# Open Inventor
## 1 - Introduzione ad Open Inventor e Visual C++

*Claudio Baldazzi* *Maurizio Patrignani*

Università degli Studi di Roma Tre
Dip. di Informatica e Automazione
Via della Vasca Navale, 79 – 00146 Roma, Italy

18 marzo 1998

## 1 Introduzione

Open Inventor è un object-oriented 3D toolkit, cioè una libreria di oggetti e metodi per la creazione di applicazioni grafiche interattive tridimensionali, sviluppata nei laboratori della Silicon Graphics. Open Inventor è scritto in C++ e basato a sua volta sulla libreria grafica OpenGL. Il toolkit mette a disposizione oggetti che possono essere composti, modificati ed estesi per creare delle scene grafiche.

Tra i "meriti" di Open Inventor c'è anche quello di aver definito uno formato standard per i file di descrizione degli oggetti tridimensionali che ha costituito una base per lo standard VRML (Virtual Reality Modeling Language).

L'Open Inventor toolkit è documentato primariamente in tre libri pubblicati dalla Addison-Wesley:

- La sorgente di informazioni più compresiva sull'uso di Open Inventor è "The Inventor Mentor" di Josie Wernecke (ISBN 0-201-62495-8).
- Nel libro "The Inventor Toolmaker", della stessa autrice (ISBN 0-201-62493-1), è mostrato come estendere Open Inventor aggiungendo nuove classi.
- Infine, il libro "The Open Inventor C++ Reference Manual" (ISBN 0-201-62491-5) contiene una versione su carta delle pagine del manuale online, per tutte le classi di oggetti Inventor.

Gli esempi contenuti in questi appunti sono tratti principalmente dal primo dei tre libri citati.

## 2 Esempi

I seguenti esempi consentono un primo approccio all'uso della libreria e costituiscono una piccola rassegna delle sue potenzialità.

## 2.1 Esempio 1: visualizzazione di un cono

Nel primo esempio viene definito e rappresentato un cono. Ciò consente di introdurre le caratteristiche di base di tutte le semplici applicazioni che fanno uso di Open Inventor, e costitisce un prototipo di programma che verrà poi modificato negli esempi successivi.

```
/*--------------------------------------------------------------
 *  This is an example from the Inventor Mentor,
 *  chapter 2, example 1.
 *
 *  Hello Cone example program; draws a red cone in a window.
 *------------------------------------------------------------*/

#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/SoXtRenderArea.h>
#include <Inventor/nodes/SoCone.h>
#include <Inventor/nodes/SoDirectionalLight.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoPerspectiveCamera.h>
#include <Inventor/nodes/SoSeparator.h>

int main( int, char** argv )
{

    // Initialize Inventor. This returns a main window to use.
    // If unsuccessful, exit.
    Widget  myWindow = SoXt::init(argv[0]); // pass the app name
    if( myWindow == 0 ) return 1;

    // Make a scene containing a red cone
    SoMaterial*     myMaterial = new SoMaterial;
    SoSeparator*    root = new SoSeparator;
    SoPerspectiveCamera*    myCamera = new SoPerspectiveCamera;

    root->ref();
    root->addChild(myCamera);
    root->addChild(new SoDirectionalLight);
    myMaterial->diffuseColor.setValue(1.0, 0.0, 0.0);   // Red
    root->addChild(myMaterial);
    root->addChild(new SoCone);

    // Create a renderArea in which to see our scene graph.
    // The render area will appear within the main window.
    SoXtRenderArea* myRenderArea = new SoXtRenderArea(myWindow);

    // Make myCamera see everything.
    myCamera->viewAll( root, myRenderArea->getViewportRegion() );

    // Put our scene in myRenderArea, change the title
    myRenderArea->setSceneGraph(root);
    myRenderArea->setTitle("Hello Cone");
    myRenderArea->show();

    SoXt::show(myWindow);   // Display main window
    SoXt::mainLoop();       // Main Inventor event loop

    return 0;

}
```

Note al codice:

linea 18 L'istruzione `SoXt::init(argv[0])` crea il collegamento tra Open Inventor e l'interfaccia grafica del Sistema Operativo ed inizializza il database di Open Inventor che conterrà gli oggetti della scena.

linea 23 La radice della gerarchia di oggetti è sempre di tipo `SoSeparator`. L'etichetta `root` è introdotta dall'utente: Open Inventor ancora non riconosce questo nodo come radice della scena (ciò verrà dichiarato alla linea 37). I nodi di Open Inventor vengono creati con l'istruzione C++ `new`, ma non si possono distruggere con la corrispondente istruzione `delete`: è lo stesso Open Inventor che amministra la cancellazione dei nodi (vedi nota successiva).

linea 25 L'istruzione `root->ref()` incrementa il contatore associato al nodo root. Ogni nodo ha associato un contatore che segnala ad Open Inventor quanti sono gli oggetti che lo istanziano. Quando il contatore è pari ad $n$ significa che $n$ oggetti hanno il nostro oggetto come componente (figlio). Si contempla, infatti, che uno stesso componente (per esempio una ruota) possa essere definito una volta sola ed utilizzato da più parti, "fattorizzando", per così dire, il codice. Quando il contatore diventa nullo Open Inventor cancella l'oggetto, presumendo che non sia più in uso. La futura radice della scena non è figlia di nessun altro nodo, e dunque ha il contatore posto a zero. Noi lo incrementiamo artificialmente per preservarlo dalle cancellazioni di Open Inventor.

linea 26-30 L'ordine con il quale i nodi vengono "appesi" come figli alla radice di tipo `SoSeparator` non è arbitrario. È necessario porre la `SoPerspectiveCamera` per prima, altrimenti gli oggetti definiti in seguito non vengono colti dalla camera. Analogamente la luce deve precedere gli oggetti che illumina, il materiale deve precedere la definizione della forma (`SoCone`), ecc.

linea 37 Il nodo `root` viene promosso come radice dello *Scene Graph*. È possibile definire diverse scene, e poi promuoverne una per ogni visualizzatore di cui si dispone.

linea 42 L'istruzione `SoXt::mainLoop()` pone il sistema in un loop infinito in cui vengono gestiti gli eventi via via provocati dall'utente (selezioni, movimenti del mouse, ...). In questo programma non abbiamo definito nessuna procedura attivata automaticamente dalle azioni dell'utente (queste particolari procedure vengono chiamate `callback`), tuttavia ci sono sempre degli eventi che Open Inventor deve gestire: per esempio la finestra va ridisegnata ogniqualvolta venga coperta e successivamente scoperta da un'altra finestra (questo evento viene detto `exposure`), ecc. ecc.

## 2.2 Esempio 2: un cono in rotazione

Questo esempio mostra come sia possibile rappresentare oggetti in movimento tramite il ricorso alle *engines*, ovvero ai *motori*, di Open Inventor. Il meccanismo è semplice: si definisce una trasformazione, che si premette all'oggetto che si vuole mettere in movimento; si definisce un contatore; si crea una connessione tra il valore del contatore e un opportuno parametro della trasformazione. Vediamo in dettaglio:

```
 1 /*--------------------------------------------------------------
 2  *  This is an example from the Inventor Mentor
 3  *  chapter 2, example 2.
 4  *
 5  *  Use an engine to make the cone spin.
 6  *------------------------------------------------------------*/

 7 #include <Inventor/Xt/SoXt.h>
 8 #include <Inventor/Xt/SoXtRenderArea.h>
 9 #include <Inventor/engines/SoElapsedTime.h>
10 #include <Inventor/nodes/SoCone.h>
11 #include <Inventor/nodes/SoDirectionalLight.h>
12 #include <Inventor/nodes/SoMaterial.h>
13 #include <Inventor/nodes/SoPerspectiveCamera.h>
14 #include <Inventor/nodes/SoRotationXYZ.h>
15 #include <Inventor/nodes/SoSeparator.h>

16 int main( int, char** argv )
17 {

18     // Initialize Inventor and Xt
19     Widget myWindow = SoXt::init(argv[0]);
20     if( myWindow == 0 ) return 1;

21     SoSeparator* root = new SoSeparator;
22     root->ref();
23     SoPerspectiveCamera* myCamera = new SoPerspectiveCamera;
24     root->addChild(myCamera);
25     root->addChild(new SoDirectionalLight);

26     // This transformation is modified to rotate the cone
27     SoRotationXYZ* myRotXYZ = new SoRotationXYZ;
28     root->addChild(myRotXYZ);

29     SoMaterial* myMaterial = new SoMaterial;
30     myMaterial->diffuseColor.setValue( 1.0, 0.0, 0.0 ); // Red
31     root->addChild(myMaterial);
32     root->addChild(new SoCone);

33     // An engine rotates the object. The output of myCounter
34     // is the time in seconds since the program started.
35     // Connect this output to the angle field of myRotXYZ
36     myRotXYZ->axis = SoRotationXYZ::X;      // rotate about X axis
37     SoElapsedTime* myCounter = new SoElapsedTime;
38     myRotXYZ->angle.connectFrom(&myCounter->timeOut);

39     SoXtRenderArea* myRenderArea = new SoXtRenderArea(myWindow);
40     myCamera->viewAll(root, myRenderArea->getViewportRegion());
41     myRenderArea->setSceneGraph(root);
42     myRenderArea->setTitle("Engine Spin");
43     myRenderArea->show();

44     SoXt::show(myWindow);
45     SoXt::mainLoop();

46     return 0;

47 }
```

Note al codice:

linea 27 Si definisce una rotazione (un nodo di tipo `SoRotationXYZ`). Il valore della rotazione rimane per il momento indefinito.

linea 36 Si specifica che la rotazione dovrà avvenire attorno all'asse delle `X`. Dal punto di vista formale si noti che `X` è un campo (evidentemente pubblico e costante) della classe `SoRotationXYZ`.

linea 37 L'oggetto di tipo `SoElapsedTime` è una *engine*, cioè un *motore*, che conta i secondi trascorsi dal momento in cui viene lanciato il programma. Si noti che l'engine è un oggetto a parte, che non viene integrato nell'architettura dello Scene Graph con un comando del tipo `root->addChild()` (non è affatto un nodo).

linea 38 L'istruzione contenuta in questa linea realizza il collegamento tra il campo `timeOut`, che contiene il valore del contatore di tipo `SoElapsedTime` e il campo `angle` della rotazione di tipo `SoRotationXYZ`.

## 2.3 Esempio 3: `trackball manipulator`

Alcuni nodi predefiniti di Open Inventor consentono di realizzare interfacce sofisticate tramite istruzioni molto semplici. In questo esempio si utilizzerà un "manipulator", ovvero un nodo che consente la manipolazione da parte dell'utente degli oggetti visualizzati. Il `trackball manipulator` è un nodo che viene visualizzato come tre anelli concentrici che abbracciano l'oggetto, e che consentendo la rotazione dell'oggetto che è al suo interno, tramite il mouse.

```
/*--------------------------------------------------------------
 *  This is an example from the Inventor Mentor,
 *  chapter 2, example 3.
 *
 *  Use the trackball manipulator to edit/rotate a red cone
 *------------------------------------------------------------*/

#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/SoXtRenderArea.h>
#include <Inventor/manips/SoTrackballManip.h>
#include <Inventor/nodes/SoCone.h>
#include <Inventor/nodes/SoDirectionalLight.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoPerspectiveCamera.h>
#include <Inventor/nodes/SoSeparator.h>

int main( int, char** argv )
{

    // Initialize Inventor and Xt
    Widget myWindow = SoXt::init(argv[0]);
    if( myWindow == 0 ) return 1;

    SoSeparator* root = new SoSeparator;
    root->ref();

    SoPerspectiveCamera* myCamera = new SoPerspectiveCamera;
    root->addChild(myCamera); // child 0
    root->addChild(new SoDirectionalLight); // child 1
    root->addChild(new SoTrackballManip);   // child 2

    SoMaterial* myMaterial = new SoMaterial;
    myMaterial->diffuseColor.setValue(1.0, 0.0, 0.0);
    root->addChild(myMaterial);
    root->addChild(new SoCone);
```

```
29     SoXtRenderArea* myRenderArea = new SoXtRenderArea(myWindow);
30     myCamera->viewAll(root, myRenderArea->getViewportRegion());
31     myRenderArea->setSceneGraph(root);
32     myRenderArea->setTitle("Trackball");
33     myRenderArea->show();

34     SoXt::show(myWindow);
35     SoXt::mainLoop();

36     return 0;

37 }
```

linea 24 Il `SoTrackballManip` è un nodo al pari degli altri, definito come figlio del nodo `SoSeparator` root. Tutto ciò che verrà definito dopo (linee 27 e 28), verrà abbracciato dalla "trackball".

### 2.4 Esempio 4: uso di un `Examiner Viewer`

Open Inventor mette a disposizione del programmatore sia degli strumenti sofisticati (engines, manipolatori, ...) per la definizione di oggetti complessi, sia degli "ambienti" nei quali rappresentarli.

In questo esempio fa uso di un `Examiner Viewer`, un visore più versatile della `render area` degli esempi precedenti, che consente di ruotare gli oggetti e di zoomare su di essi. Come si vede il codice non viene appesantito affatto, anzi viene sostanzialmente alleggerito di parte della definizione dell'interfaccia (in particolare della `SoPerspectiveCamera`), compito che viene ora assolto dall'`Examiner Viewer`.

```
/*--------------------------------------------------------------
 *  This is an example from the Inventor Mentor,
 *  chapter 2, example 4.
 *
 *  Use the Examiner Viewer to look at a red cone
 *-------------------------------------------------------------*/

#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>
#include <Inventor/nodes/SoCone.h>
#include <Inventor/nodes/SoDirectionalLight.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoPerspectiveCamera.h>
#include <Inventor/nodes/SoSeparator.h>

int main( int, char** argv )
{

    Widget myWindow = SoXt::init(argv[0]);
    if( myWindow == 0 ) return 1;

    SoSeparator* root = new SoSeparator;
    root->ref();
    SoMaterial* myMaterial = new SoMaterial;
    myMaterial->diffuseColor.setValue(1.0, 0.0, 0.0);
    root->addChild(myMaterial);
    root->addChild(new SoCone);

    // Set up viewer:
```

```
26      SoXtExaminerViewer* myViewer = new SoXtExaminerViewer(myWindow);

27      myViewer->setSceneGraph(root);
28      myViewer->setTitle("Examiner Viewer");
29      myViewer->show();

30      SoXt::show(myWindow);
31      SoXt::mainLoop();

32      return 0;

33 }
```

linea 26 Si definisce un `SoXtExaminerViewer`, che verrà ospitato dalla finestra `myWindow` definita nella linea 16. La presenza dell'`Examiner Viewer` rende inutile la definizione di una `SoPerspectiveCamera`.

## 3 Ambiente Microsoft Visual C++ e OpenInventor

Compilando le applicazioni OpenInventor con il Visual C++, bisogna configurare il sistema ed il compilatore. Se OpenInventor è stato installato, ad esempio, nella directory `c:\Inventor` l'ambiente S.O. e compilatore va configurato nel seguente modo:

1. Configurare la variabile di ambiente `INCLUDE` in modo che contenga il path in cui si trovano i file `*.h`, cioè `set INCLUDE=%INCLUDE%;c:\Inventor\include`. In alternativa si puó configurare il compilatore in modo che "conosca" tale directory.

2. Configurare la variabile di ambiente `LIB` in modo che contenga il path in cui si trovano i file `*.lib`, cioè `set LIB=%LIB%;c:\Inventor\lib`. In alternativa si puó configurare il compilatore in modo che "conosca" tale directory.

3. Configurare la variabile di ambiente `PATH` in modo che contenga il path in cui si trovano i binari e i file `*.dll` di OpenInventor. Questo si ottiene con il comando `set PATH=%PATH%;c:\Inventor\bin`. Inserire tale linea nel file `autoexec.bat` se si lavora con Windows95, mentre configurarla da Control Panel se si usa Windows NT.

In prima instanza, il progetto creato con MS-Visual C++ puó essere di tipo "console", e bisogna includere nel progetto le due librerie `inventor.lib` e `libSoXt.lib`.

La libreria grafica non è completamente compatibile ANSI C++, poiché è stata sviluppata con un compilatore un po' piú lasco rispetto ai nuovi compilatori ANSI C++. Tra i compilatori ANSI C++ vi è anche il Microsoft Visual C++, almeno nella versione 5.0. In caso di problemi di compatibilitá con l'ANSI C++ in compilazione, bisogna effettuare le seguenti modifiche nei file header di OpenInventor:

1. `Inventor/SbBasic.h` — Inserire un `#include <math.h>` all'inizio di questo file in quanto vengono usate le funzioni matematiche in alcune macro

e, se non sono definite poiché non vi è il file include, il compilatore le definisce secondo la vecchia sintassi C (K&R), e la prima volta che poi si usano, si ha un errore di compilazione.

2. `Inventor/Xt/viewers/SoXtViewer.h` — Nel costruttore della classe `SoXtViewer`, dal parametro formale `SoXtViewer::Type type = BROWSER` eliminare `SoXtViewer::` e lasciare solamente `Type type = BROWSER`. Il costrutto definito in OpenInventor è infatti inutile e illegale in ANSI C++.

3. `Inventor/Xt/viewers/SoXtFullViewer.h` — Stessa cosa di sopra riportata al costruttore della classe `SoXtFullViewer`.