

Interactive 3D Worlds with Coin and Qt

Virtual World

Qt and Coin, an Open Inventor clone, make the programming of interactive 3D worlds a lot easier than OpenGL ever has. Speed of image rendering is not always the critical requirement. Quick and easy to follow programming may produce better results.

BY STEPHAN SIEMEN

For most people OpenGL, immediately spring to mind when it comes to programming 3D worlds. Often Open Inventor by SGI or the compatible Coin (see “Open Inventor and Coin”) are a better choice: OpenGL may render graphics quickly, but it does involve complex and time-consuming programming. OpenGL’s structure and commands are closely oriented on graphics hardware. Compared to Open Inventor, OpenGL is on the same level as Assembler.

The object oriented Open Inventor Toolkit was developed in 1991 by the same SGI programmers that produced OpenGL. Open Inventor maps the functionality of OpenGL to objects. The

design particularly simplifies programming in C++. Open Inventor itself uses OpenGL to render graphics and thus inherits its superior quality and speed.

The toolkit is far more than just a library that provides classes for creating 3D graphics. Open Inventor allows you to describe and store 3D scenes in so-called scene graphs, for example. The file format developed for this purpose was used as a template for VRML (Virtual Reality Modelling Language) in 1994. Open Inventor also allows interaction with the scene. The scene graphs can be used to identify the objects and thus define appropriate reactions.

Unfortunately, Coin and SoQt are not included with every Linux distribution (see the “Installation” insert). Both packages contain exhaustive documentation.

3D Graphics with a GUI

The biggest difference between programming with Open Inventor and Coin is the GUI integration. The following examples use SoQt to interface with the Qt library, Motif variants are available from [7]. With the exception of the SoQt components, all of these examples also run on SGI’s Open Source Variant of Open Inventor; thus the names Coin and Open Inventor are interchangeable in the following sections.

As befits a programming tutorial, our first example will be a Hello World program. It initializes a viewer and displays a three dimensional text. The user can view the text from any arbitrary position. The code is shown in Listing 1 and can be downloaded from [7]. The following command compiles the program:

```
g++ HelloSoQt.cpp -o HelloSoQt \
-lCoin -lSoQt -I$QTDIR/include
```



The command binds the Coin and SoQt libraries and uses the “-I\$QTDIR/include” option to specify the location of the Qt header files. The results are shown in Figure 2.

Discovering and Exploring Virtual Worlds

The “SoQtExaminerViewer” class (line 31) provides a variety of functions that allow you to view various sections of the scene. The graphic follows mouse movements if you hold down the left mouse button. The viewer provides alternative views of the scene via the wheels and buttons on the window borders.

The three wheels are particularly interesting: two of them are in the bottom left corner, and another is located on the lower right. The right wheel, or dolly, influences the distance between the user and the scene (zoom effect). The two wheels on the left rotate the scene about its x or y axis.

Discovery Tools

A total of seven buttons are located above the dolly wheel on the right window margin:

Coin

Manufacturer: Systems in Motion

License: LGPL (GPL for Version 2.0 planned)

Stable Version: Coin 1.0.3, SoQt 1.0.1

Coin Professional: Commercial license (per developer and year 2000 US dollars), suitable for proprietary projects

Multi platform support: Coin runs on Linux, other Unix type systems and Windows; you need a C++ compiler and an OpenGL library

GUI Bindings: Currently supports Qt, Gtk+, Motif and Windows; Coin can be used without these bindings

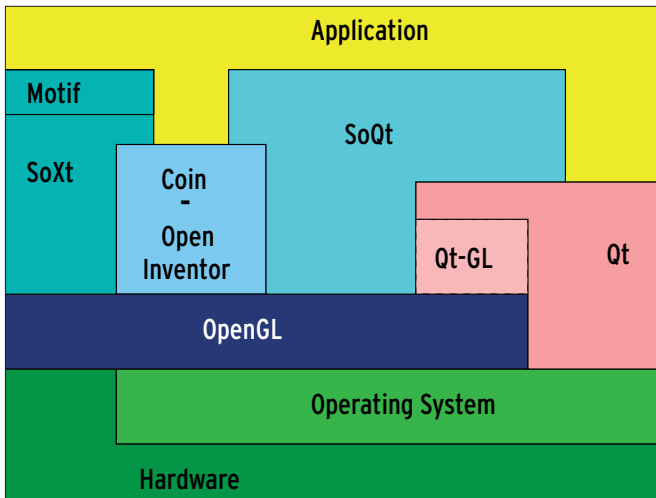


Figure 1: The Coin library mediates between the application and OpenGL. SoQt is used to bind the Qt GUI toolkit while SoXt is an alternative for Motif



Figure 2: The sample Hello SoQt program is shown in the viewer. The user can alter the viewing position

- Arrow: Edit mode
 - Hand: Interactive mode
 - House: Reset camera position to default
 - Blue House: Set a camera position (viewer angle) as the new default position
 - Eye: View the whole scene
 - Lamp: Set the focus for the zoomer (dolly)
 - Box: Toggles various transformations.
- All of these interactions move the camera and not the scene itself. Interactive

mode is automatically enabled on launching the viewer and allows the user to modify the camera position and angle as required using the GUI wheels and the mouse. Additional settings are available in the pop-up menu (right click).

The Scene Graph: Managing 3D Objects

Open Inventor uses scene graphs for the efficient management of 3D scenes (3D worlds). The scene graph has a tree structure whose roots allow access to the

objects in a scene. The elements of the scene graph are nodes, which contain various functions. Each of these nodes describes part of the image scene.

When calculating and rendering a scene, Open Inventor works its way methodically from the root to the leaves (the lowest nodes) of the scene graph, left to right and top down.

Special nodes are used to permit additional modifications, such as transformations and rotation. These modifications affect any subordinate

Listing 1: "HelloSoQt.cpp"

```

01 // SoQt header files
02 #include <Inventor/Qt/SoQt.h>
03 #include <Inventor/Qt/viewers/SoQtExaminerViewer.h>
04
05 // Coin header files
06 #include <Inventor/nodes/SoBaseColor.h>
07 #include <Inventor/nodes/SoText3.h>
08 #include <Inventor/nodes/SoSeparator.h>
09
10 int main(int argc, char **argv)
11 {
12     // Initialize SoQt library.
13     // The return value points to a Qt window
14     QWidget *window = SoQt::init("test");
15
16     // Create a "scene graph"
17     SoSeparator *root = new SoSeparator;
18     root->ref();
19
20     // Set the RGB color. Yellow in this case
21     SoBaseColor *color = new SoBaseColor;
22     color->rgb = SbColor(1, 1, 0);
23     root->addChild(color);
24
25     // Create a text
26     SoText3 *text3D = new SoText3();
27     text3D->string.setValue("Hello SoQt");
28     root->addChild(text3D);
29
30     // Create a viewer
31     SoQtExaminerViewer *b = new
32     SoQtExaminerViewer(window);
33     b->setSceneGraph(root);
34     b->show();
35
36     // Start the window
37     SoQt::show(window);
38     // Loop until exit.
39     SoQt::mainLoop();
40
41     // Delete viewer and reference for scene
42     delete b;
43     root->unref();
44     return 0;
45 }

```

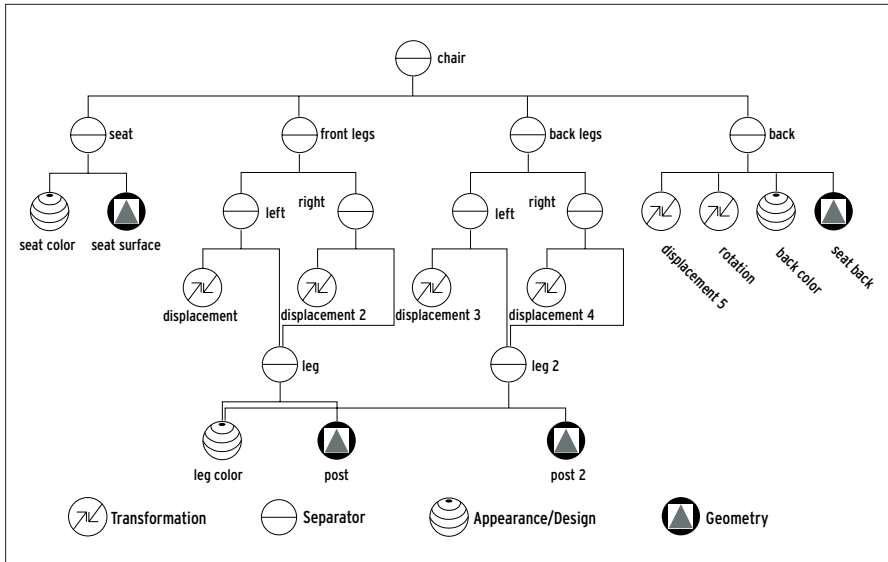


Figure 3: The scene graph divides the chair into its components: seat, front and hind legs as well as lean angle. The left and right legs differ only by their position

nodes. At this point a similarity to the OpenGL status machine becomes apparent. A property, for example the color of a 3D object (class “SoBaseColor”) remains active until a new value is assigned to the property.

The programmer can organize the objects in the scene graph to manipulate the appearance and behavior of the generated scene.

The order of the effects is important. Replacing a rotation with a translation will lead to different generated results. Open Inventor offers a variety of node types that can define surfaces, describe materials or adjust the camera lighting. Other nodes provide interfaces for inter-

active operations or describe various transformations.

Nodes: Shapes, Colors, Materials and Light

Each node is described in a class. The name of a node class starts with “So”, as in “SoMaterial” or “SoCone”. The “So” prefix is omitted when describing a scene in a file. Node names are VRML like in this case.

Each class comprises of fields that characterize the properties of the node. The SGI documentation details the fields available in each node type: the “Open Inventor Nodes Quick Reference” and “Open Inventor C++ Reference” are

available as PDF documents under [3] (enter the document name you require in the search box)

In the case of Coin and its extensions (such as SoQt) the documentation is included in the source code. You can use Doxygen to generate a HTML overview.

Inserting a New Node

The following example inserts a new node into a scene graph:

```
SoCone *cone = new SoCone;
hcone->height.setValue(4);
cone->parts.setValue("SIDES");
root->addChild(cone)
```

This creates an object of the desired class (“SoCone” in this case) and defines values for some fields. In our example

Open Inventor and Coin

In 1996 SGI handed over the development of Open Inventor as of version 2.1 to TGS [2]. This company develops and distributes the latest version (currently 3.1) commercially. In August 2000 SGI decided to publish its own Open Source Open Inventor version (2.1) for Linux [1].

The Norwegians, Systems in Motion [4], offer an implementation of their own, called Coin [5], for Windows, Linux and other Unix systems. Coin is currently available under the LGPL license (version 1.x), although version 2.0 is due to be released under GPL.

More Dynamism with Coin

Coin was chosen for this article – its development promises more dynamism than the SGI variant. Coin attempts to implement the new Open Inventor 3.x features, whereas SGI has merely ported Open Inventor 2.1 to Linux. The various extensions provided by SIM are another good reason for choosing Coin. In addition to the standard Motif binding, it also supports Qt, Gtk and Java.

These bindings are important since Open Inventor merely describes the 3D scene; the programmer must provide the window frame and the binding to the window manager. OpenGL provides the GLUT extension for this purpose, whereas Open Inventor decided on Motif to simplify this task. However, alternatives such as Gtk and Qt are simpler and more commonly available for Linux than Motif. Qt matches Coin perfectly, as both are implemented in C++.

Table 1 Shape Classes

Nodename	Meaning	Fields	Standard values
SoCone	Cone	parts	ALL (SIDES, BOTTOM)
		bottomRadius	1
		height	2
SoCube	Cube	width	1
		height	1
		depth	1
SoCylinder	Cylinder	parts	ALL (SIDES, TOP, BOTTOM)
		radius	1
		height	2
SoSphere	Sphere	radius	1
SoText2	2D text	string	Empty string
		spacing	1
		justification	LEFT (RIGHT, CENTER)
SoText3	3D text	string	Empty string
		spacing	1
		justification	LEFT (RIGHT, CENTER)
		parts	FRONT (SIDES, ALL, BACK)

the height is set to “4” and the sides are visible. Finally, the node is added to the graph (called root in this case). Table 1 includes the most important shape classes with their fields. We will be discussing how to create nodes in a subsequent article.

These elements can be combined to create complex structures, to display a chair for example. To create a graph for this purpose the scene has to be divided into individual components. The more components used to describe a scene, the more realistic the results.

A Simple Chair as an Example of a Scene Graph

A chair is fairly simple to construct; it comprises a seat, a back, and four legs. The legs are the same, apart from their position. Figure 3 shows the scene graph for this construction; every property and the accompanying geometry is described by an individual object.

The object oriented approach means that only one definition is required for duplicate objects, as multiple instances can be added to the scene graph.

In our chair example, the front and rear legs are each only described once and (following the required transformations) added to the scene graph as and when they are needed.

This also applies to the color of the chair legs. The source code for our example is available to download from [7]; Figure 4 shows the results after running the program.

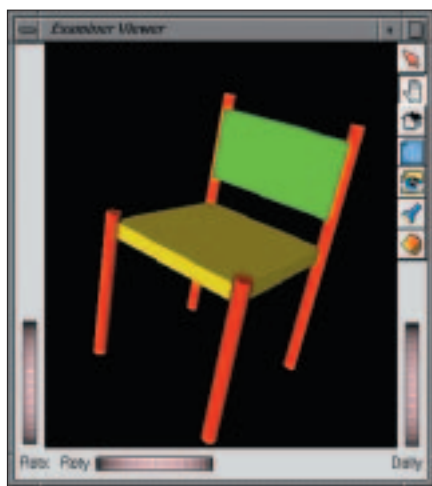


Figure 4: The chair described in the scene graph seems already very realistic, although it only has a few nodes described

Before starting to encode a program, you should take care to plan the scene graph, as repeated use of duplicate objects saves memory. If multiple instances of subgraphs (for example, the legs in our chair example) can be used, you will not only save memory, but ensure that the scene graph and the source code remain clear.

This may not seem important in the context of our example, but in the case of larger projects such as games or CAD programs, the number of objects is a vital criterion.

Documentation

If you are interested in Open Inventor and cannot wait until the next article appears, you might like to check out some interesting online sources. SGI supplies the most complete documentation [3]. The “Inventor Mentor” is the Bible for Open Inventor programmers. It

Installation

Coin requires you to pre-install OpenGL and GLUT. The rendering speed of Open Inventor mainly depends on OpenGL. The Mesa OpenGL library relies on software for 3D calculations, but optimized OpenGL drivers are available for some 3D graphics adapters. The author used a Geforce 2MX and nVidia’s drivers for XFree 4.x. You should enable hardware acceleration if possible, as Open Inventor needs a lot of power, especially in the case of interactions.

The Coin source code [6] is easy to compile and install. Simply follow the familiar steps after expanding the archive file:

```
./configure
make
make install
```

Most Linux distributions include Qt, although you will often find that the Qt libraries have been installed without the header files required for programming. You also have to compile the library with OpenGL support. For SuSE 8.1 you will need to install all the Qt 3.0.5 packages, set the “\$QTDIR” variable to “/usr/lib/qt-3.0.5” and add “\$QTDIR/bin” to your “\$PATH” variable.

The SoQt sources are also available for downloading at [6]; follow the compilation steps as described for Coin. After installing Coin and SoQt, root can invoke the “/sbin/ldconfig” command to make both libraries available throughout the system.

describes the features, such as setting light and camera positions, creating complex geometries, animating scenes and programming interactions.

In addition to the Inventor Mentor, you can also download the “Open Inventor C++ Reference Manual” from SGI. The manual describes the classes that the SGI version of Open Inventor comprises, unfortunately without the Coin extensions.

If you want to learn even more about Coin, you might like to take a look at the HTML documentation, which is included with the tool.

Conclusion and Prospects

Coin and SoQt provide for fairly simple interactive programming of three dimensional graphics, without needing to leave the (L)GPL world. The effort involved is often trivial, particularly in contrast to that of programming with OpenGL, but the results are convincing. After all Open Inventor does use OpenGL for rendering operations.

The features described in this article only scratch the surface of Open Inventor’s capabilities. Our next article discusses how programmers can provide additional interaction via the menus or with the mouse. ■

INFO

- [1] Open Source Variant of Open Inventor: <http://oss.sgi.com/projects/inventor/>
- [2] TGS: <http://www.tgs.com>
- [3] Technical documentation by SGI: <http://techpubs.sgi.com/library>
- [4] Systems in Motion: <http://www.sim.no/>
- [5] Coin: <http://www.coin3d.org>
- [6] Sourcecode for Coin and SoQt: <ftp://ftp.coin3d.org/pub/coin/src/>
- [7] Files for this article: <ftp://ftp.linux-magazin.de/pub/listings/magazin/2003/02/3d/>

THE AUTHOR

Dr. Stephan Siemen works as a scientist at the University of Essex (UK) where he is involved with creating software for 3D representation of weather systems and teaches computer graphics and programming. Additional information on this subject is available from his website at <http://prswwww.essex.ac.uk/stephan/3D/>.