

Project Progress Report
ON
“Two Dimensional Cartographic Generalization of Maps”

Project Leader

Dr. J.L.Raheja

At

Digital System Group



Central Electronics Engineering Research Institute

Submitted To:

NRDMS
Department of Science & Technology
Government of India
Technology Bhawan
New Mehrauli Road
New Delhi - 110 016

Dr. J.L.Raheja
Digital System Group
CEERI,
PILANI

CONTENTS

1. INTRODUCTION	3 - 5
➤ CARTOGRAPHIC GENERALIZATION	
➤ CARTOGRAPHIC GENERALIZATION ALGORITHMS	
2. IMPORTING A GML FILE	6 - 14
➤ INTRODUCTION	
➤ OVERVIEW OF GML	
➤ CONCEPTUAL FRAMEWORK	
➤ NEED OF GML	
➤ FEATURES	
3. SELECTION	15-19
➤ IMPLEMENTATION OF SELECTION ALGORITHM	
➤ WORK DONE	
4. POLYLINE SIMPLIFICATION	20 - 28
➤ VERTEX REDUCTION ALGORITHM	
➤ DOUGLUS PEUCKER ALGORITHM	
5. REQUIREMENT SPECIFICATION	29 – 30
6. TECHNOLOGY USED	31 - 33
7. CONCLUSION & FUTURE ENHANCEMENTS	34- 35
8. BIBLIOGRAPHY	36 - 37
9. STATEMENT OF EXPENDITURE	38
10. UTILIZATION CERTIFICATE	39 – 40
11. EQUIPMENT LIST	41

INTRODUCTION

Cartographic Generalization: State of the Art

For a long time, geospatial information was printed on paper maps whose contents were produced once for specific purposes and scales. These maps are characterized by their portability, good graphic quality, high image resolution (van Elzakker 2004) and good placement of their symbols and labels. Traditionally, these maps have been generated manually by cartographers whose work was hard and fastidious. Nowadays, thanks to technological advances, more and more maps appear on paper support as well on the display screens of computers whose sizes are often reduced. These maps are generated for more specific scales and purposes such as tourism, transportation, cadastral applications and military needs.

Due to the lack of space, the real world cannot be expressed faithfully on computers' screens. Therefore, maps are often scaled-down to fit display sizes. This operation results in the reduction of maps' legibility. In order to improve legibility, many treatments must be applied on spatial objects. Indeed, according to screen sizes and to the scale and the theme of the map, many objects may need to be displaced, scaled-down or even eliminated. In addition, some objects need to be exaggerated in order to be kept over a certain threshold of visibility. Consequently, producing maps to be visualized on screens boils down mainly to a space planning problem. In this case, the planning aims to position the spatial objects, their symbols and their labels in a legible way and to maintain the existing spatial relationships between these objects and their spatial patterns. During this planning process, spatial objects may be regarded as competing for occupation of a limited space area. They undergo different transformations that may range from exaggeration to displacement, size reduction, elimination, etc. At the end of this competition process, the amount of spatial data may be reduced and the information is adjusted to a given scale and theme. This process is called cartographic generalization (Müller 1991, Weibel and Dutton 1999). Cartographic generalization can also be defined as 'the science and the art to exaggerate the important aspects (entities) in accordance with the purpose (thematic) and the scale of a particular map and the exclusion of irrelevant details that may overload the map and confuse its user' (Hardy 1998).

The activity of generalization is one of the cornerstones involved in map design. Through generalization mechanisms the physical reality of our world is abstracted and compressed both graphically and semantically. In an artistic sense, these processes of graphic and semantic change are analogous to a caricature in which certain features are emphasized and others are not. These techniques allow us to effectively condense geographic information and re-represent it and, in so doing, emphasize issues of particular interest. A map expresses a geographical reality, according to a specific scale and purpose. This reality cannot be accurately represented when the scale of the map is reduced. Indeed, scale reduction often diminishes the map's legibility which then requires some modifications in order to be improved. The process, during which these

alterations are applied, is commonly known as cartographic generalization when intended to produce data for cartographic visualization (Bader 2001).

Cartographic generalization aims to produce a good map, balancing the requirements of accuracy, information content, and legibility (Cecconi 2003). During this process, logical and unambiguous relationships between map objects must be maintained, while aesthetic quality is preserved (Weibel and Dutton 1999).

Cartographic generalization algorithms

Cartographic generalization is a very complex process. In order to reduce its complexity, the overall process is often decomposed into individual sub-processes, called operators (Lichtner 1979, McMaster and Shea 1992). Each operator defines a transformation that can be applied to a single spatial object, or to a group of spatial objects. The following example (Figure 1), gives a classification of generalization operators proposed by ESRI (ESRI 1996):

- Elimination This operator eliminates various geographic objects because of their small size or lesser importance with regards to the map's theme (e.g. elimination of small islands, elimination of short streets).
- Simplification This operator eliminates unnecessary details of a given spatial object, without distortion of its original shape (e.g. elimination of curved lines).
- Aggregation This operator merges nearby and adjacent objects into a new, single object (e.g. the merging of nearby small lakes into a single lake).
- Size reduction This operator reduces the size of a given geographic object.
- Typification This operator reduces the density of spatial objects, as well as their levels of detail. Meanwhile, it preserves the representative distribution pattern of these objects.
- Exaggeration This operator increases the spatial extension of the geometric representation of a given object, in order to focus on its importance, and to improve its legibility.
- Classification and symbolization This operator combines items that share similar geographic attributes into a new object, which in turn has a higher level of abstraction, in addition to a new symbol.
- Displacement This operator is used to move objects in a map in order to maintain the limits of separation between them.
- Refinement This operator alters and adjusts the geometry and appearance of an object in order to improve its aesthetic visual aspect and to ensure its compatibility with reality (e.g. performing the smoothing of a given line, modification of the orientation of some symbols).

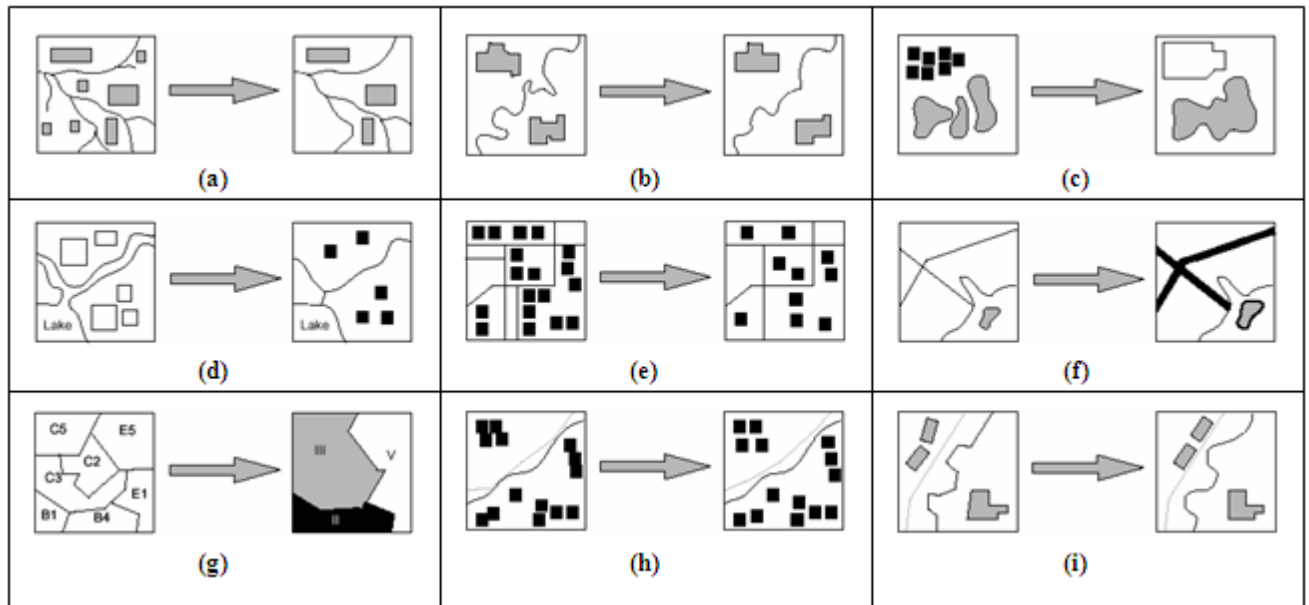


Figure 1 (a) Elimination, (b) Simplification, (c) Aggregation, (d) Size reduction, (e) Typification, (f) Exaggeration, (g) Classification and symbolization, (h) Displacement, (i) Refinement

An operator defines a transformation that can be applied to spatial data. Since this transformation depends upon the data model (e.g. raster or vector data), and on how it will be used, several algorithms may implement the same operator. For example, various algorithms have been developed for simplification (Reumann and Witkam 1974, Lang 1996, Douglas and Peucker 1973, Cromley and Campbell 1992), selection (Töpfer and Pillewizer 1996, Ruzak Mazur and Castner 1990, Thompson and Richardson 1995, Ruas 1999), and displacement (Lichtner 1979, Mackaness 1994, Roberts 1997, Jones et al. 1995, Ruas 1998, Bobrich 1996, Nickerson 1998, Bader and Weibel 1997, Lonergan and Jones 2001). Furthermore, several algorithms are generic, while others have been developed for specific types of data, such as buildings or roads (Cecconi 2003). Therefore, the choice of the suitable algorithm for each operator is important in order to make the best transformation on a given set of spatial data. This choice is usually done empirically. However, it may be based upon the efficiency and accuracy expected from the cartographic generalization process (McMaster and Shea 1992).

Multiple Representation Vs Cartographic Generalization

Multiple representation databases (Bernier et al. 2005) can quickly create maps at different scales (from predefined representations). However, it can not be an alternative to cartographic generalization. The major drawback of multiple representation is that it often generates voluminous databases and restricts the cartographic products to the predefined scales. Many research projects have addressed multiple representation in multi-scale databases, in which every object has different representations at different scales. In these databases, an object has a detailed representation at a high scale, as well as a simplified representation at a low scale. This approach reduces the complexity of multiple representation, but does not solve it.

GML Files

INTRODUCTION:

The amount of cartographic data distributed on the Internet is still increasing. Today, most of the data are raster data, but the emerging XML-standards will make it easier to distribute vector data on the Internet. One of the main advantages with vector data is that it is easier to integrate and generalize them than raster data. However, this requires a suitable technical environment to perform the data-integration and generalization.

During the last years some prototype systems for distributing vector data on the Internet have been developed. Many of these systems are based on the two XML-standards Geographic Markup Language (GML; OGC, 2002) and Scalable Vector Graphics (SVG; W3C, 2002). These two standards are complementary: GML is used for storing and distributing geographic data and SVG is used for presenting data. In many current prototype applications the cartographic data is stored in a database; when a map request is performed to the database a GML-file is created.

GML or Geography Markup Language is an XML based encoding standard for geographic information developed by the OpenGIS Consortium (OGC). It's current status is an RFC under review within the OpenGIS Consortium. The RFC is supported by a variety of vendors including Oracle Corporation, Galdos Systems Inc, MapInfo, CubeWerx and Compusult Ltd. GML was implemented and tested through a series of demonstrations which formed part of the OpenGIS Consortium's Web Mapping Test Bed (WMT) conducted in September 1999. These tests involved GML mapping clients interacting with GML data servers and service providers.

INSIGHT:

The Geography Markup Language (GML) is an XML encoding for the transport and storage of geographic information, including both the spatial and non-spatial properties of geographic features. This chapter defines the XML Schema syntax, mechanisms, and conventions that

- Provide an open, vendor-neutral framework for the definition of geospatial application schemas and objects;
- Allow profiles that support proper subsets of GML framework descriptive capabilities;
- Support the description of geospatial application schemas for specialized domains and information communities;
- Enable the creation and maintenance of linked geographic application schemas and datasets;
- Support the storage and transport of application schemas and data sets;
- Increase the ability of organizations to share geographic application schemas and the information they describe.

It defines the key concepts required to understand how the Geography Markup Language (GML) models the world. It is based on the OGC Abstract Specification which defines a geographic feature as "an abstraction of a real world phenomenon; it is a geographic feature if it is associated with a location relative to the Earth." Thus a digital representation of the real world can be thought of as a set of features. The state of a feature is defined by a set of properties, where each property can be thought of as a {name, type, value} triple. The number of properties a feature may have, together with their names and types, are determined by its type definition. Geographic features are those with properties that may be geometry-valued. A feature collection is a collection of features that can itself be regarded as a feature; as a consequence a feature collection has a feature type and thus may have distinct properties of its own, in addition to the features it contains.

OVERVIEW OF GML:

➤ DESIGN GOALS:

GML was developed with a number of explicit design goals, a few of which overlap the objectives of XML itself:

- provide a means of encoding spatial information for both data transport and data storage, especially in a wide-area Internet context;
- be sufficiently extensible to support a wide variety of spatial tasks, from portrayal to analysis;
- establish the foundation for Internet GIS in an incremental and modular fashion;
- allow for the efficient encoding of geo-spatial geometry (e.g. data compression);
- provide easy-to-understand encodings of spatial information and spatial relationships, including those defined by the OGC Simple Features model;
- be able to separate spatial and non-spatial content from data presentation (graphic or otherwise);
- permit the easy integration of spatial and non-spatial data, especially for cases in which the non-spatial data is XML-encoded;
- be able to readily link spatial (geometric) elements to other spatial or non-spatial elements.
- provide a set common geographic modeling objects to enable interoperability of independently-developed applications.

GML is designed to support interoperability and does so through the provision of basic geometry tags (all systems that support GML use the same geometry tags), a common data model (features/properties), and a mechanism for creating and sharing application schemas. Most information communities will seek to enhance their interoperability by publishing their application schemas; interoperability may be further improved in some cases through the use of profiles.

THE CONCEPTUAL FRAMEWORK

➤ FEATURES AND PROPERTIES

GML is an XML encoding for geographic features. In order to correctly interpret a GML document it is necessary to understand the conceptual model that underlies GML, which is described as in the OGC Abstract Specification.

➤ OBJECT MODEL:

A geographic feature is essentially a named list of properties. Some or all of these properties may be geospatial, describing the position and shape of the feature. Each feature has a type, which is equivalent to a class in object modeling terminology, such that the class-definition prescribes the named properties that a particular feature of that type is required to have. So a Road might be defined to have a name, a surface-construction, a destination, and a centreLine.

The properties themselves are modeled in UML as associations, or as attributes, of the feature class. The feature property type is given by the rolename from an association, or by the attribute name. The values of the properties are themselves also instances of defined classes or types. So the Road name is a text-string, the surface-construction might be a text token selected from an enumerated list, the destination is another feature of type Town, and the centreLine is a LineString, which is a geometry property. In UML it is partly a matter of taste whether a property is represented as an association or attribute, though it is common for a property with a complex or highly structured type to be modeled as an association, while simple properties are typically class attributes. If the value of a property only exists in the presence of the feature, such as the Road name, then it may use either a UML composition association or be represented as an attribute, as these two methods are functionally equivalent. However, if the value of a property is loosely bound to the object and the property value is an object that might exist independently of the feature, such as the Town that is the destination of a Road, then it must use a form of UML association called Aggregation.

- XML ENCODING OF THE OBJECT MODEL:

A feature is encoded as an XML element whose name is the feature type according to some classification. The feature instance contains feature properties, each as an XML element whose name is the property name. Each of these contains another element whose name is the type of the property value or instance; this produces a "layered" syntax in which properties and instances are interleaved.

GML adopts a uniform coding convention to help distinguish properties from instances: element names that represent instances of GML classes start with an uppercase letter (e.g. Polygon), while tags that represent properties start with a lowercase letter; all embedded words in the property name start with uppercase letters (e.g. centerLineOf).

- FUNCTIONAL VIEW OF THE OBJECT MODEL

From a functional perspective we can consider a property as a function that maps a feature onto a property value. A property is characterised by the input feature type and the type of the value that is returned. For example, suppose the feature type House has a String property called address and a Polygon property called extentOf. Using functional notation we can then write:

$$\begin{aligned} \text{address(House)} &= \text{String} \\ \text{extentOf(House)} &= \text{Polygon} \end{aligned}$$

This approach can also be applied to feature collections that have features as members:

$$\text{featureMember(FeatureCollection)} = \text{Feature}$$

➤ GEOMETRIC PROPERTIES

In general the definition of feature properties lies in the domain of application schemas. However, since the OGC abstract specification defines a small set of basic geometries, GML defines a set of geometric property elements to associate these geometries with features. The GML Feature schema also provides descriptive names for the geometry properties, encoded as common English language terms. Overall, there are three levels of naming geometry properties in GML:

1. Formal names that denote geometry properties in a manner based on the type of geometry allowed as a property value
2. Descriptive names that provide a set of standardized synonyms or aliases for the formal names; these allow use of a more user-friendly set of terms.
3. Application-specific names chosen by users and defined in application schemas based on GML

For example, a RadioTower feature type could have a location that returns a Point geometry to identify its location, and have another geometry property called extentOf that returns a Polygon geometry describing its physical structure. A geometric property can be modeled in UML as an association class.

➤ APPLICATION SCHEMAS

Three base XML Schema documents are provided by GML: feature.xsd which defines the general feature-property model, geometry.xsd which includes the detailed geometry components, and xlink.xsd which provides the XLink attributes used to implement linking functionality. These schema documents alone do not provide a schema suitable for constraining data instances; rather, they provide base types and structures which may be used by an application schema. An application schema declares the actual feature types and property types of interest for a particular domain, using components from GML in standard ways. Broadly, these involve defining application-specific types which are derived from types in the standard GML schemas, or by directly including elements and types from the standard GML schemas. The base GML schemas effectively provide a meta-schema, or a set of foundation classes, from which an application schema can be constructed. User-written application schemas may declare elements and/or define types to name and distinguish significant features and feature collections from each other; The <import> element in the Geometry schema brings in the definitions and declarations contained in the XLinks schema. The GML Geometry schema includes type definitions for both abstract geometry elements, concrete (multi) point, line and polygon geometry elements, as well as complex type definitions for the underlying geometry types. The <include> element in the Feature schema brings in the definitions and declarations contained in the Geometry schema; like the geometry schema, the Feature schema defines both abstract and concrete elements and types.

NEED FOR GML

There are already a host of encoding standards for geographic information including COGIF, MDIFF, SAIF, DLG, SDTS to name only a few. GML is a simple text based encoding of geographic features. Some of these other formats are not text based, however, some of them (e.g. SAIF) certainly are. GML is based on a common model of geography (OGC Abstract Specification) which has been developed and agreed to by the vast majority of all GIS vendors in the world. More importantly, however, GML is based on XML. There are several reasons why XML is important. To begin with XML provides a method to verify data integrity. Secondly, any XML document can be read and edited using a simple text editor. Nothing more than MS Notepad is required to view or change an XML document. Thirdly, since there are an increasing number of XML languages, it will be more and more easy to integrate GML data with non spatial data. Even in the case of non-XML non-spatial data this is the case. Perhaps, most importantly, XML is easy to transform. Using XSLT or almost any other programming

language (VB, VBScript, Java, C++, Javascript) we can readily transform XML from one form to another. A single mechanism can thus be employed for a host of transformations from data visualization to coordinate transforms, spatial queries, and geo-spatial generalization.

GML rests securely on a widely adopted public standard, that of XML. This ensures that GML data can be viewed, edited and transformed by a wide variety of commercial and free ware tools.

FEATURES ENHANCING ITS SIGNIFICANCE

➤ AUTOMATED VERIFICATION OF DATA INTEGRITY

One of the important features of XML is the ability to verify data integrity. In the XML 1.0 Recommendation this is achieved through the Document Type Definition (DTD). The DTD specifies the structure of an XML document in a such a way that a validating parser can verify that a given document instance complies with this DTD. GML is specified by such a DTD. Future versions of GML will also be supported by XML Schema, a more flexible integrity mechanism than the DTD that should become a W3C Recommendation early in 2000.

Using the GML DTD, servers and clients can readily verify that the data they are to send or receive complies with the specification. Furthermore this can be accomplished with a variety of parsing tools by at least a have a dozen different vendors on a wide variety of operating systems, databases, application servers and browsers.

➤ GML CAN BE READ BY PUBLIC TOOLS

As we have already noted, GML is text and one need have nothing more than a simple text editor to read it. GML, however, is structured, and any of a variety of XML editors can be employed to display that structure. This makes viewing and navigating GML data very easy.

➤ EDITABILITY:

Using the many XML editors ,it is also very easy to edit GML data.to add a new feature property or change a property value we just need to adjust a features geometry. These are easily accomplished with a standard XML editor. Unlike many other text based formats however there is no way you can corrupt the data using an XML editor. The editor can be made to ensure that any data which is created or modified complies with the DTD.

It is also not difficult to create a graphical editor for GML and such products are expected to appear on the market within the coming year. Again the GML DTD can be used to ensure data integrity. Note that when one edits GML graphically an intermediate graphic representation is required (perhaps SVG) which is then used to define the geometry of the associated GML feature

➤ INTEGRATION WITH NON-SPATIAL DATA

Binary data structures are typically very difficult to integrate with one another. A classic example is that of associating a text document, or a parameter list, with a separately developed and maintained spatial database of parcels or land tenure boundaries. With a binary data structure one must understand the file structure or database schema and be able to modify it. In many legacy systems using flat files the data structure cannot be modified without breaking the applications which rely on the existing data structure. With GML it is comparatively easy to provide links to other XML data elements and this will dramatically improve with the introduction of XLink and XPointer. Even links to non-XML elements can be readily handled using the well established URI syntax.

➤ TRANSFORMABILITY:

The most important aspect of XML in our view is its transformability. It is quite easy to write a transformation which carries XML data relative to one DTD to XML relative to another. This is exactly what we do when we generate an SVG graphical element stream from a GML data file. Such transformations can be accomplished using a variety of mechanisms including XSLT, Java, Javascript and C++ to name only a few. XSLT in our view is of particular interest. With XSLT it is very easy to write a style sheet which locates and transforms GML elements into other XML elements. XSLT can also make use of powerful searching syntax (XPath/XQL) so as to retrieve elements that satisfy complex boolean expressions on the elements and their attributes. Using these techniques an XSLT style sheet can perform a wide variety of querying, analysis and transformation functions. The transformability of GML also means that we can readily construct application specific indexes or at least we will be able to once XLink and XPointer implementations start to move toward reality.

➤ BEHAVIOUR TRANSPORT:

XML is a language for describing data description languages. GML does not itself encode behaviour. GML can, however, be used in conjunction with languages like Java or C++ to in effect transport geographic behaviour from one place to another. This can be done using a simple object factory which instantiates objects based on received GML data, mapping the GML element names into object classes. In the Java case this would mean mapping the GML elements into Java classes as listed in the OGC Java Simple Features RFC. This "re-hydration" of the GML data then creates Java objects which have the OGC interfaces for Simple Features . GML and Java (or COM or CORBA) Simple Features can thus get along very well with one another. In many applications one only needs the behaviour for a small number of the elements. With this approach one might receive 10,000 GML elements but only need to construct a hundred or so Java objects on an as needed basis.

Feature	Feature Code	Major Code	Minor Code	Category	Condition
Road	RD	11	1100	Highway	Metalled
		11	1300	Motorway	Metalled
		11	5300	Motorway	Unmetalled
		11	6100	Pack-track plains	Unmetalled
		11	6410	Cart-Track Plains	Unmetalled
		11	6500	Foot-path Plains	Unmetalled
		15	3000	Motorway	Metalled
		11	6300	Track follows Stream-bed	Unmetalled
Building	Ntdb:00000001	37	2100	Residential	Block Village/Town
		37	1100	Residential	Hut Temporary
		37	1200	Residential	Hut Permanent
		37	1400	Residential	Hut Oblong Permanent
		37	3020	Religious	Chhatri
		37	3040	Religious	Idgah
		37	3100	Religious	Temple

Table1: Features of GML

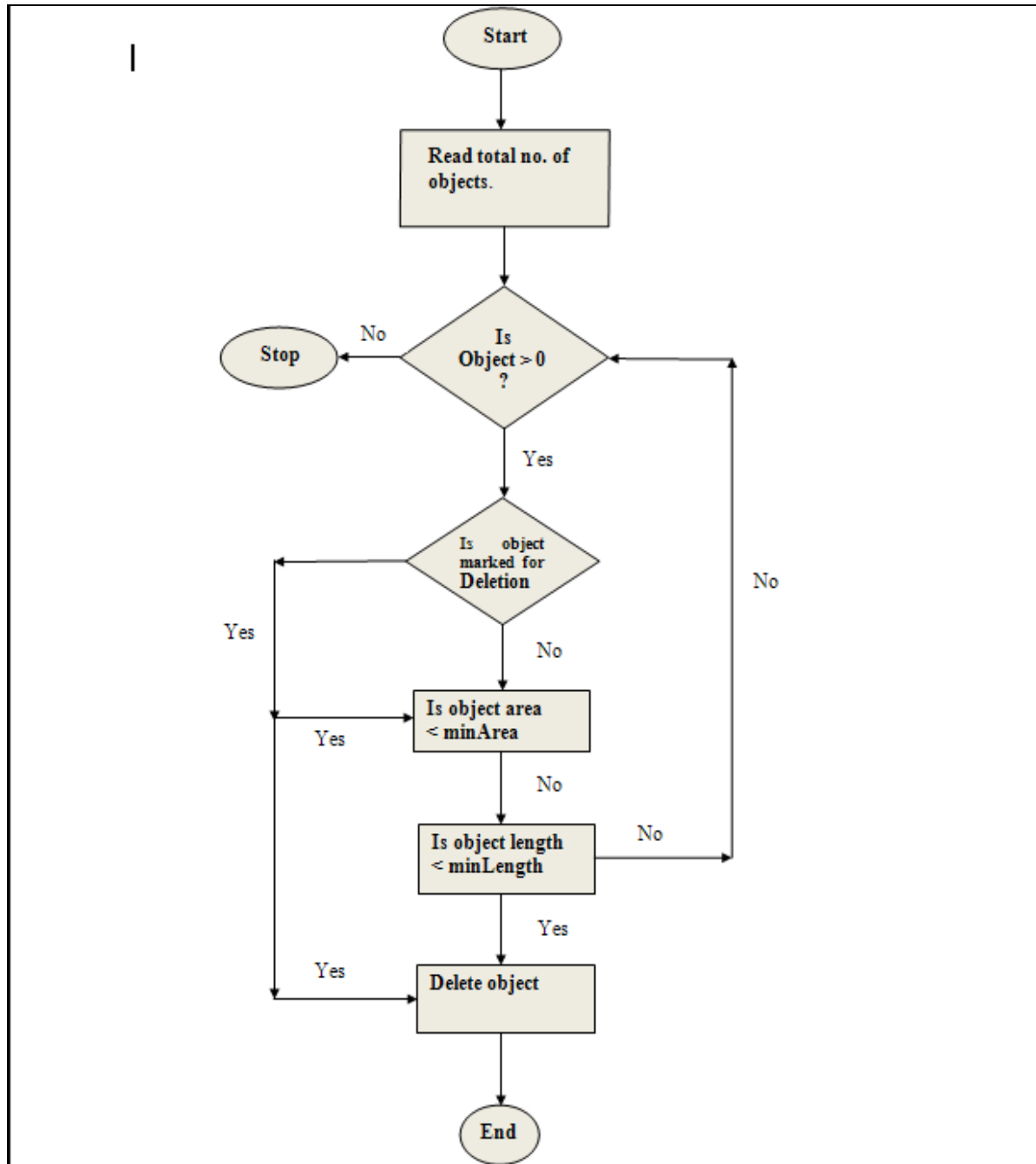
Importing a GML File:

1) GML File after being imported.

SELECTION

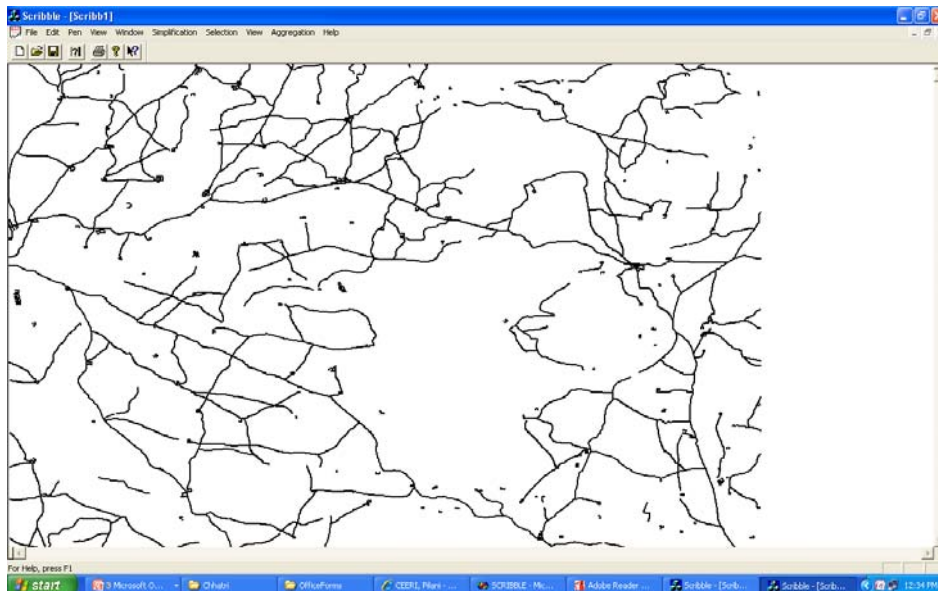
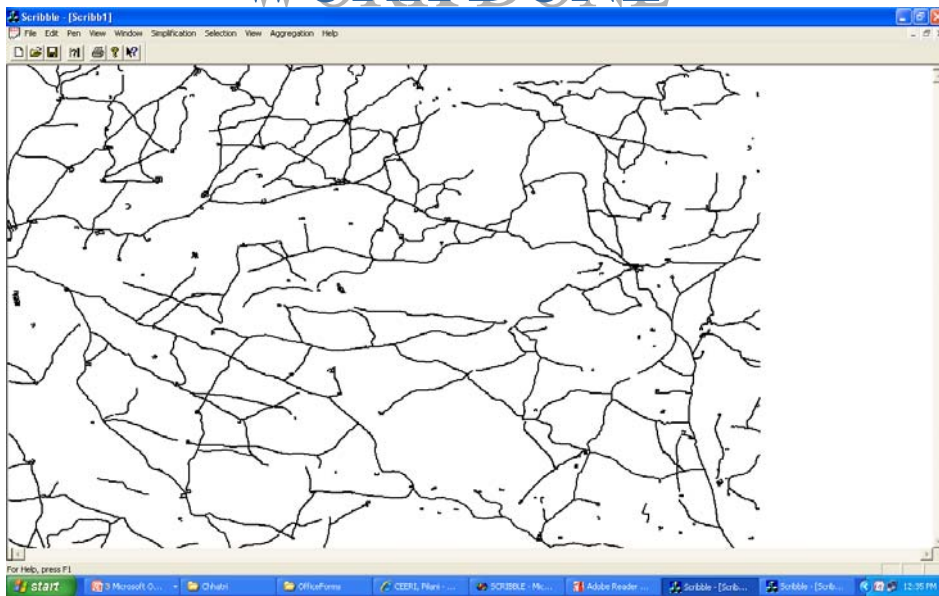
WORK DONE

Implementation of Selection algorithm



Selection Flowchart

WORK DONE

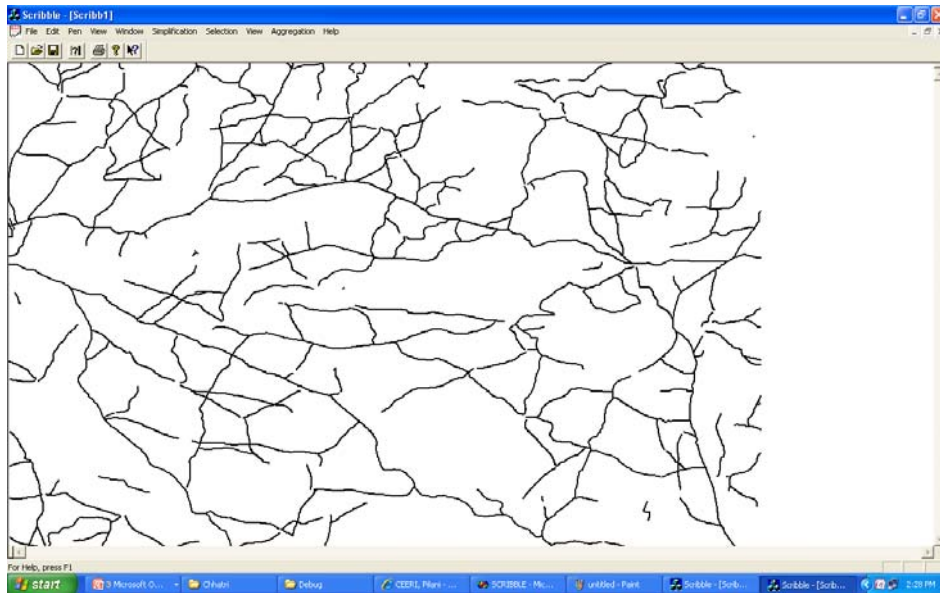
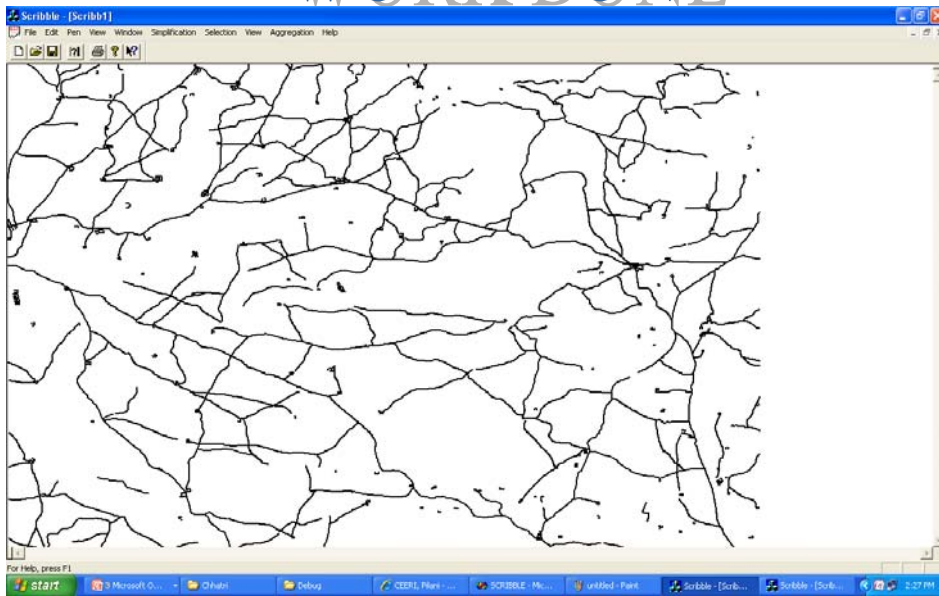


Features Selection

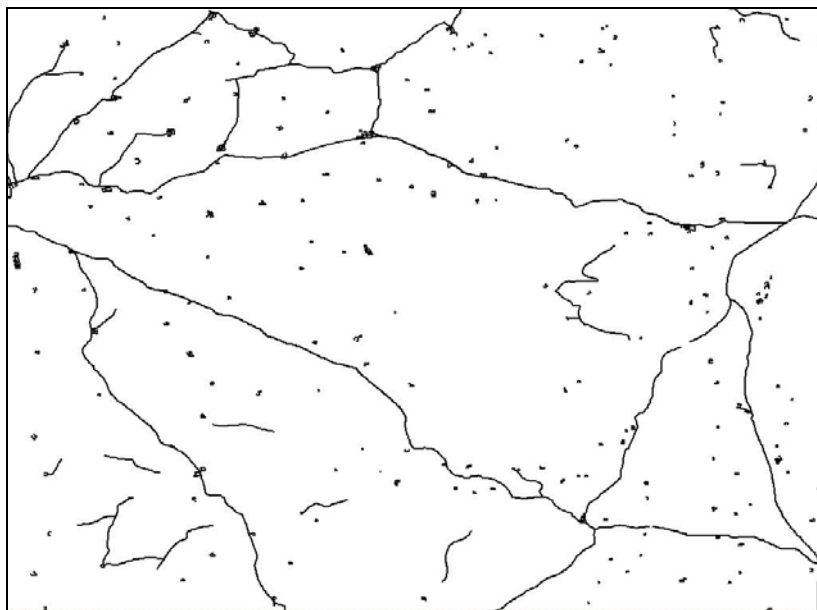
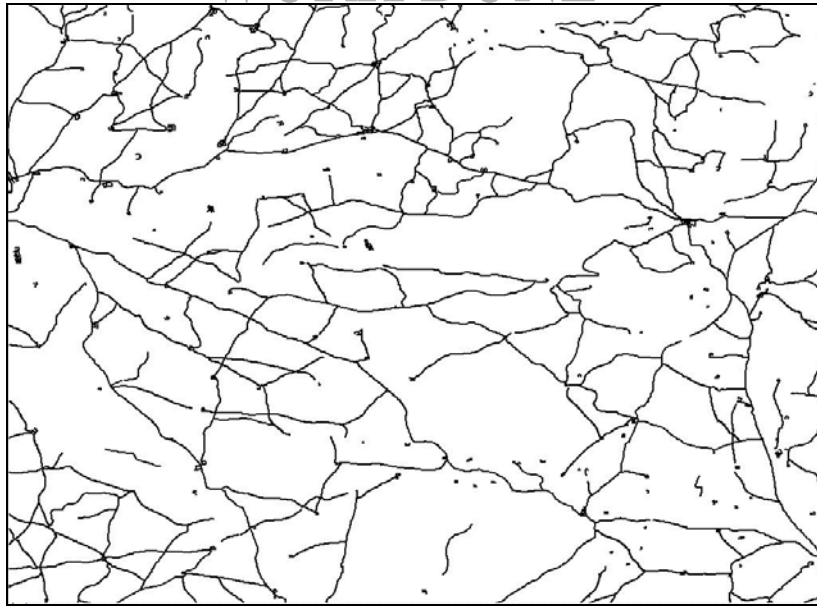
Road Type	Building	Sementic	Spatial
<input checked="" type="checkbox"/> Metalled Highway	<input checked="" type="checkbox"/> Block Village/Town	<input checked="" type="checkbox"/> Chhatri	<input type="checkbox"/> Length
<input checked="" type="checkbox"/> Metalled Motarway	<input checked="" type="checkbox"/> Hut Temporary	<input checked="" type="checkbox"/> Idgah	<input type="checkbox"/> Area
<input checked="" type="checkbox"/> UnMetalled Motorway	<input checked="" type="checkbox"/> Hut Permanent	<input checked="" type="checkbox"/> Temple	<input type="checkbox"/> Length
<input checked="" type="checkbox"/> UnmetalledPackTrackPlains	<input checked="" type="checkbox"/> Hut Oblong Permanent		<input type="checkbox"/> Area
<input checked="" type="checkbox"/> UnmetalledCartTrackPlains			<input type="checkbox"/> Length
<input type="checkbox"/> UnmetalledFootpath			<input type="checkbox"/> Area
<input checked="" type="checkbox"/> Track follows Stream-bed			<input type="checkbox"/> Length

OK
Cancel

WORK DONE



WORK DONE



Features Selection

Road Type	Building	Sementic	Spatial
<input type="checkbox"/> Metalled Highway	<input checked="" type="checkbox"/> Block Village/Town	<input checked="" type="checkbox"/> Chhatri	<input type="checkbox"/> Length
<input type="checkbox"/> Metalled Motorway	<input checked="" type="checkbox"/> Hut Temporary	<input checked="" type="checkbox"/> Idgah	<input type="text" value="1000"/>
<input checked="" type="checkbox"/> UnMetalled Motorway	<input checked="" type="checkbox"/> Hut Permanent	<input checked="" type="checkbox"/> Temple	<input type="checkbox"/> Area
<input checked="" type="checkbox"/> UnmetalledPackTrackPlains	<input checked="" type="checkbox"/> Hut Oblong Permanent		<input type="text" value="1000"/>
<input checked="" type="checkbox"/> UnmetalledCartTrackPlains			
<input checked="" type="checkbox"/> UnmetalledFootpath			
<input checked="" type="checkbox"/> Track follows Stream-bed			

OK
Cancel

Polyline Simplification

Introduction

Often a polyline has too much resolution for an application, such as visual displays of geographic map boundaries or detailed animated figures in games or movies. That is, the points on the polylines representing the object boundaries are too close together for the resolution of the application. For example, in a computer display, successive vertices of the polyline may be displayed at the same screen pixel so that successive edge segments start, stay at, and end at the same displayed point. The whole polyline may even have all its vertices mapped to the same pixel, so that it appears simply as a single point in the display!

For the sake of efficiency, one doesn't want to draw all these degenerate lines, since drawing a single point would be enough. To achieve this, one wants to reduce the vertices and edges of the polyline to essential ones that suffice for the resolution of one's application. There are several algorithms for doing this. In effect, these algorithms approximate a high resolution polyline with a smaller low resolution reduced polyline having fewer vertices. This also speeds up subsequent algorithms, such as area fill or intersection, that may be applied to the reduced polyline or polygon.

Overview

We will consider several different algorithms for reducing the points in a polyline to produce a simplified polyline that approximates the original within a specified tolerance. Most of these algorithms work in any dimension since they only depend on computing the distance between points and lines. Thus, they can be applied to arbitrary 2D or 3D curves that have been approximated by a polyline, for example, by sampling a parametric curve at regular small intervals.

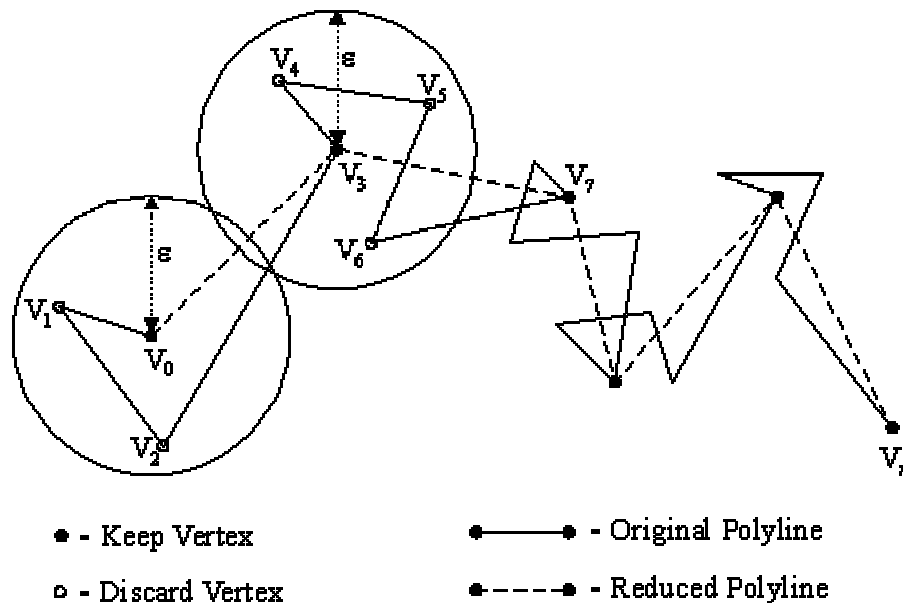
The first simplification algorithm, vertex reduction, is a fast $O(n)$ algorithm. It is the fastest and least complicated algorithm, but gives the coarsest result. However, it can be used as a preprocessing stage before applying other algorithms. This results in a faster combined algorithm since vertex reduction can significantly decrease the number of vertices that remain for input to other simplification algorithms.

The second algorithm is the classical Douglas-Peucker (DP) approximation algorithm that is used extensively for both computer graphics and geographic information systems. There are two variants of this algorithm, the original $O(n^2)$ method [Douglas & Peucker, 1973] and a more recent $O(n \log n)$ one [Hershberger & Snoeyink, 1992]. Unfortunately, as is often the case, the faster algorithm is more complicated to implement. Additionally, it is not as general, and only works for simple 2D planar polylines, and not in higher dimensions.

Vertex Reduction

In vertex reduction, successive vertices that are clustered too closely are reduced to a single vertex. For example, if a polyline is being drawn in a computer display, successive vertices may be drawn at the same pixel if they are closer than some fixed application tolerance. In a large range geographic map display, two vertices of a boundary polyline may be separated by as much as a mile (or more), and still be displayed at the same pixel; and the edge segments joining them are also being drawn at this pixel. One would like to discard the redundant vertices so that successive vertices are separated several pixels, and edge segments are not just points.

Vertex reduction is the brute-force algorithm for polyline simplification. For this algorithm, a polyline vertex is discarded when its distance from a prior initial vertex is less than some minimum tolerance $\epsilon > 0$. Specifically, after fixing an initial vertex V_0 , successive vertices V_i are tested and rejected if they are less than ϵ away from V_0 . But, when a vertex is found that is further away than ϵ , then it is accepted as part of the new simplified polyline, and it also becomes the new initial vertex for further simplification of the polyline. Thus, the resulting edge segments between accepted vertices are larger than the ϵ tolerance. This procedure is easily visualized as follows:

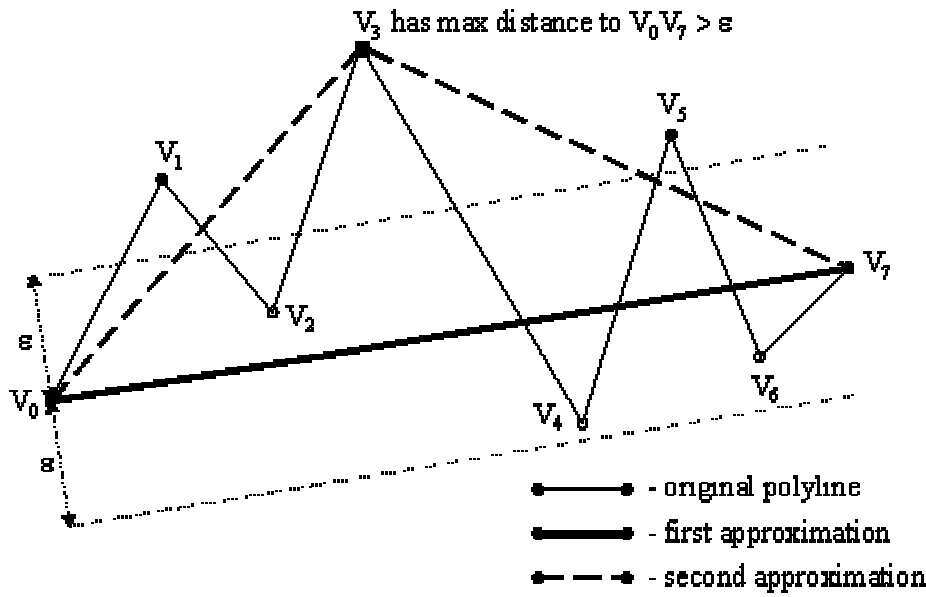


This is a fast $O(n)$ algorithm. It should be implemented comparing squares of distances with the squared tolerance to avoid expensive square root calculations.

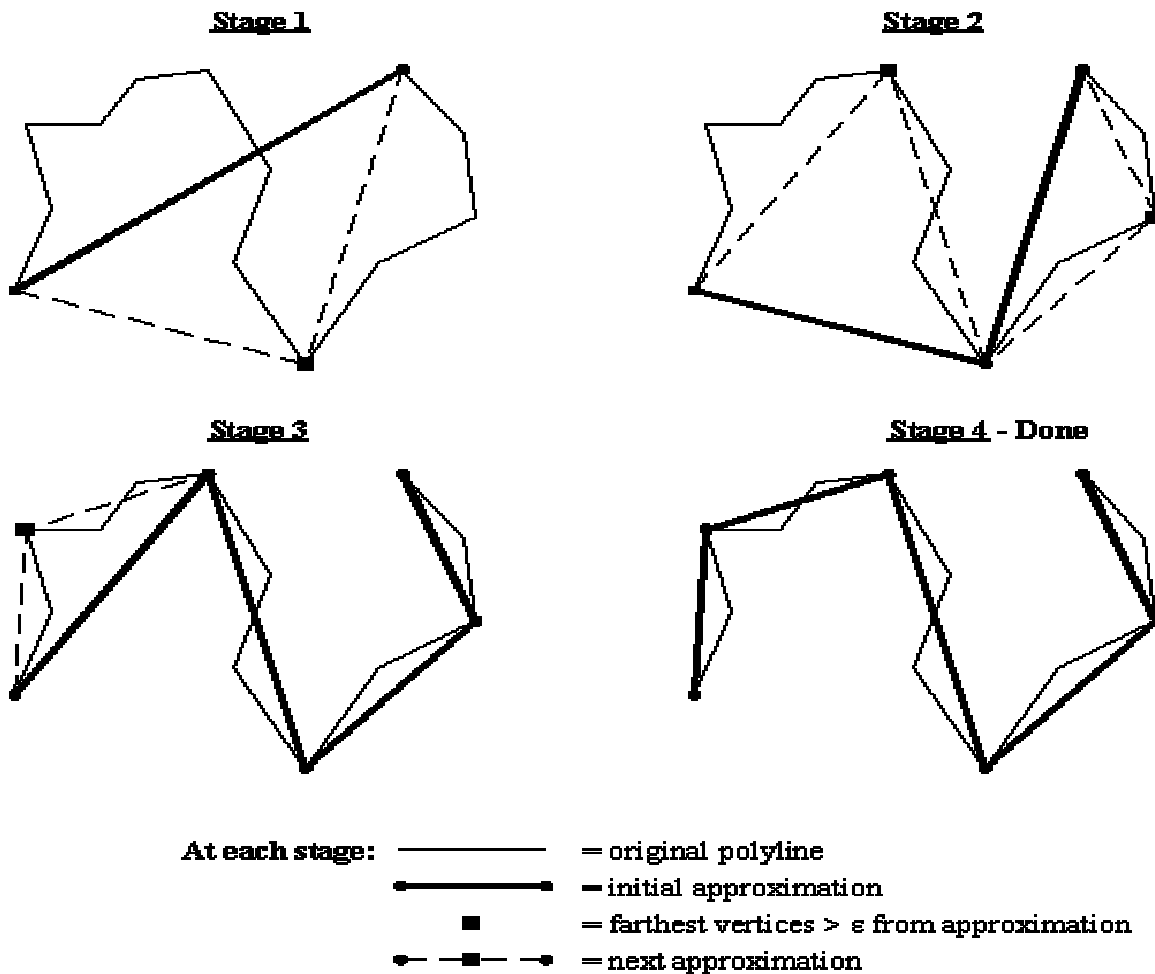
Douglas-Peucker Algorithm

Whereas vertex reduction uses closeness of vertices as a rejection criterion, the Douglas-Peucker (DP) algorithm uses the closeness of a vertex to an edge segment. This algorithm works from the top down by starting with a crude initial guess at a simplified polyline, namely the single edge joining the first and last vertices of the polyline. Then the remaining vertices are tested for closeness to that edge. If there are vertices further than a specified tolerance, $\epsilon > 0$, away from the edge, then the vertex furthest from it is added to the simplification. This creates a new guess for the simplified polyline. Using recursion, this process continues for each edge of the current guess until all vertices of the original polyline are within tolerance of the simplification.

More specifically, in the DP algorithm, the two extreme endpoints of a polyline are connected with a straight line as the initial rough approximation of the polyline. Then, how well it approximates the whole polyline is determined by computing the distances from all intermediate polyline vertices to that (finite) line segment. If all these distances are less than the specified tolerance ϵ , then the approximation is good, the endpoints are retained, and the other vertices are eliminated. However, if any of these distances exceeds the ϵ tolerance, then the approximation is not good enough. In this case, we choose the point that is furthest away as a new vertex subdividing the original polyline into two (shorter) polylines, as illustrated in the following diagram.



This procedure is repeated recursively on these two shorter polylines. If at any time, all of the intermediate distances are less than the ϵ threshold, then all the intermediate points are eliminated. The routine continues until all possible points have been eliminated. Successive stages of this process are shown in the following example.



This algorithm is $O(nm)$ worst case time, and $O(n \log m)$ expected time, where m is the size of the simplified polyline. Note that this is an output dependent algorithm, and will be very fast when m is small.

The first stage of the implementation does vertex reduction on the polyline before invoking the DP algorithm. This results in a fast high-quality polyline approximation/simplification algorithm.

Algorithm

- 1) Calculate the number of objects or buildings on the screen.
- 2) Take the first object and follow the following steps for simplification.
- 3) Follow the steps for VERTEX REDUCTION.
- 4) Call the DOUGLAS PEUKAR recursive Simplification routine.
- 5) The Simplified Polyline is obtained.

Flowchart:-

No

Yes

No

Yes

No

Yes

Yes

No

1) Loading the file for Polyline Simplification keeping the Tolerance =5

2) The file after the Polyline Simplification.

Requirement Specification

I/O Requirement

Key Board : Standard
Mouse : Standard
Monitor : VGA or XVGA

Process Requirement

Processor : Pentium IV processor.
RAM : 512 MB or Above

Software Requirement:

Language Used : Visual C++

Storage Requirement

Hard Disk Drive : 2 GB or Above

Control Requirement

Operating Systems : MS Windows XP

Technology Used

VISUAL C++ 6.0

Introduction:

Microsoft Visual C++ (often abbreviated as MSVC) is a commercial integrated development environment (IDE) product engineered by Microsoft for the C, C++, and C++/CLI programming languages. It has tools for developing and debugging C++ code, especially code written for the Microsoft Windows API, the DirectX API, and the Microsoft .NET Framework.

There are several important features of Visual C++ 6.0 which sets it apart from other languages:

Features

Code editor

Visual C++, like any other IDE, includes a code editor that supports syntax highlighting and code completion using IntelliSense for not only variables, functions and methods but also language constructs like loops and queries. Auto complete suggestions are popped up in a modeless list box, overlaid on top of the code editor. The code editor is used for all supported languages.

The Visual C++ code editor also supports setting bookmarks in code for quick navigation. Other navigational aids include collapsing code blocks and incremental search. The code editor also includes a multi-item clipboard and a task list. The code editor supports code snippets, which are saved templates for repetitive code and can be inserted into code and customized for the project being worked on. A management tool for code snippets is built in as well. These tools are surfaced as floating windows which can be set to automatically hide when unused or docked to the side of the screen.

Visual C++ features background compilation (also called incremental compilation). As code is being written, Visual C++ compiles it in the background with a view to pointing out compilation errors and warnings on-the-fly. Errors are flagged with a red wavy underline and warnings with a green underline. Background compilation does not generate executable code, and needs a different compiler than the one used to generate executable code.

Debugger

Visual C++ includes a debugger that works both as a source-level debugger as well as machine-level debugger. It works with both managed code as well as native code and can be used for debugging applications written in any language supported by Visual Studio.

In addition, it can also attach to running processes and monitor and debug those processes. If source code for the running process is available, it displays the code as it is being run. If source code is not available, it can show the disassembly. The Visual Studio debugger can also create memory dumps as well as load them later for debugging. Multi-threaded programs are also supported. The debugger can be configured to be launched when an application running outside the Visual Studio environment, crashes.

The debugger allows setting breakpoints (which allow execution to be stopped temporarily at a certain position) and watches (which monitor the values of variables as the execution progresses). Breakpoints can be conditional, that is they get triggered when the condition is met. Code can be stepped over, i.e., run one line (of source code) at a time. It can either step into functions to debug inside it, or step over it, i.e., the execution of the function body isn't available for manual inspection. The debugger supports Edit and Continue, i.e., it allows code to be edited as it is being debugged. When debugging, if any variable is hovered over by the mouse pointer, its current value is displayed in a tooltip ("data tooltips"), where it can also be modified if desired. During coding, the Visual Studio debugger lets certain functions be invoked manually from the Immediate tool window. The parameters to the method are supplied at the Immediate win

Extensibility

Visual Studio allows developers to write extensions for Visual Studio to extend its capabilities. These extensions "plug into" Visual Studio and extend its functionality. Extensions come in the form of macros, add-ins, and packages. Macros represent repeatable tasks and actions that developers can record programmatically for saving, replaying, and distributing. Macros, however, cannot be used to implement new commands or create tool windows. Add-Ins provide access to the Visual Studio object model and can interact with the IDE tools. Add-Ins can be used to implement new functionality and can add new tool windows. Add-Ins are plugged in to the IDE via COM and can be created in any COM-compliant languages. Packages are created using the Visual Studio SDK and provide the highest level of extensibility. It is used to create designers and other tools, as well as to integrate other programming languages. The Visual Studio SDK provides both unmanaged as well as a managed API to accomplish these tasks. However, the managed API isn't as comprehensive as the unmanaged one.

CONCLUSION AND FUTURE WORK

CONCLUSION

The algorithms like Selection and Simplification of Polyline have been successfully implemented and checked using the real data provided by the “Survey of India” at Hyderabad.

FUTURE WORK

The Algorithms other than the above implemented algorithms will be implemented too. The algorithms which will be implemented in the near future are as under:

- Simplification of roads (to be continued)
- Simplification of buildings
- Aggregation

Bibliography

PAPERS STUDIED:

- Sidone Christophe and Anne Ruas, “Detecting building alignments for generalization purposes,” Cogit Laboratory-IGN-France,2002.
- Yi-Chen Shao, Liang-Chien Chen, “Automatic Building Outline Reconstruction Using 2D Building Data and Stereo Images,” National Central University, Chung-Li, Taiwan.
- Monika, S., “Generalization on least square methods”, International Archives of Photogrammetry and Remote Sensing. Available at <http://citeseer.ist.psu.edu/Sestergeneralization.htm>
- Kada, M,“Generalisation of building ground planes using half spaces”, Institute for Photogrammetry (ifp), Universität Stuttgart.
- Dan Lee, “ Moving towards new technology for generalization”, ESRI USA.
- Dan Lee, “Geographic and Cartographic Contexts in Generalization”, Development Department, ESRI, Inc.,2004.Available at <http://ica.ign.fr/leicester/paper/Lee-v2-ICAWorkshop.pdf>

WEBSITES EXPLORED:

- http://en.wikipedia.org/wiki/Polygon_triangulation
- <http://www.w3.org/Mobile/posdep/GMLIntroduction.htm>

BOOKS REFERED:

- Mastering Visual C++ Michael J. Young

Annexure-XI

Statement of Expenditure 2017-18, 07
 (to be submitted financial yearwise ie. DOS* to 31st March of that financial year say 20XX, 01-04-20XX till 31.03.20XX+1 year and so on)

Sr No	Sanctioned Heads (II)	Funds Allocated (Indicate sanctioned or revised) (III)	Expenditure Incurred				Total (IV+V+VI+VII)	Balance, if any	Remarks
			1 st Year (DOS to 31 st March next year) (IV)	2 nd Year (1 st April to 31 st March next year) (V)	3 rd Year (1 st April to 31 st March next year) (VI)	4 th Year (1 st April to project completion) (VII)			
1.	Manpower costs	057000.00	0.00	0.00		0.00			
2.	Consumables	066000.00	0.00	16420.00		16420.00			
3.	Travel	050000.00	0.00	12948.00		12948.00			
4.	Contingencies	020000.00	0.00	0.00		0.00			
5.	Others, if any		0.00	0.00		0.00			
6.	Equipment	650000.00	0.00	270749.00		270749.00			
7.	Overhead expenses	157000.00	157000.00	0.00		157000.00			
8.	Total	10000000.00	157000.00	300117.00		457117.00	542883.00		

Amount to be refunded/ reimbursed (whichever is appropriate): Rs

Name and Signature of Principal Investigator: *Rehman 12.11.08*
 Date: 12.11.08

Rehman 12/11/08

Signature of Competent authority/ audit authority:
 Date: _____

* DOS : Date of Start of Project

Annexure-III

UTILISATION CERTIFICATE (2 COPIES)
FOR THE FINANCIAL YEAR - (ENDING 31st MARCH, 2008)

1.	Title of the Project/ Scheme	: Cartographic Generalization of Map Objects
2.	Name of the Institution	: CEERI, PILANI
3.	Principal Investigator	: Dr. J.L. Raheja
4.	Department of Science & Technology sanction order No & date sanctioning the project	: 20 December, 2007
5.	Head of account as given in the original sanction order	:
6.	Amount brought forward from the previous Financial year quoting DST letter no and date in which the authority to carry forward the said amount was given	i. 0.00 ii. NIL iii. NA
7.	Amount received during the financial year 2007-08 (Please give DST letter/order no and date)	i. 10,00,000.00 ii. NRDMS / 11 / 975 / 05 iii. 20.12.07
8.	Total amount that was available for expenditure (excluding commitments) during the financial year (Sr. No. 6+7)	: Rs. 10,00,000
9.	Actual Expenditure (excluding commitments) Incurred during the financial year (upto 31 st March, 2008)	: Rs. 1,57,000.00
10.	Balance amount available at the end of the financial year	: Rs 8,43,000.00

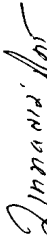
11. Unspent balance refunded, if any (please give details of cheque no etc.): Nil
12. Amount to be carried forward to the next financial year (if applicable): 8,43,000.00

UTILISATION CERTIFICATE

Certified that out of Rs_10,00,000_ of grants-in-aid sanctioned during the year 2007-08_ in favour of Director, CEERI, PILANI under this Ministry/ Department letter/ order No_ NRDMS / 11 / 975 / 05_ dated 20/12/2007_ and Rs_NIL_ on account of unspent balance of the previous year, a sum of Rs_1,57,000.00_ has been utilised for the purpose of_Overhead_ for which it was sanctioned and that the balance of Rs_8,43,000.00_ remaining unutilised ~~at the end of the year has been surrendered to Government (vide Challan no _____ dated _____)~~ will be adjusted towards the grants-in-aid payable during the next year i.e. 2008-09.

 12.11.08

Signature of PI Signature of Registrar/ Signature of Head



Accounts Officer of

the Institute

सि. ए. ए. संशोधन संस्थान

सि. ए. ए. संशोधन संस्थान

पिन-313001

21/11/08

Date 12.11.08 Date

(To be filled in by DST)

Certified that I have satisfied that the conditions on which the grants-in-aid was sanctioned have been fulfilled/ are being fulfilled and that I have exercised the following checks to see that the money was actually utilised for the purpose for which it was sanctioned:-

Kinds of checks exercised.

- 1.
- 2.
- 3.
- 4.
5. Signature: _____

Designation: _____
Date: _____

Equipments List:

Sl. No.	Generic name of the Equipment	Estimated Costs (in Foreign Currency also)*
1.	Pentium –IV- 4 Nos	03.00
2.	UPS – 4Nos	01.00
3	Laser printer (Colour)	01.00
4	Graphics library	02.50
	total	07.50

Sl. No.	Generic name of the Equipment	Estimated Costs (in Foreign Currency also)*
1.	Pentium –IV- 4 Nos	Rs. 2.71 Lakhs (Purchased)
2.	UPS – 4Nos	
3	Laser printer (Colour)	
4	Graphics library	02.50 (Not purchased)
	total	07.50

Equipment Purchased

New Equipments Requirement (without any additional funding):

Sl. No.	Generic name of the Equipment	Estimated Costs (in Foreign Currency also)*
1.	Pentium –IV- 4 Nos	Rs. 2.71 Lakhs (Purchased)
2.	UPS – 4Nos	
3	Laser printer (Colour)	
4	Graphics library-	02.50 (Not purchased)
5	Plotter	01.50 (to be purchased)
6	WorkStation	03.29 (to be purchased)
	total	07.50