Agenda for the 90's

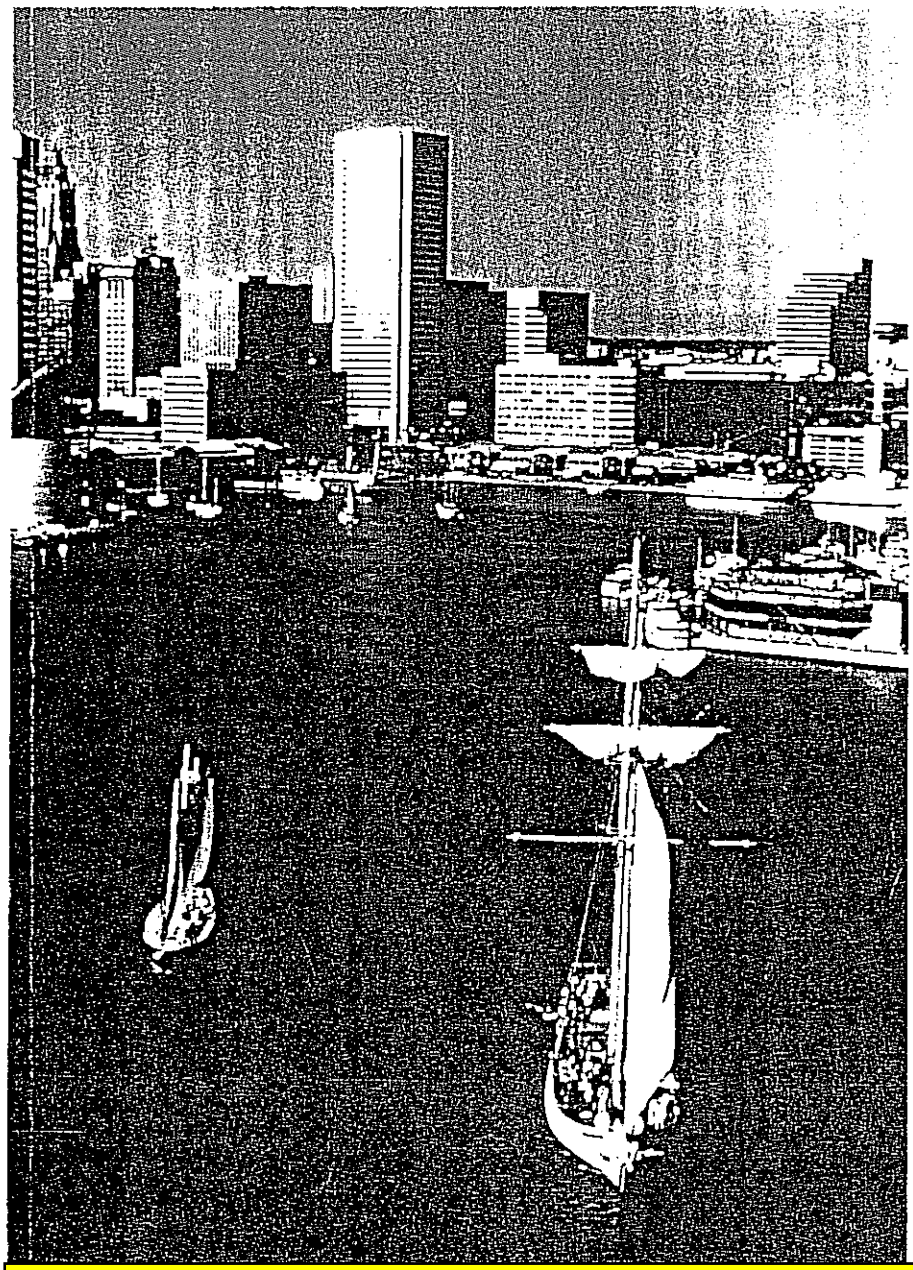
ASPRS/ACSM 1989 AGENDA FOR THE 90'S
APRIL 2-7, 1989/BALTIMORE

TECHNICAL PAPERS

1989 ASPRS/ACSM ANNUAL CONVENTION

SURVEYING & CARTOGRAPHY

VOLUME 5



1934 American Society for Photogrammetry and Remote Sensing

American Congress on Surveying and Mapping 1941

An Object-Oriented Carto-Graphic Output Package[*]

David R. Steiner
Max J. Egenhofer
Andrew U. Frank
National Center for Geographic Information and Analysis
and
Department of Surveying Engineering
University of Maine
Orono, ME 04469, USA
STEINER@MECAN1.bitnet
MAX@MECAN1.bitnet
FRANK@MECAN1.bitnet

Abstract

An object-oriented cartographic output package is proposed as a solution to the shortcomings of existing procedural graphics packages for the display of query responses to Geographic Information Systems. This paper compares the object-oriented approach with current procedural approaches for computer cartography. The concepts of data abstraction and the application of combinatoral topology to modelling graphic objects, fundamental to an object-oriented treatment of geometry, are discussed. Spatial objects can be represented by a variety of different symbols depending on the context of the map. The object-oriented approach facilitates this variation of symbology. The design and implementation of a prototype cartographic output package based on this approach are introduced.

1 Introduction

Object-oriented data modeling and database management are currently being examined as approaches to the design of the next generation of Geographic Information Systems (GIS) [Frank 1988]. A crucial component of such a spatial information system is an object-oriented graphics module which is used to present query results as maps on the display.

Existing graphics packages, such as GKS and CORE, are not satisfactory for cartographic output. These packages are intended to serve multiple types of applications, e.g. CAD/CAM, business graphics, statistics, etc. Due to this generality, there is a mismatch between the functionality supplied by these packages and what is desirable for a cartographic output system. Such mismatch can be seen in the different use of coordinate systems in business graphics and cartography.

Specific shortcomings of commericial graphic packages used for cartographic output are the use of Normalized Device Coordinates and independent scales in X and Y direction. The coordinate system of Normalized Device Coordinate assumes a viewport coordinate system with an origin in the lower left corner and ranging from 0 to 1 along both axes. This facilitates the use of different display hardware but causes difficulties when defining standard symbols.

Current packages supply the ability to specify separate X and Y scales when transforming objects from window to viewport or display device coordinates. If the user specifies a window

[*]This research was partially funded by grants from NSF under No. IST 86-09123 and Digital Equipment Corporation. David Steiner's work was supported by an Undergraduate Research Experience grant. The support from NSF for the NCGIA under grant number SES 88-10917 is gratefully acknowledged.

whose proportions do not match those of the viewport, the transformation will stretch the window more in one direction than the other so that it will fill the entire veiwport, which will produce a distorted map.

Current graphics packages are designed in a procedural style rather than descriptive. Procedural routines abstract an operation (e.g., draw line) in a single command. In a descriptive approach, users "describe" what they want done and do not have to specify, or even know, how it is to be accomplished. Thus, to draw a box, which is an object, programmers define the box and then apply the generic command "draw" to this box.

Object-oriented design provides a descriptive approach in which objects are defined that have a set of operations on them. All graphic objects have the operations "draw" and "erase." This allows several distinct advantages: device independence, generic operations, simple interfaces, information hiding, and dimension independence. This means that the user need not be concerned with whether the object is represented by a point, line, or area, or what low level drawing commands are needed to display the object on the screen since this information is part of the object. When the user wishes to display the object he or she merely issues the command "draw object" and specifies which object is to be displayed.

A desirable feature for cartographic output that is lacking in current graphics packages is a facility for the definition and composition of symbols. Since we often represent complex objects as symbols on a map, such as a building represented by a rectangle, there is a need to define a set of standard symbols to be used for these representations. This is difficult to accomplish with procedural routines that may require several procedure calls to draw the object. What we desire is to be able to request that an object be displayed and have the appropriate standard symbol assigned to it based on the current context.

To date, little effort has been made in the area of the design of an object-oriented package for representing the data graphically to the users of GIS. We propose an object-oriented graphics package, tailored to the specific needs of cartographic output. This cartographic package will overcome some of the major weaknesses of current graphics packages as applied to traditional cartographic output. The underlying spatial data model for this package is based on combinatorial topology in which simplices are used to describe the minimal objects in a dimension. More complex objects are described as aggregates of simplices, called simplicial complexes.

This paper will discuss object-oriented data representation and will provide a brief overview of some of the concepts of combinatorial topology. Next a proposed design for an object-oriented cartographic package is introduced. Finally, there will be a discussion of the implementation of a prototype of this package which is being developed on top of the VAX User Interface System.

2 Object-Oriented Data Representation

A program for cartographic output deals with concepts of points, lines, and areas, the object-oriented concept constructs a framework for these concepts and shows their linkages. Data abstraction is a method of modeling data. Egenhofer discusses the application of the abstraction methods [Brodie 1984] of classification, generalization, and inheritance to spatial objects [Egenhofer 1988a]. This discussion is summarized below. Combinatorial topology provides a means to apply these abstractions to the modeling of geometry in an object-oriented environment.

2.1 Abstraction Mechanisms

Classification is the mapping of several objects to a common class. The class characterizes the behavior of its member objects, or instances, by describing the valid operations upon them [O'Brien 1986]. All instances of a class are described by the same properties and have the same operations. For example, the building at 14 Maple St. is an instance of the class residential building and, as such, the class operations of "occupant of" and "neighbor of" apply to it.

Generalization provides different views in several different levels of detail for a class of objects. A number of classes sharing common operations are grouped into a more general superclass [Dahl 1966] [Goldberg 1983]. Subclasses and superclasses are related by an "is_a" relationship. For example, residence is a building, therefore, it is a subclass of building, while building is a superclass of residence. All operations on a superclass apply to instances of a subclass; however, the reverse is not true.

Hierarchical or single inheritance is an idealized model that implies that a class can only inherit properties from a single superclass (Figure 1). This model very often fails when applied to the real world.

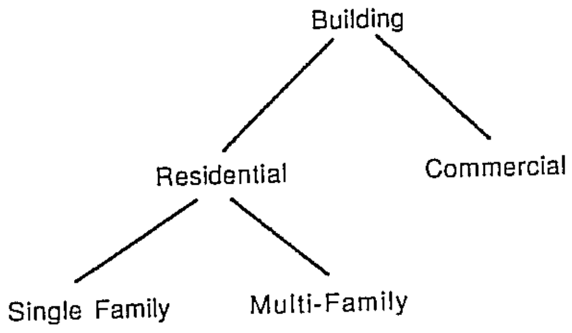


Figure 1: Hierarchical Inheritance.

Multiple inheritance [Cardelli 1984] allows classes to inherit properties from more than one superclass. It is this model which applies to spatial objects which inherit properties from their geometry and from non-spatial attributes, or lexical data (Figure 2). Examples of non-spatial or lexical properties are the type of crop grown on a parcel of farm land or the street address of a building; geometric properties are the corner coordinates of a parcel or the dimensions of a building.

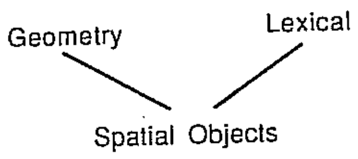


Figure 2: Multiple Inheritance in Spatial Objects.

Aggregation is the combination of several objects to form a higher-level object with its own functionality. This differs from generalization in that operations on the aggregate are not necessarily compatible with operations on the individual parts. For example, a building is an aggregation of walls, a roof, windows, doors, etc.

2.2 Modeling of Geometry in an Object-Oriented Environment

The theory of combinatorial topology provides a method to classify and formally describe sets of points. The use of combinatorial topology for modeling spatial data in GIS has been proposed [Frank 1986] and implemented [Egenhofer 1987] [Jackson 1989]. The following is a brief summary of simplices, simplicial complexes and the operations of boundary and interior.

2.2.1 Simplex

For each spatial dimension there is a minimal object which is called a simplex. The model can be extended to any n-dimensional space, where an n-simplex is the minimal object and is made up of (n+1) simplices of dimension (n-1) which are geometrically independent. For example:

- A point, a minimal 0-dimension object, is a 0-simplex.
- An edge is a 1-simplex and is bounded by two 0-simplices.
- A triangle is a 2-simplex bounded by three 1-simplices, etc.

A face is a bounding simplex of another simplex, i.e., a point is a face of an edge.

2.2.2 Simplicial Complex

A simplicial complex is defined as a finite collection of simplices where the intersection of any two is a face of both (Figure 3). This definition excludes ovelapping of simplices, for example, two 1-simplices (lines) which cross each other without a 0-simplex at the intersection. The operations on simplicial complexes include boundary and its converse, interior. A boundary operation on a complex returns the set of bounding simplices of the complex, and interior returns the set of all simplices not included in the boundary. Applying boundary twice to an object yields the empty set.

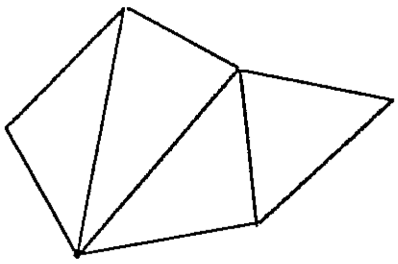


Figure 3: Four 2-simplices which intersect in three 1-simplices.

3 Object-Oriented Graphics

Object-oriented design is a method in which the software architecture is centered around objects involved rather than based on operations [Meyer 1988]. Geometry is a property of all spatial objects, and all spatial objects can be represented by graphic objects. The most abstract graphic object, which we will call the *dispObject*, requires only two high level operations. These operations on dispObjects are *draw* and *erase*. Any object to be displayed, therefore, is an instance of a subclass of dispObject and inherits the *draw* and *erase* operations from dispObject. The operations on specific objects in the subclasses are implemented specifically for those objects.

3.1 Selection of Symbology

In cartography, objects can be represented in a variety of ways depending on the context of the presentation. The object's inherent geometry or shape does not change, but the symbols used can be different from map to map.

These symbol changes can be based on several factors. One such factor is the type of map being drawn. For example, utilities such as water mains can be represented on one map schematically by single lines for pipes and symbols for valves and fittings. The same utility system can also be drawn on a map with the pipes and fittings drawn to scale (as in a construction map) (Figure 4).

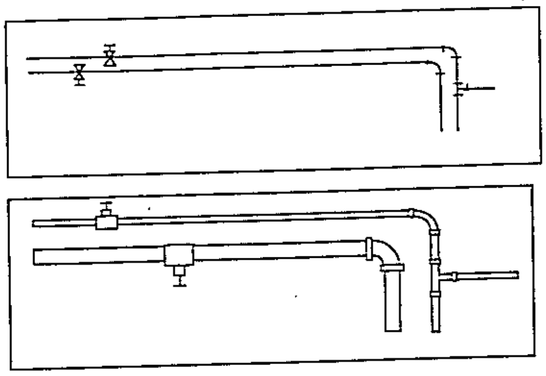


Figure 4: Variations in Representation.

Another factor affecting representation is the scale of the map. On a small scale topographic map all buildings can be shown with the square building symbol described above. On a larger scale map this representation is usually inappropriate. Users want to differentiate between buildings of different sizes and shapes by the use of various sizes of rectangles.

Information about the attributes of an objects can also be represented on a map by varying the properties of a symbol, such as the color, pattern, line weight, and intensity. Bertin provides clarification of how these variations are accomplished graphically [Bertin 1983]. These variations are discussed below.

3.1.1 Color

Color is an attractive, as well as an effective means of differentiating between objects with different properties. Representing primary roads as red lines and secondary roads as black lines is an example of this technique.

3.1.2 Pattern

Various patterns, such as hatching or cross-hatching for areas or solid vs. dashed lines, are often used to differentiate between objects and object classes. For example, various patterns can be used to show vegetation cover types on different parcels of land.

3.1.3 Line Weight

If all lines on a map are drawn with the same weight and color, interpretation of the information presented is extremely difficult. To avoid confusion, a cartographer distinguishes, for example, between boundary lines and building outlines on the map by drawing the boundaries in a lighter line weight.

3.1.4 Intensity

Unlike traditional cartographic products, which are static, interactive maps provide a dynamic representation. This includes the possibility that objects displayed on the screen can be emphasized by choosing an outstanding representation. In particular, the selection of objects by clicking on them with a pointing device, such as a mouse, requires some technique to distinguish the chosen object from the others on the screen. A common means of doing this is by highlighting the object on the screen by increasing its intensity. An object shown in green on the screen could be redrawn in a brighter shade of green when picked by the user. If a highlighting color cannot be used because it matches a color already used by another object, other methods, such as reverse video or causing the object to blink, could be substituted.

All these different methods of representation can be used in combination, a blue hatched area representing one class and a red hatched area representing another. They should also be user definable, one user choosing red rectangle for a class of objects while another chooses black.

3.2 Representation of Object Parts

If we apply the theory of combinatorial topology, we see that the geometry of objects can be represented by simplical complexes and that it is sometimes desirable to represent the various parts of the object in different manners. A linear object, such as a road, consists of a set of 1-simplices, also called edges, that are bounded by two or more 0-simplices, also called nodes, and are connected to at least one other 1-simplex. The nodes are of two types:

- Interior nodes, which connect two or more edges.
- Bounding nodes, which belong to only one edge and thus do not connect edges.

Figure 5 shows a linear object with 2 interior nodes and 3 bounding nodes.

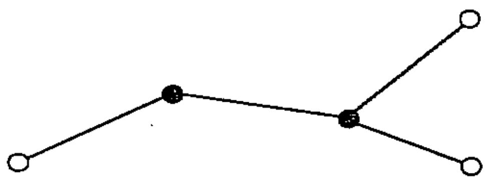


Figure 5: The interior and bounding nodes of a linear object.

When we display such an object, the various subparts–the line itself, its bounding nodes and its inner nodes–can be described individually, each with potentially different values. For example, the line in black with blue bounding nodes and red interior nodes.

This model can be extended to area and volume representations of objects, in each case considering the object itself, its inner components and its boundary components [Egenhofer 1988a]. Points, which are 0-simplices, are minimal objects and have no subparts.

4 Design of an Object-Oriented Cartographic Package

Using the model based on simplical complexes, we can now specify the objects and operations on them that are necessary for object-oriented graphics. The most general object is called a dispObject, which allows the following operations[1].

```
drawObject:   dispObject -->  .
eraseObject:  dispObject -->  .
```

[1] The operations are specified in the following form:
`operationName: parameter`$_1$ `x ... x parameter`$_n$ `--> result` and is followed by an informal description of the effect of the operation if necessary.

The "erase" operation is the inverse of the "draw" operation and has no effect if the object has not already been drawn.

These operations act on display objects that contain information on how each individual object part will be displayed. In the procedural approach, the programmer must keep track of the state of the display device; what color was last used, what line style was chosen, etc. Before the next object is drawn these parameters must be reset if necessary by issuing a series of commands. In the object-oriented approach, no previous state is assumed and the single command to draw the uses the information contained in the object itself to set the display's state.

4.1 Display Objects

Display objects contain information about which particular object in the database it represents, in the form of a reference to this object, and the display object's location on the screen, which is calculated from the object's world location using the current map projection. In addition, each display object consists of a number of subparts, each with its own display specification (*dispSpec*). The number of subparts that an object has is dependent on the spatial dimension of the object. For example, a linear object has a spatial dimension of 1 and contains three specifications for 3 subparts: the line itself, the bounding nodes, and the interior nodes. An area object has 5 subparts: the area itself, the bounding and interior edges, and the bounding and interior nodes.

It is now necessary to specify a series of operations to set the display specifications of the subparts of the object. The operations can be grouped into three classes: operations on the specification for the object itself, operations on the bounding parts, and operations on the interior parts. These operations are as follows:

```
dispObjectInitialize:   object x dimension                --> dispObject
putObjectDisp:          dispObject x dispSpec             --> dispObject
putBoundaryDisp:        dispObject x partDimen x dispSpec --> dispObject
putInteriorDisp:        dispObject x partDimen x dispSpec --> dispObject
getObjectDisp:          dispObject                        --> dispSpec
getBoundaryDisp:        dispObject x partDimen            --> dispSpec
getInteriorDisp:        dispObject x partDimen            --> dispSpec
displayObject:          dispObject                        --> .
```

In the boundary and interior operations, the dimension of the part indicates which part of the boundary or the interior is being referenced. Thus we would specify the boundary nodes' display in a linear object by using the putBoundaryDisp and passing 0 as the partDimen along with the dispObject and the dispSpec.

4.2 Display Specifications

The display specifications for each part are generally the same. Each part of the object, regardless of its dimension, has a display specification that contains information about which graphical symbol should be used to represent the object part, its scale and rotation, and whether the symbol is highlighted or not.

It should be noted that the scale and rotation mentioned here are applied to the symbol and not to the object itself. This is necessary because, if we wish to zoom in on a portion of the map being displayed, we want the symbols to be drawn larger (in a stepwise fashion) as the display scale decreases. The rotation is applied if we wish to align symbols to one another, for example, rectangles representing buildings aligned with the road they are next to.

The operations on dispSpecType are defined as follows:

```
dispInitialize:                          --> dispSpec
dispPutIntensity: dispSpec x intensity  --> dispSpec
```

```
dispPutScale:      dispSpec x scale       --> dispSpec
dispPutRotation:   dispSpec x rotation    --> dispSpec
dispGetIntensity:  dispSpec               --> intensity
dispGetScale:      dispSpec               --> scale
dispGetRotation:   dispSpec               --> rotation
```

5 Implementation of the Package

This package was implemented in Precompiled Pascal [Egenhofer 1988b] and used graphic routines supplied by the VAX User Interface System (UIS) at the lowest level. The following paragraph discusses the module dealing with display objects.

5.1 Display Objects

The most abstract object dealt with by this package is the display object. The types defined in the module for display objects are:

```
dispSpecType = RECORD
                 intensity: intensType;
                 scale: real;           {scale factor for symbol}
                 rotation: angle;      {rotation angle for symbol}
                 symbol: symbolType;
               END;
dispObjectType = RECORD
                   objectId: objectIDType;
                   location: ptType;
                   CASE Dim: dimensionType of
                     0: (nodeDisp: dispSpecType);          {point}
                     1: (edgeDisp,                          {line}
                         boundaryNodeDisp,
                         interiorNodeDisp: dispSpecType);
                     2: (areaDisp,                          {area}
                         interiorEdgeDisp,
                         boundaryEdgeDisp,
                         boundaryNodeDisp,
                         interiorNodeDisp: dispSpecType);
                   END;
                 END;
```

In the above definitions, *intensType* is an enummerated type which indicates whether the object part is highlighted or not. *objectIDType* is a unique identifier assigned to each object in the database. The *dimensiontype* indicates whether the object is a point, line, or area and determines the number of object parts in the variant portion of the record. Finally, *symbolType* contains information on which specific symbol will be used to represent that object part. This module contains the routines that perform the operations that were defined for *dispObject* in section 4.1, as well as *drawObject* and *eraseObject*. Operations on display specifications are also implemented in this module.

6 Conclusions

Users of geographic information systems need to receive answers to queries about spatial objects in a database. On very important and necessary way to display information about spatial data is in the form of a graphic or map. Standard graphics packages are designed to provide graphic routines for a wide variety of applications. This generality makes them inappropriate for cartographic use since the functionality supplied does not satisfy the needs of cartographic output. In addition, current graphics packages are procedural rather than descriptive, requiring the user to specify each step needed to display an object on the screen.

Object-oriented programming provides a solution to these problems. All spatial objects can be represented by display objects. The user needs only to issue the instruction to draw (or erase) an object and specify the object. All the information necessary to accomplish this is contained within the display object. This approach offers the advantages of simple interfaces due to generic operations, information hiding, and dimension and device independence.

Using an object-oriented approach a prototype of such a cartographic output package was designed and its implementation begun on a MicroVAX graphics workstation. This package contains the necessary operations to build a display object for a given spatial object and draw it on the display device. The display object contains the necessary information to represent the various parts of the object with the appropriate symbols.

References

[Bertin 1983] J. Bertin. Semiology of Graphics. The University of Wisconsin Press, Madison, WI, 1983.

[Brodie 1984] M.L. Brodie. On the Development of Data Models. In: M.L. Brodie et al., editors, On Conceptual Modelling, Springer Verlag, New York, NY, 1984.

[Cardelli 1984] L. Cardelli. A Semantics of Multiple Inheritance. In: G. Kahn et al., editors, Semantics of Data Types, Springer Verlag, New York, NY, 1984.

[Dahl 1966] O.-J. Dahl and K. Nygaard. SIMULA—An Algol-based Simulation Language. Communications of the ACM, 9(9), September 1966.

[Egenhofer 1987] M. Egenhofer. Appropriate Conceptual Database Schema Designs For Two-Dimensional Spatial Structures. In: ASPRS-ACSM Annual Convention, Baltimore, MD, 1987.

[Egenhofer 1988a] M. Egenhofer. Graphical Representation of Spatial Objects: An Object-Oriented View. Technical Report 83, Surveying Engineering Program, University of Maine, Orono, ME, July 1988.

[Egenhofer 1988b] M. Egenhofer and A. Frank. A Precompiler For Modular, Transportable Pascal. SIGPLAN Notices, 23(3), March 1988.

[Frank 1986] A. Frank and W. Kuhn. Cell Graph: A Provable Correct Method for the Storage of Geometry. In: D. Marble, editor, Second International Symposium on Spatial Data Handling, Seattle, WA, 1986.

[Frank 1988] A. Frank. Requirements for a Database Management System for a GIS. Photogrammetric Engineering & Remote Sensing, 54(11), November 1988.

[Goldberg 1983] A. Goldberg and D. Robson. Smalltalk-80. Addison-Wesley Publishing Company, 1983.

[Jackson 1989] J. Jackson. Algorithms for triangular Irregular Networks Based on Simplicial Complex Theory. In: ASPRS-ACSM Annual Convention. Baltimore, MD, March 1989.

[Meyer 1988] B. Meyer. Object-Oriented Software Construction. Prentice Hall, New York, NY, 1988.

[O'Brien 1986] P. O'Brien et al. Persistent and Shared Objects in Trellis/Owl. In: K. Dittrich and U. Dayal, editors, International Workshop in Object-Oriented Database Systems, Pacific Grove, CA, 1986.