A General Design Pattern for Programs of Scene Graph and its Application in a Simulation Instance

Youyi Bi[1], Carlos Domínguez[2], and Houcine Hassan[2]
[1]School of Mechanical Engineering and Automation, Beihang University, Beijing, P. R. China
[2]School of Design Engineering, Polytechnic University of Valencia, Valencia, Spain

Abstract - *The prevalence and significance of programs of Scene Graph, which mainly concern with the simulations of 3D scenes (hereinafter referred to as graphic programs), are more and more noticeable in many realms, such as Virtual Reality and physical simulators. This paper presents a general design pattern, Data-Processing-Interaction (DPI), for such programs' development. This pattern explicitly divides the development into several necessary structures, functions and corresponding solutions, which accelerates the programming process as well as provides clear application architecture. Developers can attain rich alternatives of solutions and concise decision-making process under the DPI pattern. And an instance of a simulation of motion in 3D scene will be followed. This instance clearly demonstrates the convenient features of this pattern for the preliminary preparation, the lower cost and high-efficiency of the development process.*

Keywords: Scene Graph, Design Pattern, Virtual Reality

1 Introduction

One of the most impressive achievements brought by the information technology is that scientists and engineers are able to implement their experiments and designs in computer. Especially following the rapid development of Computer Graphics and high-efficiency graphic chips, people are attempting to simulate all kinds of processes in computer, including computer aided design, robot motion, virtual manufacturing and assembling, visualization in scientific computing, 3D geological information system, medical examination and of course the most direct ones—3D games. This tendency of the wide application mainly derives from two aspects of simulation programs. First, the features of immersion and interaction greatly improve the user's sensation and intimacy with machines, especially for the fields of education, exposition, and entertainment. Another important aspect reflects on the much lower cost and higher efficiency than real physical simulation in scientific and engineering realms. Mechanical designers can detect the fatal flaws of their designs by executing interference and kinetic examinations on computer avoiding being inspected as physical entities with higher cost.

In many circumstances, a simulation program is just a simple graphic program, for which there is no necessity to build those complicated design documents under the normal regulations of Software Engineering. A scene graph [1] is a general data structure that arranges the logical and often spatial representation of a graphical scene. It is commonly used by vector-based graphics editing applications and modern computer games. The most direct example of the application of Scene Graph is virtual tour, which enables people to visit museums or palaces on computer freely and immersively.

Thus, the importance of graphic programs is calling for an easy and swift design pattern for developers. The first problem for those beginners in this filed usually is "which step should I start from?" Some researchers have already proposed their design patterns in the researches regarding graphic simulation. Closely related, C. Marcu [2] put forward a reasonable structure for his simulation program of a Fanuc M-6iB/2HS articulated robot. This structure consists of OpenGL Scene, Motion control thread, Mathematical models and the Main interface, which distinguishes the program into different functions orderly. Fu Qiang [3] also raised a MVC (Model-View-Control) design pattern in his Software for Quality Evaluation of Seismic Acquisition Data. Martin Eigner [4] presented an application to investigate and manipulate JT data, a neutral data format for products, due to the better efficiency and convenience of managing and communicating product information and data in enterprise networks. In this application, the author put forward a two-layer underlying framework, including the toolkit layer (Qt toolkit, OSG toolkit and JT toolkit) and the application layer (Extensible viewer).

However, none of them illustrated their design patterns specifically. They didn't present the details for how to prepare and implement the development process corresponding to their design patterns. Moreover, their design patterns are lack in the necessary simplicity and comprehensiveness. For example, interface between the machine and user should include other interactions like haptics and force feedback functions. Developers need a more efficient and straight design pattern.

This paper firstly focuses on the concept of the DPI design pattern in Section 2, presenting a three-layer structure and the connections between each layer. With detailed information regarding the tools used in graphic programs, the DPI pattern provides developers with concise developing principles and plentiful alternatives of solutions. Section 3 then depicts a developing instance to show the typical application of the DPI pattern and several noticeable details in the actual developing process, before Section 4 wraps up by giving a conclusion and outlook in terms of future work.

2 The DPI design pattern

The design pattern of DPI (Data-Processing-Interaction) is able to provide developers with a more explicit programming process. Most required functions of the program can be allocated into the three layers of the DPI pattern. Each layer deals with certain demands and connects with other layers smoothly. Figure 1 illustrates the basic structure of the DPI pattern.

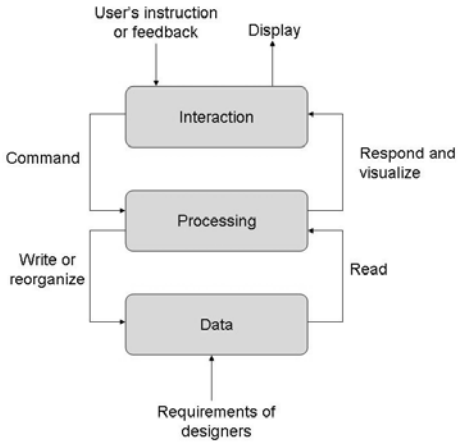

Figure 1. The basic structure of the DPI pattern

2.1 The Data Layer

Data files are the most fundamental elements to comprise a scene graph. Normally, there are five kinds of data files, and table 1 indicates their types and descriptions.

Table 1. Common data types in graphic programs

Data type	Common formats	Roles
Model	OBJ, FLT, 3DS, DAE	For rendering 3D model objects
Video	AVI, MPG, RM, WMA	For rendering dynamic texture, and multimedia
Image	BMP, GIF,JPG, PNG	For rendering texture data
Font	TTF, TTC, FON,PFB	For displaying of words and multilingual support
Others	WMV, MP3, PDF	For auxiliary effects and designers' special requirements

The most important data used for graphic programs is model, which is mainly built by various kinds of CAD software or by drawing functions from those graphic libraries like OpenGL directly in some cases. One kernel problem to be solved in the data layer is to choose reasonable modeling software. For those projects which need high mathematical or physical accuracy or high visual effect like 3D games, large commercial software is indispensable. Otherwise, those free and open-source ones are sufficient. Table 2 presents the corresponding features of the major modeling software. Developers should select the most suitable modeling software according to the actual requirements and financial conditions.

Table 2. Major modeling software's features

Name	Cost	Platform	Applications
Autodesk Inventor	$7,995	MS Windows	Creating 3D digital prototypes, visualization and simulation of products
Pro/Engineer	$4,995	MS Windows HP-UX, Unix	Solid modeling, assembly modeling and drafting, finite element analysis, and NC and tooling functionality
Solidworks	$3,995	MS Windows	CAD of industrial products
NX(UG)	$8,700	MS Windows Mac OS X	Computer-aided mechanical design, manufacturing and engineering
CATIA	$15,000	MS Windows, IBMAIX, HP-UX, Solaris	Industrial equipment, defense aerospace, automotive, consumer packaged goods
3DS Max	$3,495	MS Windows	Animation, Modeling, Visual 3D Effects
Blender	Free	MS Windows, Solaris, Linux, Mac OS X	Animation, Modeling, Visual 3D Effects. It can be even run in Pocket PC,

Another important problem is to select appropriate file formats for the models. And the selection should follow three factors. Accuracy is the primary consideration. In graphic programs, those essential information regarding the models, including geometric dimensions, colour and light, and textures, should be kept completely and consistently. Another factor is portability. There is no guarantee that a model will be used in only one application. Those neutral file formats can bring much convenience for further development. Finally, the compressibility of the model data should be concerned. Once the total volume of the model data exceeds the normal conditions of computer's hardware, developers will be busy with upgrading their computers. Table 3 indicates the features of major graphic model formats.

Table 3. The features of major general 3D model formats

Format	Origin	Descriptions and Applications
3DS	3DS Max	Open model data format of 3D studio software
AC	AC3D	Stored by the text format, used in FlightGear for scenery objects and aircraft models
DAE	COLLADA	A general open 3D model data exchange standard, using XML syntax, supported by the COLLADA library
DXF	AutoCAD	Drawing Exchange Format is developed for enabling data interoperability between AutoCAD and other programs.

IGES	Graphic standards	Widely-used format, composed of 80-character ASCII records
OBJ	Wavefront	Open and universally accepted. Including the position of each vertex, the UV position of each texture coordinate vertex, normal.
STL	Stereo-lithography CAD	Describing raw triangulated surface by the unit normal and vertices of the triangle, widely used for rapid prototyping and computer-aided manufacturing.

To sum up, choosing a suitable modeling software and file format suggests a well beginning of the development.

2.2 The Processing Layer

The processing layer manages the model data and communicates with the Interaction layer. The main functions of this layer can be classified into three aspects—data structure, visualization and specific algorithms. Figure 2 shows the structure of the Processing layer.

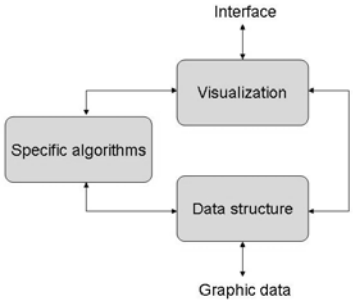

Figure 2. The structure of the Processing layer

Obviously, the primary task for the Processing layer is to organize the model data from the former layer. A well organized data structure benefits both the programmers and model builders. Actually, a scene graph is a collection of nodes in a graph or tree structure. A node may have many children but often only a single parent, with the effect of a parent applied to all its child nodes; an operation performed on a group automatically propagates its effect to all of its members. A common feature, for instance, is the ability to group related objects into a compound object that can then be moved, transformed, selected, as easily as a single object.

Furthermore, some software development kits (SDK) even encapsulate the tree structure functions into their own classes, like the OpenSceneGraph (OSG). In OSG [5], graphic model data can be managed as a leaf node, which is the most basic unit. Larger scene or more sophisticated models can comprise these basic nodes and they can be organized by superior nodes or so-called groups and so on. In this case, those algorithms from data structure, familiar to developers, like the depth-first traversal algorithm, can be applied directly.

Visualization is the central function of the Processing layer. Normally, 2D graphics can be drew and displayed straightly by the interior drawing classes of the programming language, such as the *CDC* class in Microsoft Foundation Class (MFC) and the *java.awt.Graphics2D* class in Java. For 3D graphics, OpenGL and DirectX are the most fundamental tools, which can create almost all kinds of graphs theoretically. However, OpenGL [6] is a low level library that takes lists of simple polygons and renders them as quickly as possible. To do something more practical, the programmer must break down the object into a series of simple OpenGL instructions and send them into the engine for rendering. For simple programs a tremendous amount of programming has to be done just to get started. Hence those Three-dimensional rendering engines based on such industrial standards are more popular for their integration of many common functions. An evident example is reading 3D models in graphic programs. It would be more strenuous if using OpenGL functions. Table 4 presents the characteristics of major graphics rending engines (Running platforms are inside the brackets).

Table 4. The characteristics of major graphics rendering engines

Rendering engine	Description and Application
Open Inventor (MS Windows, Linux, Unix)	A C++ object-oriented retained mode 3D graphics API designed by SGI to provide a higher layer of programming for OpenGL
Unreal (MS Windows, Linux, Mac OS Xbox, PS2 , PS3)	The most famous commercial 3D game engine, designed by Epic Games, based on C++
OpenSG (MS Windows, Linux，Solaris, Mac OS X)	A scene graph system to create real-time graphics programs, Open Source, and can be used freely, based on OpenGL, Well performed in clustering support and advanced multithreading.
OERG 3D (MS Windows, Linux，Solaris, Mac OS X)	A scene-oriented, flexible 3D rendering engine, supported by OpenGL or DirectX, realizing the scene octree, BSP tree, CLOD and paging mechanism
OpenSceneGraph (MS Windows UNIX/Linux, Mac OS X, Solaris)	An open source 3D graphics API, written in standard C++ using OpenGL, used in fields such as visual simulation, computer games, virtual reality, scientific visualization and modelling.

Additionally, specific algorithms are necessary for engineering requirements and better realistic sense. Collision detection [7] algorithm is a widely-applied example. Collision detection typically refers to the computational problem of detecting the intersection of two or more objects, which is indispensable in the simulation of robots, first-player shooter games and other physical simulations.

2.3 The Interaction Layer

The central function for the Interaction layer is to respond all kinds of commands and signals from users and transmit them to the Processing layer. The Interaction layer normally consists of the Graphic User Interface (GUI) and functions dealing with other physical reactions, including the digital gloves, the haptics and force feedback devices or sensors if necessary. The separation of the Processing layer and the Interaction layer reflects the idea of encapsulation and modularity in the Object-Oriented design. Such separation

brings better maintainability, i.e. the Processing layer can avoid modification even the GUI needs to change.

Traditional interactive devices are mainly the keyboard and mouse, while other more realistic equipments are being introduced fast. The flight simulator to help training pilots is the pioneer one. Pilots are able to grasp basic controlling skills by operating joysticks while watching the virtual flight scene. Such simulation systems have made great success in the 3D game markets, no matter people wish to shoot, drive, or even play tennis in the virtual world.

The mainstream interfaces are optional abundantly, free or commercial, cross-platform or exclusive. As the great tendency of portable smart terminals (smart phone, tablet PC, etc…) and embedded devices, free and cross-platform developing environment will be more suitable for small-scale projects for lower cost and better portability.

2.4 The connections between each Layer

The connections between each layer in the micro level are reflected in the interfaces of functions. These interfaces play the same roles as those transfer stations and conveyors in factories, which help with visualizing 3D scenes and responding to users' instructions. Specifically, the connections between the Data layer and the Processing layer mainly deal with the input and output of data, i.e. reading and writing data files.

Large projects usually require data of numerous types, which is a big challenge for low-level graphic libraries. Fortunately, customized plugin technology has been applied widely in high-level graphic rendering engines. For example in OSG, it defines a file format extension mechanism that allows users to write their own specific data import and export plug-ins. Once a file format plugin is completed, people can spread it freely between different developers and use it directly, without having to recompile the source code of engineering systems. This mechanism greatly improves the universality of graphic programs.

The connection between the Processing layer and the Interaction layer is a typical trigger-respond relation. The most common examples are the "message-response" mechanism in MFC, the "event-listener" mechanism in Java and the "signal-slot" mechanism in Qt, which are similar theoretically. Figure 3 presents the connections between each layer in the details.

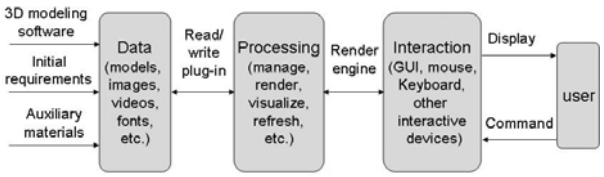

Figure 3. The connections between each layer

3 A developing instance

This instance comes from a project of Polytechnic University of Valencia: "the simulation of physical worlds to test software control agents". The actual developing of a simulation program makes the DPI pattern more convincing. The specific developing steps will be presented in this section.

3.1 Primary requirements

The main goal of this instance is to realize roaming in virtual scene. User can finally ramble in the virtual environment of a University teaching building. Through the keyboard, user can choose four directions (forward, backward, left and right) to move, rotate right or left, and make the view go up and down.

Since this project serves for the university research, another important property of this project is low cost, which means to avoid using commercial software for modeling and programming as much as possible. At the same time, because of this project being the initial part of a simulation of robot control and motion and the restrictions of computer conditions, the demand for the geometric accuracy and visual effect of scenes is moderate.

3.2 The pre-programming work

Following the above primary requirements, to determine the related work of each layer is the first step. According to the DPI pattern, the whole project is divided into several key problems in each layer. Table 5 shows the preliminary decomposition of this task. There is no necessity to make the items of this table specifically at first, for the solutions of each layer may interplay to some extent. And developers don't have to rush for starting modeling and programming immediately unless having made suitable decisions about the model format and render engine. Thus, work①③⑤ should be completed firstly in general cases.

Table 5. The decomposition of this task

The Data Layer	①To choose the modeling software and file format
	②To model objects
The Processing Layer	③To choose the graphic rendering engine
	④To program
The Interaction Layer	⑤To design graphic user interface
	⑥To program and test

Actually, the most common software for modeling buildings is 3DS Max. It's frequently used by video game developers, TV commercial studios, architectural visualization studios, movie effects and movie pre-visualization. The strong modeling function, fluent and high-performance of visual effect and the system of rich plug-ins contribute to its wide application around the world. But it

seems to cost unnecessarily to use commercial software according to the actual situations of this project. Therefore, Blender, a free and open source one is adopted.

Blender can be used for modeling, UV unwrapping, texturing, rigging, water and smoke simulations, skinning, animating, rendering, particle and other simulations, non-linear editing, compositing, and creating interactive 3D applications, including video games, animated film, or visual effects. The most amazing point of Blender is its incredibly small volume, for the application package of Blender 2.49 is only less than 32MB after installed. Its feature of cross platform even enables designers to do 3D design on mobile phones. Furthermore, Blender also provides various plug-ins for the import or export of special model file formats. These impressive features make Blender suitable modeling software for this project. Figure 4 shows the modeling process in Blender.

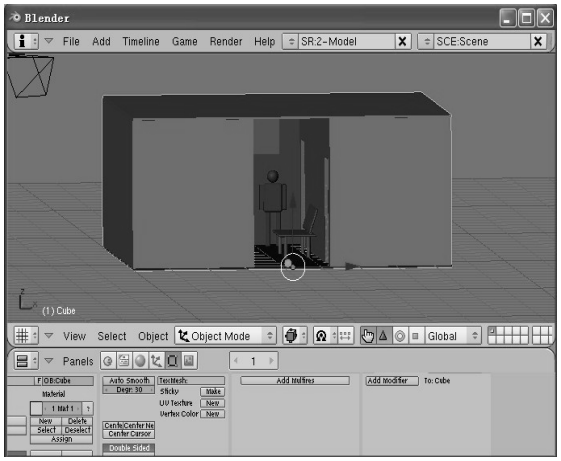

Figure 4. The modeling process in Blender

In the modeling process, especially for those large-scale projects, it is important to unify the plan and management of objects to be modeled before the formal formation. Designers should make clear of the significant properties of objects and even using the ideas of standardization and Group Technology. Usually the mechanical products could be classified into three kinds: special parts, similar parts and standard parts, and their respective proportion of the total parts is 10%, 70% and 20% approximately. Therefore, parts with similar geometric or other physical properties occupy the main aspects in the mechanical industry. Similarly, in this modeling process, similar objects can be built jointly to achieve high efficiency. In this project, the objects to be built are mainly the stuff in a teaching building, like offices, tables, benches, chairs, decorates and persons. Static objects, such as an office and inside facilities could be jointed as one node manipulated in scene graph.

And in selecting the file format for the model, the STL format was considered firstly for its feature of lightweight. STL is developed by 3D System Company and is known as the standard format for the description of parts in the Rapid Prototype field. In STL, geometric information is stored as the coordinates of the three vertexes of the triangle facet and its normal vector. An obvious advantage of the STL is the relatively low amount of data when storing models. However, when confronting the selection of the graphic rendering engine, the STL format was abandoned for its defect in the integration of color and texture information. Hence the selection of file formats and graphic engine is interplayed and it is reasonable to consider the characteristics of the rendering engine and the format of models comprehensively.

In the selection of graphic engines, OpenGL was the first consideration for its affluent development references on Internet and its wide application. However, the complexity of reading 3D model files and controlling view cameras leads to another option--- OSG, which is free and open source and very suitable for small-scale projects. As of 2006, the OSG-user mailing list exceeded 1,500 subscribers. OSG [8] is a high-level programming interface for 3D computer graphics, used in simulation, animation and visualization applications. It's built on OpenGL, which ensures that OSG is both cross platform and high performance. But it goes beyond OpenGL to provide functionality common to many 3D applications, such as 2D and 3D file loaders, texture mapped font support, level of detail control, threaded database paging and others. OSG has become widely accepted by both academia and industry for its rich feature set and liberal open source license.

3.3 The application design and result

After the decision of rendering engine, the design of interface and application architecture can be started. Figure 5 shows the main classes in this program. Among these classes, the Core_OSG class is responsible for calling the read/write mechanism of OSG library to read model data. The View_control class is in charge of calling the view control functions from OSG library. The Application class is served as the basic structure of this application. The interface deals with user's commands and the Dialog class provides users with extra control parameters.

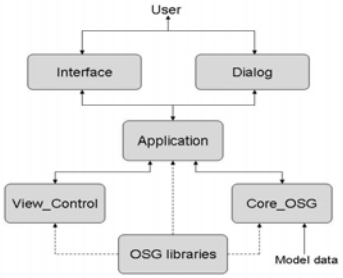

Figure 5. The main classes in the program.

Fortunately, Blender offers a special plug-in for OSG, which can export the Blender model file as the OSG format. As mentioned above, since the OSG encapsulates many advanced functions, like reading model files and controlling cameras directly, the programming process is simplified

greatly. Figure 6 is part of essential codes for controlling camera views of scene graph.

```
bool View_Control::handle(const osgGA::GUIEventAdapter &ea, osgGA::G
   UIActionAdapter &us)
{
        //get the coordinates of the mouse's cursor
          float mouseX = ea.getX();
          float mouseY = ea.getY();

          switch(ea.getEventType())
{
          case(osgGA::GUIEventAdapter::KEYDOWN):
                        {
  //press the "A" key or left arrow key, the view is moved left
            if (ea.getKey () == 0x41||ea.getKey () == 0x61)
{
ChangePosition(osg::Vec3 (0, m_fMoveSpeed * cosf(osg::PI_2+m_vRotation.
  _v[2]), 0)) ; //change the view location
ChangePosition(osg::Vec3  (-m_fMoveSpeed  *sinf(osg::PI_2+m_vRotation._
   v[2]), 0, 0)) ;
                        return true;
}
```

Figure 6. Essential codes for controlling camera views.

And the developing environment was chose as Qt finally. The most distinguished advantages of Qt are the portability, compatibility and flexibility. Using the Qt programming environment we can obtain a portable simulation program running in different operating systems, and even in the embedded devices. Qt framework is used for GUI, providing with all the functionality needed to develop advanced GUI applications on desktop and embedded platforms. Additionally, since Qt is a C++ based environment, a number of C++ libraries can be included for better development, such as OSG, which provides more advanced functions of rendering and view control based on OpenGL. After the decision of the three-layer‘s design, modeling and coding can start. Figure 7 shows the effect of main interface. The scene graph is presented in the view area. In the menu bar, user could read different files of scene graph to be simulated from the “File” menu. In the “Simulate” menu, user could enter related parameters to control the quantity and location of specific objects in scene graph.

Figure 7. The effect of main interface

4 Conclusions

The DPI provides developers with a simple and high-efficiency design pattern for graphic programs. The explicit classification of a task is the decisive point. Developers could quickly make clear of their thoughts and choose the most suitable solutions among those alternative options. As a matter of fact, the heavy work on writing complete developing document is simplified as only to solve several kernel problems. Once they are solved, the modeling and programming work can be executed almost simultaneously if the human resource is adequate, which will shorten the developing time further. The further work of DPI will be focused on establishing more elaborate and general connections between each layer and the possibility of auto-programming of graphic programs.

5 Acknowledgements

This work has been partially funded by Escuela Técnica Superior de Ingeniería del Diseño, Universidad Politécnica de Valencia.

6 References

[1] Avi Bar-Zeev. Scene graphs: Past, Present and Future , http://www.realityprime.com/articles/scenegraphs-past-present-and-future, 2007

[2] C. Marcu, Gh. Lazea, S. Herle, R. Robotin, L. Tamas. 3D Graphical Simulation of an Articulated Serial Manipulator based on Kinematic Models. 19th International Workshop on Robotics in Alpe-Adria-Danube Region, 2010.

[3] FU Qiang, XIAO Yunshi, YUE Jiguang. QESADSYS V1.0: A New Cross-Platform Software for Quality Evaluation of Seismic Acquisition Data. World Congress on Software Engineering, 2009.

[4] Martin Eigner, Florian Gerhardt. Extensible JT Open Tool for Prototypical Process Support, based on OpenSceneGraph and QT. 13th International Conference on Computer Supported Cooperative Work in Design, 2009.

[5] Paul Martz. OpenSceneGraph Quick Start guide http://www.openscenegraph.org/osgwiki/pmwiki.php/ Documentation/ QuickStartGuide

[6] Dave Shreiner. OpenGL programming guide, Addison-Wesley Publishing Company, 2009.

[7] Yang Shixing, Cao Mingliang. Step into OpenSceneGraph. Virtual Reality Lab of Zhengzhou University, 2008.

[8] Wang Rui, Qian Xuelei. The design and practice of OSG rendering engine, Tsinghua University Press, 2009.