Interactive 3D Worlds with Coin and Qt

# Moving Objects

Coin and a few lines of C++ will allow you to implement even the most ambitions ideas, creating and animated 3D world from a few simple objects with textures and complex VRML models. BY STEPHAN SIEMEN

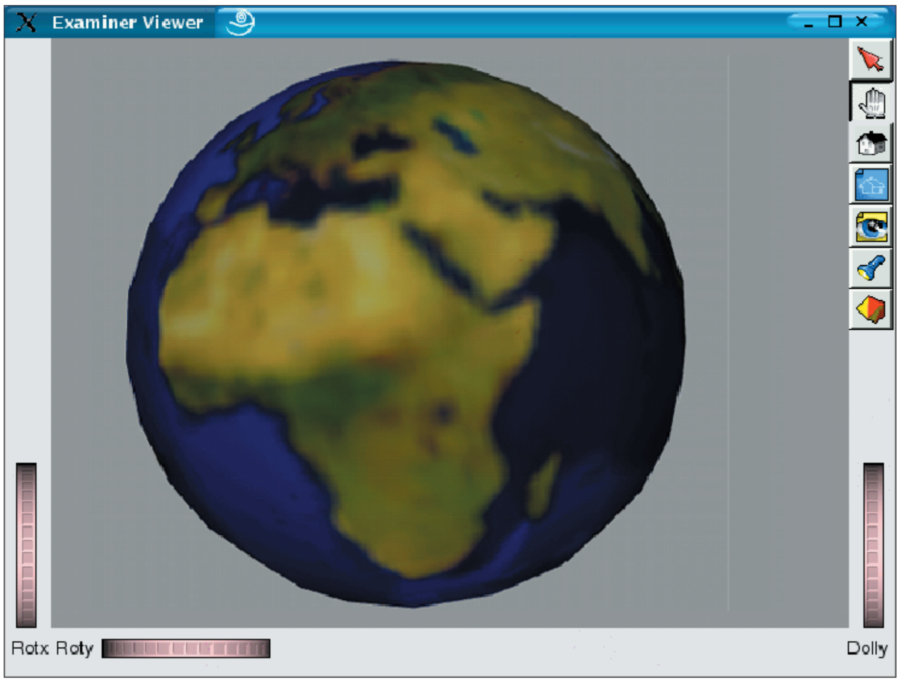

Figure 1: The earth in 3D. A sphere and a suitable surface texture are all you need for this graphic

3D objects do not really look realistic, until you apply a texture to them. Coin and C++ (see the "Review" box) provide you with simple means to use this approach, and this does not involve tedious manual programming of objects. Open Inventor, and thus Coin, can integrate VRML 1.0 files for which tried and trusted graphic modeling tools are available. Animation allows the scenes you develop really come to life.

The scene graph is the common denominator that connects the various elements of a 3D worlds. It is a classic case of "divide and conquer", a complex scene is far easier to program when divided into smaller parts. Before programming a scene it is advisable to create a list of the required objects and their characteristics. A group node is needed for each geometrical type; the required transformations (such as rotation or scaling) are added first, followed by the material attributes (such as the color) and finally by the actual geometry. This order is important as Open Inventor uses OpenGL for rendering.

When subdividing the scene, you should attempt to reuse as many objects as possible, as this approach saves code and is easier to keep track of. One possible approach would be to write functions that use a pointer to return smaller scene graphs or groups. This would allow more convenient integration of available components into scene graphs.

Programmers tend to swap out code recurring segments to libraries, graphics programming is no exception. Libraries can be used to integrate pre-defined descriptions for recurring, complex objects, the most common variant being three dimensional scenes, described by scene graphs of their own, and 2D images (also referred to as textures in this context). The latter are applied to the surfaces of other objects.

**Listing 1: Drawing the Earth**

```
SoSeparator* drawEarth()
{
   // Objects for group nodes and texture
   SoSeparator *earth = new SoSeparator;
   SoTexture2  *texture_earth = new SoTexture2;

   // Name of texture file
   texture_earth->filename = "worldmap.rgb";

   // Add texture to group node
   earth->addChild(texture_earth);

   // Add sphere to group node
   // => Draw texture on surface of sphere
   earth->addChild(new SoSphere);

   return earth;
}
```

## Textures

Textures are designed to make other objects appear more realistic. Open Inventor offers two flavors: textures that are described by a matrix stored in memory, or as a two dimensional file based image. The former approach is extremely time-consuming and normally unnecessary. In order to load a texture from a file, the program first creates an object of the "SoTexture2" class, which is then passed the name of the image file:

```
SoTexture2 *texture = ↵
new SoTexture2;
texture->filename.setValue↵
("texture.rgb");
```

The texture object must be inserted into the scene graph before the object that it is applied to. Open Inventor provides two ways of applying textures to objects: it can either apply a single (default) or multiple instances of texture to an object. Coin uses the "simage" library [5] to load textures and supports the following formats: JPEG (".jpg"), GIF (".gif"), Targa (".tga"), PIC (".pic"), SGI RGB (".rgb", ".bw") and XWD (".xwd").

Listing 1 shows some code that applies a world map to a sphere. The code is implemented as a function that returns a pointer of the "SoSeparator" type. The group node contains a texture called "SoTexture2", and a sphere, "SoSphere". The file "worldmap.rgb" is available from [8]; and was originally an Inventor Mentor sample file.

Figure 1 shows the results of running the function within the context of a scene graph and rendered by "SoQtExaminer".

## External Geometries

As previously mentioned, multiple instances of an object with various sizes and aspects can appear at multiple positions within a scene graph. As Open Inventor describes the shape independently of the position, orientation and aspects (such as color), programmers can re-use an object arbitrarily.

As the example with the chair in part one [1] showed, the objects only need to be defined once. A large amount of code is required to describe an object's geometry, particularly in the case of more complex objects. To keep large-scale projects simpler, it normally makes sense to store complex objects in separate files.

Open Inventor uses a special file format for this purpose, just like VRML. The file suffix is ".iv", and the content can be either ASCII or binary. It is normally preferable to store smaller scenes in (readable) ASCII format. Larger scenes should be stored in binary format to save space and allow Coin to load them more quickly.

The "loadGeometry()" function described in Listing 2 returns a pointer to a a scene graph stored in a file. The function contains several error handling routines to ensure that only valid scenes will load.

Some more sample Open Inventor files are available from [8], such as "boeing767.iv" that describes the geometry of a Boeing 767 jet. Figure 2 shows the scene in "SoQtExaminer". A quick look at the ASCII text version soon shows how complex the scene description is.

In graphical programming, modeling tools are normally used to develop complex objects (such as figures and vehicles). The designer can then export the objects to the required file format (Open Inventor ".iv" or VRML ".wrl"), and Coin loads the files at runtime. Lots

### Listing 2: Loading Geometry from a File

```
SoSeparator* loadGeometry(const char *filename)
{
   // Root of scene graph
   SoSeparator *file_scene = new SoSeparator;

   // Handler for Open Inventor file
   SoInput myScene;

   // Open scene file
   if (!myScene.openFile(filename))
   {
      printf("Error loading file '%s'\n", filename);
      return NULL;
   }

   // Is the file format valid?
   if (!myScene.isValidFile())
   {
      printf("File '%s' is not a valid Inventor file\n", filename);
      return NULL;
   }

   // Read scene and add to group node 'file_scene'
   file_scene = SoDB::readAll(&myScene);

   if (file_scene == NULL)
   {
      printf("Error reading file '%s'\n", filename);
      myScene.closeFile();
      return NULL;
   }

   // Close file
   myScene.closeFile();

   return file_scene;
}
```

### Review

The first part of our mini-series on Coin [1] described how three dimensional graphics are produced in C++ using Coin and SoQt . Coin [2] is a free clone of Open Inventor (SGI [3] and TGS [4]), which is based on OpenGL. The examples in this and the previous article do not contain Coin specific code. Thus, the programs should run with any Open Inventor version and any clone, with the possible exception of the window system binding: Qt is used in place of Motif here.

One advantage of Open Inventor in comparison with OpenGL is the fact that the former uses scene graphs to describe 3D scenes. The scene graph is a tree structure, where nodes are used to store 3D elements. The position in the graph defines where and how will Coin displays a node. This allows you to store a scene or part of it, and program its behavior (interaction and animation).

**Listing 3: Earth Rotation**

```
#include <Inventor/Qt/SoQt.h>
#include <Inventor/Qt/viewers/SoQtExaminerViewer.h>
#include <Inventor/SoInput.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoSpotLight.h>
#include <Inventor/nodes/SoScale.h>
#include <Inventor/nodes/SoTexture2.h>
#include <Inventor/nodes/SoTranslation.h>
#include <Inventor/nodes/SoRotationXYZ.h>
#include <Inventor/nodes/SoSphere.h>
#include <Inventor/engines/SoTimeCounter.h>
#include <Inventor/engines/SoCalculator.h>

// Insert Code for Listing 1 and 2 here

int main(int argc, char ** argv)
{
  // Initialize SoQt (creates a Qt window)
  QWidget *window = SoQt::init("main");

  // Create scene graph
  SoSeparator *root = new SoSeparator;
  root->ref();

  // Spotlight scene: also creates shadow
  SoSpotLight *light = new SoSpotLight;
   light->location.setValue(0,0,2);
   light->direction.setValue(0,0,-1);
   light->cutOffAngle = 1.5;
  root->addChild(light);

  // Group node for rotating earth
  SoSeparator *earth = new SoSeparator;

  // Set rotation node
  SoRotationXYZ *earthrotation = new SoRotationXYZ;
   earthrotation->axis.setValue("Y");
  earth->addChild(earthrotation);

  // Add earth to scene
  earth->addChild(drawEarth());

  // Set Counter
  SoTimeCounter *counter = new SoTimeCounter;
   counter->max=360;
   counter->step=1;
   counter->frequency=0.03;

  // Convert values: Degrees -> Rad
  SoCalculator *converter = new SoCalculator;
   converter->a.connectFrom(&counter->output);
   converter->expression.set1Value(0,"oa=a/(2*M_PI)");

  // Connect counter to earth rotation node
  earthrotation->angle.connectFrom(&converter->oa);

  // Add earth group node
  root->addChild(earth);

  // Create group node for plane
  SoSeparator *plane = new SoSeparator;

  // Move plane from center of scene
  SoTranslation *altitude = new SoTranslation;
   altitude->translation.setValue(0,0,1.2);
  plane->addChild(altitude);

  // Scale plane down and turn through 90
  SoScale *scale = new SoScale;
   scale->scaleFactor.setValue(0.0025,0.0025,0.0025);
  plane->addChild(scale);
  SoRotationXYZ *course = new SoRotationXYZ;
   course->axis.setValue("Y");
   course->angle = 1.5707963;
  plane->addChild(course);

  // Read plane geometry from file
  plane->addChild(loadGeometry("boeing767.iv"));

  // Add plane to scene
  root->addChild(plane);

  // Create viewer
  SoQtExaminerViewer *b = new SoQtExaminerViewer(window);
   b->setSceneGraph(root);
   b->setHeadlight(FALSE);
   b->show();

  // Show windows and wait for "Exit"
  SoQt::show(window);
  SoQt::mainLoop();

  // Delete viewer and scene reference
  delete b;
  root->unref();

  return 0;
}
```

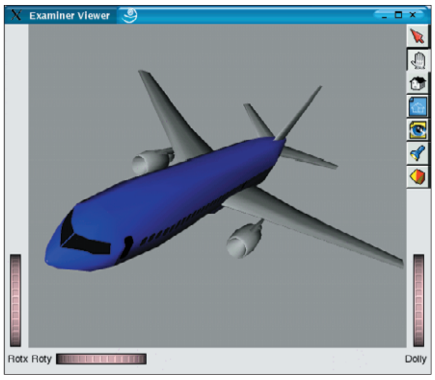

Figure 2: The plane model is described in a text file which is loaded by a Coin program and integrated into the scene graph

of modeling tools are available and some of them are free for private use. They normally support multiple file formats.

## VRML

Not all modeling programs can export 3D objects to the Open Inventor file format, although most tools will understand the common VRML format (Virtual Reality Modeling Language) that Open Inventor is based on. As you can imagine, it is quite simple to convert VRML files to Open Inventor files. To convert a VRML 1.0 ASCII format file to Open Inventor format, you simply edit the first line:

```
#VRML V1.0 ascii
```

The same line would read as follows in Open Inventor:

```
#Inventor V2.0 ascii
```

VRML 1.0 and Open Inventor tags are identical, although later VRML versions contain objects that Open Inventor does not understand. Version 2.0 of Coin, which is currently planned, will provide support for newer VRML versions however. If your modeling software does not support VRML or Open Inventor, you might like to try a 3D file converter, such as 3dc [6], or see [7] for further tools.

## Animations

So far we have only discussed static objects, although users could view the scenes from several sides as the programs used the "SoQtExaminerViewer" class. Open Inventor also provides scene animation classes. There are two steps required. The first is to program an object that triggers changes in the animation – this is also referred to as an engine. There are two basic groups of engines: counters and computers. The second step involves assigning the output from the engine to an object attribute in a scene graph, allowing the attribute to react to that output.

To allow the globe in Figure 1 to rotate about its own axis, you first add a "SoRotationXYZ" class object to the graph, ensuring that it occurs at a position before the globe. The class has two fields: a rotational axis (axis) and an angle of rotation (angle). The rotational characteristics will produce the desired animation when linked to an engine.

Listing 3 creates a node group that will draw a rotating globe when called by the "drawEarth()" function in Listing 1. A "SoTimerCounter" class engine creates the required angles. However, Open Inventor expects angles as radials (and thus as floating point values), and

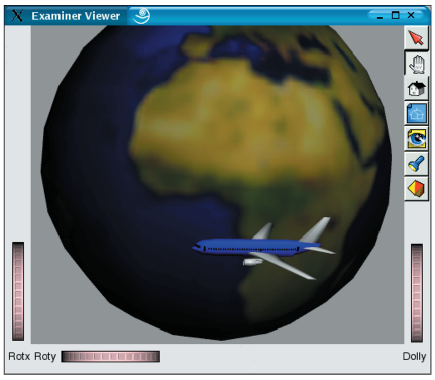

Figure 3: A plane flying around the rotating earth in 3D. It's simple to program this animation with Coin

"SoTimerCounter" supplies integers. "SoCalculator" takes care of converting integers to radials, using common arithmetic expressions to reformat the input values.

## Take off

To complete the example, we added the plane shown in Figure 2 to the scene. Figure 3 shows the results, and Figure 4 the scene graph used to create them. The scene graph does not contain aspect or geometry descriptions, as they are either stored in an external file (and loaded by a call to "loadGeometry()"), or described by a function ("drawEarth()" in this case). The nodes used here contain groups and transformations.

Open Inventor offers a whole range of additional features, providing scope to define animations that use other engine types. The online documentation for Coin [2] provides detailed descriptions of all these engines, indicating their effect and potential. ■

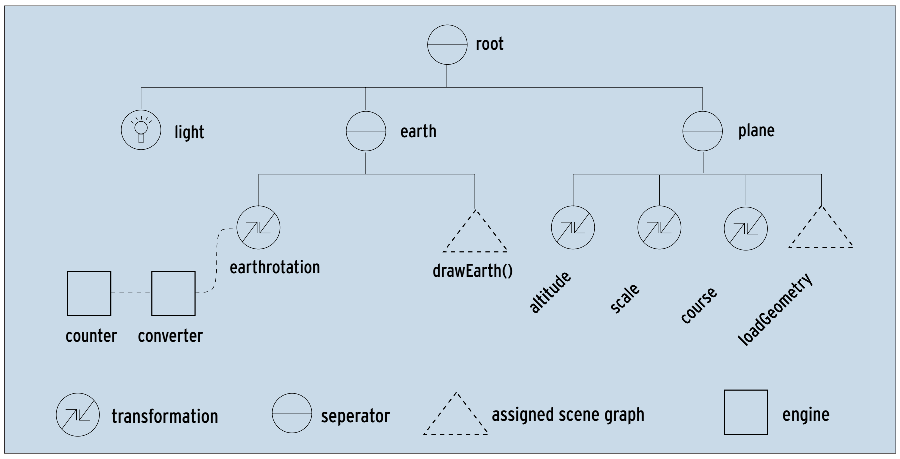

Figure 4: The scene graph for this example contains the earth and the plane as group objects. The counter ensures that the earth will rotate about its own axis

### INFO

[1] Stephan Siemen, "Virtual Worlds : Linux Magazine Issue 28, p72
[2] Coin: *http://www.coin3d.org*
[3] Open Source Variant of Open Inventor: *http://oss.sgi.com/projects/inventor/*
[4] TGS: *http://www.tgs.com*
[5] Simage library: *ftp://ftp.coin3d.org/pub/coin/src/*
[6] 3dc, a 3D converter: *http://www.on-the-web.ch/3dc/*
[7] Additional information: *http://prswww.essex.ac.uk/stephan/3D/*
[8] Files for this article: *ftp://ftp.linux-magazin.de/pub/listings/magazin/2003/03/3d/*