Erschienen: Computers & Geosciences, 26(2000), 67-76, Elsevier Science Verlag.

An Object-Oriented Approach for Integrating 3D Visualization Systems and GIS

JÜRGEN DÖLLNER and KLAUS HINRICHS

FB 15, Institut für Informatik, Westfälische Wilhelms-Universität,
Einsteinstr. 62, D-48149 Münster, Germany
Phone: (++49) 251 / 83 - 32700, FAX: (++49) 251 / 83 - 33755
(e-mails: dollner@uni-muenster.de, khh@uni-muenster.de)

Abstract – Visualization has become an integral part in many applications of GIS. Due to the rapid development of computer graphics, visualization and animation techniques, general-purpose GIS can no longer satisfy the multitude of visualization demands. Therefore, GIS have to utilize independent visualization toolkits. This article examines how visualization systems can be used with and integrated into GIS. We analyze several key characteristics visualization toolkits should satisfy in order to be used efficiently by GIS. We show how GIS can provide visualization and animation features for geo objects by embedding the visualization system using object-oriented techniques. The concepts are described along with a new visualization and animation toolkit which provides extensible object-oriented technology for the development of visualization components for 2D, 3D, and time-varying data. The design of this visualization toolkit concentrates on a seamless integration of application-specific geo-data into visualization components, an open interface for different rendering techniques, and an advanced management of data dynamics.

Key Words: ViSC, Visualization Framework, Object-oriented Visualization, Computer Animation.

1 Introduction

Visualization has become an integral part in many GIS and their applications. New advanced applications of GIS, e.g., in exploratory cartographic visualization, meteorological simulations, or seismic exploration, process three-dimensional and time-varying data. Interactive 3D visualization evolves as a main stream technology both in science and industry, and is available even on low-cost PCs.

General-purpose GIS were originally designed for the classical 2D application areas, and hence do often not reflect the state of the art in visualization. Moreover, if a GIS implements its own visualization subsystem, it will be difficult or even impossible to integrate cutting-edge visualization technology. For example, state of the art visualization toolkits support real-time volume rendering for 3D data sets based on a clever usage of textures. Since the 3D graphics library OpenGL (Woo, Neider, and Davis, 1997), the de facto industry standard for 3D graphics, is available on most of today's machines, high-level visualization techniques are no longer restricted by graphic software limitations.

A few new, innovative toolkits for visualization and 3D graphics have been developed in the past, for example *vtk* (Schroeder, Martin and Lorensen, 1998), *OpenInventor* (Wernecke, 1994), and *MAM/VRS* (Döllner and Hinrichs, 1997). These toolkits consist of sets of classes which can be included and compiled into application programs. In contrast to closed visualization systems such as AVS (Upson and others, 1989; for an overview see Slocum, 1994) these toolkits provide the prerequisites for being extended and enriched by new visualization components due to their object-oriented architecture.

In contrast to more monolithic GIS such as ARC/INFO (Morehouse, 1989), GIS should be based on a software architecture which allows developers to use or to integrate independent visualization toolkits, and to extend the visualization capabilities with respect to the application domain by customized, application-specific visualization components. The GIS architecture should ensure that the application developer can choose the most appropriate visualization library and switch to another visualization toolkit if necessary.

There exist different strategies for coupling a visualization toolkit with a GIS. If the visualization toolkit has to be used as an independent system, e.g. Geomview (Phillips, Levy, and Munzner, 1993), the GIS has to communicate with the visualization system by import and export files (loose coupling). The main disadvantage of this approach is that visualization features are restricted by the file exchange format and that dynamic or large data sets can hardly be transferred. Alternatively, the visualization toolkit could be integrated as part of the GIS (tight coupling), communicating either through shared data structures at the object level, or by object communication services (e.g., CORBA, ActiveX). The main disadvantage of this approach is that the GIS has to link visualization objects and geo objects by customized software components. However, this approach allows for a direct and efficient communication between visualization toolkit and GIS.

In this paper, we describe how to integrate a GIS with a visualization toolkit by *embedding visualization objects* into the GIS and by *specializing visualization classes* by object-oriented techniques such as subclassing. In order to adapt the visualization system to the requirements of the GIS, visualization objects have to provide a flexible, efficient, and adaptable interface. The tight coupling of application-specific data management and the visualization objects is realized by view classes which use the concept of *iterators* to directly exploit data structures for specifying visualization primitives. Since the application data structures are integrated with the visualization toolkit, data redundancies between the components are prevented, and the semantics of application-specific data is available in all parts of the system.

2 GIS Requirements for Visualization Toolkits

Visualization toolkits used by GIS must meet the requirements of the visualization of typical geo data and typical manipulation tasks. We identified several key characteristics visualization toolkits should satisfy in order to be used efficiently by GIS. These characteristics include an efficient incorporation of geo data in visualization objects; an integrated management of data dynamics; high-level 3D interactive manipulation features; exchangeable rendering systems; and facilities for designing and recording animation sequences. These characteristics are discussed in the following. We have designed and implemented these concepts in an object-oriented manner in a new, prototype visualization toolkit, called MAM/VRS, which serves as a customizable visualization subsystem of any GIS.

2.1 Realization of GIS Visualization Requirements in MAM/VRS

The visualization toolkit MAM/VRS is an open framework for developing components for geo-based visualization and animation. MAM, the *Modeling and Animation Machine*, supports high-level modeling of 3D geometry and its dynamics. VRS, the *Virtual Rendering System*, is an object-oriented 3D rendering system which provides a variety of graphics primitives and rendering techniques.

2.1.1 Application Data Structures for Graphics Primitives

Geo objects provided and maintained by GIS have to be transformed or linked to visualization objects in order to provide data to the visualization toolkit. Both the implementation effort and the efficiency depend on the type of coupling. In MAM/VRS, visualization objects use application data as much as possible without copying the data into internal data structures. Most MAM/VRS visualization 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.

2.1.2 Integrated and Sophisticated Management of Time

Data dynamics encoded in time-variant geo data can be visualized by temporal animation. However, time can be used also to visualize features other than time. Non-temporal animations, for example, can be used to visualize spatial data uncertainty. For a detailed discussion of the dynamic variables see MacEachren (1994) who points out that time as “*powerful variable [...] is likely to make the most substantial impact on maps as visualization tool in GIS*”. Thus, the visualization toolkit has to provide mechanisms for controlling and manipulating time in geo objects and for modeling the global time flow.

MAM/VRS models the geometry of a visualization and its dynamics symmetrically 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 its acceleration with respect to the wind information at its current position).

2.1.3 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. 3D interaction represents also the technical foundation for intelligent dynamic maps. 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 path to a certain altitude above the 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. Since all visual objects can be associated with tag objects, the GIS can annotate any type of information it needs for identifying geo objects in virtual scenes.

2.1.4 Multiple 3D Rendering Techniques

GIS applications have different rendering requirements: real-time rendering for interactive GIS applications 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 (Upstill, 1990), POV Ray (POV Team, 1998), and the light simulation system Radiance (Ward, 1994) for high-quality rendering with global illumination effects. New rendering systems can be integrated by implementing so called virtual *rendering engines*. Fig. 1 shows a terrain rendered with OpenGL. To facilitate the navigation in the terrain, sun flares have been added.

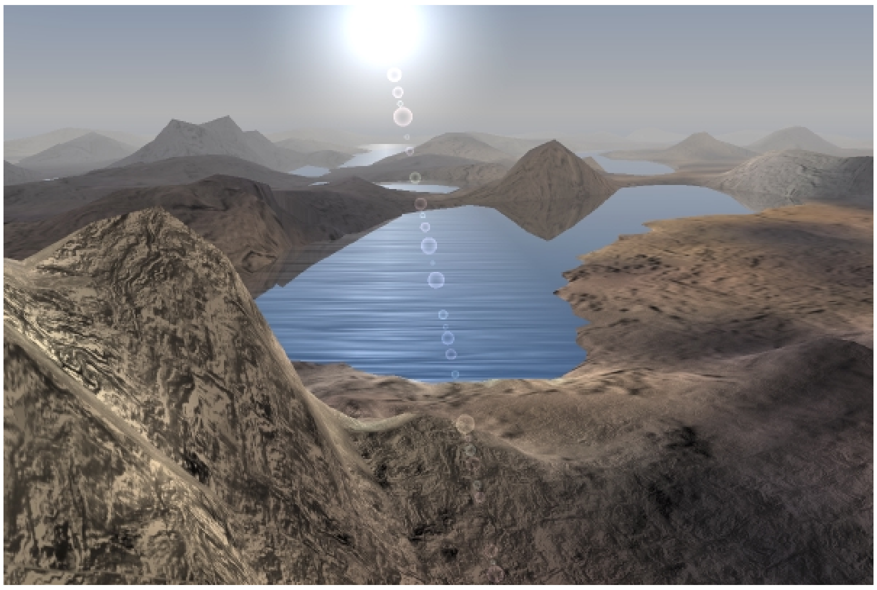

Figure 1: *Real-time terrain visualization using OpenGL 2D textures and 2D detail-textures. Lens flares are superimposed by transparent textures.*

2.1.5 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.

2.2 Architecture of the MAM/VRS Visualization Toolkit

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 (Döllner and Hinrichs, 1997b) 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 MAM/VRS toolkit is implemented as a C++ library. User interface bindings exist for the Microsoft Foundation Classes MFC, OSF/Motif, and Tcl/Tk. On Windows platforms, visualization components can be developed as ActiveX components.

MAM/VRS objects can be used within the scripting language Tcl/Tk (Ousterhout, 1994) in order to support rapid prototyping of GIS applications, to take advantage of the powerful and portable Tk user interface toolkit, and to support interactive development and testing. The different application programming interfaces ensure that the toolkit is independent of the window system and the low-level 3D rendering library and that MAM/VRS is portable. The visualization toolkit can be used if at least OpenGL is available on the target platform (e.g., Windows95/NT, most Unix systems). Figure 2 shows the overall architecture of MAM/VRS.

3 Integration of Geo-Data and Visualization Objects

The interface between geo-data and visualization objects must be designed very carefully because the interface has a major impact on the transparency of the system and its overall performance.

If we choose a loose coupling, geo data structures and visualization objects 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

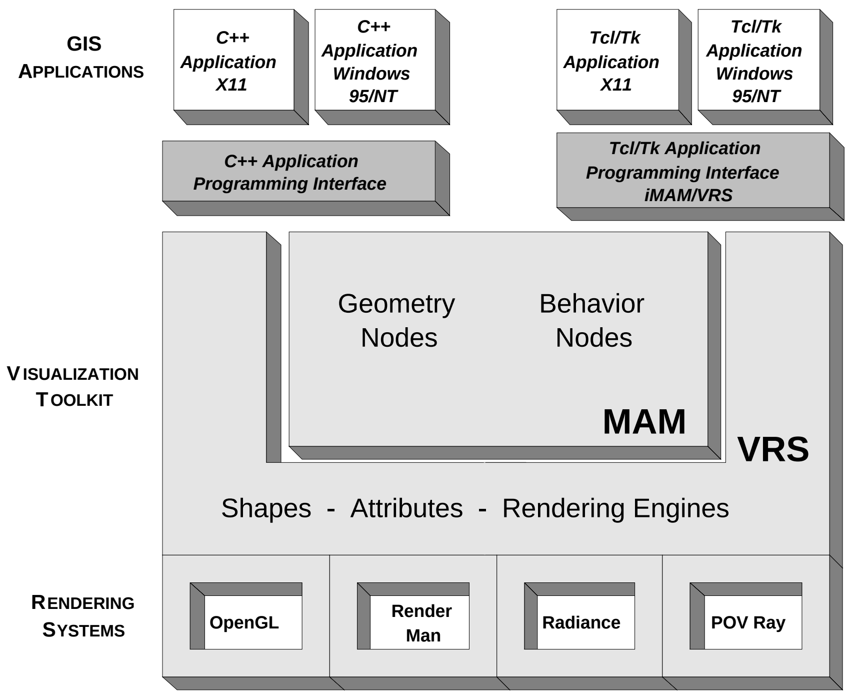
GIS Applications
C++ Application X11
C++ Application Windows 95/NT
Tcl/Tk Application X11
Tcl/Tk Application Windows 95/NT
C++ Application Programming Interface
Tcl/Tk Application Programming Interface iMAM/VRS
Visualization Toolkit
Geometry Nodes
Behavior Nodes
MAM
VRS
Shapes - Attributes - Rendering Engines
Rendering Systems
OpenGL
Render Man
Radiance
POV Ray

Figure 2: *The architecture of the MAM/VRS visualization toolkit.*

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 and visualization toolkit is chosen. While most geo-data structures have been designed to meet the requirements of GIS applications, the data model provided by MAM/VRS reflects the needs of 3D graphics and animation. Of course, there is in general a certain analogy in the class hierarchies. For example, vector-based data classes provided by an application, e.g., polylines, polygonal regions, and solids, can be represented by sets of line segments, triangles, and simplices provided by the visualization toolkit. Furthermore, 2D and 3D meshes of triangles and simplices available in the visualization toolkit can be used directly for the visualization of the corresponding raster-based classes in most applications. However, we cannot expect to merge these class hierarchies due to their different semantics and requirements. The visualization toolkit should allow the GIS developer to add whatever is needed to visualize a specific type of geo object by subclassing existing visualization classes. In our approach, we integrate geo data structures and visualization by *visualization view classes* embedded in the visualization toolkit and by providing a uniform *embedding in a scripting language*.

3.1 Visualization View Classes

Visualization view classes manage the mapping of application objects to graphics objects. By application object we mean any type of object which contains geo data, for example a DEM object. With respect to object-oriented design patterns (Gamma et al., 1995), a visualization view class is both an *adapter class* which converts the interface of one class into another interface clients expect, and a *mediator class* which encapsulates how two types of objects interact. In general, a visualization view class will base the mapping on the geometric and thematic data. In particular, the tight coupling of the application objects and graphics objects is realized by specialized view objects, the *iterator objects*. The iterators provide an efficient way to establish a direct link between application kernel and visualization toolkit. An overview of the architecture of such an integrated system is given in figure 3.

A visualization view class is associated with application classes and derives for these application classes iterator classes. These iterator subclasses are specific to the application 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 application 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.

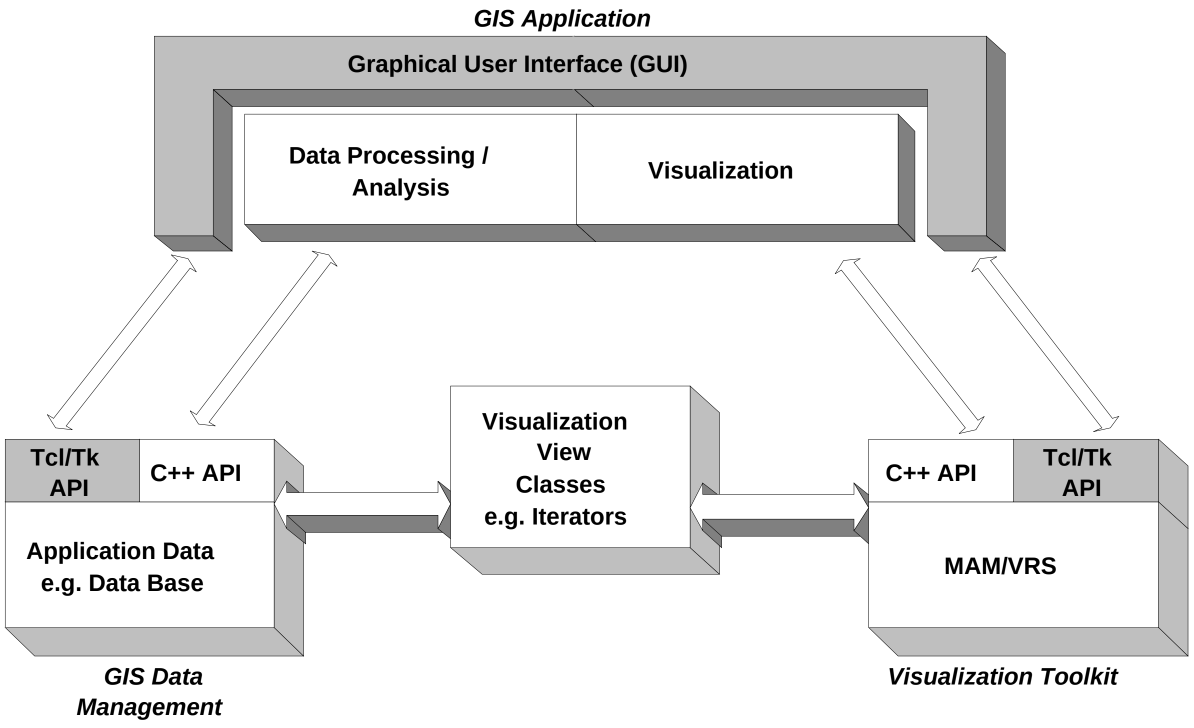
GIS Application
Graphical User Interface (GUI)
Data Processing / Analysis
Visualization
Tcl/Tk API
C++ API
Application Data e.g. Data Base
Visualization View Classes e.g. Iterators
C++ API
Tcl/Tk API
MAM/VRS
GIS Data Management
Visualization Toolkit

Figure 3: *Architectural overview of the integration of a GIS application and the visualization toolkit MAM/VRS.*

Example:

Consider the visualization of a digital elevation map (DEM) which could be represented in a 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 *Mesh* provided by MAM/VRS. *Mesh* objects represent triangulated *n x m* meshes. The interface between *Grid2D* and *Mesh* is established by the iterator classes *Grid2DVertexIterator* and *Grid2DNormalIterator*, both are derived from class *Iterator< Vector>*, the base class of all MAM/VRS iterators. Each iterator object is associated with an *Grid2D* object and a *Mesh* object.

If the DEM is requested to render itself, the *Mesh* 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 *Grid2DVertexIterator* return *Vector* objects, the 3D point and 3D vector class of MAM/VRS. Iterator objects of class *Grid2DNormalIterator* return *Vector* 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 application objects has been linked to graphics objects. The thematic part of the application 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 *Grid2DVectorIterator* objects and *Grid2DNormalIterator* objects. To visualize the land use 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 *Landuse-ColorIterator* which has to return *Color* objects. A *Color* object contains the RGB coefficients and the transparency coefficient. The color iterator will actually use the associated land use geo object to inquire land use 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 *DEMHeightColorIterator.*

Both land use 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.

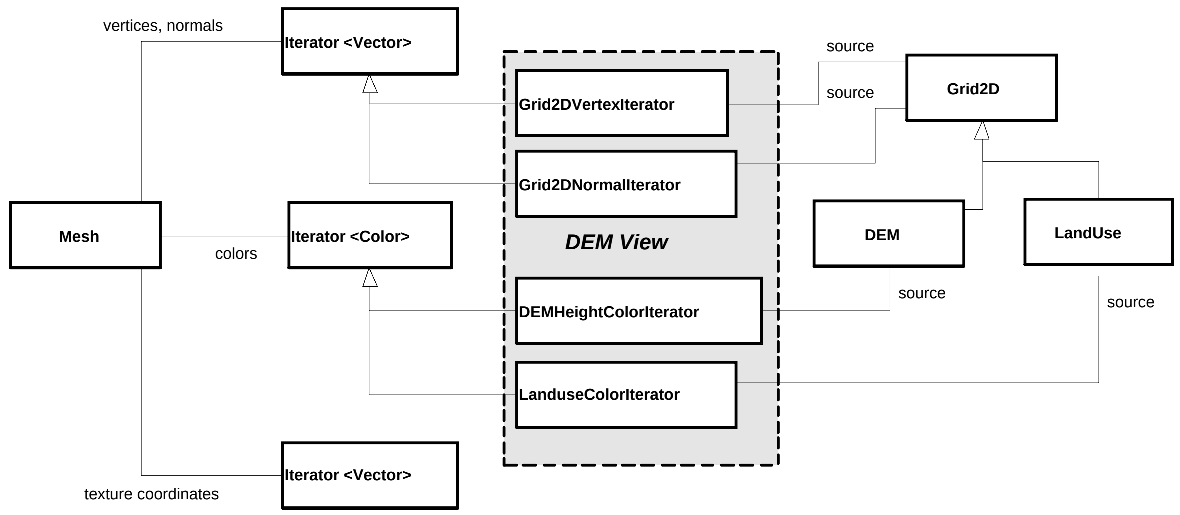

Figure 4: *Class relations for the DEM visualization.*

.

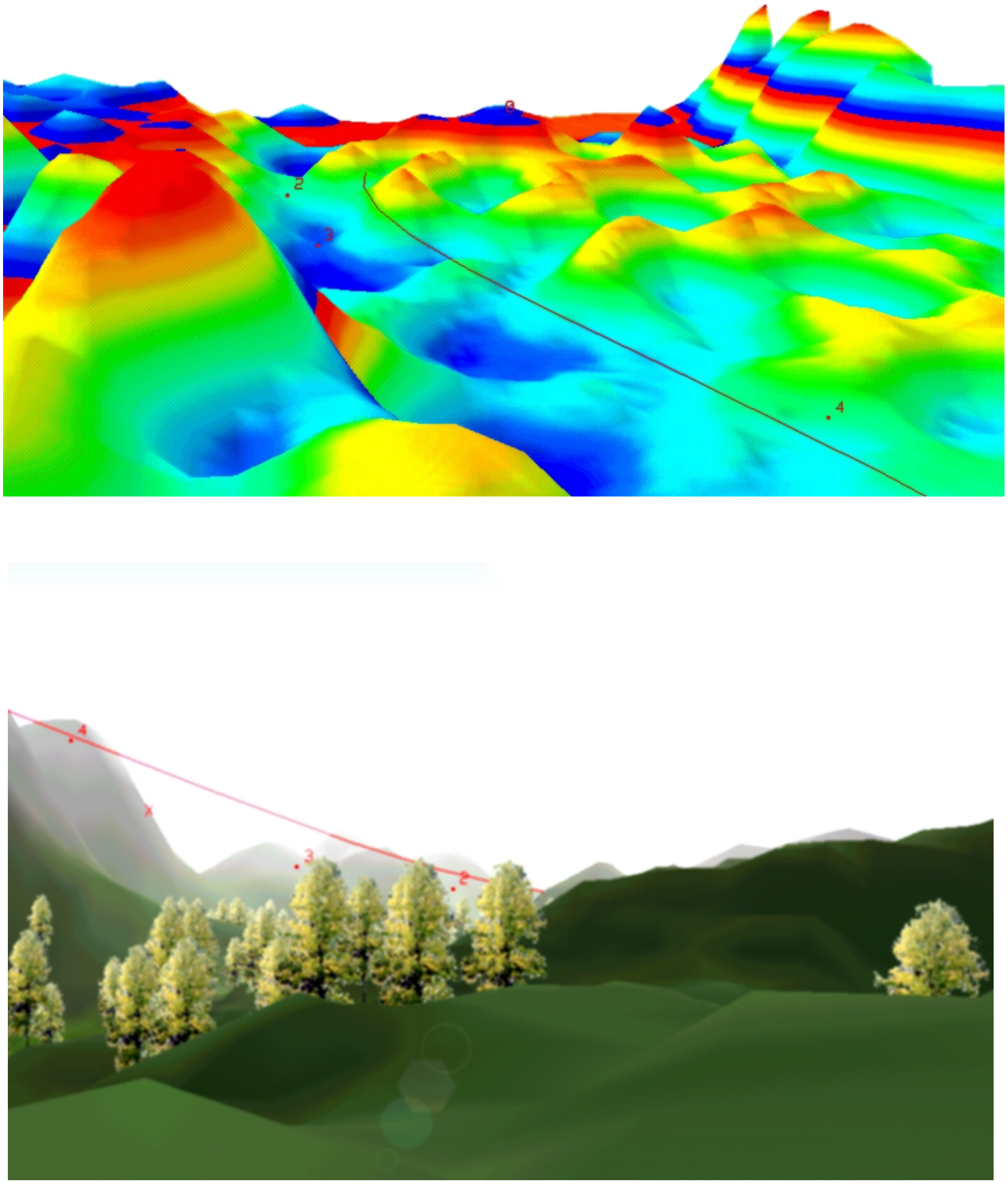

Figure 5: *Sample DEM visualization. (a) Hypsometric color tinting applied to the terrain. The flight path is visualized by a polyline. (b) Visualization of the forest land use; trees are visualized by transparent textures.*

Obviously, the concept of embedded data in graphics objects does not require explicit conversion of data between application and visualization subsystem, and thereby an erroneous data redundancy between both components is prevented. The visualization directly accesses the objects of the application and transfers the information contained in the application into a format which can be processed by the low-level rendering engines underlying the visualization toolkit. This transfer is done each time an application 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 application objects such that cached data is always updated if application objects have been modified.

The tight coupling of application and visualization subsystem offers extensible and customizable visualization techniques to GIS applications. For an application of our approach, see Bernard et al. (1998) who describes an interoperable object-oriented GIS-framework for managing, modeling, and visualizing high-dimensional spatio-temporal data by an integrated system based on MAM/VRS and the iterator concept. Figure 5 shows sample visualizations taken from this application

3.2 Interactive Modification of Geo Objects

As a consequence of the tight coupling of GIS and visualization subsystem, it is easy to implement interactive modification techniques for geo objects. The visualization toolkit MAM/VRS provides *tag objects* to assign application-specific identifiers and group identifiers to visual objects. Tags represent the hooks for any type of geo object editor. Tags are created by visualization view classes and are associated with the graphical representations of geo 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 geo object editor. In addition, interaction techniques can use the built-in ray tracer to inquire spatial relationships of 3D objects.

Since geo objects and their visual representations are connected by iterators, modifications apply directly to the visual representation as well as to the application. 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 application objects.

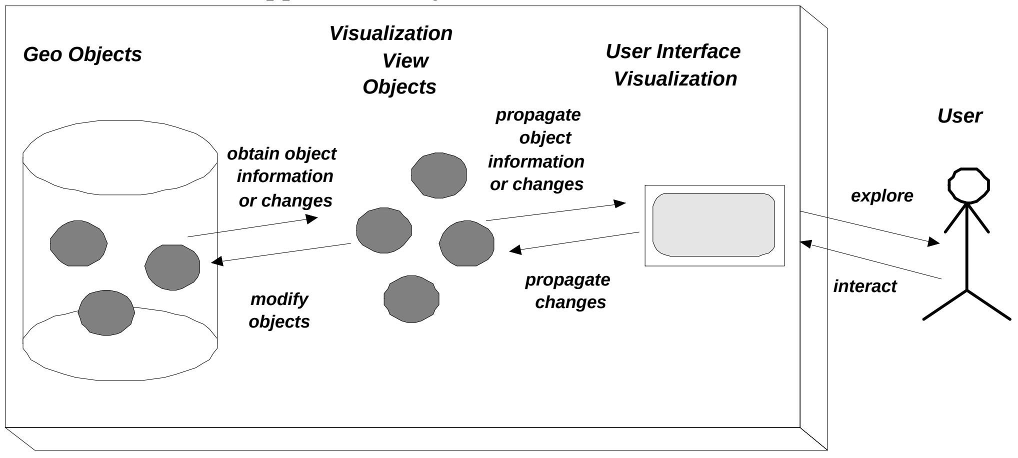
Geo Objects
Visualization View Objects
User Interface Visualization
User
obtain object information or changes
modify objects
propagate object information or changes
propagate changes
explore
interact

Figure 6: *Interaction between geo objects and their visual representations.*

3.3 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 (Ousterhout, 1998). Their main disadvantage is that they cannot implement large object-oriented systems. The interpretative tool command language *Tcl* (Ousterhout, 1994), its GUI toolkit *Tk*, and the object-oriented Tcl extension *[incr Tcl]* (McLennan, 1995) together represent a widely used and one of the most powerful object-oriented scripting languages.

The visualization toolkit MAM/VRS is implemented by C++ libraries. To integrate the system at the application programming interface level, we have embedded it 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 visualization toolkit is available through the [incr Tcl] interface. However, there is no lack of performance since the visualization functionality is not executed in the interpretative scripting language but in the native C++ code.

If the application developer extends the visualization toolkit 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 MAM/VRS which should also be available to the [incr Tcl] interface. To support the automatic integration of application-specific classes, a generator 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] also proved to be a valuable tool for developing user interfaces. 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++.

4 Conclusions

GIS application development can profit from the integration of object-oriented visualization toolkits. This integration saves development and implementation effort and ensures that up-to-date visualization techniques are available to GIS applications.

We identified customizability as one of the main criteria for choosing a visualization toolkit. Most likely, no visualization toolkit will offer all the primitives or techniques required for a concrete GIS application. Therefore, the ability to customize the toolkit by means of object-oriented techniques such as subclassing and template instantiation represents a very important characteristic a visualization toolkit should provide.

Geo objects and their visual representations are best linked by coupling both types of objects using mediators, e.g., iterator objects. In this way, both types of objects remain independent and can be developed within their own class hierarchy, but can communicate efficiently. In our experience, writing iterator classes can be easily implemented since they are just wrappers for existing GIS and visualization classes.

Time management features and animation features play a central role when visualizing the dynamics of data. The visualization toolkit should provide building blocks for specifying time flows and animation processes, otherwise animations have to be realized by the GIS developer.

Visualization toolkits for GIS should be able to use different rendering systems. Although today OpenGL represents a stable and powerful platform, this might change in the future due to the development of new rendering techniques. The exchange of a rendering system should not cause major code revisions but should be supported by the visualization toolkit like in the case of the Virtual Rendering System VRS.

GIS development can further be facilitated by an object-oriented scripting language which interfaces both GIS objects and visualization objects. It allows for interactive and rapid prototyping of database, visualization, and user interface components. The complete functionality of the visualization toolkit is accessible within the scripting language without a significant loss of performance if the scripting language extension is based on delegation to the C++ implementation.

Our future work includes the design and implementation of a geo visualization toolkit and the development of visualization strategies for high-dimensional dynamic geo-data. Along with this project, the MAM/VRS visualization library will be extended and provided under the GNU library public license (`http://wwwmath.uni-muenster.de/~mam`).

Acknowledgments – The authors would like to express their appreciation to Oliver Kersting and Tobias Gloth for their contributions to the MAM/VRS toolkit.

References

Bernard, L., Schmidt, B., Streit, U. and Uhlenküken, C. (1998) Managing, Modeling, and Visualizing High-dimensional Spatio-temporal Data in an Integrated System. *GeoInformatica* **2**(1), 59-77.

Döllner, J. and Hinrichs, K. (1997a) Object-Oriented 3D Modeling, Animation. *Journal of Visualization and Computer Animation* **8**(1), 33-64.

Döllner, J. and Hinrichs, K. (1997b) The Design of a 3D Rendering Meta System. Proceedings of *Eurographics Workshop on Programming Paradigms for Graphics `97,* Budapest, Hungary.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995) *Design Patterns. Elements of Reusable Object-Oriented Software,* Addison-Wesley.

MacEachren, A. (1994) Time as A Cartographic Variable. In *Visualization in Geographical Information Systems,* eds. Hearnshaw, H. and Unwin D. J., pp. 115-130. John Wiley & Sons Ltd.

McLennan, M. (1995) [incr Tcl] Object-Oriented Programming in Tcl/Tk. *Proceedings of the Tcl/Tk Workshop 1993.*

Morehouse, S. (1989) The Architecture of ARC/INFO. *Auto-Carto 9 Conference*, 266-277.

Ousterhout, J. (1994) *Tcl and the Tk Toolkit*, Addison-Wesley.

Ousterhout, J. (1998) Scripting: Higher Level Programming for the 21st Century. *IEEE Computer* **31**(3), 23-30.

Phillips, M., Levy, S. and Munzner, T. (1993) Geomview: An Interactive Geometry Viewer. *Notices Am. Math. Soc.* **40**(8), 985-988.

POV Team (1998) *Persistency of Vision Ray Tracer (POV-Ray)*. Version 3.0, Technical Report, 1998.

Schroeder, W., Martin K. and Lorensen, B. (1998) *The Visualization Toolkit*, 2nd edn., Prentice Hall.

Slocum, T. (1994) Visualization Software Tools. In *Visualization in Modern Cartography*, eds. MacEachren, A. M. and Taylor, D. R. F., pp. 91-96, Pergamon Press, Oxford.

Upson, C., Faulhaber, Th., Kamins, D., Laidlaw, D., Schlegel, D., Vroom, J., Gurwitz, R. and van Dam, A. (1989) The Application Visualization System: A Computational Environment for Scientific Visualization. *IEEE Computer Graphics & Applications* **9**(4), 30-42.

Upstill, St. (1990) *The RenderMan Companion. A Programmers's Guide to Realistic Computer Graphics*, Addison-Wesley.

Ward, G. (1994) The Radiance Lighting Simulation and Rendering System. *Computer Graphics (Proceedings of SIGGRAPH '94)*, **28**(2), 459-472.

Wernecke, J. (1994) *The Inventor Mentor : Programming Object-Oriented 3D Graphics with Open Inventor*, Addison-Wesley.

Woo, M., Neider, J. and Davis, T. (1997) *OpenGL Programming Guide*, 2nd edn., Addison Wesley.