An Object-Oriented Solution to an Interdisciplinary 3D Visualization Tool

Craig Sinclair, Todd Little, M. Ahsan Rahi

Summary

This paper describes the design and implementation of an interdisciplinary 3D visualization tool. An object-oriented approach was used to overcome deficiencies within 3rd party graphical toolkits and incompatible data formats. It also facilitated multidisciplinary visualization by providing a coherent framework for combining drawing technologies from the different discipline. The tool kit was designed to be extensible and supportable since it was expected to continually evolve to meet new market requirements.

Introduction

Improvements in computational and acquisition technology has rapidly been met by geoscientists developing more complex geophysical, geological and reservoir simulation models. These models have become so complex that to understand them requires interactive three dimensional visualization. For studies that integrate these models an integrated visualization tool helps gain insight into the interrelationship of the data as it progresses through each geoscience discipline.

We set out to provide a single visualization tool that addresses the needs of the geoscience disciplines to facilitate an integrated approach to exploration and production. Our object-oriented approach gives a practical, extensible and supportable solution for applications developers to provide end users with the desired multidisciplinary visualization capability.

The problems that arise when dealing with visualization for multiple disciplines are largely due to the dissimilar representation of similar underlying data. Geophysical models deal with large volumes of geometrically regular data whereas reservoir simulation models typically deal with smaller volumes of irregular data. To develop a visualization tool suitable for either of these cases requires exploiting the implicit nature of the data to optimize the drawing and memory management portions of the code. The prevailing approach in the industry thus far has been to create separate specialized programs for the various disciplines. This has led to applications that interact on the data level but cannot render data simultaneously to a single scene.

Object oriented technology is well suited to creating a visualization tool that must combine similar but different graphical grid types, data models and user interfaces. Being able to derive specialized classes from a base class allows common implementation to be shared between radically different derivations of the same item, such as a grid. Objects can be implemented for the different disciplines with optimized rendering, data storage and memory management routines. A single application will not need to be aware of which implementation is used but will interact with all the objects through a common interface in a seamless manner. In this way data from different disciplines can be viewed together in a single scene.

Current Three Dimensional Technology Alternatives

Three dimensional visualization technology is rapidly expanding, creating a period of transition where technology tries to provide solutions for new domains. Display list systems (also called retained mode), such as HOOPS[1] or PHIGS, have been very successful in three dimensional CAD applications. However, scientific application developers have found it difficult to use this paradigm due to the dynamic nature and sheer volume of scientific data. Dynamic data changes the scene from frame to frame requiring the graphics data base in a display list system to be continuously edited and rendered. Also, using a graphics data base causes duplication of information and requires implicit structures to be defined in explicit geometry. These problems in using display based systems have pushed geoscience application developers towards immediate mode libraries such as Silicon Graphics' GL which exhibits better performance and resource utilization for dynamic data. However, immediate mode systems are not without their drawbacks either. Display list systems currently provide the only acceptable solution to scalable hardcopy. They also provide better portability and features than immediate mode systems. It is apparent that neither system by themselves can provide a complete solution for three dimensional visualization of the type envisioned here. A system which combines the best of both graphical system would provided the best solution.

Visualization environments such as AVS, IBM's Data Explorer and SGI's IRIS Explorer have been used for visualizing geoscience data. These systems provide well-defined environments in which visual networks create three dimensional scenes from data sources. A major problem with these systems is the amount of resources necessary to render pictures. There are two reasons for this heavy demand: the systems require data to be mapped to internally supported data types and furthermore, the

data flow architecture tightly ties the data to the visualization. Mapping data from one form to another can result in duplicate copies of the data; one for the application and one for the visualization system. The second cause, that of tying the data to the visualization occurs because the visual networks are set up using a data flow methodology. This requires all input stubs of a node to be primed before the node will fire. Therefore, in typical networks, all the geometry and attributes required for a scene must be supplied at the same time resulting in an excessive amount of system resources (i.e., memory) being used. This problem is further compounded when viewing data from multiple disciplines in the same scene.

Integrated Visualization

Simple three dimensional viewing works well for static data. However, with interdisciplinary analyses, data can no longer be considered a static entity. To effectively handle integration, data should be thought of as a stream or evolution of numbers. Each stream has several sources of input, which, through calculations and 'massaging', produce several output branches. The output branches are used to feed the input branches of subsequent streams. For example, geological zone data from wells is converted, through interpretation and calculations, into grid layers. The layers are then used to generate a reservoir grid and upscaled attributes for reservoir engineering. There is a many-to-many relationship between the input and output branches. To provide an iterative approach to seismic, geological and engineering disciplines, the streams as well as the data must be retained. As changes are made to the source branches, the change must be propagated through all the streams that are interconnected. The transition within the streams can be visually verified by displaying the input and output branches together. Three dimensional concurrent viewing of input and output sources provides a powerful tool for editing and viewing the iterative propagation of dynamic data through the disciplines.

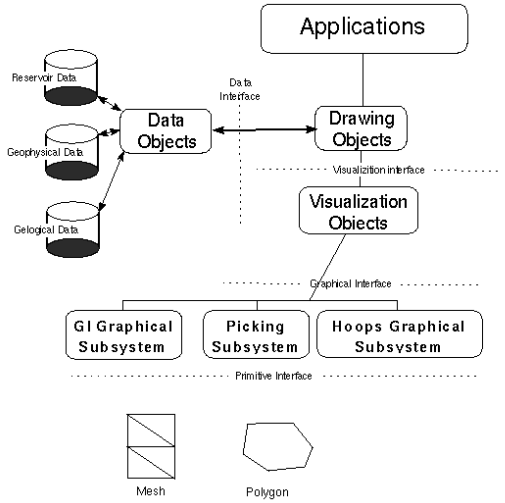
Object Oriented Solution

At the time this project was initiated there were no graphical layers or environments that addressed all our concerns for speed, efficiency, flexibility and features. It was decided therefore to develop a tool kit that would provide the necessary components to create integrated applications. The tool kit would provide high level abstractions for each discipline and use existing graphical systems for the actual rendering.

Objects provide a nice paradigm for creating the building blocks for an integrated viewer. This can best be illustrated through the example of a grid. Each discipline has the concept of a "grid" making it an ideal candidate to be shared. But is a seismic grid the same as a reservoir engineering grid? This is certainly implied by the name, but when you start talking to a geophysicist and an engineer it becomes apparent that there are fundamental differences in the manner each of these disciplines defines a grid. There is still some commonality in that they both describe an array of numbers with a logical ordering. In our design a Grid Object is used to represent grid data and its corresponding topology.Common elements such as the data and functions which define the logical ordering are part of the Grid Object class. The specialization required for each discipline are part of the objects derived from the Grid Object class. This separates out only the unique functionality keeping the commonality grouped together.

Architecture

The basic concept in our object oriented design is to use objects to create high level tools called drawing objects that can display geophysical, geological or reservoir data. The raw input data and the subsequent graphical representation by the underlying graphic subsystems is completely separated from the implementation of the Drawing Objects, see figure 1. Access to the data and the visualization system is through specialized objects that implement a generic interface.

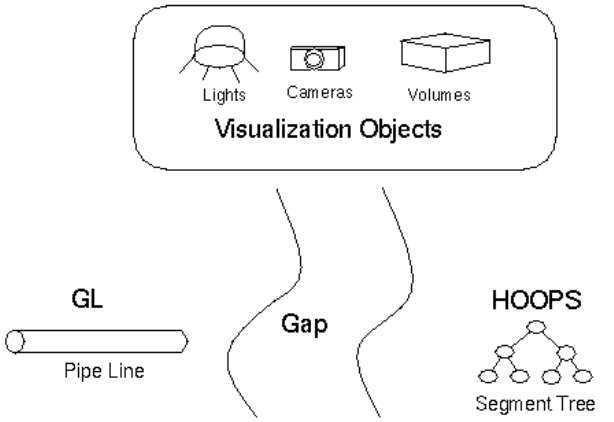


The Visualization objects dynamically switch between GL, HOOPS (for PostScript, image and CGM output) and picking. This design is a hybrid between driver technology found in classical functional graphic systems and a completely object based system.
Removing the data from the graphic implementation was done for efficiency and to allow greater flexibility in accessing different data sources. When dealing with large volumes of data, it is often more efficient and sometimes necessary to handle the data outside the geometric database or system. Our experience indicates geoscience applications have a better understanding of their data than

the underlying visualization system and can therefore use implicit information to store and retrieve it more effectively.

Separating the data, visualization and high level objects in this manner provides a versatile system. Data access and output can be programmatically selected allowing a great deal of flexibility within an application. Since the objects interact in a generic fashion a building block approach can be taken in constructing the application. Objects can be combined differently to provide multidisicplined integrated 3D-viewers or a 3D-viewer of a single discipline and data source.

Visualization Objects

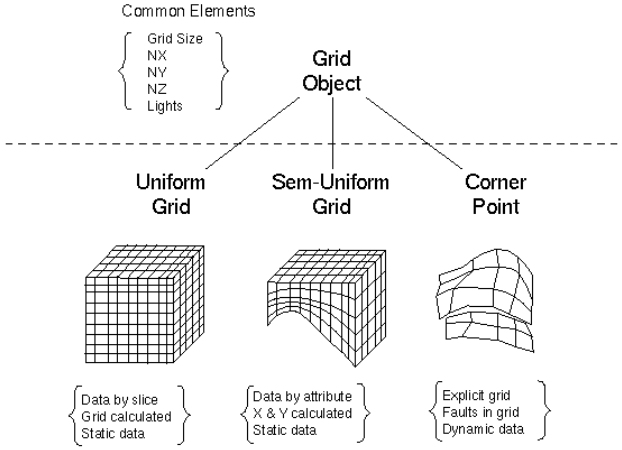


The visualization objects insulate the application from the underlying graphic subsystems. The objects dynamically handle identical graphical requests based on which graphic system (i.e., GL or HOOPS) is selected programmatically. This approach was used to obtain adequate interactive drawing times for dynamic data through Silicon Graphic's GL library. However, HOOPS, a display list system, is used to render a picture to hardcopy or an X display. The visualization layer, see Figure 2, goes beyond just providing a procedural interface, it bridges the fundamental differences between display based and immediate mode architectures. There is not enough commonality between display list and immediate modes to allow a simple procedural mapping to work. The Visualization Objects provide generic calls that are independent of the graphics systems by retaining enough information to render to either system. The amount of information stored for windows, viewports, cameras, etc. is small compared to the geoscience data and does not severely impact memory usage. In effect, the system is semi-display based where graphical objects are retained making it easy for an application to use them while having data, the resource killer, used in an immediate mode fashion.

All the visualization objects dynamically switch from one driver to the next. Unfortunately, inheritance in C++ is not dynamic after an object has been instantiated. This meant the polymorphic nature of objects was not sufficient to implement the dynamic behavior. Instead, our Visualization Objects use functional pointers to allow an object to alter how it handles calls. To change from GL to HOOPS Visualization Objects simply change their pointers. Visualizations objects which are derived from other Visualization Objects (i.e., a Graphics Area which is a specialized window for rendering three dimensional pictures) must change their whole inheritance tree. All our class definitions are composed of generic functions (functions that are common to all graphic systems), dynamic functions (functions that depend on the graphic system with corresponding pointers) and state information (encapsulated data that is common to all graphic systems).

Drawing Objects

Drawing Objects are our high level implementations of grids and other data types. Each different Drawing Object contains an optimized drawing algorithm for the area it deals with. The implementation is graphic system independent. Rendering by the drawing objects is done through the visualization objects.



Currently, we have implemented three types of grids: uniform, semi-uniform and corner point. Each grid has logically ordered data of size NX by NY by NZ. However, the geometric descriptions are different. The uniform grid is geometrically constant in three dimensions. This means that the geometry can be implied and calculated as needed. The semi-uniform grid is geometrically constant areally, but varies in the Z direction. Therefore the Z grid corner points must be defined by NX+1 by NY+1 by NZ+1 values, while X and Y corners are implied. Corner point grids allow discontinuities in all directions and are thus defined using 8 values of X, Y, Z per cell.

Our uniform, semi-uniform and corner point are different types of logical grids so they inherit the Grid Object class, see Figure 3. The derived classes contain data and operations specific to each grid. For example, a corner point grid optimizes drawing by removing faces that are invisible. It does this by maintaining a list of faces that are adjacent and therefore invisible if they are in the interior of grid. However, in Uniform grids this optimization is not required because all faces are guaranteed to be adjacent to another face if they are not on a boundary. Although the list of differences among the grids can be extensive, it does not result in the duplication of functionality; common data and functions are shared through the inheritance of the GridObject class.

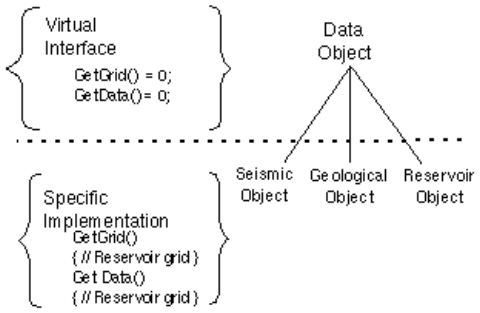
Grid Objects define a virtual draw method which must be implemented in the derived classes. Applications draw grids by accessing this method and can ignore what type of grid is actually being shown. The draw method, in turn, renders their cell representation accessing methods defined in the Visualization Objects (i.e., polygons or meshes).

This design has shown to be extensible over time. A prototype seismic viewer was implemented and then shelved for 12 months. When it was resurrected the compile indicated all the modifications that must be made to the interface. By making minor adjustments the seismic viewer was updated in seven days even though there was a significantly revamped data base and graphical toolkit sitting underneath the application.

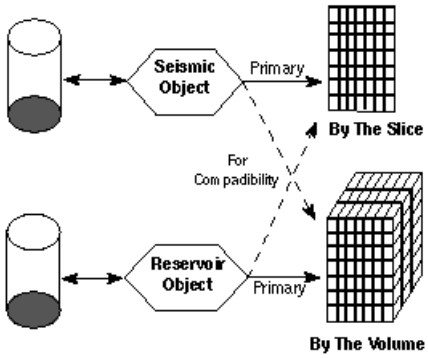
A key component to our system is how the Grid Objects efficiently and portably access data. Grid Objects are designed in a demand driven [4] fashion which means that they make requests to Data Objects at the time they require the data and not before. A demand driven system, by definition, does not propagate data until it is required. When the data is requested there is a backward chain of requests until the data is returned. In the context of the Grid Objects this simply means the grid data is loaded from outside sources when it is required in the rendering algorithm and not before. Therefore, the Grid Objects can trade off between memory requirements and access speed to optimize drawing.

Data Object

Data Objects contain functions that feed data to the Grid Objects. The functions declared within the Data Object are pure virtual functions, see figure 4a, which means the actual implementations for the functions must be defined in a derived class. In other words, for a Grid Object to be used, an object must be defined which is derived from a Data Object and this object must implement the pure virtual functions declared in the Data Object. The derived object will typically do this by accessing a specific source of data. Currently, we have created three derived types, one for 3D seismic data, one for geological data and one for reservoir data. The reservoir Data Object has been further refined to access different sources of reservoir data.



Any object derived from a Data Object can feed data to an object derived from a Grid Object, see figure 4b. This allows objects to be mixed and matched within applications providing the programmer with the building blocks to create integrated viewers. This process is simplified by the fact that all Data Objects are self contained and communicate through a well-defined interface.



Application

Applications can be built by combining Data Objects, Grid Objects and Visualization Objects in different combinations. For example, a viewer which displays reservoir data in a Motif Window can be created by using a Data Object that reads reservoir data, a corner point Grid Object and a Visualization Object for Motif windows, see figure 5. This can be then repeated for geological data creating an application which displays two types of data. To create an integrated viewer the Grid Objects are simply combined into one window. The Grid Objects are all derived from the same base class so the application can ignore the fact that the grids are displaying different types of data in the same window.

Several applications have been created on top of Grid Objects and Data Objects such as a seismic viewer, a geological viewer, 3DVIEW a reservoir visualization system, and an integrated viewer which displays geological and reservoir data. Each application has the ability to use multiple scenes. This allows the user to create multiple instances of the same grid, different grids (within the same a model), different models or even different types of models. The integrated viewer can also incorporate multiple grids into each scene. The additional grid can be the same grid, a different grid, a grid from a different model or from a different discipline. In interactive mode each viewing area is brought up as a Motif Main Window providing the user with all the familiar window controls provided by Motif. In hard copy mode each scene is created as a separate window and layered on the page as they appeared on the screen.

Conclusion

The design and use of object oriented technology has resulted in a system that is flexible, extensible and portable. We have created reusable components for the different geoscience disciplines. This has simplified implementation and maintenance activities such as testing, porting and debugging. Updates, re-organization and concurrent development have thus been possible throughout the life cycle of the system.

The geoscience objects provide the framework for creating integrated applications. Grid Objects and Data Objects can be mixed together in various fashions providing a flexible system for combining data from different disciplines. Interdisciplinary analysis is now augmented by simultaneously viewing different geoscience data within a single application.

References

1. Wiegand, G., Covey, R., and Couch, P.: HOOPS Reference Manual (Version 3.2)

2. Stroustrup, B.: The C++ Programming Language -- 2nd ed., Addison-Wesley Co., New York (1993).

3. Bahrs, P., Dominick, W., and Moreau, D.:"GO||: An Object-Oriented Framework for Computer Graphics," Computer Graphics Using Object-Oriented Programming (1992) 111-136

4. Wadge, W. and Ashcroft, E.: Lucid, the Dataflow Programming Language, Academic Press, Inc., London (1985)