Building an Interoperable GIS: Integration of an Object-Oriented GIS Database Kernel and a Visualization Framework*

L. Becker, H. Ditt, J. Döllner, K. H. Hinrichs, J. Reiberg, A. Voigtmann

FB 15, Institut für Informatik, Westfälische Wilhelms-Universität,
Einsteinstr. 62, D-48149 Münster, Germany
email: {beckelu, ditt, dollner, khh, reiberg, avoigt}@math.uni-muenster.de

Abstract

Geo-Information Systems (GIS) are used in various application areas of the geosciences. Advanced GIS-applications like meteorological simulations operate on three-dimensional and time-varying data. An extensible database kernel supporting three-dimensional time-varying data forms a promising base for implementing such applications since it can be adapted to the individual data modeling needs of an application. An extensible visualization framework supporting the visualization of data and their dynamics can be adapted to the specific visualization needs of GIS-applications. The Geo Object-Oriented DAtabase Core GOODAC is an extensible database core which supports the development of new-generation GIS-applications. We describe the integration of GOODAC with the visualization and animation framework MAM/VRS which provides extensible object-oriented technology for the development of scientific visualization components for 2D, 3D, and time-varying data.

1 Introduction

During the last couple of years Spatial- and Geo-Information Systems (GIS) have been used in various application areas, like environmental monitoring and planning, rural and urban planning, and ecological research. These classical applications usually are restricted to two-dimensional data and require visualization capabilities which resemble classical paper maps. New advanced applications cover three-dimensional and time-varying data found, for example, in meteorological simulations or seismic exploration.

Currently available GIS were originally designed for the classical application areas mentioned above. Therefore, these GIS are closed systems primarily designed for the storage, management, and visualization of two-dimensional geo-data. A prominent example of such a system is ARC/INFO [More89]. However, new application areas require extensible technologies both for the database kernel which is responsible for storing and managing various types of 2D, 3D, and time-varying data, and for the visualization system which must be capable of displaying the data and their dynamics in multiple variations.

In this paper, we describe the concepts of integrating the geo-database core GOODAC and the visualization and animation framework MAM/VRS. Furthermore, we discuss the benefits for

* This work is supported by the DFG (Deutsche Forschungsgemeinschaft) under grant STR172/8-1.

GIS resulting from this integration. The goal of our research is to provide an extensible and powerful framework for the development of new-generation GIS applications.

The Object-Oriented Geo-Data Model OOGDM and its prototype implementation, the Geo Object-Oriented Database Core GOODAC, provide extensible object-oriented database technology for GIS ([BVH96], [VBH96b], [VBH97]). The data model supports commonly found types of geo-data, and GOODAC realizes an extensible database core supporting the development of new-generation GIS applications on top of GOODAC.

The visualization and animation framework MAM/VRS ([DH97a], [DH97b], [DH97c]) represents an open environment for developing components for geo-based visualization and animation. MAM, the Modeling and Animation Machine, supports high-level modeling of 3D geometry and its dynamic. VRS, the Virtual Rendering System, is an object-oriented 3D rendering system which provides a variety of graphics primitives and rendering techniques.

The tight coupling of GOODAC and MAM/VRS is realized by view classes which use a new approach, the so called *iterators*, to directly exploit database structures for realizing graphics primitives. Since the application is integrated with the database and the visualization framework, data redundancies between the components are prevented and the semantics of application-specific data is available in all parts of the system. Both the data management and the visualization components of a GIS application built on top of GOODAC and MAM/VRS can be customized to fit exactly the individual needs of the application.

GOODAC and MAM/VRS are implemented as C++ libraries. Both are embedded in the object-oriented scripting language [incr Tcl] in order to support rapid prototyping of GIS applications, to take advantage of the powerful and portable Tk user interface toolkit [Ou94], and to support interactive development and testing.

The remainder of this paper is structured as follows. In section 2 we review the data model OOGDM and the database kernel GOODAC. Section 3 presents an overview of the visualization and animation framework MAM/VRS. In section 4 we discuss strategies for integrating database kernel and visualization framework. Section 5 concludes this paper with a discussion of the benefits of our integrated environment for GIS and with an overview of future work.

2 The Geo Object-Oriented Database Core GOODAC

During the last decade object-oriented databases have been subject to intensive research. There are various reasons for the attractiveness of these systems but a very important advantage is that object-oriented data models offer numerous benefits for data modeling of non standard applications. However, GIS have adopted object-oriented database technology and models in part only. Hence, GOODAC tries to provide the benefits of this current database technology for the development of GIS applications.

2.1 OOGDM - The Data Model of GOODAC

GOODAC's data model OOGDM ([BVH96], [VBH96a], [VBH96b]) is an object-oriented data model allowing GIS application developers to define an object-oriented model for their individual application. The core of OOGDM is given by a hierarchy of classes which cover most kinds of spatial data found in GIS-applications. OOGDM supports raster- and vector-

based data in 2D, $2\frac{1}{2}$D, and 3D data space. Furthermore, OOGDM is capable of handling time-varying data by the incorporation of concepts from temporal databases [VBH96b]. A commonly used set of operations including, for example, geometric operations, topological predicates and operations, and direction predicates is also available within the data model.

2.2 The Object Definition and Query Language of GOODAC

Associated with OOGDM are an Object Definition Language OOGDM-ODL and an SQL-like query language OOGQL (Object-Oriented Geo Query Language) [VBH96a]. OOGDM-ODL supports the creation of new classes for a database in a C++-like style. The object definition language satisfies the individual modeling needs of an application developer who realizes GIS applications: The application developer may either extend the data model by deriving application dependent classes from the predefined class hierarchy of OOGDM or by defining classes independently of OOGDM.

Queries to an OOGDM-based database can be formulated in the SQL-like query and manipulation language OOGQL. The design of OOGDM-ODL and OOGQL resembles and extends the languages proposed in the *Object Database Standard* [ODMG96] by concepts for spatial and temporal data. OOGQL can be customized by an extension mechanism.

2.3 The Prototype

GOODAC ([BVH96], [VBH97]) is an extensible GIS database core which realizes the OOGDM class hierarchy. OOGDM-based GIS applications are developed on top of GOODAC in C++. The object-oriented database system *ObjectStore* [LLOW91] has been chosen as an implementation base for GOODAC. ObjectStore provides support for object storage, multi-user access, and transaction management. GOODAC is basically realized by a two-layer architecture which is shown in figure 1. The top layer is the *descriptive layer* which is the view an application developer and a user of the system have. The data model of this layer is OOGDM. The bottom layer of GOODAC is the *representation layer.* The data model of this layer is the so called *representation data model* consisting of various representations for the geometry of OOGDM-objects, various implementations for geometric and topological operations, index structures, and stream based query processing methods.

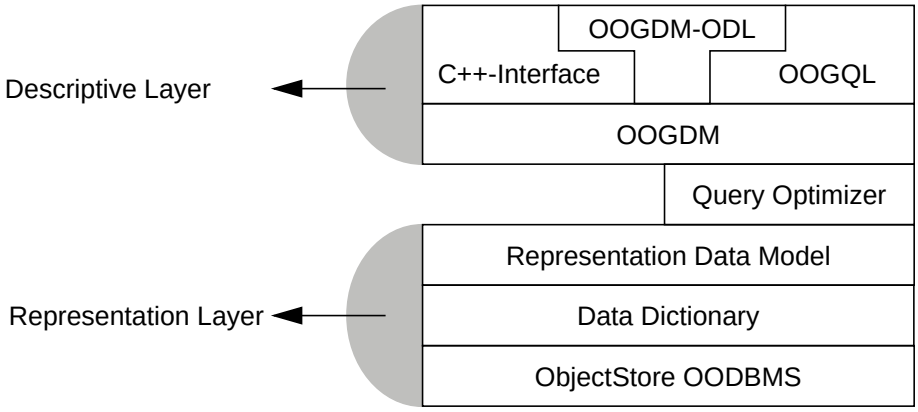

Figure 1: *The architecture of GOODAC.*

The core of application development with GOODAC is the definition of new classes for the database of the application in OOGDM-ODL. These class definitions are precompiled into C++ code. Afterwards, the application developer has to add the code for the methods defined

for these new classes and for the rest of the application program. The code for modifying and reading all attributes defined in the OOGDM-ODL class definitions as well as code for the predefined functionality of OOGDM-based classes is automatically generated by the system.

To meet the requirements of various GIS-application areas, GOODAC has been designed to be an extensible system. System and application programmers may add representations of types (e.g., new representations for the geometry of spatial types), methods and functions for operations (e.g., new GIS functionality), index structures (e.g., new spatial and spatio-temporal indices), query processing methods, and extensions of the query optimizer.

Due to the object-oriented paradigm GOODAC realizes a new form of interoperability where applications integrate seamless with the database kernel, i.e., the applications directly work on the objects and classes provided by the database kernel. Hence, no conversion between the main memory data structures of the applications and the data representation in the database is required, and the semantics of the stored data is available to each application querying and manipulating a database. This kind of coupling is usually not supported by classical closed GIS.

2.4 Supporting Data Integrity

In GIS it is very important to ensure the integrity of the stored data. Usually, attribute values are restricted to a certain domain (e.g., population ≥ 0), or upon an update of the database related attributes or objects are changed (e.g., the population of a city must be updated if the population of a city district changes). Constraints and triggers are well known techniques in database systems which ensure the integrity of the database by

- protecting the attributes against incorrect values (constraints).
- automatically updating attributes upon certain events (trigger).

Hence, triggers provide some kind of active behavior to the database. Since *ObjectStore* does not support constraints and triggers, GOODAC realizes its own concept for constraints and triggers. This support of automated integrity checks simplifies the implementation of pre- and postprocessing steps in GIS applications implemented on top of GOODAC. However, it is not our intention to realize an *active database system* [ACT96] since these systems require a much more sophisticated realization of active features than our approach to constraints and triggers provides.

Constraints and triggers are realized by an extension of our object definition language and by corresponding extensions of GOODAC. OOGDM-ODL allows the definition of constraints and triggers for each class. Triggers are declared in an ECA-style (Event-Condition-Action) like rule. The conditions and actions for constraints and triggers are expressed by statements in OOGQL. The support for time-varying data is integrated into the constraint and trigger mechanism. Details of our concept of constraints and triggers can be found in [DBVH97].

3 The Visualization Framework MAM/VRS

Most existing GIS provide integrated visualization capabilities. However, in the last couple of years computer graphics and computer animation have made progress at an impressive speed. Therefore, many GIS cannot benefit from these developments (e.g., photorealistic and pseu-

dorealistic rendering techniques, interactive manipulation of 3D objects, etc.) because their graphics subsystems are either closed or linked to a specific rendering toolkit. In particular, high-end graphics hardware is poorly used by GIS visualization subsystems. For example, on high-end graphics workstations, OpenGL can superimpose ordinary textures with detail textures in order to represent an additional layer of information. However, this new feature cannot be accessed through black-box visualization components.

3.1 Design Issues for GIS Visualization Frameworks

In contrast to traditional visualization frameworks and graphics capabilities of common GIS, the object-oriented software system for interactive, animated 3D graphics MAM/VRS is based on an open architecture which concentrates on the following aspects:

3.1.1 Application Data Structures for Graphics Primitives

MAM/VRS graphics objects use as much as possible application data without copying the data into internal data structures. Most MAM/VRS graphics objects require so called *iterator objects* provided by the application and use these iterators to *embed* the necessary data. A 3D point set object, for example, does not maintain an array of coordinates. It is associated with an iterator object and uses that iterator to inquire the coordinates each time the point set is rendered. It is up to the iterator‘s implementation to define how that data is calculated or how and where the data is stored.

3.1.2 Integrated and Sophisticated Management of Time

MAM/VRS visualizations are specified by two types of graphs: *geometry graphs* and *behavior graphs*. A geometry graph represents hierarchically nested 3D scenes in analogy to VRML scene graphs. A behavior graph complements a scene description by representing its dynamic aspects such as animations or user interaction capabilities. More abstract, behavior graphs model the time and event flows of a visualization. MAM/VRS provides high-level time building blocks which deform or distribute time according to time layouts. They are useful to build complex animations, such as a semantic-guided flight across a landscape (e.g., the virtual camera could control the acceleration with respect to the landuse information underneath its current position).

3.1.3 Integrated 3D Interaction Capabilities.

3D interaction is important for the direct manipulation of geo objects. For example, in order to place a new building into a virtual landscape, the user needs a precise control mechanism in 3D space. MAM/VRS supports 3D interaction by an internal ray-tracer. The ray-tracer calculates distances and positions of 3D rays and 3D objects. For example, one could simulate a flight across a landscape and constrain the flight altitude to a certain distance above ground; the virtual camera would send out test rays in order to check and adjust its altitude. Note that ray-tracing does not refer to the image synthesis process: here, ray-tracing is an analytical tool applied to geo objects.

3.1.4 Multiple 3D Rendering Techniques

GIS applications have different rendering requirements: real-time rendering for interactive access must use a different rendering technology than high-quality image productions used for

computer generated videos. VRS ensures that the same visualization application can change the underlying 3D rendering library without having to recode the application because of VRS's uniform and object-oriented interface. Currently, VRS supports OpenGL for real-time rendering, and RenderMan, POV Ray, and Radiance for high-quality rendering with global illumination effects. New rendering systems can be integrated by implementing so called virtual *rendering engines*.

3.1.5 Automated Production of Computer Video Sequences

The generation of computer video sequences is a time consuming task. In particular, if data sets are large, the requirements of time and space are enormous. MAM/VRS facilitates the design and realization of computer animations due to its built-in time management and multiple 3D rendering techniques. An animation can be planned and modified with a fast real-time rendering system. To produce the final video sequence, a high-quality rendering system can be plugged in without writing additional code.

3.2 Architecture of the MAM/VRS Framework

The architecture basically consists of two layers, the MAM graphics layer and the VRS rendering layer. The *rendering layer* is responsible for the image synthesis based on low-level 3D rendering libraries, whereas the *graphics layer* is responsible for composing 3D scenes and specifying their dynamics.

The Virtual Rendering System ([DH97b]) provides *graphics objects* which represent graphical entities, e.g., colors, textures, geometric transformations, and shapes. Shapes represent concrete 2D or 3D objects. The appearance of shapes is modified by graphical attributes. Graphics objects are processed and evaluated by *rendering engines* which map graphics objects to appropriate calls of the underlying 3D rendering systems. The application can define new mapping techniques by so called *shape painters* and *attribute painters*. Painters are objects which encapsulate the code for the actual mapping. This way, developers can add application-specific rendering functionality to their visualization system.

VRS is a *thin* object-oriented layer. Its virtual rendering engines do not have a significant impact on the rendering performance compared to applications which access a rendering system directly. Moreover, the OpenGL rendering engine has been fine-tuned to achieve almost the same performance as native OpenGL programs.

The Modeling and Animation Machine provides higher-level modeling techniques for visualization. MAM specifies *geometry nodes* and *behavior nodes*, and it is responsible for the management of *geometry graphs* and *behavior graphs*. Geometry graphs consist of geometry nodes, and behavior graphs consist of behavior nodes. To visualize graphics objects, they have to be associated with geometry nodes. To animate them, they are associated with behavior nodes. VRS and MAM are tightly coupled because MAM's geometry nodes and behavior nodes manipulate and operate on associated, shared graphics objects provided by VRS.

The framework is implemented in C++. User interface bindings exist for Windows, OSF/Motif, and Tcl/Tk. Due to different application programming interfaces and independence from window systems and low-level 3-D rendering libraries, the portability of MAM/VRS is guaranteed. Figure 2 shows the overall architecture of MAM/VRS.

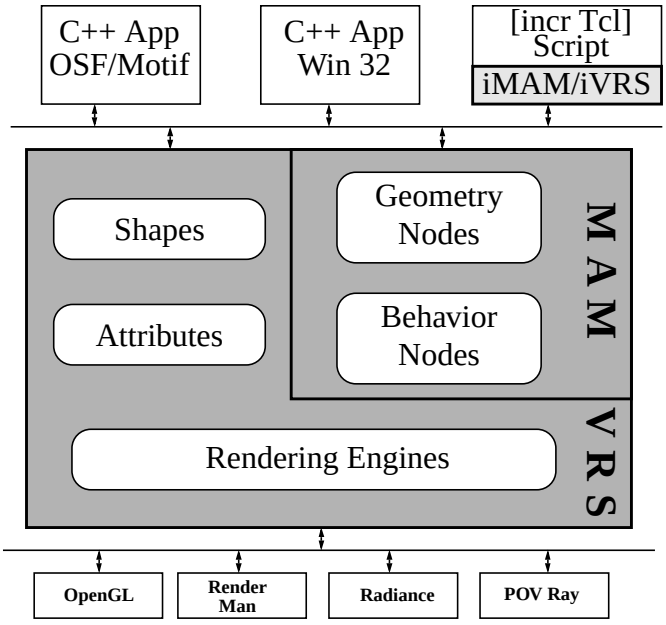

Figure 2: *The Architecture of MAM/VRS.*

4 Integration of Database Kernel and Visualization

GOODAC and MAM/VRS provide database and visualization capabilities required to develop GIS applications. The interface between the database kernel, the visualization framework, and the GIS application must be designed very carefully because the interface has a major impact on the overall system performance.

If we choose a loose coupling, database kernel and visualization framework exchange data basically in three steps:

1. Convert objects of the source component into an exchange format.
2. Store these intermediate objects.
3. Convert the intermediate objects into objects suitable for the target component.

This loose coupling of components leads to problems resulting from the potential data redundancy and from loss of information during the conversion processes. Hence, the semantics of the objects used by the application program is usually not accessible by the visualization component which naturally restricts visualization techniques. The loose coupling is the most popular interface provided by classical GIS.

In our approach, a tight coupling of GIS application, database kernel, and visualization framework is chosen. A GIS application shares the data types used in the database kernel by deriving application-dependent spatial classes from the OOGDM class hierarchy. The usage of unique data types within the application and the database provides efficient processing of the persistent data without data redundancy and conversion operations.

The integration of the database kernel and the visualization framework requires a different strategy since both systems have been developed independently and can be used independently of each other. While OOGDM has been designed to meet the requirements of GIS applica-

tions, the data model provided by MAM/VRS reflects the needs of 3D graphics and animation. Of course, there is a certain analogy in their class hierarchies. For example, vector-based data classes provided by the database kernel, e.g., polylines, polygonal regions, and solids, can be represented by sets of line segments, triangles, and simplices provided by the visualization framework. Furthermore, 2D and 3D meshes of triangles and simplices available in the visualization framework can be used for the visualization of the raster-based classes of the database kernel. However, we cannot expect to merge these class hierarchies due to their different semantics and requirements. In our approach, we integrate database and visualization by *visualization view classes* embedded in the visualization framework and by providing a uniform *embedding in a scripting language*.

4.1 Visualization View Classes

Visualization view classes manage the mapping of database objects to graphics objects. In general, a visualization view class will base that mapping on the geometric and thematic data. The tight coupling of the database objects and graphics objects is realized by iterator objects. The iterators provide an efficient way to establish a direct link between database kernel and visualization framework. An overview of the architecture of our integrated system is given in figure 3.

A visualization view class is associated with database classes and derives for these database classes iterator classes. These iterator subclasses are specific to the database classes and may take advantage of their internal data representation. The main purpose of iterators is the sequential access of geometric or graphics data in a form suitable for MAM/VRS. The conversion is carried out on the fly without an intermediate storage. Furthermore, visualization view classes instantiate MAM/VRS graphics objects and connect them to iterators.

Visualization view classes can map database objects to graphics objects in various ways. Since most classes of graphics objects in MAM/VRS rely on embedded data provided by iterators, visualization techniques can be realized efficiently.

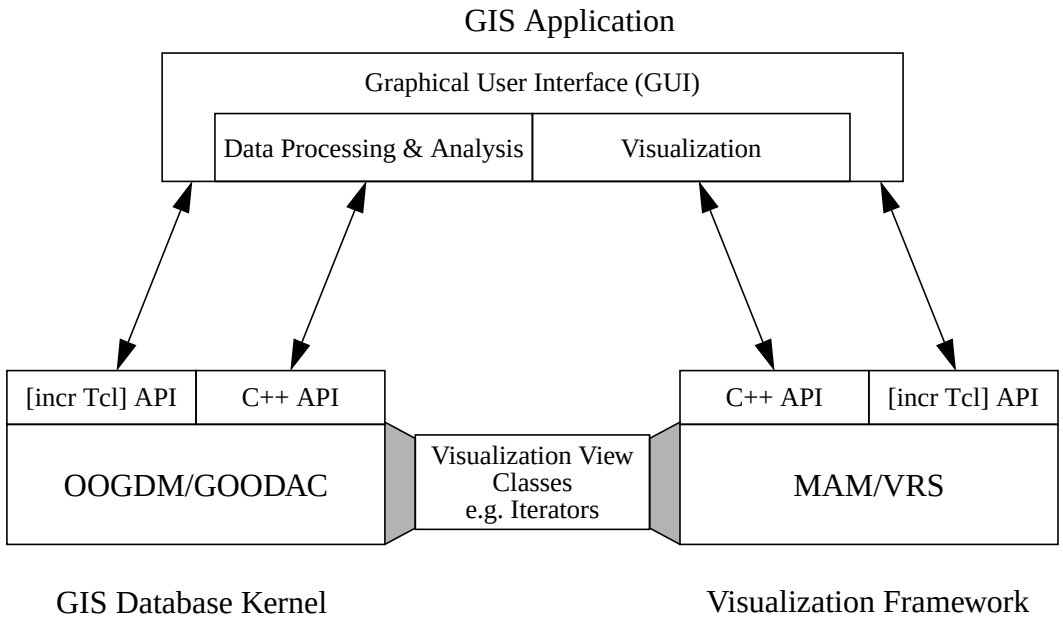

Figure 3: *Architectural Overview of the Integration of GOODAC and MAM/VRS.*

Example:

Consider the visualization of a digital elevation map (DEM) which is represented in the database by the class *DEM*, a subclass of class *Grid2D*, i.e., a two-dimensional matrix of elevation information. Figure 4 shows the class hierarchy described in this example. A DEM is visualized by an object of class *RMesh* provided by MAM/VRS. *RMesh* objects represent triangulated *n x m* meshes. The interface between *Grid2D* and *RMesh* is established by the iterator classes *RGrid2DVertexIterator* and *RGrid2DNormalIterator,* both are derived from class *MIterator<MVector>*, the base class of all MAM/VRS iterators. Each iterator object is associated with an *Grid2D* object and an *RMesh* object.

If the DEM is requested to render itself, the *RMesh* object actually uses the iterator objects to inquire the coordinates and the color data. Both types of iterators perform a row-by-row scan of the DEM‘s elevation matrix. Iterator objects of class *RGrid2DVertexIterator* return *MVector* objects, the 3D point and 3D vector class of MAM/VRS. Iterator objects of class *RGrid2DNormalIterator* return *MVector* objects which represent surface normal vectors of a DEM entry. The computation of the normal vectors can be based on different rules, e.g., geometric normals. Iterators perform their computations on the fly, i.e., no data storage or duplication is required.

So far, the geometry of database objects has been linked to graphics objects. The thematic part of the database objects can be mapped either by additional graphics objects or by color information applied to existing graphics objects. It is up to the visualization strategy how to map thematic data.

Example:

Consider again the visualization of a DEM. We have described how spatial coordinates and the normal vectors are embedded in graphics objects by *RGrid2DVectorIterator* objects and *RGrid2DNormalIterator* objects. To visualize the landuse information for a DEM, we assign colors to the mesh based on a thematic color scale. Colors are assigned to a mesh by a color iterator. In the example, we derive a color iterator class *RLanduse-ColorIterator* which has to return *MColor* objects. An *MColor* object contains the RGB coefficients and the transparency coefficient. The color iterator will actually use the associated database landuse object to inquire landuse information for a spatial position.

Alternatively, we could color the mesh with respect to the height of DEM entries, whereby the colors are calculated based on a color scale. This approach can be implemented by the iterator subclass *RDEMHeightColorIterator.*

Both landuse and height information layers are visualized by the same paradigm: color iterators infiltrate directly application semantics in MAM/VRS objects. We could also design an appropriate texture coordinate iterator to map an additional information layer onto the mesh.

Iterator classes are introduced at the level of user-accessible classes of the data model OOGDM. For each user-accessible class of OOGDM, the vertex and normal iterator classes are provided by default.

Obviously, the concept of embedded data in graphics objects does not require explicit conversion of data between database kernel and visualization framework, and thereby an erroneous data redundancy between both components is prevented. The visualization directly accesses the objects of the database and transfers the information contained in the database into a format which can be processed by the low-level rendering engines underlying the visualization

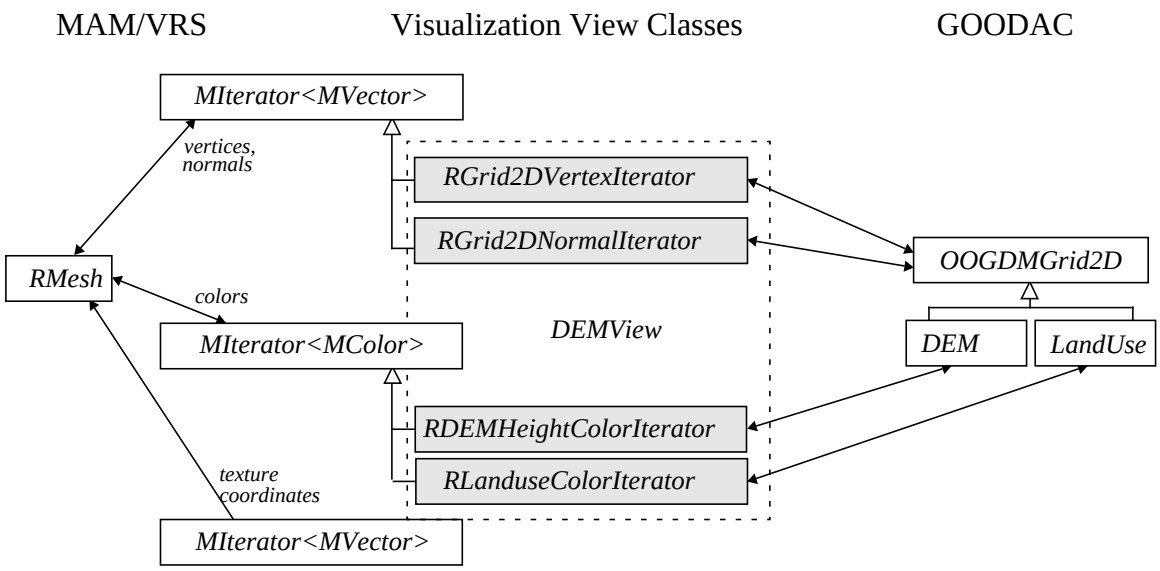

Figure 4: *.Class Relations for the DEM Visualization.*

framework. This transfer is done each time a database object is rendered on the screen. To provide smooth animations and to speed up user-interaction (e.g., interactive exploration of 3D scenes) graphics objects can be cached. However, the visualization view classes keep track of modifications to the database objects such that cached data is always updated if database objects have been modified.

The tight coupling of database and visualization offers extensible and customizable visualization techniques to GIS applications. To prove the applicability of our approach, we have realized a GIS application for the modeling of *nocturnal cold air drainage flows*. Figure 5 shows an example visualization taken from this application.

4.2 Interactive Modification of Database Objects

As a consequence of the tight coupling of database and visualization, it is easy to implement interactive modification techniques for database objects. The visualization framework MAM/VRS provides *tag objects* to assign application-specific identifiers and group identifiers to graphics objects. Tags represent the hooks for any type of database editor. Tags are created by visualization view classes and are associated with the graphical representations of database objects. The tags are used to formulate scene requests and to build object-specific interactions. Interaction techniques can use tags to identify the relevant objects and to report them to the database editor. In addition, interaction techniques can use the built-in ray tracer to inquire spatial relationships of 3D objects.

Since database objects and their visual representations are connected by iterators, modifications apply directly to the visual representation as well as to the database. Moreover, 3D interaction techniques developed in computer graphics can be exploited. For example, the height of a building located in a DEM can be interactively modified by a 3D scroll bar (i.e. a cylinder erected upon the landscape with a small slider box). Figure 6 illustrates the data flow during an interactive modification of database objects.

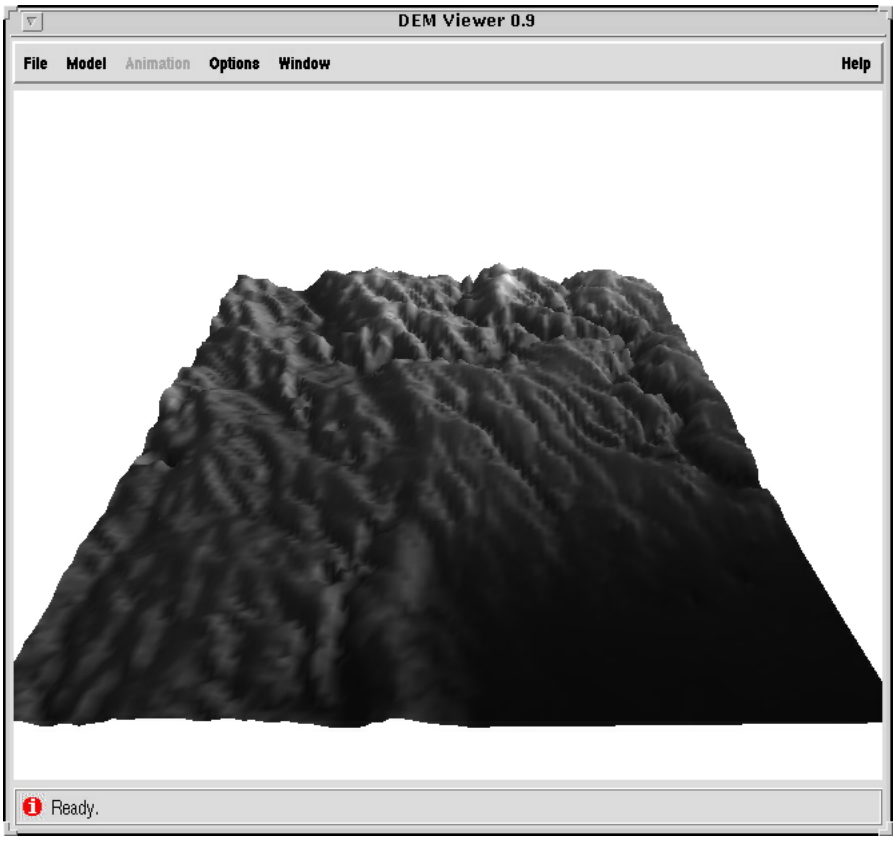

Figure 5: *Sample DEM Visualization.*

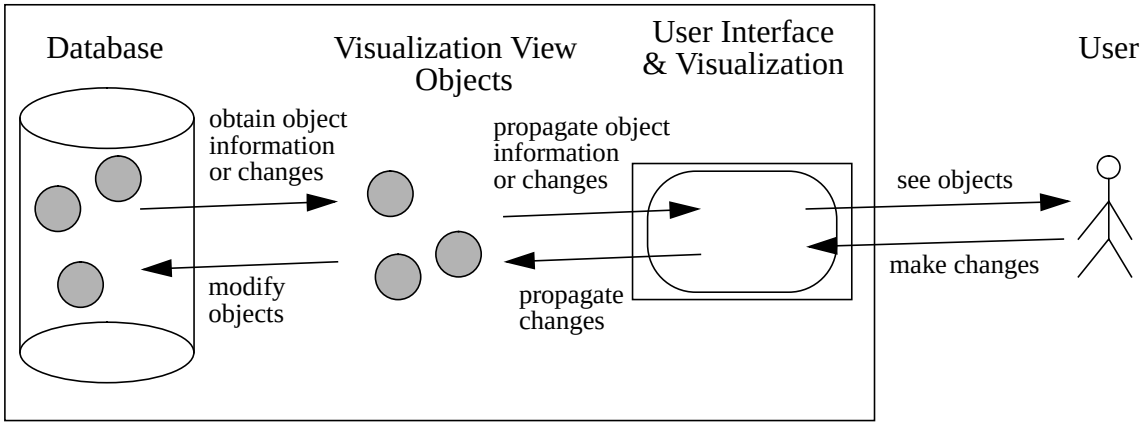

Figure 6: *Data Flow during Visualization and Modification of Database Objects.*

4.3 Embedding in an Object-Oriented Scripting Language

In general, scripting languages offer many advantages for high-level programming compared to system programming languages because they are easier to learn and understand, provide a tight binding to user interfaces, and allow for rapid and interactive prototyping. Their main disadvantage is that they cannot implement large object-oriented systems. The interpretative tool command language *Tcl* [Ou94], its GUI toolkit *Tk*, and the object-oriented Tcl extension

[incr Tcl] [ML97] together represent a widely used and one of the most powerful object-oriented scripting languages.

Both the database kernel GOODAC and the visualization framework MAM/VRS are implemented by C++ libraries. To integrate both systems at the application programming interface level, we have embedded them into the object-oriented scripting language [incr Tcl].

Our approach for embedding C++ classes in [incr Tcl] is based on the following concept: For each C++ class there is one “mirror class” in [incr Tcl] whose class members have a one-to-one relationship to their C++ counterparts. For each object created in the C++ world, there is a “mirror object” in the [incr Tcl] world. A mirror object delegates all requests to its corresponding C++ object. That delegation guarantees optimal flexibility and performance: The complete functionality of the database kernel and the visualization framework is available through the [incr Tcl] interface. However, there is no lack of performance since the database and the visualization functionality is not executed in the interpretative scripting language but in the native C++ code.

If the application developer extends the database kernel or the visualization framework by new classes derived from the existing class hierarchies, these classes must be available to the [incr Tcl] interface as well. The same applies to application-dependent classes developed independently of GOODAC and MAM/VRS which should also be available to the [incr Tcl] interface. To support the automatic integration of application-specific classes, a generator [DH97d] has been developed which creates the [incr Tcl] classes from information obtained by parsing the corresponding C++ header files.

The interpretative object-oriented scripting language [incr Tcl] proved to be a valuable tool for developing user interfaces, too. Tk offers a platform-independent collection of easy to customize user interface components. Due to the nature of a scripting language, user interface design can be done interactively. New parts of an application can also be developed in the scripting language, can be tested within the interpretative environment, and can later be ported to C++.

5 Conclusions and Future Work

Traditional GIS are based on a closed architecture, primarily and historically designed for handling 2D geo data. However, new applications, e.g., in geology or meteorology, require three-dimensional and time-varying data. This shift has to be reflected by an advanced architecture for GIS.

The integration of the object-oriented GIS database kernel GOODAC and the object-oriented visualization and animation framework MAM/VRS has a series of benefits for the development of GIS:

- GIS benefit from object-oriented database technology provided by GOODAC. Geo objects, for example, are inherently persistent without having to write persistency routines.
- Geo objects are efficiently infiltrated in the visualization framework without loss of semantics. The integration of our database kernel and visualization framework ensures that no data redundancies occur, i.e., the visualization obtains its data directly from the

database. This option is not available with systems operating independently and exchanging data, for example, via a file system.

- Both GOODAC and MAM/VRS can be extended to meet the requirements of individual GIS applications. The database kernel can be extended to efficiently support the storage and retrieval of data for a wide range of GIS applications. The visualization framework can be extended by application-specific visualization classes for advanced and optimized usage.
- 2D, 3D, and time-varying data are explicitly supported by both the database kernel and the visualization framework. The dynamics of data can be visualized by interactively exploring the data or by animation sequences.
- GOODAC provides tools for the development of GIS-applications in C++. For example, using the OOGDM-ODL class definitions the precompiler automatically generates the required C++-class definitions. Furthermore, constraints and triggers support the realization of application-dependent pre- and postprocessing methods.
- The construction of interactive editors for database objects is facilitated since all components are integrated and data semantics is accessible within all system components. Interaction techniques benefit from the built-in interaction capabilities of MAM/VRS.
- A GIS application can choose between different rendering systems and integrate future rendering systems due to the virtual rendering system.
- Developers benefit from the object-oriented scripting language because it shortens development time and allows for interactive and rapid prototyping of database, visualization, and user interface components. The complete functionality of the database kernel and visualization framework is accessible within the scripting language without a significant loss of performance due to the embedded C++ classes.

Future work on MAM/VRS and GOODAC includes the development of new visualization and interaction strategies for high-dimensional data, the development of an *OpenGIS*-interface, the addition of classes for the *FGDC*-standard of geo-spatial metadata, and the realization of further query processing methods.

Acknowledgments

The contributions of Lars Bernard, Jan Budde, Torsten Kähler, Holger Lange, Benno Schmidt, and Tilmann Steinberg to the GOODAC prototype and of Tobias Gloth, Oliver Kersting, Björn Lojewski, and Marc Nienhaus to MAM/VRS are gratefully acknowledged.

Literature

[ACT96] ACT-NET Consortium: *The Active Database Management System Manifesto: A Rulebase of ADBMS Features*, ACM SIGMOD Record, 25 (3), 1996, 40 - 49.

[BVH96] L. Becker, A. Voigtmann, K. H. Hinrichs: *Developing Applications with the Object-Oriented GIS-Kernel GOODAC*, Proc. 7th Int. Symp. On Spatial Data Handling (SDH'96), Delft, The Netherlands, 1996, 5.A.1 - 5A.18.

[DBVH97] H. Ditt, L. Becker, A. Voigtmann, K. H. Hinrichs: *Constraints and Triggers in an Object-Oriented Geo Database Kernel*, Proc. 8th Int. Workshop on Database and Expert Systems Applications (DEXA'97), Toulouse, France, 1997, 508 - 515.

[DH97a] J. Döllner, K. H. Hinrichs: *Support of Explicit Time and Event Flows in the Object-Oriented Visualization Toolkit MAM/VRS*, in: Proceedings of Visualization and Mathematics `97, Berlin-Dahlem, Germany, 1997.

[DH97b] J. Döllner, K. H. Hinrichs: *The Design of a 3D Rendering Meta System*, in: Eurographics Workshop on Programming Paradigms for Graphics `97, Budapest, Hungary, 1997.

[DH97c] J. Döllner, K. H. Hinrichs: *Object-oriented 3D Modeling, Animation, and Interaction*, The Journal of Visualization and Computer Animation, 8, 33-64, 1997.

[DH97d] J. Döllner, K. Hinrichs: *Object and Class Management in a Hybrid Architecure using [incr Tcl] and C++*. Interner Bericht, University Muenster, 1997.

[ML97] M. McLennan: *[incr Tcl] - Object-Oriented Programming in Tcl/Tk*, online document, see http://www.tcltk.com/itcl/

[ODMG96] R. G. G. Catell (ed.): *The Object Database Standard: ODMG-93, Release 1.2*, Morgan-Kaufmann Publishers, San Francisco, CA, 1996.

[LLOW91] C. Lamb, G. Landis, J. Orenstein, D. Weinreb: *The ObjectStore Database System*, Communications of the ACM, 34 (10), 1991, 50 - 63.

[More89] S. Morehouse: *The Architecture of ARC/INFO*, Auto-Carto 9 Conf., S. 266-277.

[Ou94] J. Ousterhout: *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.

[VBH96a] A. Voigtmann, L. Becker, K.H. Hinrichs: *An Object-Oriented Data Model and a Query Language for Geo Information Systems*, Bericht Nr. 5/96-I, Institut für Informatik, Westf. Wilhelms-Universität, Münster, Germany, 1996.

[VBH96b] A. Voigtmann, L. Becker, K. H. Hinrichs: *Temporal extensions for an Object-Oriented Geo-Data-Model*, Proc. 7th Int. Symp. On Spatial Data Handling (SDH'96), Delft, The Netherlands, 1996, 11A.25 - 11A.41.

[VBH97] A. Voigtmann, L. Becker, K. H. Hinrichs: *Physical Design Aspects of an Object-Oriented Geo Database Kernel*, Proc. 8th Int. Workshop on Database and Expert Systems Applications (DEXA'97), Toulouse, France, 1997, 529 - 534.