INTERACTIVE MANIPULATION IN OSG LIBRARY

Vít Štancl
Doctoral Degree Programme (1), FIT BUT
E-mail: stancl@fit.vutbr.cz

Supervised by: Přemysl Kršek
E-mail: krsek@fit.vutbr.cz

ABSTRACT

In this paper we purpose to describe theory basis and implementation methods used for adding interactive manipulation possibility to Open Scene Graph library.

1. INTRODUCTION

Open scene graph (OSG) library does not straightly support interactive manipulations with objects. This paper describes library that adds this functionality. The term manipulation means for us translation and rotation (around one axis or, generally, rotation of trackball type).

This work uses the mouse device as an input source. There are two methods frequently used to interactively manipulate some on-screen element – drag and move is the first one, the second is: select and then use some proxy geometry to manipulate the object.

The same methods are frequently used in many 3D modeling. Inspiration can be searched in Open Inventor, Coin3D library, Gleem manipulating library or 3D Studio Max and similar modeling software.

The whole action can be divided into the several parts:

1) We must decide which object has to be manipulated. It can be called the *active object*.
2) The active object must optionally signalize its new state (it is selected) or show some additional geometry, which can be used to control the movement.
3) From mouse movement the algorithm must be able to decode what user wants.
4) It is often needed to send some messages to the application. At least the application must know that the scene graph is changing or it is changed.

The presented *Draggers library* covers all four parts of this process.

2. OSG LIBRARY

Open Scene Graph is 3D graphics library for C++ programmers. OSG is licensed as Open Source, so the source code is freely available, modifiable and redistributable under terms of the Open Scene Graph Public License (OSGPL).

The whole scene is represented with graph structure consisting of terminal and nonterminal nodes. It means that objects which share some properties (color, drawing mode, rotation, level of detail etc.) can be grouped and common properties can be set to a whole group.

There are several types of nodes in scene graph. Some of them handle geometry (`osg::Geode`), some are used for grouping (`osg::Group` class and its successors). Extensions of OSG library that define new nodes are often called *node kits*. Our node kit is named *Draggers library.*

3. DRAGGERS LIBRARY BASICS

As it was said, the whole system is build over the OSG library. A scene in OSG library is described by directed acyclic graph. This graph is composed of graphic nodes (implementing some 3D elements) and grouping nodes. One of the nodes is special – all nodes in graph have it as a parent. This node is titled *root*.

There is a special class of nodes in OSG library which is responsible for geometrical transforms of its children. It is `osg::MatrixTransform` (and its successors). It is derived from a group node and all included children are transformed with transformation matrix stored in this matrix node.

The hierarchy of classes is visible on figure 2. The new successor of matrix transform node is derived as the basic class for interactive sub tree manipulation – `Draggers::CManipulatingMatrix`. This class includes two groups of children. First one is used for normal sub-tree nodes, the second one contains special `osg::Group` derivate named `Draggers::CBasicDragger`.

This class implements manipulating methods that transforms preceding manipulating matrix. Existing successors implement rotation, translation and trackball algorithms. The second function of this class is to store proxy geometry that can be shown, when user selects handled object – for example stylized arrow for moving.

There are some new tasks connected with this solution:

1) To find the right transformation matrix in scene graph. There can be more manipulated objects in the scene and also more transformation matrices.
2) To transform mouse moves over the view window to 3D transformations. The manipulated geometry is in undefined position. User needs to move with it in predictable way.
3) To handle mouse events and to pass them to an algorithm connected with manipulation matrix node.

3.1. SELECTING THE OBJECT

When user presses the mouse button, program must decide which object is visible on the screen straightly under the mouse cursor. Possible way how to find the selected geometry node in OSG is to use *visitors* mechanism. Visitors in OSG are functional classes passed to the scene graph root node. The visitor traverses the whole scene graph and it can call methods of traversed nodes or it collects some information.

The intersection visitor is specialized version of basic visitor. It includes data structure which describes a line segment and it finds all intersections of this segment with scene

geometry. All intersections are stored in a list and they are sorted depending on distance from the beginning of the stored line segment.

We can send intersection visitor with a line segment directing straightly to the scene and parallel with a global z-axis. Two remaining coordinates (x and y) are given by position of the mouse cursor. Intersection visitor tries to find some intersections with scene geometry.

All what is needed to do is:

1) Find if there is a geometry node on the beginning of the list.
2) If it is, we must decide if it has to be interactively manipulated.
3) Then we must get the manipulating node (that contains manipulation matrix, as we said) and do the manipulations.

When user presses the button the object selection algorithm is:

```
Send intersection visitor with a line segment.
If there are intersections on the list then
    while (node <> root node) and (node <> manipulating matrix node) do
      node = node.parent
    If the node = manipulating matrix node then
      start manipulation (or show some additional geometry)
```

Now the active object is found and selected.

Active object can show additional geometry. In our implementation is this geometry used for requested transformation possibility choice. For example - dragging the stylized arrow causes linear translation of the active object.

3.2. CONVERTING THE MOVE OF THE MOUSE TO THE 3D-TRANSFORMATION

Mouse is moving in 2D space, scene and manipulated objects are in 3D space. When user wants to use a mouse for 3D object transformation, some translating function must be found between 2D and 3D spaces. This function must fulfill predictable effect expectations. Manipulation possibilities can be divided into three courses of study – rotation (around given fixed axis), translation (also along given fixed axis) and trackball simulation (free rotation). But first of all must be known, what is seen in the view window.

Position of visible object

Mouse movements can be transformed to the object space using *projection matrix* All transformations are stored in `osg::TransformMatrix` node class objects (or in inherited classes). To compute projection matrix, algorithm must traverse from object of interest to the root node and use every transformation matrix on the path. The resultant is then multiplied by scene projection matrix and window matrix (that stores scene-space to window pixel-space transformation). And the result is object-window projection matrix. It will called *OW matrix* in the following text.

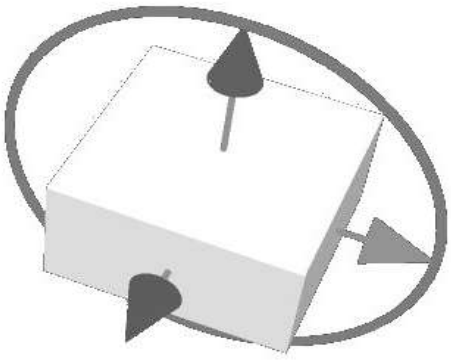

Figure 1: Manipulated object with proxy geometry visible

Rotating the object

Rotation in real world can be achieved using force and lever (or handle). User is accustomed to this principle. And the same method is used in Draggers library.

Rotation axis is defined when the Dragger object is created. Th lever length is calculated. Intersection point (IP) is defined as intersection of beam fired to scene from mouse cursor position and geometry that will be rotated. This point and its projection to the rotation axis defines length of the lever. Used force (value and direction) is obtained as a perpendicular part of mouse shift vector to the rotation axis vector projected on screen using OW matrix.

Moving the object

Move in Draggers library is defined as a simple translation along selected direction. This direction cannot be changed until the move is finished.

Move direction vector is predefined again. Using OW matrix, the predefined move direction is projected on the screen. Move length is defined as a projection of mouse shift vector to the projected move direction vector. This projection is computed as a dot product of these vectors. Afterwards the object is transformed.

Virtual local trackball

At the beginning of trackball usage two vectors are stored. These two vectors are projection of X and Y screen axes to the local object space using inverse OW matrix. Projected vectors define what user sees on screen as a new natural coordinate system for the selected object(s). Difference in X-axis direction is utilized as a rotation angle around projected Y-axis and complementary the difference in Y-axis direction is used for computing rotation angle around projected X-axis.

3.3. COMMUNICATION WITH OUTER WORLD

Draggers library uses own event handler class. This class is responsible for event processing or passing. Manipulation matrix group also implements callback function registration and checkout.

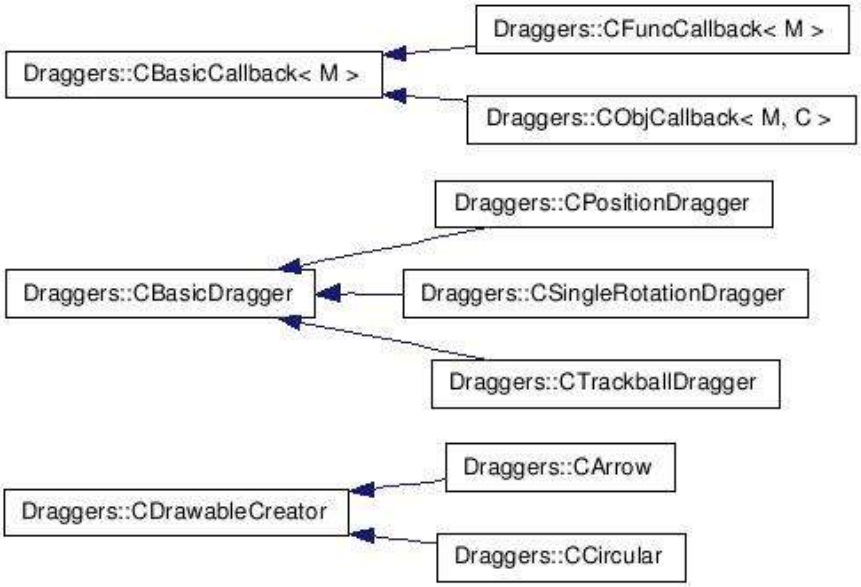

Figure 2: Class hierarchy

4. CONCLUSION

Described Draggers library presents an extension of OSG library to an interactive domain. Use sample (object with manipulation geometry shown) is visible on figure 1. Used algorithms are not completely new, but are used in a new way. The whole system is tightly linked with OSG library, but can be simply converted to any comparable system. Library is completed and ready for use.

REFERENCES

[1] Burns Don, Osfield Robert: Open Scene Graph- An Open Source Solution for Real Time Image Generation, Image 2003, http://www.openscenegraph.org/documentation/IMAGE2003/IMAGE_2003.pdf

[2] Barros L.M.: A Short Introduction to the Basic Principles of the Open Scene Graph, 17.8.2005,http://www.stackedboxes.org/~lmb/OSG/ASIttBPoOSG--02005-10-23.pdf

[3] Gregory M. Nielson, Dan R. Olsen, Jr.: Direct manipulation techniques for 3D objects using 2D locator devices, Symposium on Interactive 3D Graphics, p.175 - 182

[4] Open Inventor library, 29.3.2007, http://oss.sgi.com/projects/inventor

[5] Coin3D graphics toolkit, 29.3.2007, http://doc.coin3d.org

[6] Gleem: OpenGL Extremely Easy-to-use Manipulators, 29.3.2007, http://alumni.media.mit.edu/~kbrussel/gleem/