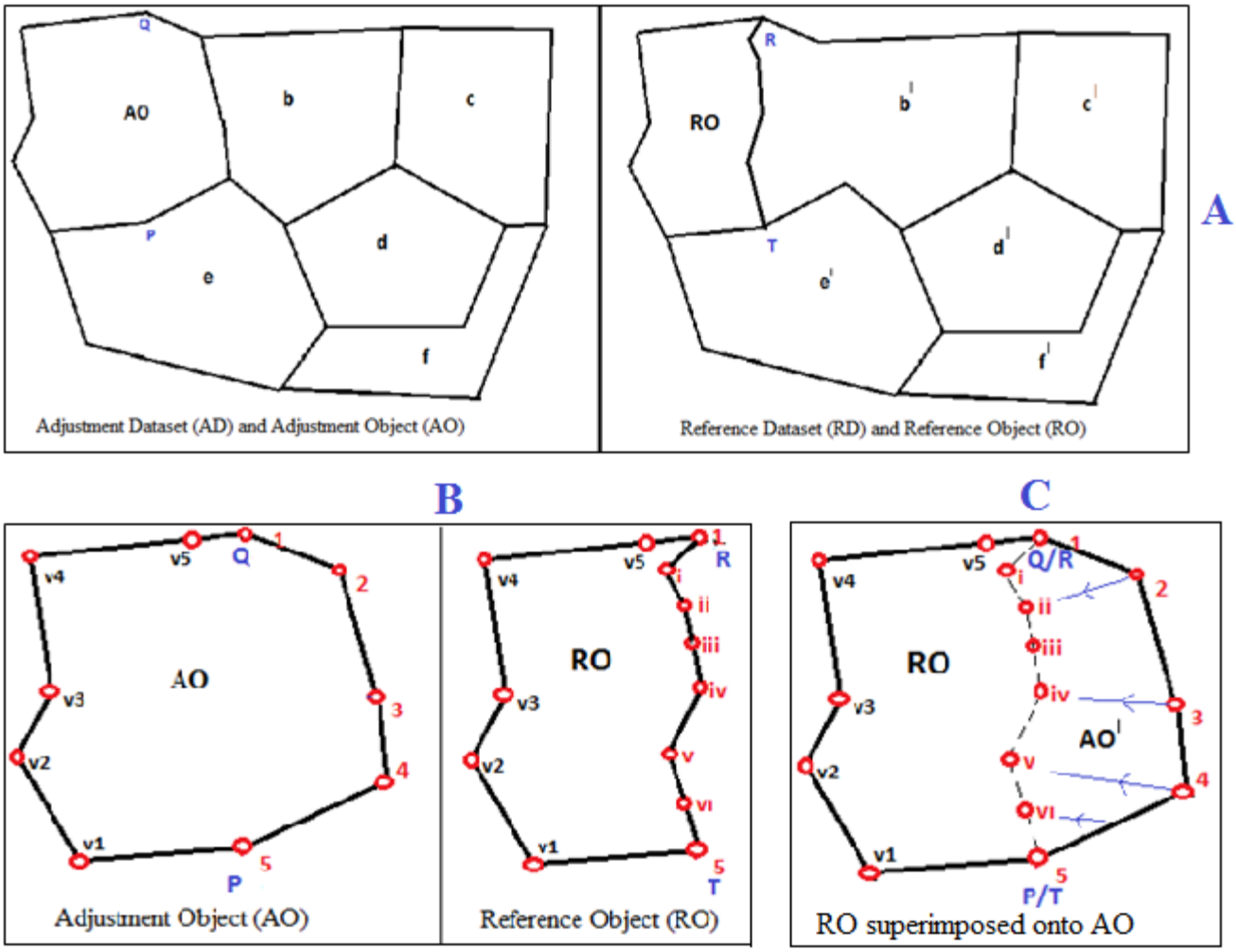


**Figure 5: Adjustment and Reference Datasets and Objects**

The algorithm compares datasets (AD and RD) to find geometrical variations by reading x-y coordinates for every vertex on each object in the datasets. For any corresponding objects in AD and RD (determined using primary attribute) whose geometry vary are marked as objects for adjustment as shown on Figure 5A that has two objects on AD (AO and b) on the left varying from the corresponding objects (RO and $b^1$) on RD on the right. But the adjustment process does not handle simultaneously adjacent two objects. One object is handled in the first round and if there are still variations, then the next object is handled in second round.

For each pair of marked corresponding objects say AO on AD and RO on RD as shown on Figure 5B (extracted from Figure 5A), all the x-y coordinates are read as sets of m by n matrixes. On Figure 5B, vertices of AO (P/5, v1, v2, v3, v4, v5, Q/1, 2, 3, and 4) and RO (T/5, v1, v2, v3, v4, v5, R/1, i, ii, iii, iv, v, and vi) read clockwise, we note that vertices P/5, v1, v2, v3, v4, v5, and Q/1 on AO are the same as vertices T/5, v1, v2, v3, v4, v5, R/1) on RO respectively. The beginning vertex (P) and end vertex

(Q) of AO are marked to get polyline PQ and the beginning vertex (T) and end vertex (R) of RO are marked to get polyline TR. That means polylines PQ and TR are the same. That part of AO is not updated and it is remaining section of AO - polyline QP that is adjusted using values from polyline RT.

To put that into perspective, we superimposed the RO onto AO from Figure 5B to give us Figure 5C that shows variation in size amounting to object $AO^{I}$. The arrows indicate direction in which to move vertices so that AO is adjusted to reflect the situation as depicted on RO. The algorithm makes the number of vertices the same and computes adjustment distance between corresponding vertices on AO and RO followed by the difference between adjustment polyline QP on AO and the reference polyline RT on RO as vectors of change in x-y values. This is followed by computing total length of polylines QP and RT. The length of polylines is the sum total of the distance between vertices. The distance between vertices is obtained as the resultant vector of change in x-y coordinates. For example distance for segment (D2-3) between vertex 2 and vertex 3 is computed using: $d_{2\text{-}3} = \sqrt{(X3 - X2)^2 + (Y3 - Y2)^2}$ with dx23 = (X3 – X2) and dy23 = (Y3 – Y2), Where X3 and Y3 are the coordinates of vertex 3, X2 and Y2 are the coordinates of vertex 2, and dx23 and dy23 are the difference in x and y values.

The algorithm computes the length of the polylines using formula

Length of QP on AO = $d_{1\text{-}2} + d_{2\text{-}3} + d_{3\text{-}4} + d_{4\text{-}5}$

Length of RT on RO = $d_{1\text{-}i} + d_{i\text{-}ii} + d_{ii\text{-}iii} + d_{iii\text{-}iv} + d_{iv\text{-}v} + d_{v\text{-}vi} + d_{vi\text{-}5}$

### 3.2.2 Making the Polylines and Vertices same

Adjustment polyline QP and corresponding RT (Figure 6-A) are selected and if the polylines have varying length, QP must be increased or reduced to match the RT.

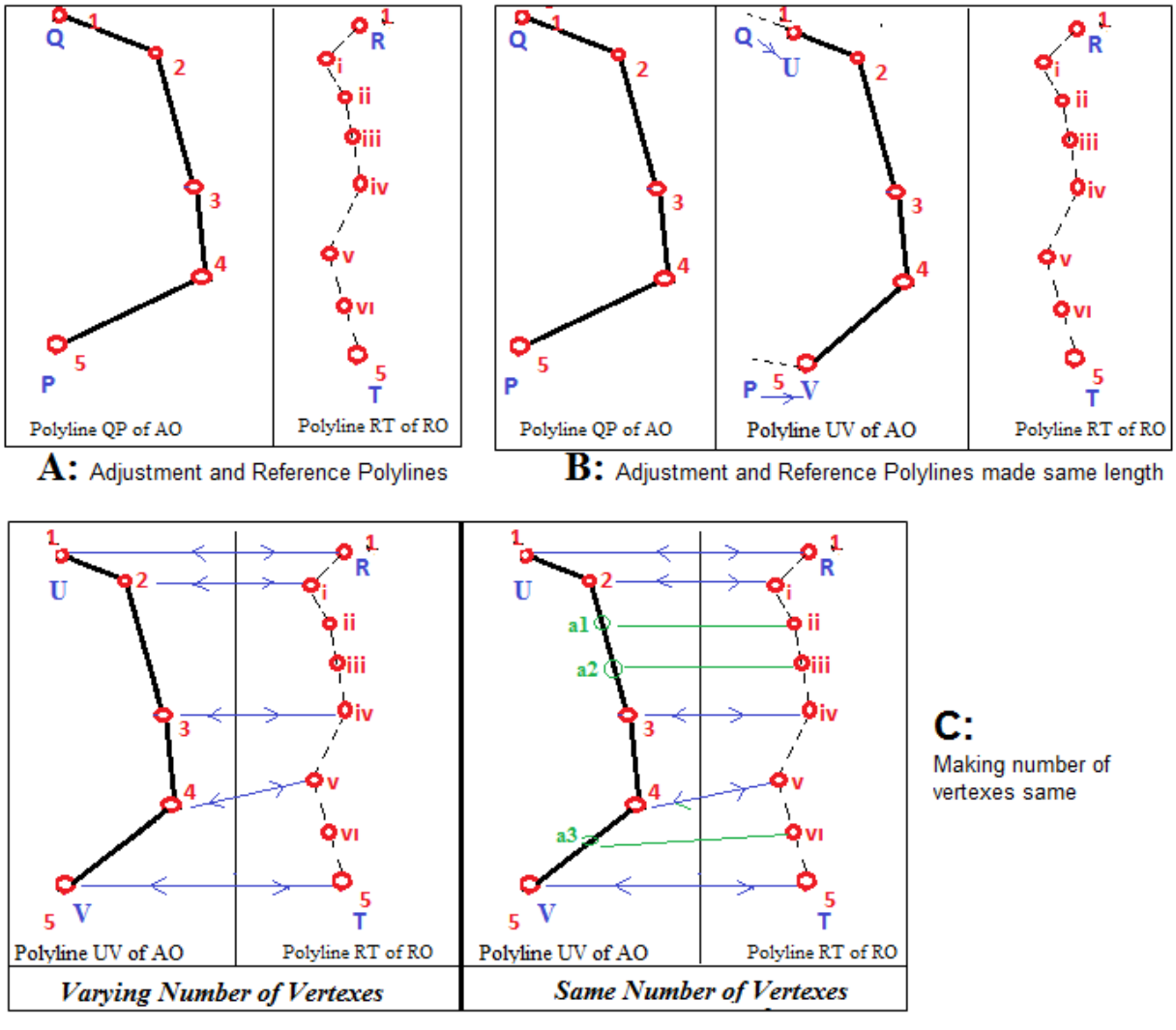


**Figure 6: Adjustment and Reference Polylines**

From Figure 6-A, we observe that QP is longer than RT, so the length of QP is reduced to be same as RT to get polyline UV on Figure 6-B. The algorithm accomplishes that by reading the x-y coordinates of the first vertex (Q/1) on polyline QP and substituting its values with those of the first vertex (R/1) on polyline RT. Then reading the x-y coordinates of the last vertex (P/5) on polyline QP and substituting its values with those of the last vertex (T/5) on RT and the intermediate vertices are moving using average of the end points. That gives a polyline UV the same resultant distance as RT.

In order to update UV using the vertices' values from RT there is needed to make the number of vertices on the two polylines the same. This involves inserting new vertices on polyline with fewer vertices. The rule is never delete vertices as they may contain information about topology, attributes, and may be holding relationship with adjacent objects. To enable mapping vertices of RT on UV, new vertices are inserted between the existing ones as shown on Figure 6-C with UV having 5 vertices (red/bold circles labeled 1-5) and polyline RT with 8 vertices. The algorithm accomplishes this by counting vertices on UV and RT. The polyline with fewer vertices is picked and insertion of more vertices is done e.g. vertices a1, a2, and a3 are inserted on UV. The algorithm compares the ratio of the distance between vertices on RT with those on UV and wherever distance on UV is more than the corresponding distance on RT, a vertex is inserted. The inserting just happens around the middle of the neighboring vertices. The x-y values of each inserted vertex are the average of the neighboring vertices on the left and right of that vertex being inserted. The process continues until the number of vertices is the same on the two polylines.

### 3.2.3 Computation of Adjustment Values

After making the number of vertices the same, algorithm reads all x-y coordinates of vertices into m by n matrix, where n is the number of vertices on that polyline and m is the polylines. If it is a polygon, then it is n -1 since the first and last vertices are the same for polygons.

Using formula dx = Xr – Xa and dy = Yr - Ya, difference between every two corresponding vertices computed, where

Xr and Yr are coordinate reading of the vertex on RT
Xa and Ya are coordinate reading of the vertex on UV
dx and dy are difference in coordinates between vertices on RT and UV

Using Xuv = Xa + dx and Yuv = Ya + dy, vertex values of UV on adjustment object are computed;
Where Xuv and Yuv are coordinates of the final adjusted object. There is need to compute these as sometime the Xuv and Yuv of updated object may need to be plotted directly.

The coordinate values Xuv and Yuv are used to over-write the values in the x-y coordinate matrixes of the UV polyline. This moves UV to proper location (Xuv, Yuv). The process is continued for each object in the dataset being adjusted. List of adjustment values are employed

List of x values ($Xuv_1$, $Xuv_2$, $Xuv_3$, $Xuv_4$……..$Xuv_n$)
List of y values ($Yuv_1$, $Yuv_2$, $Yuv_3$, $Yuv_4$……..$Yuv_n$)

### 3.2.4 Writing the adjusted dataset onto disk

The last step is to create GIS file from matrix onto disk. Three files for each shapefile are created with the same base name but varying file extensions. The extensions are .dbf (attribute format - columnar attributes for each shape, in dBase IV format), .shp (shape format - stores the geometry of the objects), and .shx (shape index format - a positional index of the object geometry to allow seeking forwards and backwards quickly). For example a shapefile of districts will have districts.dbf, districts.shp, and districts.shx

## 3.3 Pseudo-code for the algorithm

Read the data sets RD and AD
Compute polylines QP and RT length on the two objects AO and RO from the two layers RD and AD
If $QP_{AO}$ is longer than $RT_{RO}$,
    Increase $RT_{RO}$ to the length of $QP_{AO}$.

Elseif
    $RT_{RO}$ is longer than $QP_{AO}$
    Increase the $QP_{AO}$ to the length of $RT_{RO}$
Else
End
If $RT_{RO}$ has more vertices than $QP_{AO}$
    Inset more vertices $QP_{AO}$
    Until number of vertices $QP_{AO}$ = number of vertices on $RT_{RO}$
    Replace the vertices values of $QP_{AO}$ with values of $RT_{RO}$
Elseif $RT_{RO}$ has less vertices than $QP_{AO}$
    Insert more vertices on $RT_{RO}$
    Until number of vertices on $QP_{AO}$ = number of vertices on $RT_{RO}$
    Replace the vertices values of $QP_{AO}$ with values of $RT_{RO}$
Else
End
Write the GIS file on the disk.

# 4 Coding the Algorithm

## 4.1 Applications used

The algorithm was implemented in MATLAB (MathWorks www.mathworks.com) - a high-level language with interactive environment for scientific computing and it provides a good environment for developing and testing algorithms. The results were tested using Quantum Geographical Information System (QGIS) (www.qgis.org) - an open source GIS application developed by Open Source Geo-spatial Foundation (www.osgeo.org) whose functionality can be extended by developing plugins using python language (www.python.org) or C++ (www.cplusplus.com).

## 4.2 The Algorithm's Codes

The algorithm was coded using the standard approach in MATLAB. Below is part of the code to help understand how the coding took place. We have picked a part that accomplishes inserting vertices. In the algorithm there are six situations faced when inserting vertices.

a) polyline AD is shorter and has more vertices than polyline RD
b) polyline AD is shorter and has less vertices than polyline RD
c) polyline AD is longer and has more vertices than polyline RD
d) polyline AD is longer and has less vertices than polyline RD
e) polyline AD is longer and has same vertices as polyline RD
f) polyline AD is shorter and has same vertices as polyline RD

We used first situation where polyline AD is shorter and has more vertices than polyline RD to explain how the algorithm coding was done. We have numbered the different lines in code to help follow the explanation.

```
Line 01: %================================================================================
Line 02: %1) polyline AD is shorter and has more vertices than polyline RD
Line 03: if (noVerticesToInsert < 0) and (lengthdiff > 0)
Line 04:     noVerticesToInsert = abs(noVerticesToInsert);
Line 05:     % Create data to insert
Line 06:     xNewVertices = xvaluesRD;
Line 07:     yNewVertices = yvaluesRD;
Line 08:     vCount = 1; % counting number of vertices being inserted
Line 09:      while vCount <= noVerticesToInsert % runs noVerticesToInsert-1 times
Line 10:        % Compute the number of vertices on ployline
Line 11:        % length is function for computing number of vertices
Line 12:        % it also counts NaN (marker for end of polyline, so we substract one
Line 13:        noVertices = length(xNewVertices)-1;
Line 14:        vLoc = 1; % indicator of the location on ployline
```

```
Line 15:        lengthPoly = 0;
Line 16:        while vLoc < noVertices
Line 17:              % Compute length between the different vertices and sum it up
Line 18:              % remember the distance computation function between vertices
Line 19:              % vdist = distance(x1, y1, x2, y2) =
Line 20:              % sqrt((x1*x1)*(y1*y1)+(x2*x2)*(y2*y2))
Line 21:              x1 = xNewVertices(vLoc);
Line 22:              y1 = yNewVertices(vLoc);
Line 23:              vLoc = vLoc + 1;
Line 24:              x2 = xNewVertices(vLoc);
Line 25:              y2 = yNewVertices(vLoc);
Line 26:              % distance is a function for computing distance between two vertices
Line 27:              lengthSegment = distance(x1,y1,x2,y2);
Line 28:              lengthPoly = lengthPoly + lengthSegment;
Line 29:              % put the value in the distance vector
Line 30:              lengthSegVec(vLoc-1) = lengthSegment;
Line 31:        end
Line 32:        % find the longest segment on the polyline
Line 33:        longestSegment = max(lengthSegVec);
Line 34:        sCount = 1; % segment conuter on polyline
Line 35:        while sCount < noVertices
Line 36:            if lengthSegVec(sCount) >= longestSegment
Line 37:              segmentToInsert = lengthSegVec(sCount);
Line 38:              % Define the row where we will insert.
Line 39:              columnToInsert = sCount+1;
Line 40:              % Create vertices that we will insert
Line 41:              xVerticesToInsert = (xNewVertices(sCount)+xNewVertices(sCount+1))/2;
Line 42:              yVerticesToInsert = (yNewVertices(sCount)+yNewVertices(sCount+1))/2;
Line 43:              % Now insert vertices into lineVertices
Line 44:              % starting at row columnToInsert
Line 45:              xNewVertices = [xNewVertices(1:columnToInsert-1), ...
Line 46:              xVerticesToInsert, xNewVertices(columnToInsert:end)];
Line 47:              yNewVertices = [yNewVertices(1:columnToInsert-1), ...
Line 48:              yVerticesToInsert, yNewVertices(columnToInsert:end)];
Line 49:              % increase the vertex counter after inserting
Line 50:              vCount = vCount+1;
Line 51:              sCount = sCount+1;
Line 52:            else
Line 53:              sCount = sCount+1;
Line 54:        end
Line 55:      end
Line 56:      % to check id it is passing here
Line 57:      testif = 10;
Line 58: end
Line 59: % create duplicate structure as AD
Line 60: UVfinal = ADupdated(noObj);
Line 61: % transfer the new data
Line 62: ADupdated(noObj) = setfield(UVfinal,'X', xNewVertices);
Line 63: ADupdated(noObj) = setfield(ADupdated(noObj),'Y', yNewVertices)
Line 64: %================================================================================
```

We can observe that this part of code is separated from others by dotted on Line 01 and line 64. Functions have introductory comment on line 02, so the system analyst/ developers can understand the action of each code. Every specific action in the algorithm is introduced with an explanation of what is accomplished and were included as comments in the algorithm like on line 5, 10, 17, 32, etc. to make the code easy to read, there are spaces at the end of action like on line 31, 54, 55, 58, etc. For any "if statement" or other nest statements, the actions it accomplishes are indented under it to show that they are sub-actions like on line 03, 36, etc. In addition, individual action lines that make-up the nest statements have comments of what is being accomplished like on lines 08, 34, etc.

## 5 Running the Algorithm

The following conditions were used in testing the algorithm. So, datasets with those compositions and make-up were used:-

a) Where the adjustment object had less vertices compared to reference object
b) Where the adjustment object had more vertices compared to reference object
c) Where the adjustment object had the same vertices as reference object
d) Where the adjustment polyline was longer than the reference polyline
e) Where the adjustment polyline was shorter than the reference polyline
f) Where the adjustment dataset had more objects than the reference dataset
g) Where the adjustment dataset had fewer objects than the reference dataset
h) Where the adjustment dataset was in the same location as the reference dataset
i) Where the adjustment dataset was in a different location compared to reference dataset
j) Where the adjustment dataset had only one object for geometry updating
k) Where the adjustment dataset had many objects to adjust
l) Where the datasets have only points
m) Where the datasets have only polylines (lines and arcs)
n) Where the datasets have only polygons
o) Where the datasets were adjacent

All the above conditions were tested, but let use an example of many objects needing adjustment with adjacent objects - districts of Uganda as per 2006 and 2010. The dataset of 2006 had more administrative sub-divisions (districts, counties, and sub-counties) but with only 80 districts while the dataset of 2010 had 112 districts but with less administrative sub-divisions. The task was to come up with a dataset having the current 112 districts with the sub-divisions of counties and sub-counties with their attributes.

Running the algorithm on the datasets, the resultant dataset had 112 districts with counties, sub-counties, and attributes as of 2006 dataset. The process involves comparing all objects in the two datasets with the focus on the smaller objects (sub-divisions) representing sub-counties and counties that make up districts in Uganda (see Figure 7), then adjusting the polygons in 2006 using 2010 as RD.

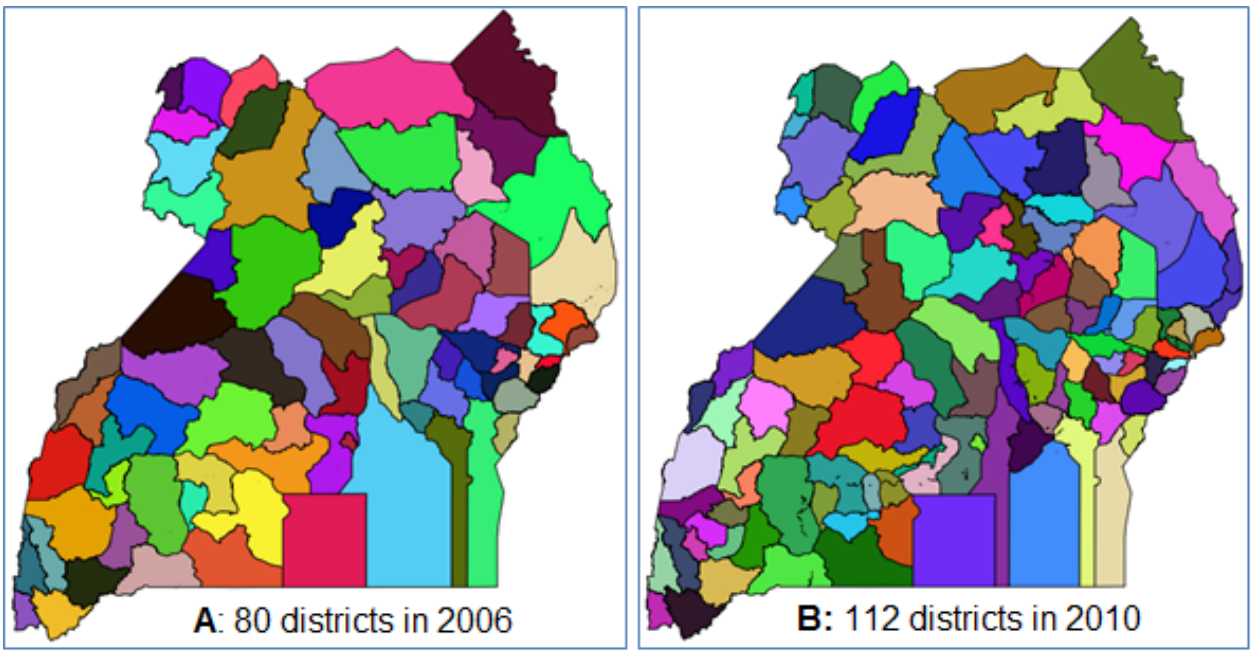


**Figure 7: Uganda districts**

The algorithm starts by reading all the objects available in both AD and RD, followed by comparing structures to check for geometry type (points, lines, polygons) and this is stored as text in matrix. The number of objects inside the structure is computed and is used to determine the number of iterations to

perform on the structure during the adjustment process. Adjusted procedure and after changing the x-y values, the algorithm compares the attributes that gives meaning to the shapes by promoting the user to identify the primary attribute for each object. If the user decides to add more attributes to the primary ones, then she/he indicates so. The algorithm proceeds by reading the attributes from the matrix and appending them to the attributes in the data structure of the target dataset, which are written into a data structure matrix and translated into GIS file.

### 5.1 Summary of Algorithms Outcomes

All the different testing conditions and running the algorithms produced results with the following changes or modifications being noticed:-

a) Where the AD had less or the same vertices as the RD, the resultant shapefile had the number of vertices equal to the number of vertices on the RD.
b) Where the AD had more vertices compared to RD, the resultant shapefile had the number of vertices equal to number of vertices as the AD.
c) In all conditions the final length of polylines adjusted were equal to the length of the reference polylines
d) Irrespective of the number of objects in the AD and RD, the final dataset always had the most number of objects in respect to AD or RD
e) The final location of AD was in the location of RD expect for complementary or adjacent datasets
f) In all conditions the AD and RD maintained attributes
g) The relationship (topology) between objects in the final adjust dataset was maintained
h) The final location of datasets was that of RD
i) Any object that needed adjustment was updated
j) The final geometry type was as per RD

## 6 Conclusion

Being able to read the geometry values from one object, then using them to adjust another object's geometry provides as easy way of adjusting geometry of one dataset with changes from another. Since all GIS packages are able to handle x-y coordinates, there is need to take advantage of these values so that the difference change individually in x-y is computed and applied directly to objects during updating and adjusting of its geometry. The use of the actual x-y coordinates eliminates the unnecessary computation that would involve comparing and determining the resultant adjustment values for the movement and direction of the object geometry. The other advantage is that there is no need to worry about adjusting the adjacent features as it is automatically done as the process takes place when topology is present.

With that, we avoid applying several techniques and procedures to achieve object adjustment like going through clipping, creating and managing intersections, having separate operations like for polygons only. This work directly contributes to the different efforts taking place in area of data and geometry clipping, intersection, merging, and integration like works by Liu et al. [2007], Xu et al. [2005], Hoel, et al. [2003], Ware and Jones [1998], Li et al. [2008], Vatti [1992], Bentley and Ottmann [1979], Sester et al. [2007], and Martinez et al.[2009].

## References


BAYER, T. (2008) The importance of computational geometry for digital cartography. Geoinformatics FCE CTU.

BENTLEY, J. L. & OTTMANN, T. A. (1979) Algorithms for reporting and counting geometric intersections. IEEE Transactions on Computers, 100, 643-647.

ESRI (2004) XML Schema of the Geodatabase - An ESRI Technical Paper, ESRI 380 New York St., Redlands, CA 92373-8100, USA www.esri.com, ESRI.

HOEL, E., S. MENON, et al. (2003). Building a robust relational implementation of topology. Lecture notes in computer science: 508-524.

LI, C., LIU, Y., WANG, Q. & TAI, Y. (2008) A GIS Database Updating Algorithm Based on GPS Surveying Data. 2008 International Conference on Computer Science and Software Engineering.

LIU, Y. K., WANGA, X. Q., BAO, S. Z., GOMBOSI, M. & ZALIK, B. (2007) An algorithm for polygon clipping, and for determining polygon intersections and unions. Computers & Geosciences, 33, 589-598.

MARTINEZ, F., RUEDA, A. J. & FEITO, F. R. (2009) A new algorithm for computing Boolean operations on polygons. Computers & Geosciences, 35, 1177–1185.
MOOSAVI, A. and A. A. ALESHEIKH (2008). Developing of Vector Matching Algorithm Considering Topologic Relations. Proc. of Map Middle East - MME08 PN 40.
NAJAR, C., RAJABIFARD, A., WILLIAMSON, I. & GIGER, C. (2006) A Framework for Comparing Spatial Data Infrastructures An Australian-Swiss Case Study. GSDI-9 Conference Proceedings. Santiago, Chile.
OOSTEROM, P. V. (1994) An R-Tree Based Map-overlay Algorithm. EGIS Foundation.
SESTER, M. (2000) Generalization based on Least Squares Adjustment. International Archives of Photogrammetry and Remote Sensing. Amsterdam.
SESTER, M., GÖSSELN, G. V. & KIELER, B. (2007) Identification and adjustment of corresponding objects in datasets of different origin. 10th AGILE International Conference on Geographic Information Science 2007. Aalborg University, Denmark.
VATTI, B. R. (1992) A generic Solution to Polygon Clipping. Communications of the ACM, 35.
WARE, J. M. and C. B. JONES (1998). Matching and aligning features in overlayed coverages, ACM.
XU, W., S. ZHU, et al. (2005). "A Hybrid Approach of Data Integration and GIS for Spatial Analysis in Bank and Capital Market." Environmental Informatics Archives 3: 364 - 371.