# IRIS Performer 2.3 Installation and Porting Guide for Linux

007-4253-001

CONTRIBUTORS

Written by Ken Jones

Edited by Rick Thompson

Engineering contributions by Allan Schaffer

# Record of Revision

| Version | Description |
| --- | --- |
| 001 | November 1999<br>Original publication. |

# Contents

IRIS Performer 2.3 Installation and Porting Guide for Linux 1

# About This Guide

This guide contains installation instructions for IRIS Performer 2.3 for Linux and a description of differences between IRIS Performer on the IRIX operating system and the Linux operating system.

IRIS Performer is the premier high-performance 3D rendering toolkit for developers of real-time, interactive graphics applications. IRIS Performer dramatically simplifies development of complex applications such as those for visual simulation, simulation-based design, virtual reality, interactive entertainment, broadcast video, CAD, and architectural walk-through while providing a high-performance portability path across the entire SGI product line and now Linux.

## Related Publications

The following documents contain additional information that may be helpful:

- *IRIS Performer Getting Started Guide*

  The guide introduces the most important concepts and classes in the Performer libraries. Use this guide to quick-start your programming using the IRIS Performer API.

- *IRIS Performer Programmer's Guide*

  The guide explains the techniques and effects of IRIS Performer.

- *IRIS Performer Class Reference Guide for C Programmers*

  This reference guide lists the classes in all of the IRIS Performer C programming libraries along with the methods of each class.

- *IRIS Performer Class Reference Guide for C++ Programmers*

  This reference guide lists the classes in all of the IRIS Performer C++ programming libraries along with the methods of each class.

## Obtaining Publications

To obtain SGI documentation, go to the SGI Technical Publications Library:

```
http://techpubs.sgi.com
```

## Conventions

The following conventions are used throughout this document:

| Convention | Meaning |
|---|---|
| `command` | This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures. |
| *variable* | Italic typeface denotes variable entries and words or concepts being defined. |
| **`user input`** | This bold fixed-space font denotes literal items that the user enters in interactive sessions. Output is shown in nonbold, fixed-space font. Also, function names with parentheses following the name - for example, **`glPolygonMode`**() - and arguments to command line options. |
| [ ] | Brackets enclose optional portions of a command or directive line. |
| . . . | Ellipses indicate that a preceding element can be repeated. |

## Reader Comments

If you have comments about the technical accuracy, content, or organization of this document, please tell us. Be sure to include the title and document number of the manual with your comments. (Online, the document number is located in the front matter of the manual. In printed manuals, the document number can be found on the back cover.)

You can contact us in any of the following ways:

- Send e-mail to the following address:

  ```
  techpubs@sgi.com
  ```

- Use the Feedback option on the Technical Publications Library World Wide Web page:

`http://techpubs.sgi.com`

- Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system.
- Send mail to the following address:

  Technical Publications
  SGI
  1600 Amphitheatre Pkwy., M/S 535
  Mountain View, California 94043-1351

- Send a fax to the attention of Technical Publications:

  +1 650 932 0801

We value your comments and will respond to them promptly.

# Product Overview

IRIS Performer provides a powerful and extensible programming interface (with ANSI C and C++ bindings) for creating real-time visual simulation applications, virtual sets, performance animations, virtual reality systems, engineering visualizations, and other interactive graphics applications. IRIS Performer is the flexible, intuitive, toolkit-based solution for developers who want to optimize graphics performance on their OpenGL-capable systems.

IRIS Performer 2.3 for Linux combines SGI's commitment to high-performance, scalable, interactive rendering with the reduced cost and cross-platform availability of the Linux operating system. This release is fully API-compatible with existing IRIX-based IRIS Performer applications and is a full distribution, including the core run-time libraries and file loaders, development header files, sample source code, and manual pages.

## IRIS Performer Components

IRIS Performer consists of two main libraries, `libpf` and `libpr`, and four associated libraries:

- `libpf` and `libpr`
- `libpfdu`
- `libpfdb`
- `libpfui`
- `libpfutil`

For aid in application development, IRIS Performer also includes sample application source code ranging from simple programs to illustrate particular features to the comprehensive, GUI-driven file viewer `perfly`.

### libpf and libpr

The basis of IRIS Performer is the performance rendering library `libpr`, a low-level library providing high-speed rendering functions based on **`pfGeoSet`**, efficient graphics state control using **`pfGeoState`**, and other application-neutral functions.

Layered above `libpr` is `libpf`, a real-time visual simulation environment providing a high-performance database rendering system that takes best advantage of your hardware and OpenGL.

### libpfdu

The database utility library `libpfdu` provides powerful functions for defining both geometric and appearance attributes of three dimensional objects, encourages sharing of state and materials, and generates efficient triangle strips from independent polygonal input.

### libpfdb

The database library `libpfdb` uses the facilities of `libpfdu`, `libpf`, and `libpr` to import database files in more than fifty industry-standard database formats. These loaders also serve as a guide to developers creating new database importers.

### libpfui

The `libpfui` library contains user-interface building blocks for creating manipulators and user-interface components common to many interactive applications. This library has both a C and C++ API.

### libpfutil

Completing the suite of libraries is `libpfutil`, the IRIS Performer utility library. It provides a collection of convenience routines implementing tasks such as smoke effects, multichannel option support, graphical user-interface tools, X-event collection and management, and traversal functions.

## Further References

General information about IRIS Performer, including a product overview, technical data, news, white papers, training offerings, support information, and links to partner products can be found on the IRIS Performer web site:

```
http://www.sgi.com/software/performer
```

Substantial documentation of the features and API of IRIS Performer 2.3 for Linux is available online:

```
http://www.sgi.com/software/performer/linux-developer.html
```

You can meet the IRIS Performer developers and learn a lot from fellow IRIS Performer users on the info-performer mailing list. To join, refer to the following Web page:

```
http://www.sgi.com/software/performer/mailinglist.html
```

# Installing IRIS Performer on Linux

## Dependencies and Supported Platforms

IRIS Performer 2.3 for Linux has been tested on a variety of systems running Red Hat Linux versions 6.0 and 6.1 and with several vendors' graphics accelerators. The software is understood to be functional with any Linux distribution (including SGI Linux, Red Hat, Linux-Mandrake, SuSE, Caldera OpenLinux, Debian, and others) with the following dependencies:

- glibc version 2.1.1
- XFree86 version 3.3.3.1
- OpenGL or equivalent
- Motif or equivalent

If you need to download any of the required software above, refer to the online IRIS Performer 2.3 for Linux FAQ for instructions:

```
http://www.sgi.com/software/performer/linux-faq.html#5
```

IRIS Performer for Linux requires the following hardware:

- 200MHz Pentium-class CPU or better
- 48MB RAM or greater (memory requirements vary with database size)
- 1024x768 video resolution with 64K colors or better

**Note:** IRIS Performer 2.3 for Linux operates on Pentium-based Linux systems only.

## Installation Instructions

IRIS Performer 2.3 for Linux is available in three common packaging formats:

- RPM

  Red Hat Package Management Format for systems running Red Hat Linux version 6.0 or 6.1.

- DEB

  Debian installation format for systems running Debian Linux.

- TGZ

  GNU-compressed `tar` format for other Linux systems.

Choose the packaging format corresponding to the distribution of Linux installed on your system.

## Installing the RPM Files

1. Ensure that you have the software listed as dependencies already installed on your system.

   See the prior section “Dependencies and Supported Platforms” on page 5.

2. Download or copy the IRIS Performer 2.3 for Linux distribution into a temporary location, such as `/usr/tmp`.

   **Note:** In some cases, the installation tools for certain Linux distributions will prevent you from loading IRIS Performer 2.3 until the dependencies are already present on your system.

3. Log in as `root`.
4. As `root`, execute this command:

   ```
   rpm -Uvh /usr/tmp/performer_*-2.3linux-1999112102.i386.rpm
   ```

   (Replace `/usr/tmp` with the temporary location where you stored the files.)

5. You may now archive or delete the temporary files.

   Your installation is complete.

## Installing the DEB Files:

1. Ensure that you have the software listed as dependencies already installed on your system.

   See the prior section “Dependencies and Supported Platforms” on page 5.

2. Download or copy the IRIS Performer 2.3 for Linux distribution into a temporary location, such as `/usr/tmp`.

   **Note:** In some cases, the installation tools for certain Linux distributions will prevent you from loading IRIS Performer 2.3 until the dependencies are already present on your system.

3. Log in as `root`.
4. As `root`, execute the following commands in order:

   ```
   dpkg --install /usr/tmp/performer-eoe_2.3linux-1999112102_i386.deb
   dpkg --install /usr/tmp/performer-dev_2.3linux-1999112102_i386.deb
   dpkg --install /usr/tmp/performer-demos_2.3linux-1999112102_i386.deb
   ```

   (Replace `/usr/tmp` with the temporary location where you stored the files.)

5. You may now archive or delete the temporary files.

   Your installation is complete.

## Installing the TGZ Files

1. Ensure that you have the software listed as dependencies already installed on your system.

   See the prior section “Dependencies and Supported Platforms” on page 5.

2. Download or copy the IRIS Performer 2.3 for Linux distribution into a temporary location, such as `/usr/tmp`.

   **Note:** In some cases, the installation tools for certain Linux distributions will prevent you from loading IRIS Performer 2.3 until the dependencies are already present on your system.

3. Log in as `root`.

4. As `root`, execute the following commands in order:

   ```
   cd /
   tar xzvf /usr/tmp/performer_eoe-2.3linux-1999112102.tgz
   tar xzvf /usr/tmp/performer_dev-2.3linux-1999112102.tgz
   tar xzvf /usr/tmp/performer_demos-2.3linux-1999112102.tgz
   ```

   (Replace `/usr/tmp` with the temporary location where you stored the files.)

5. You may now archive or delete the temporary files.

   Your installation is complete.

# Porting IRIS Performer Applications to Linux

Porting software from one system type to another is an art which requires an in-depth understanding of the abilities and idiosyncrasies of both the source and target platform. This section is intended for experienced programmers and experienced users of IRIS Performer. This section describes the issues you are likely to encounter when bringing your existing IRIX code base to Linux.

IRIS Performer 2.3 for Linux is fully API-compatible with existing IRIS Performer 2.2 applications built for IRIX. We have made an effort to make porting as easy as possible by leaving the API mostly unchanged and adding prominent warning messages to your program output if unsupported functionality is utilized. It is trivial to port most simple programs and even more intricate applications like `perfly` can be ported to Linux with only a small amount of effort. However, your application may be considerably more complex or rely on functionality that is not available in Linux or that is not supported in this release. For most applications, this will not be the case.

The following sections categorize the porting issues in the following manner:

- header files
- endianness
- compiler differences
- features and functionality not supported in this release
- guidelines for new applications

## Header Files

Some of the IRIX-based C or C++ header files your program uses through `#include`'s may not be available on Linux platforms, or the structures they define may be contained in a different header file than on IRIX. Such issues can usually be resolved by visual inspection of the header files themselves.

Using the C language version of the IRIS Performer example application `perfly` as an example, the following code was changed to remove headers:

```
#ifndef __linux__
    #include <sys/sysmp.h>
    #include <bstring.h>
    #else
    #include <limits.h>
    #include <string.h>
    #endif
```

The prototypes for **`bzero`**(), **`bcopy`**(), etc. are located in `string.h` instead. The `#define` for `PATH_MAX` is in `limits.h`.

## Endianness

IRIS Performer 2.3 for Linux is only available for Intel x86-based systems. The little-endian structure of the Intel x86 architecture is the opposite of that of big-endian MIPS-based IRIX; so, any code or data that assumes big-endian operation will fail sometimes in an unpredictable manner.

A typical case where endianness issues are seen is when using data or retained database files that were generated on IRIX or other big-endian systems. The database loaders shipped with IRIS Performer 2.3 for Linux have been modified to automatically identify such files and correct the byte ordering in the file, accordingly. If you have implemented your own file loaders, you should make similar changes.

See the source code of the `bin` or `pfb` loaders shipped in `/usr/share/Performer/src/lib/libpfdb/libpfbin/pfbin.c` and `/usr/share/Performer/src/lib/libpfdb/libpfpfb/pfpfb.c` for a simple and a complex example of the changes necessary. In the case of `bin` format, the assumption is made that all input files will contain big-endian data so a byte flip of integer data is always necessary. In the case of `pfb` format, a more sophisticated approach is taken that detects the endianness of the data file by comparing a header key to its representation in the native format. If they match, no changes are necessary. If they are in the opposite byte order, then an alternate **`fread`** function is used that will swap the bytes.

Endianness issues arise in many other cases, such as when defining RGB color values in a single 32-bit variable. For example, the following code from IRIX defines an array of eight RGB color values:

```
static const uint colors[MAX_SIZE] =
{
   0xff080000,
   0xff190100,
   0xff2a0100,
   0xff3b0100,
   0xff4c0200,
   0xff5d0240,
   0xff6e0280,
   0xff7f02c0
};
```

This would need to be changed in Linux (preferably also using an `#ifdef __linux__ / #endif` preprocessor directive so that the code could still be shared between the two platforms) to the following:

```
static const uint colors[MAX_SIZE] =
{
   0x000008ff,
   0x000119ff,
   0x00012aff,
   0x00013bff,
   0x00024cff,
   0x40025dff,
   0x80026eff,
   0xc0027fff
};
```

## Compiler Differences

The compilers shipped with most Linux distributions do not automatically zero variables and pointers as they are declared. This may expose hidden bugs in your code; so, you should take extra care to set variables before using them. As an example, `var` needs to be initialized to `NULL` in the code below:

```
void myfunc()
   {
      static pfGroup *var;
      if (!var)
```

```
    {
    /* initialize var */
    }
    /* use var */
}
```

A good C program checker such as `lint` may help you to detect such bugs.

## Features and Functionality Not Supported in This Release

Certain IRIX functionality or OpenGL functionality specific to SGI IRIX-based systems is not available on Linux platforms and, likewise, certain functionality from IRIS Performer 2.2 on IRIX is not yet available in IRIS Performer 2.3 for Linux.

This section is an overview of the features commonly used in IRIX-based IRIS Performer applications that are not supported in this release. Results stemming from the use of such features is undefined.

Ignoring the settings your program has made, in many cases IRIS Performer 2.3 for Linux will add prominent warning or informational messages to your program output if unsupported functionality is utilized and then continue normally . However, in other cases your program will fail or behave in an unexpected manner.

### General IRIS Performer Functionality

The following are general aspects of IRIS Performer functionality not supported in this release:

- Multiprocessing (forked `CULL`, `DRAW`, X Input, `DBASE`, `COMPUTE`, `ISECT`, threaded `CULL`, threaded `LPOINT`, etc.) is not supported. Only `PFMP_APPCULLDRAW` mode is available.
- Multipipe operation (**`pfMultipipe`**, **`pfHyperpipe`**, etc.) is not supported. Only a single graphics pipeline can be utilized.
- Shared arenas (**`pfGetSharedArena`**) and related functions are not supported. **`pfGetSharedArena`**() will always return `NULL`.

### SGI-Specific OpenGL Features or IRIX-Specific Functionality

The following are SGI-specific OpenGL features or IRIX-specific functionality that are not supported in the current release:

- Data pools (**pfDataPool**)
- Real-Time processor control (**pfuProcessManager**)
- Video retrace timing control (**pfVClock**)
- Clip textures (**pfClipTexture**, **pfMPClipTexture**, **pfImageTile**, etc.)
- Antialiasing (**pfAntialias**)
- Projected or shadowed light sources (part of **pfLightSource**)
- Dynamic video resize (**pfVideoChannel**)
- Calligraphic light points (**pfCalligraphic**)
- SGI Video Channel Extensions (**pfVideoChannel**)
- Direct file I/O (**pfFile**)

## Guidelines for New Applications

IRIS Performer 2.3 for Linux only operates in a single-process model, which allows certain poor coding practices to be utilized with regards to program structure. However, a future release of IRIS Performer for Linux will support the multi-process functionality of the **pfPipe** object for forked cull, draw, intersections, database paging, and asynchronous compute functionality familiar to IRIX users. In preparation for these future releases, here are some very important things to remember so that your new application will work when multiprocessed. For more detailed information on proper multiprocess-ready coding practices, refer to the *IRIS Performer Programmer's Guide*.

- When allocating any data that must be shared by the APP, ISECT, DBASE, COMPUTE, CULL, or DRAW stages of the pipeline, such as geometry data, be certain to allocate them from the shared arena and with functionality that uses **pfMalloc**(). Note this example:

```
pfGeoSet *gs = pfNewGSet(pfGetSharedArena());
```

- Any actual pointers to shared memory that should be visible from the different pipeline stages should be allocated between **pfInit**() and **pfConfig**(). Data allocated to global variables initialized after **pfConfig**() will only be visible to the process that did the allocation. This is a correct example:

```
SharedData *s;
pfInit();
pfMultiprocess(PFMP_DEFAULT);
/* allocate shared data before fork()'ing parallel processes */
s = (SharedData*) pfMalloc(sizeof(SharedData),
pfGetSharedArena());
pfConfig();
```

- Only issue OpenGL commands in a pipeline initialization callback, channel draw callback, or node draw callback. When multiprocessing, the draw callbacks will be properly executed in the DRAW process, which is the only process with a valid GLX context.