Overview of ArcGIS® Topology

An ESRI® White Paper • October 2004

ESRI 380 New York St., Redlands, CA 92373-8100, USA • TEL 909-793-2853 • FAX 909-793-5953 • E-MAIL info@esri.com • WEB www.esri.com

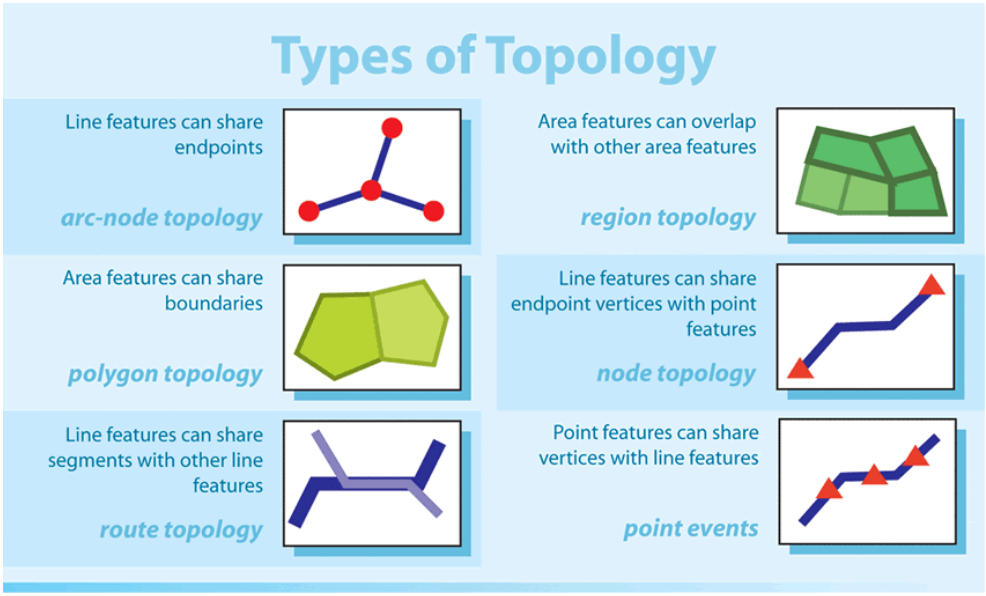
Overview of ArcGIS Topology

An ESRI White Paper

Contents **Page**

Overview of ArcGIS Topology

At the core of geographic information system (GIS) technology lies an essential concept: topology. What is topology? Technically, and in GIS terms, topology defines the spatial relationships between features. It is the "glue" that holds everything together and is based on sets of rules. Topology rules are embedded in spatially intelligent software. As an example, here is an explanation of the commonly used term "arc–node topology." The vertices (*x,y* pairs) define the shape of the arc. The endpoint of each arc is called a node. Every arc has two nodes: a "from" node and a "to" node. Arcs are only joined at nodes. These are the basic topology rules for the arc–node topology type. The following diagram shows a representation and brief explanation of several different topology types.



The ability to store these topological rules and relationships efficiently is one of the essential functions of a GIS product. That is what allows fast processing of large data sets and provides a robust environment or model analyses. In the current generation of comprehensive GIS products, such as ESRI's ArcGIS®, topology rules are integrated as a tool set with an associated set of behaviors.

Topology is employed to

- Manage shared geometry. For example, adjacent polygons, such as parcels, have shared edges; street centerlines and census blocks share geometry; adjacent soil polygons share edges.

- Define and enforce data integrity rules (e.g., no gaps between features, no overlapping features).
- Support topological relationship queries and navigation (e.g., feature adjacency or connectivity).
- Support sophisticated editing tools that enforce the topological constraints of the data model.
- Construct features from unstructured geometry (e.g., construct polygons from lines).

Coverage Topology Storage

ArcInfo® coverage users have a long history and appreciation for the role that topology plays in maintaining the spatial integrity of their data.

In a coverage, the feature boundaries and points are stored in a few main files that are managed and owned by ArcInfo Workstation. The ARC file held the linear or polygon boundary geometry as topological edges, which were referred to as arcs. The LAB file held point locations, which were used as label points for polygons or as individual point features such as for a wells layer. Other files were used to define and persist the topological relationships between each of the edges and the points. For example, one file called the polygon–arc list (PAL) file listed the order and direction of the arcs in each polygon. In ArcInfo, software logic was used to assemble the coordinates for each polygon (from the ARC and PAL files) for display, analysis, and query operations. The polygons were assembled during run time when needed.

The coverage model had numerous advantages.

- It used a simple structure to maintain topology.
- It enabled edges to be digitized and stored once and shared by many features.
- It could represent polygons of enormous size (with thousands of coordinates).
- The topology storage structure was intuitive. Its physical topological files were readily understood by users.

Coverages also had some disadvantages.

- Some operations were slow because many features had to be assembled on the fly.
- Until features (such as polygons, regions, and multipart lines or routes) had been constructed, the coverage was not ready to use. If edges were edited, the topology had to be rebuilt. (Note: Partial processing was eventually used, which required updating only the changed portions of the coverage.)
- Only a single user could update a topology at a time. Users would tile their coverages and maintain a tiled database for editing.

Shapefiles and Simple Geometry Storage

In the early 1980s, coverages were seen as a major improvement over the older polygon- and line-based systems in which polygons were held as complete loops. All the coordinates for a feature were stored in each feature's geometry. Before the coverage and ArcInfo came along, these simple polygon and line structures were used. They were

simple but had the disadvantage of double-digitized boundaries (i.e., two copies of the coordinates of adjacent polygons with shared edges would be contained in each polygon's geometry). The main disadvantage was that GIS software at the time could not maintain shared edge integrity. In addition, storage costs were enormous, and each byte of storage came at a premium. During the early 1980s, a 300 MB disk drive was the size of a washing machine and cost $30,000. Holding two or more representations of coordinates was expensive. Thus, the use of a coverage topology had real advantages.

During the mid 1990s, interest in simple geometric structures grew because disk storage and hardware costs were coming down, GIS data sets were more readily available, and the work of GIS users was evolving from primarily data compilation activities to include data use and sharing.

Users wanted faster performance for data use (e.g., no need to derive polygon geometries when we need them; just give us the coordinates of these 750 polygons as fast as possible). Thousands of GIS systems were in use, and numerous data sets were readily available. Maintaining integrity of spatial data was now possible and more reasonable.

Around this time, ESRI developed and published its ESRI® shapefile format. Shapefiles used a very simple storage model for feature coordinates. Each shapefile represented one type of feature (point, line, or polygon) and contained a simple file of each feature along with a well-defined and simple storage model for the feature's coordinates. A few years later, ArcSDE® pioneered the same simple storage model in relational databases. ArcSDE is an advanced data server, providing a gateway for storing, managing, and accessing spatial data in any of several commercial database management systems. We were beginning to see the real possibility of leveraging the relational database management system (RDBMS) for GIS data management.

Shapefiles became ubiquitous, and using ArcSDE, this "simple features" mechanism became the fundamental feature storage model in RDBMSs. ESRI was the lead author of the Open Geospatial Consortium, Inc. (OGC), and ISO Simple Features Specification because we recognized the significance of this simple storage mechanism.

It had clear advantages.

- The complete geometry for each feature was held in one record. No assembly was required.
- The data structure (physical schema) was simple, fast, and scalable.
- It was easy for programmers to write interfaces.
- It was interoperable. Many programmers wrote simple converters to move data in and out of these simple geometries from numerous other formats. Shapefiles were widely applied as a data use format.

Its disadvantages were that maintaining the data integrity that could be provided by topology was not easy. As a consequence, users applied one data model (such as coverages) for editing and maintenance and another for deployment and use (such as shapefiles or ArcSDE layers).

Users would edit their data in coverages, computer-aided design files, and so forth. Then they would convert their data into shapefiles for use. Thus, even though the simple features structure was an excellent direct use format, it was not readily edited and maintained using shared geometry. Direct use databases would become out of date and have to be refreshed. They worked, but there was a real lag time for information update. Topology was missing.

What GIS required and what the geodatabase topology model implements now is a mechanism that stores features using the simple features geometry but enables topologies to be used on this simple, open data structure. This means that users can have the best of both worlds—a transactional data model that enables editing, rich data modeling, and data integrity and a data storage mechanism that employs simple feature geometry. This direct use data model is fast, simple, and efficient and can now be directly edited and maintained. With the release of ArcGIS 9, users are now discovering the advantages of this new data model and the significance of the topology implementation.

Geodatabase Topology Storage

Recall in the coverage model that when a polygon or a multipart line was needed, its geometry would be assembled on the fly from its constituent parts (from the ARC, LAB, and PAL files). For example, users would get the coordinates for each polygon by looking up the coordinates from each arc and assembling them according to the ordered list in PAL. The ArcInfo software was used to perform this topological operation on the coverage's topological data structure.

At ESRI, we investigated a number of alternative implementations of the logical topology model and software that would support this functionality. It quickly became apparent that a kind of reverse process was possible during shapefile editing in which features would be decomposed into their topological elements of edges, faces, and nodes. In other words, deriving topological elements—such as nodes, edges, and faces—on the fly was just as reasonable an approach as on-the-fly feature geometry assembly was in the coverage days.

This required a few key ingredients.

- A persisted topological index on the coordinates of the simple features
- Advanced software logic that knew how to discover the topological elements in the feature geometry
- An editing and data management framework that could maintain topological integrity
- A data management framework that enabled users to define the integrity rules and topological behavior of their feature classes
- Tools to validate, discover, identify, and resolve geometric errors in the features that were discovered through a topological analysis of the information

In addition, the software and database schemas had to scale to support enormous data volumes and many simultaneous editors.

An Evolved Solution

In effect, topology must be considered as a complete data model (objects, integrity rules, editing and validation tools, a topology and geometry engine that would process data sets of any size and complexity, and topological operators) as well as the storage format or set of record types.

A complete GIS is not just a data structure but also includes the tools, application logic, and software that enables a data structure to be used to accomplish real work.

We wanted to find the mechanism that would

- Support massively large databases of millions of features.
- Perform editing and maintenance by many simultaneous editors.
- Support feature geometry that was always ready to use and always available.
- Support topological integrity and behavior.
- Go fast and scale for many users and many edits.
- Be flexible and simple.
- Leverage RDBMS.

After testing numerous implementations, we made the pragmatic decision to use simple features storage for all features in the geodatabase and to index the shared coordinates on each feature using a topological index. In ArcSDE and the geodatabase, we already had experience using indexes and software logic to accomplish many operations on the simple features storage model. We knew the clear advantages of direct use data formats that would scale and be multiuser.

In the case of topology, we implemented a specialized coordinate index in which shared coordinates between features and across feature classes were the same in each feature's geometry and a topological index allowed us to build relationships between shared coordinates. These coordinates could be stored as part of each feature's simple geometry, and the topological indexes on the coordinates would enable each shared coordinate to be used as foreign keys to other features. In other words, we could readily define and use the relationships between shared coordinates for performing topological updates.

During editing and update, as features are added they are directly usable. At any time, users can choose to validate the updated areas, which are tracked as updates are made to each feature class.

What this meant was that topological primitives (e.g., nodes, edges, and faces) and their relationships to one another could be efficiently discovered and assembled using the coordinates and their topology indexes in each feature class. In the coverage, we calculated the feature geometry on the fly from the topological elements when it was needed. In this case, we could reverse the process and calculate the topological primitives on the fly when they are needed along with their relationships to other features.

This has several advantages.

- Simple feature geometry storage is used, which is efficient and scales to large sizes and numbers of users.

- This physical data model is transactional and is multiuser. By contrast, the older topological storage models will not scale and are extremely difficult to make

transactional for multiple editors and numerous other GIS data management work flows. This means that geodatabase topologies need not be tiled, many users can update the topological database simultaneously, and it can grow to any size (millions of features).

- This topology implementation is additive. Users can typically add it to an existing schema of related feature classes instead of redefining and converting all of their existing feature classes to new data types containing topological primitives.
- There is only one data model for geometry editing and data use, not two or more.
- Topology implementation is interoperable because all feature geometry storage adheres to simple features specifications from OGC and ISO.
- Data modeling is more natural because it is based on user features (such as parcels, streets, soil types, and watersheds) instead of topological primitives (such as nodes, edges, and faces). Users will begin to think about the integrity rules and behavior of their actual features instead of the topological primitives. For example, how do parcels behave? This will enable stronger modeling for all kinds of polygon and line features. It will improve our thinking about streets, soil types, census units, watersheds, rail systems, geology, forest stands, and so on.
- In cases in which users want to store the topological primitives, it is easy to create and post topologies and their relationships to tables for various analytical and interoperability purposes (such as wanting to post these features in an Oracle Spatial warehouse that stores tables of topology primitives).

At a pragmatic level, the ArcGIS topology implementation works. It scales to extremely large geodatabases and multiuser systems without loss of performance. It includes rich validation and editing tools for building and maintaining topologies in geodatabases. It includes rich and flexible data modeling tools that enable users to assemble practical, working systems on any relational database.

GIS requires more than a storage model. It requires a complete solution. A GIS also requires rich tools for data definition to define the integrity rules and schema of the information. A GIS requires tools for editing, mapping, analysis, management, and geoprocessing. GIS application logic must be included for analyzing and validating feature geometry as well as for applying logic to perform GIS transactions and services. The application logic needs to reside in many possible locations in an enterprise GIS. Thus, GIS software must be deployable in desktops and central GIS servers and embedded in custom or focused applications. It must be available through industry-standard programming languages (APIs such as .NET, Java, and C++).

ArcGIS was built to provide a complete solution for geographic information management and, as such, is a complete system for managing topology in geographic databases. It builds on the best of coverages, shapefiles, and simple feature geometries and is a pragmatic and scalable implementation. We have tested this topology model in large GIS implementations with large numbers of users. We know that this implementation is robust, scalable, and workable.