Tutorial 4: Understanding functors

Franclin Foping
franclin@netcourrier.com

May 12, 2008

Abstract

Right now, you know how OSG works, but you have not yet written a piece of code in OSG. Don't worry we will start coding. In this tutorial I will continue the preparation process for OSG. This time, I will talk about functors: a powerful abstraction aiming at dividing interobjects communication.

1 Structure of the tutorial

As usual, the first section (section 2) of the tutorial will focus on the motivation. I will give an answer to the following question: why and where do we need functors in OSG? The tutorial will be closed by a conclusion.

2 Motivation

As you have learnt from previous tutorials, OSG makes use of STL and design patterns. However some important parts in OSG use functors. Although there are many tutorials on functors freely available on the internet, I think this one will be useful. In OSG, **callbacks** are implemented using functors so you have to get accustomed to them!

Alexandrescu (2001)
Gamma, Helm, Johnson & Vlissides (1995)

3 Introduction

Functors are a generalisation of the **command design pattern** described by Gamma et al. (1995). Basically, a functor is a class which overloads the function operator (**operator()**). The following listing shows an example of this concept.

```
1 class myNodeCallback: public osg::NodeCallback
2 {
3   public:
4    void operator()(osg::Node* node, osg::NodeVisitor* nv)
5          {//Do something here!}
6   private:
7    //Class data members
8 }
```

The most interesting part of the previous code is certainly what is written at the line 4. What is interesting here is the fact you can put information on the node in that class. This is critical and we will see how it works when I will talk about callbacks. For now, I just want you to understand the rationale of functors.

4 Sample usage in OSG

An example will be given in the second part of these tutorials when we will talk about dynamically updating nodes in our scene. For now, just to grasp the basic idea. Later on, things will be clearer.

5 Conclusion

Throughout this tutorial, I talk about installing OSG on Linux, hopefully everything went ok with you, if not please feel free to drop me an email. Even in this tutorial there is no exercise, we will soon start coding. It is important to install OSG on your Linux before we go on otherwise you will

be lost in the subsequent tutorials. The next tutorial will focus on some important C++ skills for OSG developers. These skills are so important that they deserve a separate tutorial. Fasten your belt because we will start hardcore coding in the next tutorial. It will also be the first tutorial with some exercises for you to practise, remember practice makes perfect!

References

Alexandrescu, A. (2001), *Modern C++ Design: Generic Programming and Design Patterns Applied*, Addison-Wesley.

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