Open Inventor:

How to Write an Open Inventor File Translator

CONTRIBUTORS

Written by Josie Wernecke and Eleanor Bassler
Engineering contributions by Rikk Carey and Paul Strauss

© Copyright 1994, Silicon Graphics, Inc.— All Rights Reserved
This document contains proprietary and confidential information of Silicon Graphics, Inc. The contents of this document may not be disclosed to third parties, copied, or duplicated in any form, in whole or in part, without the prior written permission of Silicon Graphics, Inc.

RESTRICTED RIGHTS LEGEND
Use, duplication, or disclosure of the technical data contained in this document by the Government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013 and/or in similar or successor clauses in the FAR, or in the DOD or NASA FAR Supplement. Unpublished rights reserved under the Copyright Laws of the United States. Contractor/manufacturer is Silicon Graphics, Inc., 2011 N. Shoreline Blvd., Mountain View, CA 94039-7311.

Silicon Graphics and IRIS are registered trademarks and IRIX and Open Inventor are trademarks of Silicon Graphics, Inc. Apollo is a registered trademark of Apollo Computer, Inc. FrameMaker is a registered trademark of Frame technology, Inc. Hewlett-Packard is a registered trademark of Hewlett-Packard Company. IBM is a registered trademark of International Business Machines Corporation. Macintosh is a registered trademark of Apple Computer, Inc.

Open Inventor: How to Write an Open Inventor File Translator

Contents

1. **Introduction** 1

What Is in This Document? 2

Things You Need to Know about Inventor 2

- Scene Database 2
- Nodes and Fields 3
- Node Icons 4
- Scene Graphs 4
- Group Nodes 5

2. **Translating Files into Inventor Format** 7

General Steps 7

SGO File Format 8

- SGO Quadrilateral List 9
- SGO Triangle List 9
- SGO Triangle Mesh 10

Reading the File Header 10

Initializing the Inventor Database 11

- Initializing the Database 11
- Creating the Root Node 11
- Setting Up Default Attributes 11

Entering the Object Read Loop 13

Writing the Database to a File 15

Complete Sample Program 16

Sample Results 24

Using the File Translator in Another Program 25

3. Tips and Guidelines 27
Tips 27
    Testing the Results 27
    Creating an Efficient Scene Graph 28
    Verifying Values 28
    Automatic Normal Generation 29
Guidelines for Writing an Inventor File Translator 29
    File Suffix 29
    Application Name and Command Line Syntax 29
    Error Handling 30
    Manual Page 30
    Conventions Used in Inventor Files 31

Figures

Figure 1-1 Exchanging 3D data between Inventor and non-Inventor applications 1
Figure 1-2 Node icons 4
Figure 1-3 Simple scene graph 4
Figure 1-4 Example of separator groups 5
Figure 2-1 SGO file format 8
Figure 2-2 SGO quadrilateral list object 9
Figure 2-3 SGO triangle list object 10
Figure 2-4 Nodes created during initialization 12
Figure 2-5 Inventor nodes for a face set object 14
Figure 2-6 Inventor database after reading the SGO object TRILIST 14

Chapter 1

Introduction

An important feature in Open Inventor is its **3D Interchange File Format**, which provides a much-needed standard for exchanging 3D data among applications. Inventor's file format supports both ASCII and binary files.

- The ASCII file format is a simple, human-readable representation of a 3D scene database.
- Binary files are written in a machine-independent format. Although the binary format is not public, tools are readily available for converting an Inventor ASCII file to binary format.

Translator programs make possible the flow of 3D data between application programs using Open Inventor and non-Inventor. Figure 1-1 illustrates this data flow.

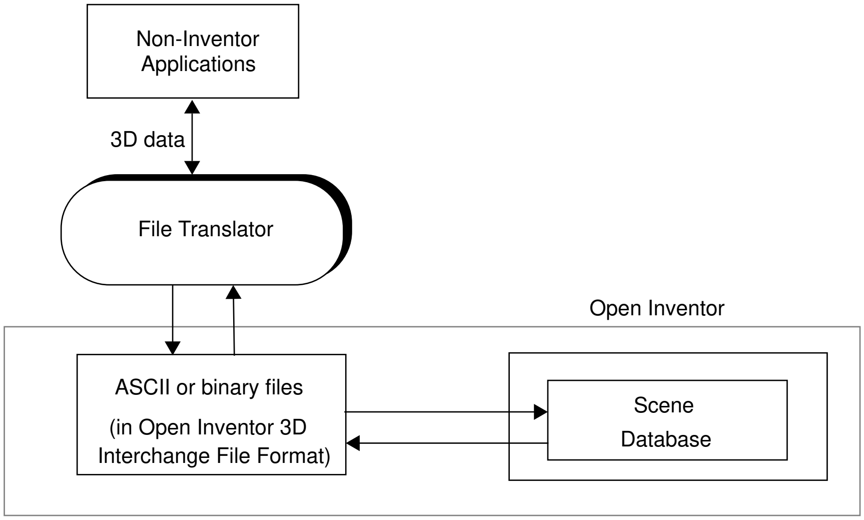

Figure 1-1 Exchanging 3D data between Inventor and non-Inventor applications

Other types of data have been converging on standard formats such as TIFF, GIF, PICT, and PostScript, but there is currently no clear winner for a standard 3D format. Once a standard emerges, all 3D developers and users can benefit from a common file format that allows data exchange among members of a world-wide audience. With a shared file format, users can cut and paste among a variety of applications on the desktop, and developers have access to a wide range of tools and code to boost their productivity. In addition, developers need only write one translator to and from Inventor format, rather than a whole collection of file translators for each 3D file format in the industry.

What Is in This Document?

This document provides the necessary basic information to enable you to write a program that translates existing graphics files into Open Inventor format. It includes the following sections:

- Background information on the Inventor scene database
- An example of a simple file translator
- Tips and suggestions for writing file translators

For a more complete description of Inventor objects, creating a scene database, and applying actions, see *The Inventor Mentor,* Chapters 1 through 5 and Chapter 9.

Things You Need to Know about Inventor

The following paragraphs outline basic concepts of an Inventor scene database.

Scene Database

An Inventor *scene database* is a collection of 3D objects and properties arranged appropriately to represent a 3D scene or data set. Inventor programs create or read their own copies of scene databases each time they execute. A scene database resides in the program’s memory while the program is running, unlike a traditional database that resides on disk and is shared by many running programs.

Nodes and Fields

A node is an object that represents a 3D shape, property, or group. *Shape nodes* represent 3D geometric objects. *Property nodes* represent appearance and other qualitative characteristics of the scene. *Group nodes* are containers that collect nodes into graphs. Other important nodes include camera, light, and callback nodes.

Nodes contain both data and functions. The data elements contained in a node are called *fields*. When you create a node, all the fields within that node are created as well, and each field automatically contains a default value. For example, a point light node contains these fields:

Field	Default Value	Meaning
on	TRUE	whether light is active
intensity	1.0	value between 0.0 and 1.0 (maximum illumination)
color	1.0 1.0 1.0	red, green, blue color of light
location	0.0 0.0 1.0	position in *x*, *y*, *z*

See the *Open Inventor Nodes Quick Reference* for a complete alphabetical listing of all Inventor nodes as well as their fields and default values.

Node Icons

Figure 1-2 contains the legend for nodes used in the diagrams in this document.

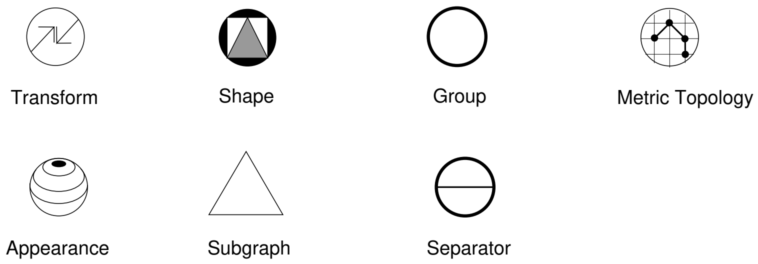

Figure 1-2 Node icons

Scene Graphs

A *scene graph* is an ordered collection of nodes. Hierarchical scene graphs are created by adding nodes as *children* of group nodes. Figure 1-3 shows a simple scene graph. The top node of the graph (in this figure, "top") is called the *root* node. The Inventor scene database can contain any number of scene graphs, each consisting of a related set of 3D objects and attributes. Typically, a 3D scene or a set of object files contains only one scene graph. However, this is not a restriction.

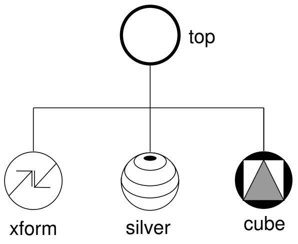

Figure 1-3 Simple scene graph

Group Nodes

A group node contains an ordered list of children that are traversed from left to right. A *separator* group, shown in Figure 1-4, is a special type of group node. Nodes under the separator group do not affect nodes in the graph after the separator.

When you write a scene database, the objects in the database are written from top to bottom and from left to right. Objects lower (and to the right) in the scene graph *inherit* the attributes and values set by objects that precede them. If you do not want subsequent objects to inherit certain properties or values, use a *separator* group, which pushes and pops properties during traversal (see Figure 1-4; the red material in the separator A group does not affect the cone in the separator B group). The root node of a scene graph is usually a separator.

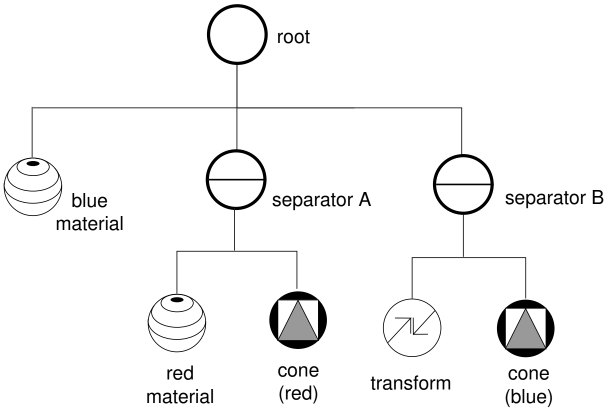

Figure 1-4 Example of separator groups

Chapter 2

Translating Files into Inventor Format

This section outlines a general methodology for writing an Inventor file translator and presents a sample file translation program. The sample program translates Silicon Graphics Object (SGO) data files into Open Inventor files. This example is presented as one possible way to write a file translator. There are, of course, many other possible solutions, depending on your needs, the type of data you are working with, and the structure of the files you are translating. Chapter 11 in *The Inventor Mentor* describes the Inventor file format in great detail.

General Steps

The basic steps in an Inventor file translation program can be summarized as follows:

1. Read and verify the file header of the input file (if applicable).
2. Initialize the Inventor database. This step includes creating the root node or nodes of the database and setting up nodes containing any default attributes that will be used by other nodes in the scene.
3. Enter the object read loop. Read the first object from the input file, generate an Inventor object, and put it into the database. Continue reading objects from the input file until all objects are read and translated.
4. Clean up the Inventor database. Reorganize it for maximum efficiency.
5. Write the Inventor database to a file.

The following subsections discuss each step in more detail. The complete program is shown in “Complete Sample Program” on page 16.

SGO File Format

Since the program example translates files from SGO file format, you'll need to know something about this format before you look at the example in detail.

Figure 2-1 shows the basic structure of the SGO file format. The first word in the file must be the SGO magic number, a code number used to identify the file (0x5424). The magic number is 4 bytes long.

The objects in SGO files are constructed from three data types: quadrilateral lists, triangle lists, and triangle meshes. One SGO file can contain any number of objects of differing type. An identifying token (4 bytes) precedes the data for each object:

`OBJ_QUADLIST`	`(= 1)`	`quadrilateral list`
`OBJ_TRILIST`	`(= 2)`	`triangle list`
`OBJ_TRIMESH`	`(= 3)`	`triangle mesh`

The end of data token is `OBJ_END` (= 4). This token is placed after the data for the last object in the file.

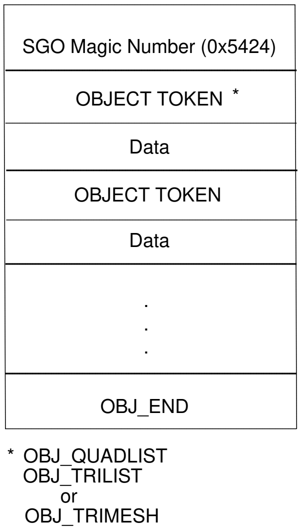

Figure 2-1 SGO file format

SGO Quadrilateral List

Figure 2-2 shows the structure of an SGO quadrilateral list object. The object begins with the object token, followed by the size (in 4-byte words) of the data for this object. Next follow nine words of data for each vertex in the object. As shown at the bottom of Figure 2-2, the first three words are the *xyz* components of the normal vector at the vertex. The next three words are the *RGB* color components at the vertex. The last three words are the *xyz* coordinates of the vertex itself.

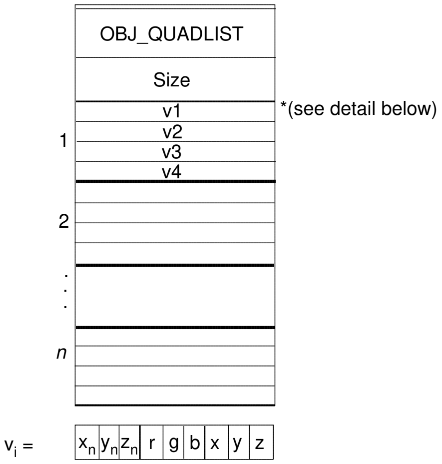

Figure 2-2 SGO quadrilateral list object

SGO Triangle List

Figure 2-3 shows the structure of an SGO triangle list object. The object begins with the object token, followed by the size (in 4-byte words) of the data for this object. Next follow nine words of data for each vertex in the object. As shown at the bottom of Figure 2-3, the first three words are the *xyz* components of the normal vector at the vertex. The next three words are the *RGB* color components at the vertex. The last three words are the *xyz* values of the vertex itself.

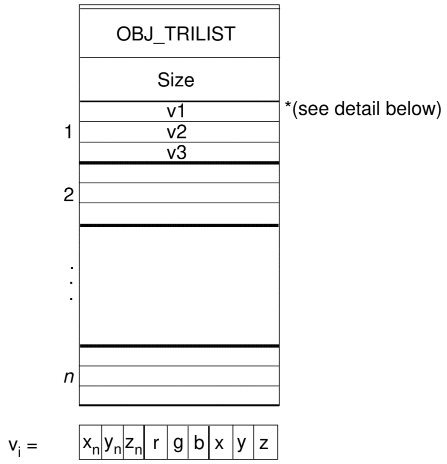

Figure 2-3 SGO triangle list object

SGO Triangle Mesh

In the interest of brevity, the SGO triangle mesh object is not discussed in this document or illustrated in the example.

Reading the File Header

The first step in the file translation program is to check that the header of the input file is in the expected format. SGO files, used in our example, are identified by a special code number, 0x5424, known as the SGO "magic number." If the file header does not contain the magic number, the function returns FALSE.

```
static SbBool
readHeader()
{
   long   magic;

   return (fread(&magic, sizeof(long), 1, stdin) == 1 &&
           magic == SGO_MAGIC);
}
```

Initializing the Inventor Database

This step comprises three parts:

- initializing the Inventor database
- creating a root node for the database
- creating nodes with default attributes that will be used by other nodes in the database

Initializing the Database

The following code initializes the Inventor database:

```
SoDB::init();
```

Creating the Root Node

The root node for the database is typically a separator node (see“Group Nodes” on page 5). The root node is always *referenced*, as shown below. This ensures that the root node is not accidentally deleted. In most cases, you will use a separator as your root node. If in doubt, use a separator.

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

Setting Up Default Attributes

This step involves setting up default or global attributes that are the same for all objects in the scene graph.

SGO objects include values for normals and colors along with each index. In Inventor, you need to specify how this information is applied to the shape objects in the scene graph. For example, a color can be applied, or “bound,” to an entire shape, to each face in the shape, or to each vertex in the shape.

In our example, we want the normals and materials to be bound to each vertex in the shape object. This is termed *per-vertex binding*. The sample program creates the following two Inventor nodes and adds them to the scene graph:

SoMaterialBinding	tells how to bind the specified materials to the shape node
SoNormalBinding	tells how to bind the specified normals to the shape node

Note: While the sample program does not translate SGO triangle mesh objects, your program may require this translation. If so, you should note that this object presents a special case because it lists colors and normals in an arbitrary order and then indexes into the list. For triangle mesh objects, you should use Inventor's *per-vertex indexed* binding. When no indices are present, this binding defaults to per-vertex binding (normals and materials are used in order).

See *The Inventor Mentor*, Chapter 5, for a detailed description of binding nodes.

Here is the code that creates the material and normal binding nodes and specifies per-vertex binding for both nodes:

```
mtlBind = new SoMaterialBinding;
normBind = new SoNormalBinding;
mtlBind->value = SoMaterialBinding::PER_VERTEX;
normBind->value = SoNormalBinding::PER_VERTEX;
root->addChild(mtlBind);
root->addChild(normBind);
```

At this point, the Inventor database looks like Figure 2-4.

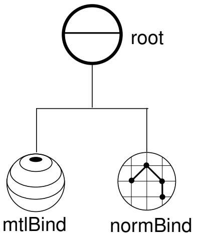

Figure 2-4 Nodes created during initialization

Entering the Object Read Loop

The object read loop is where the majority of the database objects are created. This loop reads the first object from the input file. It determines which type of object follows and calls one of three functions: **readQuadList()**, **readTriList()**, or **readTriMesh()**. Then it generates Inventor nodes to represent the corresponding data and puts those objects into the database that was initialized in step 1. This loop continues reading objects from the input file until all objects have been read and translated into Inventor nodes.

The SGO objects OBJ_QUADLIST and OBJ_TRILIST are translated into the Inventor shape node **SoFaceSet**. The OBJ_TRIMESH object is not implemented for this example.

As described earlier, SGO objects contain the following nine words of data for each vertex:

- Normals (*x*, *y*, *z*)
- Colors (*r*, *g*, *b*)
- Coordinates (*x*, *y*, *z*)

In Inventor, each of these three sets of information is contained in a separate node as follows:

- **SoNormal** - contains all vertex normals for a shape
- **SoBaseColor** - contains the red/green/blue values for the base color of the vertices (An **SoMaterial** node could be used here, but **SoBaseColor** is more efficient since only the dif