Programming on Silicon Graphics Systems: An Overview

Document Number 007-2476-001

CONTRIBUTORS

Written by Eleanor Bassler
Edited by C. Kleinfeld
Photography by Nancy Cam

Programming on Silicon Graphics Systems: An Overview
Document Number 007-2476-001

Contents

List of Figures

List of Tables

About This Guide

If your goal is to write application programs that run on Silicon Graphics® computers, and if you know quite a bit about UNIX® but not much about the IRIX™ operating system, this book is for you. It doesn't tell you everything you need to know to write your IRIX application, but it does give you an overview of IRIX and the tools and libraries it provides, and it tells you where to go to learn more.

The IRIX operating system is Silicon Graphics' implementation of the UNIX operating system. All application programs written for Silicon Graphics platforms run in the IRIX environment. Writing an application with a graphical user interface that runs on Silicon Graphics platforms requires the IRIX operating system, a compiler for your source code, tools to build your user interface, tools to debug and tune your program, and typically one or more application libraries. This guide describes the software Silicon Graphics provides to perform these tasks.

Figure i illustrates the relationship between Silicon Graphics software, your application, and your user. In this example, an application uses one of the application libraries, Open Inventor, to create an image and display it on the monitor.

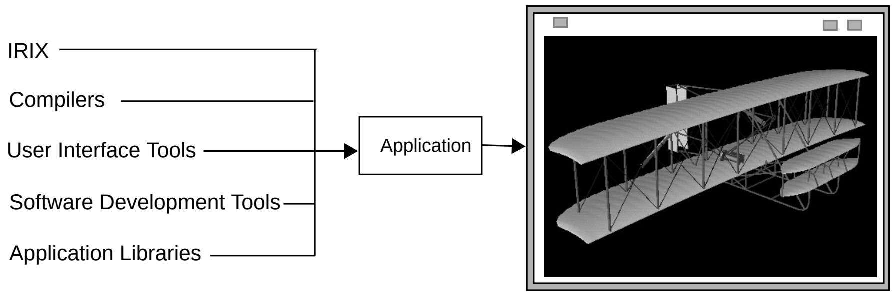

Figure i Building an Application with Silicon Graphics Software

What This Guide Contains

This introduction to programming on Silicon Graphics computers contains the following chapters:

- Chapter 1, “The IRIX Operating System,” briefly describes IRIX, the UNIX standards with which it complies, and the features added beyond UNIX to support the graphics and multiprocessing capabilities of Silicon Graphics platforms.
- Chapter 2, “IRIX Developer Documentation,” describes the documentation available for IRIX application developers.
- Chapter 3, “Compilers,” briefly describes the compilers available to IRIX developers.
- Chapter 4, “User Interface Tools,” tells you about the tools Silicon Graphics provides to help you develop a graphical user interface for your application.
- Chapter 5, “Software Development Tools,” describes the tools you can use to compile, debug, and tune your application. It also tells you about configuration management and version control software.
- Chapter 6, “Application Libraries,” provides an introduction to the application libraries you can use for graphics, image processing, digital media, and printer / scanner management applications.

What You Should Know Before Reading This Guide

This guide assumes that readers are experienced programmers who are familiar with a UNIX programming environment, but not necessarily familiar with IRIX. An understanding of object-oriented programming is helpful when reading parts of this book.

Chapter 1

The IRIX Operating System

This chapter provides a brief overview of the IRIX operating system, Silicon Graphics' implementation of the UNIX operating system. It describes IRIX compliance with standards, summarizes the features unique to IRIX (such as support for graphics hardware), briefly discusses writing device drivers (which run as part of the IRIX kernel), and lists the hierarchy of tools and libraries you can use when you write an IRIX application.

About the IRIX Operating System

IRIX, Silicon Graphics' implementation of the UNIX operating system, is based on UNIX System V, Release 4 (SVR4).

Standards Compliance

IRIX provides standard SVR4 programming interfaces and BSD networking, and complies with the following standards:

- System V Interface Definition, Issue 3 (SVID3), which is the defining document for SVR4. IRIX provides the SVR4 Applications Programming Interface (API) and the Applications Binary Interface (ABI) as defined in SVID3.
- *X/Open Portability Guide, Issue 3 (XPG3),* which specifies a set of programming interfaces to be provided by operating systems in order to facilitate the writing of portable programs.
- POSIX P1003.1, another standard for portable programming.
- X Window System, Version 11, Release 6 (X11R6). IRIX provides the industry-standard X Window System™.

- OSF / Motif™ Release 1.2. IRIX includes IRIS IM™, the Silicon Graphics port of the industry-standard OSF / Motif user-interface toolkit.
- OpenGL™. IRIX provides a full implementation of this standard for 3D graphics.

Internationalization

Internationalization is the process of making a program capable of running in more than one spoken-language environment without recompiling. Internationalized software can be made to produce output in a user's native language, to format data (such as dates and currency values) according to local custom, and in other ways make the software more comprehensible for people whose culture is different than that of the original software developer.

Table 1-1 lists the internationalization features supported by IRIX and the standards with which this implementation complies.

Table 1-1 IRIX-Supported Internationalization Features

Feature	**Standard**
Locales	ANSI C and POSIX (ISO0045-1)
XPG / 3 message catalogs and interpretation of locale strings	*X/OPEN Portability Guide, Issue 3* (XPG / 3)
Multi-National Language Support (MNLS) message catalogs	UNIX System V Release 4
Input methods Text rendering Resource files	X11R6

The chapter titled "Internationalization" in *Topics in IRIX Programming* describes functions you can use and guidelines you can follow to create an application that runs in more than one spoken-language environment.

Networking

The network programming facilities available with the IRIX operating system include the following:

- The Transport Layer Interface (TLI) defined in ISO-OSI, using System V Release 4 STREAMS modules.

  The International Standards Organization (ISO) has developed a standard known as the Reference Model of Open Systems Interconnection (ISO-OSI). This model uses a layered view of networking. The TLI, which defines an interface between two of these layers, provides a set of functions that applications can call to perform various network operations. TLI conforms to the MIPS® ABI (Application Binary Interface).

- Network interfaces (sockets) defined by the 4.3 release of the Berkeley Software Distribution (BSD).

  The 4.3BSD Inter-Process Communication (IPC) facility provides a socket interface that enables low-level access to network addressing and data transfer. IRIX supports *libsocket*, the standard SVR4 method of accessing the BSD4.3 networking interface. The *libsocket* library provides a socket interface that is MIPS ABI compliant.

- An implementation of the Sun Microsystems® Remote Procedure Call (RPC) library.

  RPC implements a remote procedure call model, in which a procedure executing on a remote system can be treated as a local procedure call by the calling application. RPC enables synchronous execution of procedure calls on remote hosts and provides transparent access to network facilities.

The IRIX operating system implements the Internet Protocol (IP) suite. The IP suite is a collection of layered protocols developed by the Department of Defence Advanced Research Project Agency (DARPA). The two most widely used IP protocols are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). The IRIX operating system implements the Internet Protocol suite and UNIX domain sockets using the 4.3BSD UNIX socket mechanism.

X Window System

The Silicon Graphics native X Window System is based on the X Version 11 Release 6 (X11R6) standard that supports multiple visuals, overlay windows, the X Input Extension, the Display PostScript™ Extension, the Shape Extension, and the Shared Memory Extension. Read “X Window System” on page 31 for more information about the X Window System on Silicon Graphics platforms.

Silicon Graphics Extras

In addition to the standard UNIX features described in “About the IRIX Operating System”, IRIX provides software that supports the graphics and multiprocessing capabilities of Silicon Graphics platforms. These added features include:

- Fast 3D graphics
- Parallel programming
- Real-time enhancements
- High-performance I/O
- An icon-oriented interface to IRIX

Fast 3D Graphics

Most Silicon Graphics systems are equipped with specialized graphics hardware and software. Programs can take advantage of this graphics support by using:

- OpenGL interfaces—the industry standard interfaces for 3D rendering across multiple platforms
- Open Inventor™—a graphics library, built on OpenGL, for creating interactive graphics applications
- IRIS Graphics Library™ (IRIS GL)—a library of subroutines for creating color graphics and animation

- IRIS Performer™—a graphics library, built on IRIS GL, for creating real-time graphics and visual simulation applications
- ImageVision Library™—an image processing library built on IRIS GL

"Graphics Libraries" on page 55 and "ImageVision Library" on page 63 describe these libraries in greater detail.

Parallel Programming

IRIX is designed to take advantage of multiple processors and provides interfaces to support parallel programming. These interfaces include:

- Low-overhead inter-process communication routines—The IRIX IPC mechanisms use a shared arena (shared memory) for communication between processes. This arena is mapped into a process's user space, which means most of the shared arena IPC functions don't have to make system calls. This keeps overhead lower than for standard System V IPC. Read the chapter titled "Inter-Process Communication" in *Topics in IRIX Programming* for more information about shared arenas.
- Multiprocessor control routines—IRIX supports the sysmp(2) commands MP_RESTRICT and MP_MUSTRUN, and the runon(1) and mpadmin(1) commands. These allow you to control the distribution of processes among the processors in a multiprocessor system. Read the *REACT/Pro Release Notes* for more information about these commands.
- Shared address space (sproc)—This system call creates a new process that is a clone of the calling process and that shares the virtual address space of the calling process.

Real-Time Enhancements

IRIX includes a number of extensions to enable programs to achieve real-time response. You can use these features to

- accurately time events
- control allocation of memory to the process
- provide for priority scheduling

The *REACT/Pro Release Notes* tell you where you can find more information about the mechanisms available to help you achieve real-time performance and how to use these features together to create the best real-time environment.

High-Performance I/O

IRIX includes several special programming interfaces for high-performance I/O. These include:

- Memory-mapped files. You can use the mpin(2) and munpin(2) commands to lock data into memory, thus reducing I/O activity.
- Asynchronous I/O—IRIX has added support for asynchronous I/O in accordance with the specification in POSIX 1003.4a Draft 12.
- Direct I/O—This bypasses the system buffer cache.

These interfaces are described in the *REACT/Pro Release Note*s.

Indigo Magic Environment

The Indigo Magic™ desktop is an end-user environment that provides an icon-oriented interface to the IRIX system. Users can launch applications and select files using icons instead of pathnames. You can integrate your application into the desktop environment to give it the Indigo Magic look and feel.

"Indigo Magic Desktop Integration" provides an overview of desktop integration. The *Indigo Magic Desktop Integration Guide* covers the topic in detail.

Device Drivers

A device driver enables communication between a user process and a peripheral device. In addition to kernel-level drivers, there are user-level drivers and STREAMS modules. Device drivers perform functions such as taking the device online and offline, transmitting data from the kernel to the device, receiving data from the single device and passing it to the kernel, and handling and reporting I/O errors.

Silicon Graphics provides drivers for many popular devices. However, you still may find that you need to write a device driver to support a device for which no driver is available.

The IRIX filesystem provides a device-independent interface that allows device drivers to be opened, read, written, and closed as though they were files. The program issues standard system I/O calls, and the system then calls the driver you've written to handle the device.

Figure 1-1 illustrates the relationship between the various software modules that connect the user application and the device with which it needs to communicate.

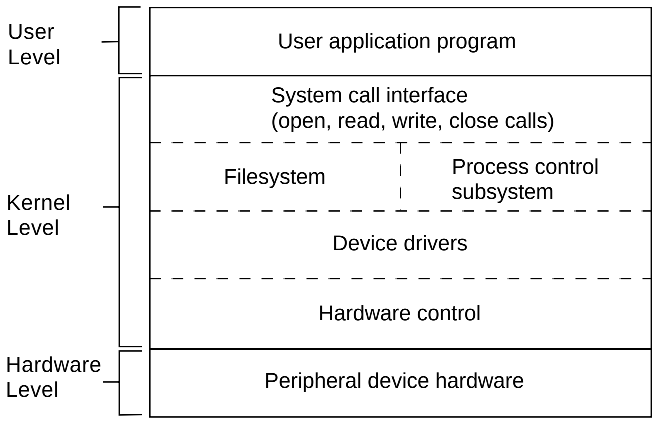
User Level
Kernel Level
Hardware Level
User application program
System call interface (open, read, write, close calls)
Filesystem
Process control subsystem
Device drivers
Hardware control
Peripheral device hardware

Figure 1-1 Driver Position in the Kernel

The kernel interfaces for IRIX device drivers are

- SVR4 Device Driver interface/Driver Kernel Interface (SVR4 DDI/DKI). IRIX uses the multiprocessor version of DDI/DDK.
- SVR4 STREAMS Interface, documented in the *UNIX System V Release 4 Streams Programming Guide.*

- IRIX 5.x Data Link Provider Interface (DLPI).
- User-level device drivers for VME, EISA, and SCSI.

The following list describes some Silicon Graphics-specific features you may need to include in a device driver:

- Memory mapping extensions (map and unmap)—the Silicon Graphics-specific memory mapping functions used by a device driver.
- Utility extensions—the Silicon Graphics-specific utility functions a developer needs to include in a device driver.
- Data structure extensions—data structure extensions supported by Silicon Graphics include *eisa_dma_buf* and *eisa_dma_cb*.
- Four new kernel definitions.

Writing an Application Program

The IRIS Developer's Option (IDO) is required for writing applications that run on Silicon Graphics platforms. This option provides the basic software and documentation for the development environment. "The IRIS Developer's Option" on page 13 tells you more about the IDO.

When you write an application for a Silicon Graphics platform, you can choose from a number of languages, toolkits, and libraries supplied by Silicon Graphics for developers. Your choice depends on the nature of your application and, to some degree, on your personal preference. Here are some choices you can make and the options available under IRIX:

- A programming language. Choose a language that is best suited for your application. Silicon Graphics provides compilers for C, C++, Fortran, Pascal, and Ada®. If your application runs on a multiprocessing platform and performance is a key consideration, you may want to choose either Power C or POWER Fortran, the versions of C and Fortran with special support for multiprocessing.
- A graphical user interface. If you plan to incorporate a graphical user interface in your application, you can use one of the user interface toolkits provided by Silicon Graphics rather than writing your own. These toolkits are IRIS ViewKit™ and IRIS IM, which is Silicon Graphics' port of the industry-standard OSF/Motif. They allow you to

build an interface for your application that looks like the interfaces of other applications running in the Indigo Magic desktop environment.

- Software development tools. When you order the IDO, you receive tools for compiling, debugging, and tuning your application. These tools include the compiler driver, the *dbx* debugger, and several performance tuning tools. You can also order the CASEVision WorkShop, which provides several interactive, graphical tools for debugging and tuning your application. If your project is large, you may need configuration management and version control. You can use the CASEVision™/Tracker and CASEVision/ClearCase tools for these tasks.
- An application library. You may want to use one of the application libraries developed by Silicon Graphics. These libraries provide tools for developers of 3D graphics, image processing, digital media, and printer/scanner management applications.

Figure 1-2 shows, in a hierarchical arrangement, the components you can select.

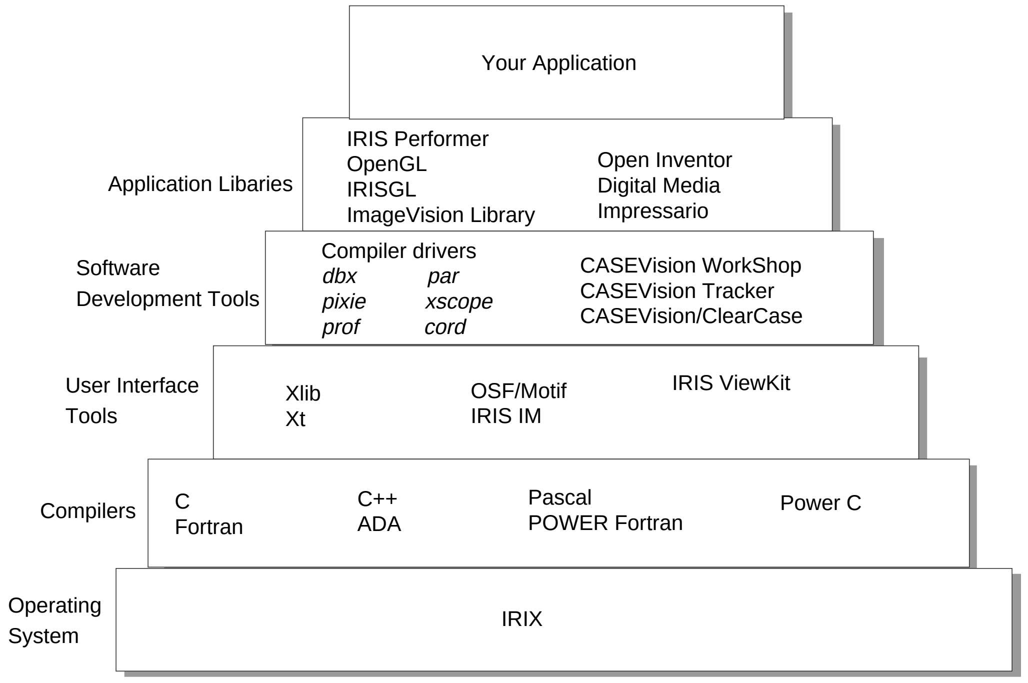

Figure 1-2 Component Hierarchy of an IRIX Application Program

Now that you have an overview of the IRIX programming environment, you're ready to learn more about the tools and libraries available and the documentation Silicon Graphics provides for reading about them. Chapter 2 lists the IRIX programming documentation. Chapters 3 through 6 describe IRIX compilers, tools, and application libraries in greater detail.

Chapter 2

IRIX Developer Documentation

This chapter tells you about the documentation Silicon Graphics provides to describe the operating system, compilers, tools, and libraries you can use to develop your applications. It presents this information in three forms:

- A documentation roadmap that lists the available developer documentation in graphical form.
- A list of the manuals provided to everyone who orders the IRIS Developer's Option (IDO).
- An annotated list of all IRIX developer documentation, grouped by function:

Table 2-1	Operating system manuals
Table 2-2	Programming language manuals
Table 2-3	Documentation about user interface toolkits
Table 2-4	Manuals about software development tools
Table 2-5	Application libraries

A Documentation Roadmap

The diagram in Figure 2-1 shows the documentation provided by Silicon Graphics for application developers.

Operating System

IRIX

- **Programming on Silicon Graphics Systems: An Overview**
- **Topics in IRIX Programming**
- **IRIX Network Programming Guide**

Device Driver

- **IRIX Device Driver Programming Guide**
- **IRIX Device Driver Reference Pages**

Compilers

C++

- C++ Programming Guide
- C++ Language System Overview
- C++ Language System Guide

C

- **C Language Reference Manual**
- IRIS Power C User's Guide

Fortran

- Fortran 77 Language Reference Manual
- Fortran 77 Programmer's Guide
- POWER Fortran Accelerator User's Guide

Pascal

- Pascal Programming Guide

Ada

- Ada X/Motif Interface (AXI) Programmer Reference Manual

User Interface Tools

Window System

- **Xlib Programming Manual**
- **X Toolkits Intrinsics Programming Manual**
- **X11 Input Extension Library Specification**

Desktop Environment Integration

- **Indigo Magic Desktop Integration Guide**
- **Silicon Graphics User Interface Guidelines**

User Interface Toolkits

- IRIS Viewkit Programmer's Guide
- **OSF/Motif Programmer's Guide**
- **OSF/Motif Programmer's Reference**
- **OSF/Motif Style Guide**
- **IRIS IM Programming Notes**

Tools

Compiling and Performance Tuning

- **MIPS Compiling and Performance Tuning Guide**

Debugging

- ***dbx* User's Guide**
- ***dbx* Quick Reference**

Programming Tools

- CASEVision/WorkShop User's Guide, Vol. 1–2
- CASEVision Environment Guide
- CASEVision/WorkShop Mega Dev User's Guide
- CASEVision/WorkShop Pro MPF User's Guide

Version Control/ Configuration Management

- CASEVision/Tracker Design Guide
- CASEVision/Tracker User's Guide
- CASEVision/ClearCase

Application Libraries

OpenGL

- **OpenGL Programming Guide**
- OpenGL Reference Manual
- **The OpenGL Porting Guide**

IRIS GL

- Graphics Library Programming Guide, Vol. 1–2
- Graphics Library Programming Tools and Techniques

Open Inventor

- The Inventor Mentor
- The Inventor Toolmaker
- Open Inventor C++ Reference Pages

Performer

- IRIS Performer Programming Guide
- IRIS Performer Reference Pages
- IRIS Performer Quick Reference

ImageVision Library

- ImageVision Library Programming Guide
- ImageVision Library Inheritance Hierarchy

Digital Media

- **IRIS Digital Media Programming Guide**

Printer/Scanner Management

- Impressario Programming Guide

Document titles shown in **bold** are part of the IDO (IRIS Developer's Option)

Figure 2-1 Documentation for Developers

The IRIS Developer's Option

The IRIS Developer's Option (IDO) gives you the basic software and documentation you need to write applications for Silicon Graphics platforms. When you order the IDO, you'll receive online manuals for the products that are part of the IDO. You can view these manuals using IRIS InSight™, an online documentation viewer. The manuals you receive as part of the IDO are:

- *Programming on Silicon Graphics Systems: An Overview*
- *Topics in IRIX Programming*
- *IRIX Network Programming Guide*
- *IRIX Device Driver Programming Guide*
- *IRIX Device Driver Reference Pages*
- *C Language Reference Manual*
- *dbx User's Guide*
- *dbx Quick Reference*
- *OpenGL Programming Guide*
- *OpenGL Porting Guide*
- *Indigo Magic Desktop Integration Guide*
- *Silicon Graphics User Interface Guidelines*
- *Xlib Programming Manual*
- *X11 Input Extension Library Specification*
- *X Toolkit Intrinsics Programming Manual*
- *IRIS IM Programming Notes*
- *OSF/Motif Programmer's Guide*
- *OSF/Motif Programmer's Reference*
- *OSF/Motif Style Guide*
- *IRIS Digital Media Programming Guide*
- *MIPS Compiling and Performance Tuning Guide*
- *Software Packager User's Guide*

If you want to order printed versions of these manuals, call SGI Express at 1-800-800-7441, ext. 4650, and supply the desired product codes. Refer to the *Documentation Catalog for IRIX 5.3*, viewable from IRIS Insight, to determine product codes and other information about ordering manuals.

An Annotated List of Manuals

The five tables in this section list the manuals available for developers, along with the product name for each manual and a brief description of the content of the manual.

Operating System Level Documentation

Table 2-1 lists manuals containing information about topics at the operating system level. You receive online versions of these manuals when you order the IDO.

Table 2-1 Operating System Level Manuals

Product Name	Title	Description
IDO	*Programming on Silicon Graphics Systems: An Overview*	Provides an overview of the IRIX operating system, programming languages, software development and user interface toolkits, and application libraries.
	Topics in IRIX Programming	Describes selected topics in IRIX programming, including interprocess communication, file and record locking, fonts, and internationalization.
	IRIX Network Programming Guide	Describes various approaches to writing software that sends or receives information through a network.

Table 2-1 (continued) Operating System Level Manuals

Product Name	Title	Description
	IRIX Device Driver Programming Guide	Tells you how to write device drivers to control peripheral devices.
	IRIX Device Driver Reference Pages	Contains reference pages that describe the functions and data structures used in writing device drivers.

Compilers

Table 2-2 lists the manuals containing information about programming languages supported by Silicon Graphics. Some of these manuals come with IDO. Others you receive only when you order the associated product.

Table 2-2 Compiler Manuals

Product Name	Title	Description
C++	*C++ Programming Guide*	Tells you how to compile, link, and run a C++ program. Documents the interface between C and C++, and the differences between the 64- and 32-bit versions of the compiler. Describes how to use the Delta / C++ compiler.
	C++ Language System Overview	Contains an overview of the language features of C++. Describes the differences between C and C++.
	C++ Language System Library	Introduces the iostream support in the C++ library and describes facilities for using complex-number arithmetic.
IDO	*C Language Reference Manual*	Contains a summary of the syntax and semantics of the C programming language as implemented on Silicon Graphics platforms.

Table 2-2 (continued) Compiler Manuals

Product Name	Title	Description
IRIS Power C	*IRIS Power C User's Guide*	Describes how to use IRIS Power C, a C compiler for developers who want to make efficient use of IRIX multiprocessors by executing parts of their programs concurrently.
Fortran	*Fortran 77 Language Reference Manual*	Describes the Fortran 77 language specifications as implemented on the IRIS workstations.
	Fortran 77 Programmer's Guide	Describes the implementation of Fortran 77 for IRIX and the IRIS workstations.
POWER Fortran	*POWER Fortran Accelerator User's Guide*	Describes the features of the POWER Fortran Accelerator™ (PFA), a source-to-source preprocessor that allows you to run Fortran 77 programs on multiprocessor systems.
Pascal	*Pascal Programming Guide*	Describes the general syntax of the Pascal programming language, including data structures and program flow control.
IRIX5 AXM	*Ada X/Motif Interface (AXI) Programmer Reference Manua*l	Contains information about programming in the AXM environment, including the Ada preprocessor, the Statistical Analyzer, and interfaces to libraries written in other languages.

User Interface Tools

Table 2-3 lists the manuals containing information about the user interface libraries and toolkits supported by IRIX. Some of these manuals come with the IDO. Others you receive only when you order the associated product.

Table 2-3 Documentation About User Interface Tools

Product Name	Title	Description
IDO	*Xlib Programming Manual*	Describes the X library, the C interface to the X Window System.
	X11 Input Extension Library Specification	Describes the input extension to the X11 server. This extension supports the use of additional input devices beyond the pointer and keyboard devices defined by the core X protocol.
	X Toolkit Intrinsics Programming Manual	Describes how to use the Xt Intrinsics library to write X Window System programs.
	OSF/Motif Programmer's Guide	Describes how to use the OSF / Motif API to create Motif applications.
	OSF/Motif Programmer's Reference	Contains descriptions of the OSF / Motif toolkit, window manager, and user interface language commands and functions.
	OSF/Motif Style Guide	Provides a framework of behavior specifications to guide developers in the design and implementation of products consistent with the OSF / Motif user interface.
	IRIX IM Programming Notes	Describes how to develop applications using IRIS IM. Contains advice for X and Xt programmers about programming in the Silicon Graphics X environment.

Table 2-3 (continued) Documentation About User Interface Tools

Product Name	Title	Description
	Indigo Magic Desktop Integration Guide	Explains how to integrate applications into the Indigo Magic desktop environment.
	Silicon Graphics User Interface Guidelines	Helps you create products whose user interface is consistent with other applications in the Indigo Magic desktop environment.
C++	*IRIS ViewKit Programmer's Guide*	Describes how to create programs using IRIS ViewKit, a toolkit that provides user interface facilities for applications.

Software Development Tools

Table 2-4 contains a list of the manuals describing the tools for compiling, debugging, and tuning your application. Some of these manuals come with IDO. Others you receive only when you order the associated product.

Table 2-4 Software Development Tools Manuals

Product Name	Title	Description
IDO	*MIPS Compiling and Performance Tuning Guide*	Describes the compiler system, dynamic shared objects (DSOs), and program debugging tools. It explains ways to improve program performance using *prof*, *pixie*, and the optimization options.
IDO	*dbx User's Guide*	Describes how to use *dbx*, a source level debugger, to debug C, C++, Fortran 77, Pascal, and assembler programs. This includes how to execute a program using *dbx*, examine source code, control program execution, debug machine language code, and debug multiple processes.
CASEVision WorkShop	*CASEVision/WorkShop User's Guide - Volume 1*	Describes how to use the Debugger and Static Analyzer tools in the WorkShop toolset.
	CASEVision/WorkShop User's Guide - Volume 2	Describes how to use the Performance Analyzer, Tester, and Build Manager tools in the WorkShop toolset.
	CASEVision Environment Guide	Describes the common environment that all the tools in the CASEVision product line share.

Table 2-4 (continued) Software Development Tools Manuals

Product Name	Title	Description
	CASEVision/WorkShop MegaDev User's Guide	Describes the C++ Browser and the Fix and Continue utilities. The C++ Browser lets you view the structure of any set of C++ classes. Fix and Continue allows you to redefine functions and then continue execution without recompiling.
	CASEVision/WorkShop Pro MPF User's Guide	Describes the Pro MPF tool, an interactive, visual comparison of the original source with transformed, parallelized code.
	CASEVision/Tracker Design Guide	Describes the CASEVision / Tracker, a tool for creating systems to track bugs and enhancement requests.
	CASEVision/Tracker User's Guide	Describes how to use the Request Tracking System (RTS), a system for tracking bugs and requests for enhancements.
	CASEVision/ClearCase	Describes the CASEVision / ClearCase software configuration management system.
DESK	*Software Packager User's Guide*	Describes how to use the Software Packager (*swpkg*), a graphical tool for packaging software for installation on Silicon Graphics workstations. Products packaged with Software Packager can be installed with Software Manager (*swmgr*), an Indigo Magic Desktop utility for installing software.

Application Libraries

Table 2-5 contains a list of the manuals describing the graphics, image processing, digital media, and printer / scanner libraries available on an IRIS system. Some of these manuals come with IDO. Others you receive only when you order the associated product.

Table 2-5 Application Libraries Manuals

Product Name	Title	Description
IDO	*OpenGL Programming Guide*	Describes how to use OpenGL, a platform-independent standard for rendering 3D graphics.
	OpenGL Reference Manual	Contains the reference pages for OpenGL, the OpenGL Utility Library (GLU), and GLX—the OpenGL extension to X. Also includes an overview and summary of OpenGL routines and commands.
	The OpenGL Porting Guide	Provides directions, hints, and tips for porting your IRIS Graphics Library software to OpenGL.
Graphics Library	*Graphics Library Programming Guide, Volumes 1-2*	Describes the IRIS Graphics Library API.
	Graphics Library Programming Tools and Techniques	Describes useful software tools and programming techniques for use with IRIS GL.

Table 2-5 (continued) Application Libraries Manuals

Product Name	**Title**	**Description**
Open Inventor	*The Inventor Mentor*	Provides basic information on programming with Open Inventor.
	The Inventor Toolmaker	Provides advanced information on extending Open Inventor by creating new C++ classes and customizing existing classes.
	Open Inventor C++ Reference Pages	Contains the C++ reference pages for Open Inventor.
ImageVision	*ImageVision Library Programming Guide*	Describes how to use the ImageVision Library to perform image processing tasks.
	The ImageVision Library Inheritance Hierarchy	A quick reference card that shows the entire inheritance hierarchy of the ImageVision Library.
IRIS Performer	I*RIS Performer Programming Guide*	Provides an overview of IRIS Performer and describes the API of IRIS Performer's two main libraries: *libpf*—the high-level visual simulation library, and *libpr*—the low-level high-performance graphics library.
	I*RIS Performer Reference Pages*	Contains the reference pages for all the functions in the two primary IRIS Performer libraries and the adjunct IRIS Performer utilities library.
	I*RIS Performer Quick Reference*	Lists the function prototype for each function in the IRIS Performer libraries: *libpr*, *libpf*, *libpfutil*, and *libpfsgi*.

Table 2-5 (continued) Application Libraries Manuals

Product Name	**Title**	**Description**
IDO	*IRIS Digital Media Programming Guide*	Describes the API's of the libraries that comprise the IRIS Digital Media Development Environment. This environment includes the Audio, Audio File, CD, DAT, MIDI, Video, IndigoVideo™, Compression, and Movie Libraries.
Impressario	*Impressario Programming Guide*	Describes Impressario™, which provides tools for developers who need to print and scan from their applications, or who need to write printer or scanner drivers.

Chapter 3

Compilers

This chapter describes the compilers supported by IRIX and the programming language standards adhered to in the implementation of these compilers. Read "Creating Executable Files" on page 41 to learn how to compile and debug programs written in these languages. Refer to Table 2-2 on page 15 for a list the manuals you can read to learn more about the topics discussed in this chapter.

C and C++

Silicon Graphic provides compilers for both the C and C++ programming languages.

The IRIX C compiler conforms to the ANSI C standard as well as "traditional C", the dialect of C defined by Kernigan and Ritchie in *The C Programming Language*. Compiler options allow you to compile programs written in "traditional C", pure ANSI C, or ANSI C with Silicon Graphics extensions. ANSI C is part of the IRIS Developer's Option (IDO).

The Silicon Graphics implementation of C++ conforms to the standard as defined in *The Annotated C++ Reference Manual* by Margaret Ellis and Bjarne Stroustrup.

The IRIX operating system supports two versions of the C++ compiler.

NCC	This is a 32-bit native compiler that implements all the features of the language described in *The Annotated C++ Reference Manual* by Margaret Ellis and Bjarne Stroustrup.
DCC	This is the Delta / C++ compiler that is available as part of the CASEVision / Workshop Pro C++ package. Delta C++ is an extension to C++. It is a native compiler (not a preprocessor). It supports dynamic classes, which minimizes the need for recompilation if you modify classes.

Fortran 77

Fortran, as implemented on Silicon Graphics IRIS workstations, contains the full ANSI Programming Language Fortran (X3.9-1978). It has extensions that provide full VMS Fortran compatibility to the extent possible without the VMS operating system or data representation. It also contains extensions that provide partial compatibility with programs written in VMS Fortran and Fortran 77.

The Parallelizing Compilers

If your application runs on a multiprocessor platform and performance is a critical issue, you may want parts of the program to run concurrently. Two Silicon Graphics compilers, C and Fortran, have preprocessors that analyze source code and produce, where possible, object code that utilizes the multiprocessor environment.

Power C

IRIS Power C allows your program to make efficient use of Silicon Graphics multiprocessor platforms by generating code segments that execute concurrently. Power C consists of the standard C compiler and a preprocessor that automatically analyzes sequential code to determine where loops can run in parallel. The preprocessor generates a modified version of the source code with multiprocessing directives added. The C compiler, when it compiles the modified source code, interprets the directives and produces object code that uses multiple processors. An advantage of Power C is that you can use it to recompile existing serial C programs so that they run efficiently on multiprocessor computers without hand recoding.

The IRIS Power C Analyzer (PCA) is the C code optimization preprocessor that detects potential parallelism in C code. It also performs other optimizing tasks. The Power C Analyzer can

- direct C code to run in parallel
- determine data dependencies which might prevent code from running concurrently

- distribute well-behaved loops and certain other code across multiprocessors
- optimize source code

You can use PCA either as a standalone tool or as a phase of the C compiler. You can also enter the directives that produce concurrent code directly into your program rather than using the PCA. Figure 3-1 illustrates the role of the PCA in producing an executable module that can utilize more than one processor on a multiprocessor system.

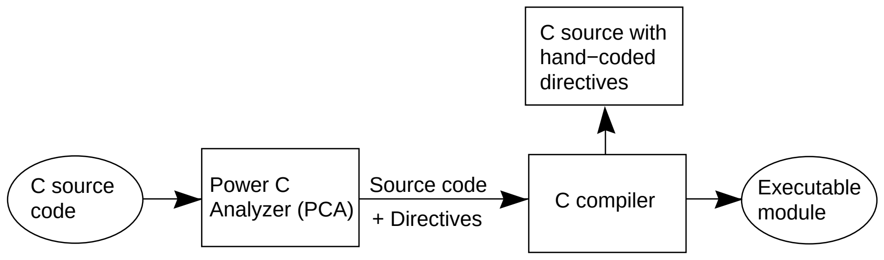

Figure 3-1 Using Power C to Produce a Parallelized Program

Power C can produce a listing containing information about the loops that it parallelizes and those that it cannot. Using this information, you may be able to modify your source so that a subsequent Power C compilation produces more efficient code. You can select a PCA compilation of your source code by specifying the *pca* compiler driver option when you compile your program.

Power Fortran Accelerator

The Power Fortran Accelerator (PFA) is a source-to-source preprocessor that enables you to run existing Fortran 77 programs efficiently on the Silicon Graphics POWER Series™ multiprocessor systems.

PFA analyzes a program and identifies loops that don't contain data dependencies. It is a preprocessor that automatically inserts special compiler directives into a Fortran program to produce a modified copy of the source. The Silicon Graphics Fortran 77 compiler can then interpret these directives to generate code that can run across all available processors. Because the directives inserted by PFA look like standard Fortran 77 comment statements, PFA does not affect the portability of the code to non-Silicon Graphics systems. Figure 3-2 illustrates the role of PFA in producing an executable module that can utilize more than one processor on a multiprocessor system.

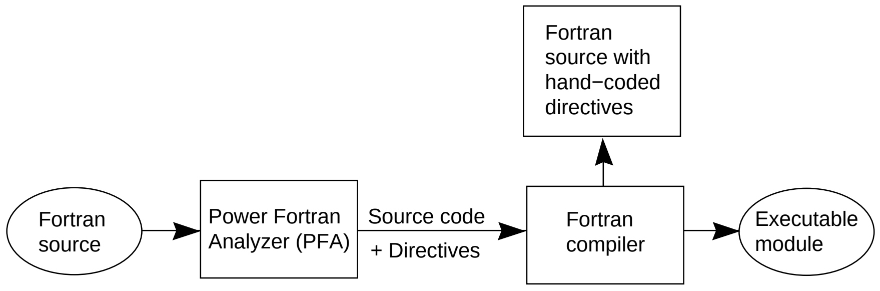

Figure 3-2 Using PFA to Produce a Parallelized Program

PFA can, if you request it, produce a listing file explaining which loops were parallelized and if not, why not. You may be able to use this information to modify your application for more efficient use of multiple processors.You can select a PFA compilation of your source code by specifying the *pfa* compiler driver option when you compile your program.

Pascal

The Pascal language supported by the Silicon Graphics Pascal compiler is an implementation of ANSI Standard Pascal (ANSI / IEEE770X3.97-1983). This implementation complies with ANSI requirements except for some extensions. These extensions include:

- names—allows underscores in identifiers and use of lowercase for public names
- constants—four extensions for constants
- statement—seven statement extensions
- declaration—six declaration extensions
- predefined procedures—four new procedures

- predefined functions—20 new functions
- predefined data types and predefined data type attributes
- extensions that affect the compile process

Ada

The Silicon Graphics AXM (Ada X / Motif) Ada Development System provides an Ada development environment for Silicon Graphics platforms. It consists of

- Ada compiler (ANSI / MIL-STD-1815A)
- X11 and OSF / Motif
- Bindings to the IRIS Graphics Library
- Non-intrusive, symbolic debugger

MP / Ada 6.2 is a multiprocessor Ada development system. It’s built using the POSIX-compliant threads model for Ada tasking. If your application runs on a multiprocessor platform, MP / Ada6.2 allows Ada tasks in this application to run concurrently.

Chapter 4

User Interface Tools

This chapter describes the user interface libraries and toolkits you can use to create a graphical interface between your application and its users. Refer to Table 2-3 on page 17 for a list of the manuals you can read to learn more about the topics discussed in this chapter.

X Window System

A window system allows a user to run several tasks at the same time and to view and control each of these tasks from a separate window. Silicon Graphics implements the X Window System, a hardware- and operating system-independent windowing system. The X Window System is a portable, network-based windowing system whose portability allows you to create applications that can run on many different workstations. You can compile and execute portable X application code in IRIX without modification.

The Silicon Graphics native X Window System is based on the X Version 11 Release 6 (X11R6) standard. The IRIX extensions to X include

- support for multiple visuals
- overlay windows
- the Display PostScript extension
- the Shape Extension that supports non-rectangular windows
- the X Input Extension that supports devices other than keyboard and mouse
- the Shared Memory Extension
- simultaneous display on multiple graphics monitors

- support for OpenGL
- PEX Version 5.1 supports 3D graphics

Xlib and Xt

The X library, known as *Xlib*, is the C-language programming interface to the X Window System. It is the lowest level programming interface to X. You can use *Xlib* to build a graphical user interface for your application, although most application developers choose higher level tools.

It's difficult to build applications with a graphical user interface that uses a low-level library such as *Xlib*. The *Xt* Intrinsics library (or simply *Xt*) simplifies this task by providing a library of C language routines designed to facilitate the interface to Xlib. Xt is a standard established by the X Consortium that provides an object-oriented programming style in the C language. Xt routines can be used to create interface components called widgets. OSF / Motif, for example, uses Xt to create its widget set (see "OSF / Motif and IRIS IM").

Silicon Graphics implements Xlib and Xt as defined by the standard and supports them as dynamic shared libraries.

4Dwm

The IRIS Extended Motif Window Manager, *4Dwm*, is an X Window System client based on *mwm*, the Motif Window Manager. In addition to the standard functions found in *mwm*, *4Dwm* provides:

- support for multiple screens
- session management functions that support the Indigo Magic desktop
- support for the desk overview feature of the Indigo Magic desktop, which allows the user to organize windows into related groups called desks
- control of background images on desks, which switches the screen background automatically when the user switches desks
- communication between the application and *4Dwm* using the *tellwm* program

For more detailed information about *mwm* and *4Dwm*, read the *mwm* and *4Dmw* reference pages.

User Interface Toolkits

Silicon Graphics provides developers with several user-interface toolkits that simplify the development of a graphical user interface. These toolkits supply a set of objects that appear in graphical form on the screen and allow users to interact with an application by manipulating these objects.

OSF/Motif and IRIS IM

The industry-standard OSF / Motif library provides user-interface objects (called widgets) to be used with Xt. The objects defined by OSF / Motif include menus, scrollbars, dialog boxes, and command buttons.

IRIS IM is Silicon Graphics' port of OSF / Motif for use on Silicon Graphics IRIS workstations. Figure 4-1 shows the relationship of an application to the various user interface libraries you might choose to use.

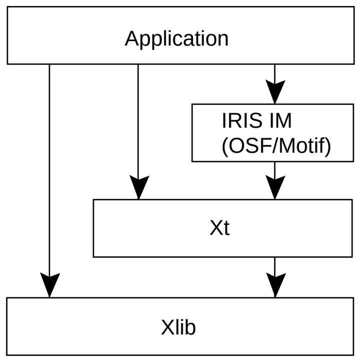

Figure 4-1 Hierarchy of User Interface Toolkits

The application programming interface (API) to IRIS IM is the same as that for OSF / Motif. However, Silicon Graphics has modified the appearance of some IRIS IM widgets so that they conform to the Silicon Graphics user interface style. You can select either the OSF / Motif or the IRIS IM widget set for use with your application, although using IRIS IM widgets is preferable because their appearance conforms to the Indigo Magic style.

In addition to the low-level building blocks such as buttons and scrollbars that IRIS IM provides, it also supplies some related software that isn't part of the standard OSF/Motif. This includes the GLwMDrawingArea widget for IRIS IM programs that use OpenGL to draw to a window within an IRIS IM application.

IRIS ViewKit

IRIS ViewKit is a C++ toolkit that provides user interface facilities for applications. It defines a collection of high-level components that you typically must implement in all applications, for example, components such as windows, menus, and dialogs. IRIS ViewKit components are designed to implement as many commonly used features as possible. To build a user interface component for your application, you can create a subclass of an IRIS ViewKit component and define any application-specific behavior. Also, with ViewKit classes as a base, you can create your own library of reusable components.

IRIS ViewKit is based on IRIS IM and uses IRIS IM widgets to implement all of its user interface components. You can make IRIS IM and X calls directly from your IRIS ViewKit application, but using ViewKit components directly is simpler and faster than creating your own components from low-level widgets. Figure 4-2 shows the relationship of an application to IRIS ViewKit and the libraries upon which it is built.

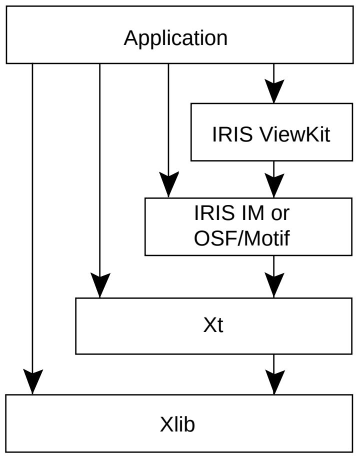

Figure 4-2 IRIS ViewKit in the Developer Environment

An IRIS ViewKit component is a C++ class that encapsulates sets of widgets and methods for their manipulation. Figure 4-3 shows a graphical representation of a portion of the ViewKit class hierarchy.

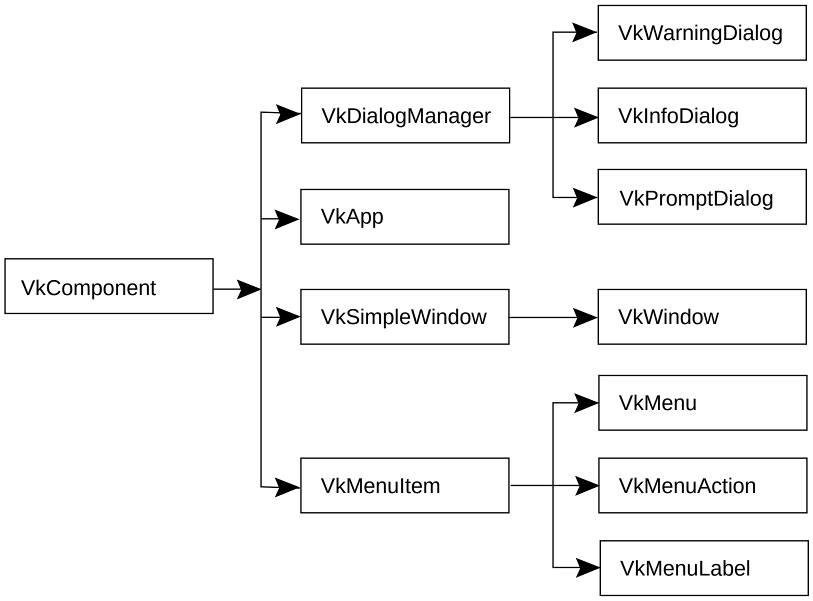

Figure 4-3 A Portion of the IRIS ViewKit Class Hierarchy

The functions of some of these IRIS ViewKit base classes are:

- VkComponent—This abstract class defines the basic structure and protocol for all ViewKit components.
- VkDialog Manager—This is the base class for specific dialog classes.
- VkApp—This base class handles application-level tasks such as Xt initialization, event handling, window management, and cursor control.
- VkSimpleWindow—This base class implements a simple top-level window (one that does not require a menu bar).
- VkMenu—This base class provides a standard set of functions for accessing and manipulating menu items.

Indigo Magic Desktop Integration

The Indigo Magic desktop is an end-user environment that provides an icon-oriented interface to the IRIX operating system and filesystem. Users can launch applications and select files using icons instead of typing shell commands and pathnames. Integrating your product into this desktop is an important part of your application. It is much easier for users of your application who are familiar with the desktop to get started if your interface conforms to the Indigo Magic standard.

Learning About Silicon Graphics User Interface Style

Silicon Graphics provides user interface style guidelines for developers of software products used on Silicon Graphics workstations. The *Silicon Graphics User Interface Guidelines* describes these style guidelines. Its purpose is to help you create products that are consistent with other applications and that integrate seamlessly into the Indigo Magic desktop environment.

The guidelines cover these areas:

- The design of icons for your application programs and files—they should be meaningful and behave appropriately in response to user actions.
- The appearance of your application's windows and the expected behavior these windows should support—such as when users should be able to size, move, and minimize windows.
- The individual components of your desktop interface, such as menus, controls, dialogs, and use of color.

Integrating Your Application Into the Desktop

Integrating your application into the desktop is an important step in creating your product. Since your users are already familiar with the appearance and behavior of applications in the desktop environment, you can simplify their use of your application by conforming to this standard. Here are the steps to follow to integrate your application:

1. Achieve the Indigo Magic look and feel
2. Create desktop icons for your application
3. Package your application for installation

Achieving the Indigo Magic Look and Feel

Here are some of the things you want to consider to achieve the Indigo Magic look and feel:

- Use schemes—Silicon Graphics includes schemes in its implementation of Xt. Schemes allow you to provide default colors and fonts for your application while also ensuring that users can easily select other colors and fonts according to their individual needs and preferences. If you provide default colors, your application will use the same colors and fonts as other applications on the desktop using default colors.
- Use enhanced widgets—IRIS IM provides new and enhanced widgets that are part of the Indigo Magic look and feel. Some of these widgets are the Color Chooser, the Dial, and the Thumbwheel. You can use these widgets by linking to the appropriate library.
- Provide window, session, and desk management for your application—Most users of Silicon Graphics systems use *4Dwm*, which is based on *mwm* (the Motif window manager).
- Create minimized windows—You can customize the minimized version of your application's window so it's easily recognized when a user clicks the minimize button.

Creating Desktop Icons

When you create icons for your application, perform these tasks:

- Draw the icons—Use IconSmith™, a tool for drawing desktop icons.
- Program the icon—Define the look and behavior of the icons in your application, for example, the actions to be taken if a user double-clicks an icon.
- Install the icon in the icon catalog—Use the *iconbookedit* command to install your icon in the desktop's Icon Catalog.

Packaging Your Application for Installation

Silicon Graphics recommends that you use the software packaging tool Software Packager (*swpkg*) to package your application for installation. This allows your users to easily install your application using Software Manager (*swmgr*), Silicon Graphics software installation program.

Chapter 5

Software Development Tools

After you've written the source code for your application, you typically compile, debug, and link it. You may also want to optimize the performance of the resulting executable code. This chapter describes the tools you can use to perform these tasks. It tells you how you can

- create executable modules from your source programs
- use IRIX tools to debug, analyze, and optimize your program
- use CASEVision programming tools to debug and optimize your program
- use CASEVision tools to perform version control and configuration management

Refer to Table 2-4 on page 19 for a list of the manuals you can read to learn more about the topics discussed in this chapter.

Creating Executable Files

An executable file (or executable) is a program that is ready for execution. To create an executable version of your application, compile your source code and link it with the appropriate libraries. This section briefly describes how to create executables.

Compiler Drivers

A compiler driver is a program that prepares your application for execution by calling subsystems that compile the source into object code, and then linking together object files, default libraries, and any other libraries you specify. Here are some of the compiler drivers Silicon Graphics provides and the programming language each supports:

Compiler Driver Name	Source Language
f77	Fortran
cc	C
pc	Pascal
CC	C++

When you invoke a compiler driver, specify the name of one or more source files and, optionally, one or more of the compiler driver options. The compiler drivers share a common set of options, although a few options are unique to a single driver or have different meanings for each driver. For example, the *–c* option suppresses the linker step for all drivers, while the *–C* option has one meaning for C and another for Pascal and Fortran. You can use compiler driver options to control these functions:

- program compilation
- program linking
- production of profiling information for performance tuning (see "Performance Tuning Tools")

Read the *MIPS Compiling and Performance Tuning Guide* for more information about compiler drivers. The reference page for each driver contains detailed information about command options for that driver.

Object Files and Dynamic Linking

Object files generated by Silicon Graphics compilers are in the Executable and Linking Format (ELF). ELF is the format specified by the System V Release 4 Applications Binary Interface (SVR4 ABI).

ELF provides support for dynamic shared objects (DSOs). A DSO is an object file that's shared by multiple applications as they are executing. The object code of a DSO is position-independent and can be mapped into the virtual address of several processes. DSOs are loaded at runtime (instead of at linking time) by the runtime loader, *rld*.

DSOs replace the static shared libraries provided with releases of IRIX prior to IRIX 5.0. You can, and generally should, use them in place of archive libraries. Using DSOs with your application provides you with several benefits. These include the following:

- Overall memory usage is minimized because code is shared.
- Executables linked with DSOs are smaller (and take less disk space) than those linked with unshared libraries because the shared objects are not part of the executable file image.
- Executables containing DSOs don't have to be relinked if the DSO changes.
- DSOs are easier to use and build than the static shared libraries available in releases of IRIX prior to IRIX 5.0.

Most libraries supplied by Silicon Graphics are available as DSOs. When you invoke a compiler driver to build an executable file from your source program, the driver links to DSOs unless you specify otherwise.

You can build your own DSOs if you have the IRIS Developer's Option installed on your system. You don't have to make any changes to your source code to make it part of a DSO. Just use *ld* with the *–shared* option to build the DSO.

IRIX Tools for Debugging and Tuning Your Program

IRIX provides a number of standard UNIX tools to aid you in debugging and tuning the performance of your program. It also provides performance tools developed at Silicon Graphics.

Debugging Tool

IRIX provides *dbx*, a source level debugger that allows you to debug C, C++, Fortran 77, Pascal, and assembler code. You can execute a program under *dbx* control to

- examine source code
- examine and change data

- control program execution
- debug machine language code
- debug multiple processes.

You can also use *dbx* to examine a core file, if a program crash occurs, to determine the point at which the crash occurred.

To use *dbx*

- compile the source using an appropriate compiler driver option, usually *–g*. This produces an executable containing symbol table information that is used by *dbx* during program execution.
- execute the program under *dbx* control.

Tools for Object File Query and Manipulation

Object files generated by Silicon Graphics compilers are in the Executable and Linking Format (ELF), the format specified by the SVR4 ABI. IRIX provides several tools to query and manipulate object files:

dis	Dissembles an object file into machine instructions.
elfdump	Lists the contents of an ELF-format object file.
file	Provides descriptive information on the properties of a file.
nm	Lists symbol table information.
size	Prints the size of each section of an object file.
strip	Removes symbol table and relocation bits from an object file, which saves space after you've debugged your program.

Performance Tuning Tools

IRIX provides several tools that you can use to optimize the performance of your application. Table 5-1 summarizes these tools and the paragraphs following the table describe each tool in greater detail.

Table 5-1 Summary of Performance Tuning Tools

Name of Tool	Function
prof	Measures the amount of time spent in various parts of a program—used in conjunction with the *–p* compiler driver option.
pixie	Counts the number of times basic blocks in a program are executed—used in conjunction with the *prof* program.
par	Traces system calls and scheduling activity.
cord	Rearranges procedures in your program to reduce paging and reduce instruction cache mapping.
xscope	Monitors connections between an X server and a client.

Using *prof* and *pixie*

You can use the profiling tools *prof* and *pixie* to find areas of your program where most of the execution time is spent. With this information, you can concentrate your effort on improving code efficiency in these parts of the program. The profiling tools provide two kinds of information:

- Program counter (pc) sampling, which measures the amount of execution time spent in various parts of a program. It does this by interrupting program execution every 10 milliseconds and recording the value of the program counter.
- Basic-block counting, which counts the number of times each basic block executes. A basic block is a region of the program that can be entered only at the beginning and exited only at the end.

To obtain pc sampling data, specify the *–p* compiler driver option when you prepare your program for execution. A program prepared using this option produces pc sampling data during program execution. To view the pc sampling data, run the *prof* program. *prof* analyzes the data files generated during program execution and produces a formatted listing.

Figure 5-1 shows the steps necessary to get pc sampling information.

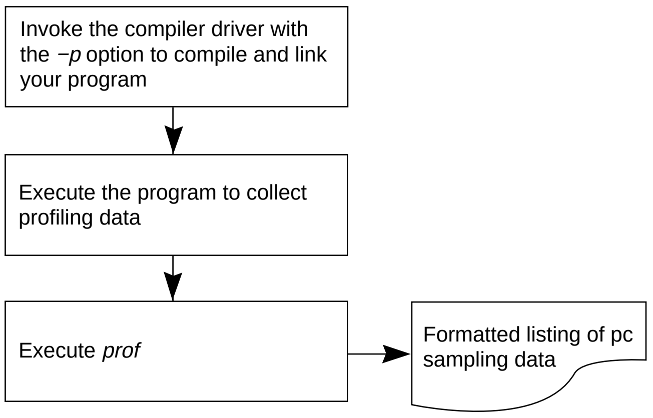
Invoke the compiler driver with the *−p* option to compile and link your program
Execute the program to collect profiling data
Execute *prof*
Formatted listing of pc sampling data

Figure 5-1 Using *prof* to Obtain pc Sampling Information

To obtain block counting information, use the *pixie* program. *pixie* reads an executable program, partitions it into basic blocks, and writes (instruments) an equivalent program containing additional code that counts the execution of each basic block. You can execute this instrumented program to obtain a file containing basic block counts.

To obtain a formatted listing of the block count information, run the *prof* program using the *–pixie* option. Figure 5-2 shows how you can obtain basic block counts for your program.

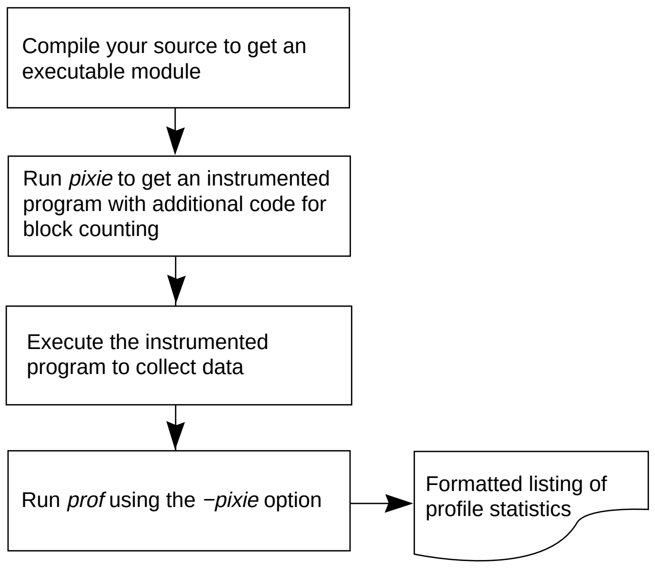
Compile your source to get an executable module
Run *pixie* to get an instrumented program with additional code for block counting
Execute the instrumented program to collect data
Run *prof* using the −*pixie* option
Formatted listing of profile statistics

Figure 5-2 Using *pixie* to Collect Basic Block Counts

par

The *par* system utility program traces system calls and scheduling activity. You can use it to trace the activity of a single process, a related group of processes, or the system as a whole. *par* prints a report showing all system calls made by the specified processes, along with arguments and return values.

cord

The *cord* program rearranges procedures in an executable object to reduce paging and achieve better instruction cache mapping. Typically, the order specified either minimizes paging or maximizes the likelihood that data items are in cache when needed. Figure 5-3 shows how you can use *cord* to rearrange the procedures in your program.

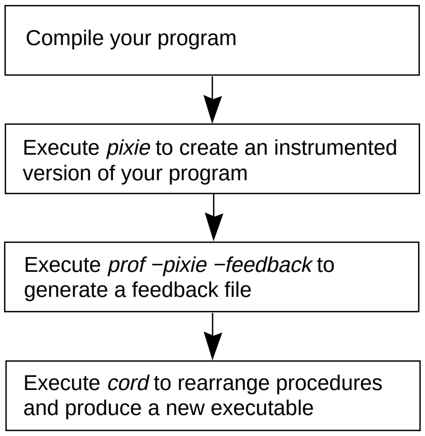
Compile your program
Execute *pixie* to create an instrumented version of your program
Execute *prof −pixie −feedback* to generate a feedback file
Execute *cord* to rearrange procedures and produce a new executable

Figure 5-3 Using *cord* to Rearrange Procedures

xscope

The *xscope* program monitors connections between an X server and a client. It prints the contents of each request, reply, error, or event that is communicated between the server and client. This information can be useful in debugging and performance tuning of X servers and clients.

The CASEVision Programming Tools

Silicon Graphics provides several interactive, graphical programming tools to aid you in debugging and tuning your application. These computer-aided software engineering (CASE) tools, which comprise the CASEVision product line, operate in a common environment that provides a consistent user interface. "The CASEVision Environment" on page 49 describes this environment.

The CASEVision tools include:

- CASEVision / WorkShop
- CASEVision / WorkShop MegaDev
- CASEVision / WorkShop Pro MPF

Note: The CASEVision / WorkShop tools are not part of IDO—you have to order them separately.

The CASEVision Environment

All tools in the CASEVision product line share a common environment. Some common facilities and features of this environment are:

- CASEVision tools use standard IRIS IM elements such as the File Browser (used to save and load files) and offer easily set X defaults.
- The CASEVision environment offers a comprehensive online help system with context-sensitive access.
- All CASEVision tools provide access to source code through a common text editor called Source View. Source View provides a window displaying lines of text in a source code file and offers simple text editing features and the ability to fork other text editors such as *vi* or *emacs*.
- Many CASEVision tools provide graphical representations of code, such as function call trees or class hierarchies. These have common features for manipulating the presentation so that you can focus on the data of specific interest or get a larger overview.

CASEVision/WorkShop

CASEVision / Workshop is a software development environment that helps you visualize your code. It's a set of tools that use an object-oriented application framework and an IRIS IM interface with user-selectable color schemes. The WorkShop toolset includes five graphical tools:

- Debugger
- Static Analyzer
- Performance Analyzer
- Tester
- Build Manager

Figure 5-4 gives you an example of what an interface to a CASEVision tool looks like. This example, output from the Performance Analyzer, shows the total time spent in several routines, and the time each routine actually executed, exclusive of the time spent in called routines.

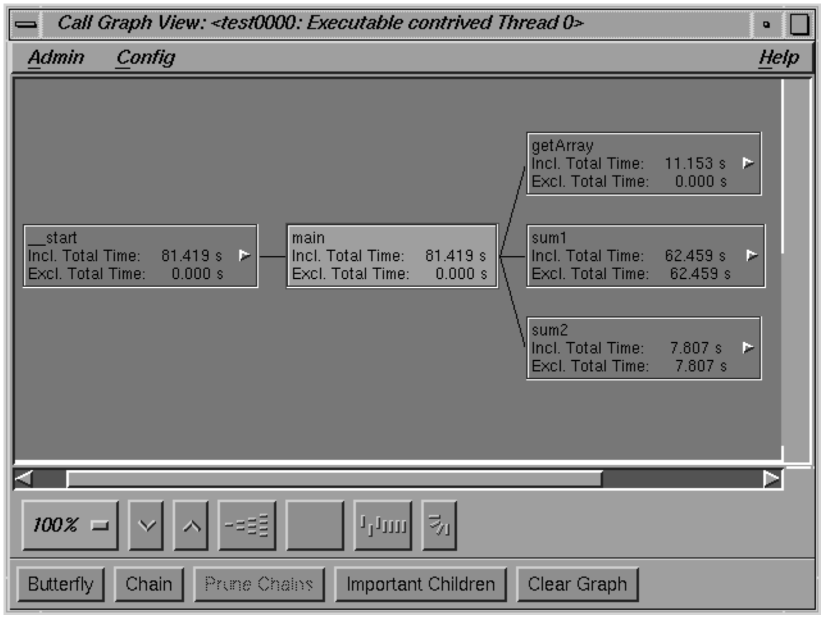
Call Graph View: <test0000: Executable contrived Thread 0>
Admin
Config
Help
getArray
Incl. Total Time: 11.153 s
Excl. Total Time: 0.000 s
__start
Incl. Total Time: 81.419 s
Excl. Total Time: 0.000 s
main
Incl. Total Time: 81.419 s
Excl. Total Time: 0.000 s
sum1
Incl. Total Time: 62.459 s
Excl. Total Time: 62.459 s
sum2
Incl. Total Time: 7.807 s
Excl. Total Time: 7.807 s
100%
Butterfly
Chain
Prune Chains
Important Children
Clear Graph

Figure 5-4 An Example of a CASEVision User Interface

The WorkShop Debugger

The WorkShop Debugger is a source-level debugging tool that provides special windows (views) for displaying program data and execution status as the program executes. The Debugger lets you set various types of traps (breakpoints) and watch points.

The WorkShop Static Analyzer

The WorkShop Static Analyzer helps you analyze source code written in C, C++, or Fortran by showing you the code's structure (graphically or in text format). It also shows you how the functions within programs call one another, where and how variables are defined, how files depend on one another, where you can find macros, and many other structural details that help you understand the code.

The Performance Analyzer

You can use the Performance Analyzer to define and run experiments that collect performance data. The Performance Analyzer uses this data to produce charts, tables, and annotated code that help you analyze the performance of your program.

WorkShop Tester

WorkShop Tester is a software quality assurance toolset for software and test engineers and their managers who are involved in the development, test, and maintenance of long-lived software projects.

WorkShop/Build Manager

You can use the WorkShop / Build Manager to compile software without leaving the WorkShop environment. You can look for problems using the WorkShop analysis tools (Static Analyzer, Debugger, and Performance Analyzer), make changes to the source, suspend your testing, and then recompile. The Build Manager has two components:

- Build View—for compiling, viewing compile error lists, and accessing the code containing the errors in Source View (the CASEVision editor) or an editor of your choice. Build View helps you find files containing compile errors so that you can quickly fix them, recompile, and resume testing.
- Build Analyzer—for viewing build dependencies and recompilation requirements and accessing source files.

Build View uses the UNIX *make* facility as its default build software. Although Build Analyzer determines dependencies using *make*, you can use the build software of your choice.

CASEVision/WorkShop MegaDev

CASEVision / WorkShop MegaDev is a suite of graphical, interactive computer-aided software engineering (CASE) tools designed especially for programmers developing and maintaining C++ libraries and applications.

It contains the C++ Browser and the Fix and Continue utilities. The C++ Browser lets you view the structure of any set of C++ classes. It provides a global, graphical view of interclass relationships such as inheritance, containment, and interaction within a set of classes. The Fix and Continue utilities allow you to redefine functions in your program and then continue execution without recompiling.

CASEVision/WorkShop Pro MPF

Fortran 77 programmers can use the WorkShop Pro MPF Parallel Analyzer View to view the structure of multiprocessing applications. This tool reads analysis files generated by the Power Fortran Accelerator (PFA) and provides a visual comparison of the original source with the parallelized code.

The function of the Parallel Analyzer View's is integrated with that of CASEVision / WorkShop to allow examination of a program's loops in conjunction with a performance experiment on either a uni- or multiprocessor execution of the program.

Version Control and Configuration Management

CASEVision provides a tool for creating tracking systems and another for software configuration management.

CASEVision/Tracker

CASEVision / Tracker is a tool for creating tracking systems for bugs and enhancement requests. It allows you to design and create the tracking database and the interface programs.

The Request Tracking System (RTS) was designed using the CASEVision / Tracker tool. RTS is a system for tracking bugs and requests for enhancements. RTS is designed to:

- meet the basic request tracking needs of most software organizations with only minor modification
- serve as a functioning starter example of a Tracker-based system for those organizations that want to create their own systems

CASEVision/ClearCase

CASEVision / ClearCase is a software configuration management system that’s specifically designed for large-scale, long-lived software projects. ClearCase simultaneously manages multiple versions of evolving software and tracks versions used in software builds. It performs builds and rebuilds of individual programs or entire releases according to user-defined specifications, and enforces project-defined policies.

ClearCase provides enhanced version control of all UNIX (or IRIX) filesystem objects, binary sharing to minimize rebuilds and unnecessary copies or links, and complete build auditing. It also supports parallel builds across the network.

Chapter 6

Application Libraries

Silicon Graphics provides several application libraries that you can use in writing your applications. These libraries provide tools for developing programs in the following areas:

- Graphics
- Image processing
- Digital media
- Printer / scanner management

Refer to Table 2-5 on page 21 for a list of the manuals you can read to learn more about the topics discussed in this chapter.

Graphics Libraries

Silicon Graphics supports four graphics libraries.

- OpenGL provides low-level graphics routines and is an interface to the graphics hardware.
- Open Inventor is a toolkit, built on OpenGL, that allows you to create interactive graphics applications.
- IRIS Graphics Library is a library of subroutines for creating 2D and 3D color graphics and animation.
- IRIS Performer is a toolkit for creating real-time graphics and visual simulation applications.

OpenGL

OpenGL is a software interface to graphics hardware. It consists of about 120 commands that you can use to specify the objects and operations needed to create interactive programs that produce color images of moving 3D objects. OpenGL is an industry standard for 2D and 3D graphics rendering and is a part of the IDO.

OpenGL uses a client-server model for interpretation of commands. An application using OpenGL can run under IRIX on any Silicon Graphics platform and be rendered on a platform with another operating system and window system, provided the implementation of OpenGL on each platform conforms to the standard.

OpenGL doesn’t include commands for performing windowing tasks or obtaining user input. For that you must work through the windowing system that controls your hardware. Since the OpenGL application programming interface (API) is independent of hardware platforms, window systems, and operating systems, porting among conforming implementations of OpenGL is an easy task.

OpenGL allows you to build the models you need from a small set of geometric primitives—points, lines, and polygons. It doesn’t provide high-level commands for describing models of 3D objects—Silicon Graphics provides higher level graphics libraries for these tasks. One of these libraries is Open Inventor, which is built on OpenGL and uses OpenGL calls for rendering. You can write your application using the OpenGL API, although it’s often easier to use one of the higher level libraries. Using the higher level graphics libraries makes rendering operations transparent to your application, which allows you to concentrate on your application rather than on the time-consuming details of rendering.

Note: IRIS Performer and the ImageVision Library are being converted from IRIS GL (the precursor to OpenGL) to the OpenGL standard.

The OpenGL library contains functions for

- rendering primitives (points, lines, polygons)
- controlling colors and lighting

- using texture mapping to add surface characteristics to geometry
- setting and controlling transformations

Figure 6-1 shows an object created by OpenGL—a sphere illuminated by a light source.

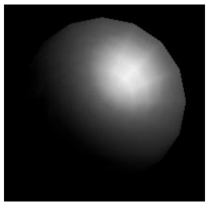

Figure 6-1 Using the Lighting Feature of OpenGL

Figure 6-2 shows another object created by OpenGL—a texture-mapped Bezier surface mesh.

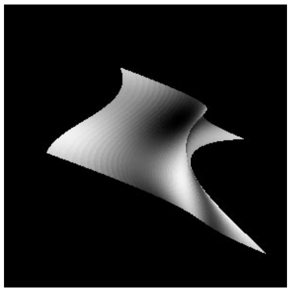

Figure 6-2 Using the Texture-Mapping Feature of OpenGL

OpenGL provides a small but powerful set of rendering commands, and all higher-level drawing must be done using these commands. To simplify your programming tasks, OpenGL provides a Utility Library (GLU) that includes routines that encapsulate OpenGL commands. These GLU routines perform tasks such as:

- drawing common objects such as spheres, cylinders, and disks
- manipulating images used in texturing

- handling simple non-convex polygons
- setting up matrices for a variety of viewing orientations and projections

Open Inventor

Open Inventor is an object-oriented toolkit that provides objects and methods for creating interactive 3D graphics applications. This toolkit contains 3D objects you can use to represent your 3D physical models, as well as objects that allow you to interactively operate on these models.

Figure 6-3 shows a single scene from a racing game created using Open Inventor. In this game, mouse buttons control the speed and position of a car as it moves along a track. Open Inventor creates scenes showing the car, the track, and the terrain that appears as the car moves along the track.

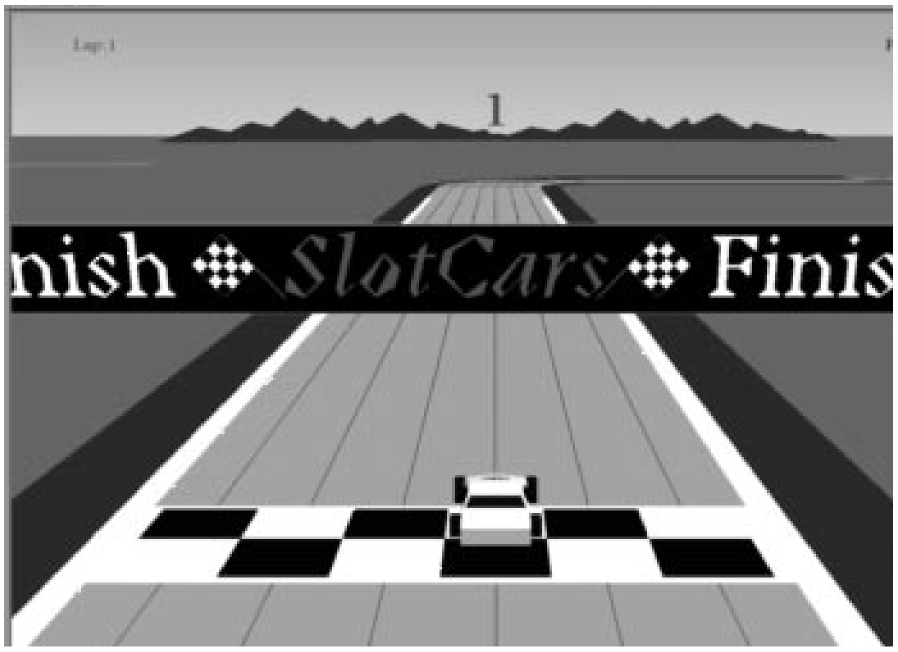

Figure 6-3 A Scene Created by Open Inventor

Open Inventor is written in C++ but also includes C bindings. It is object-oriented and extensible. The Inventor toolkit is based on OpenGL and provides a library of objects you can use, modify, and extend to meet your needs.

Figure 6-4 illustrates the architecture of an Open Inventor application. The Open Inventor components shown in the figure are described in the paragraphs following the figure.

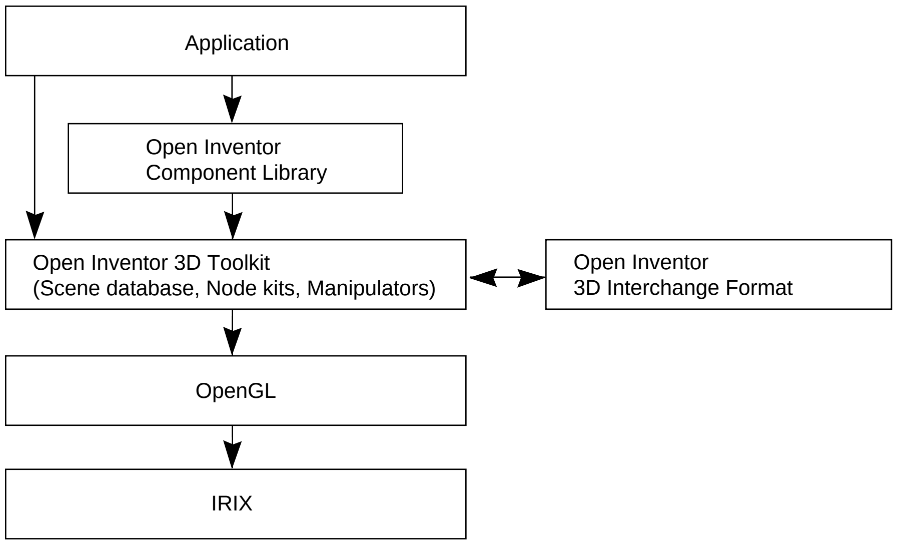
Application
Open Inventor Component Library
Open Inventor 3D Toolkit (Scene database, Node kits, Manipulators)
Open Inventor 3D Interchange Format
OpenGL
IRIX

Figure 6-4 Open Inventor Architecture

Open Inventor Toolkit

The Open Inventor Toolkit provides three programming tools that you can use in your Open Inventor application.

Scene database	A scene database is a collection of 3D objects and properties arranged to represent a 3D scene. A scene is composed of nodes that define all information about an object—its shape, size, coloring, surface texture, and location in 3D space. You can use this information to render the object or to vary it in a variety of ways—for example, to move the object or change the way it looks. Some objects, called engines, are used to animate part of a scene.

Manipulators	A manipulator is a special kind of node that reacts to user events. Manipulator objects allow users to interact with 3D objects on a screen. Manipulators allow rendering into a scene and provide a means for translating user-initiated events into changes to the scene database.
Node kits	A node kit is a collection of nodes grouped together to provide a simplified model. Open Inventor provides these ready-made kits to make building a structured scene database easier. The kit provides the basic structure of an object, but allows you to define information specific to your object. For example, the shape node kit describes the shape but allows you to define a geometric specification, material, a lighting model, texture, and other properties of the shape.

Component Library

The Component Library is a convenience library for programmers who use X Window System and X-based toolkits such as Xt and Motif. It contains an event translator that converts X events into Open Inventor events.

3D Interchange File Format

Open Inventor includes an interchange file format for exchanging 3D objects and scenes between applications. Objects in the scene database can be written to a file during the execution of your program, in either ASCII or binary form.

IRIS Graphics Library

The IRIS Graphics Library (IRIS GL) is a library of subroutines for creating 2D and 3D color graphics and animation.

Here are some of the things IRIS GL allows you to do:

- draw graphics primitives such as points, lines, polygons
- draw characters and define fonts
- use color modes and color maps to control the way colors are displayed
- use double buffering to create animated graphics

- perform coordinate transformations
- define and manipulate light sources to create lighted scenes
- use texture mapping to add surface characteristics to geometry

IRIS GL is a predecessor of OpenGL, the industry standard for graphics applications. Silicon Graphics' application libraries that were originally built on IRIS GL are moving to the OpenGL standard. Figure 6-5 shows relationship of an application to IRIX GL and to the ImageVision Library and IRIS Performer, the libraries currently built on IRIS GL.

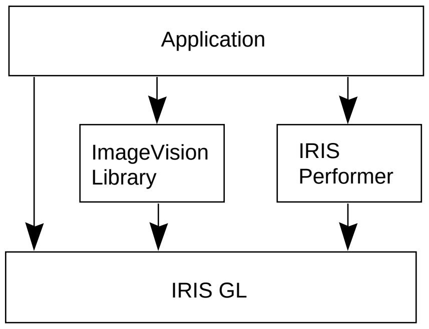

Figure 6-5 IRIS GL in the Developer Environment

Note: You should never reference both IRIS GL and OpenGL in a single application. This means you should not use a higher-level library based on OpenGL (for example, Open Inventor) in the same application in which you use a library based on IRIS GL (for example, the ImageVision Library).

IRIS Performer

IRIS Performer is a software development environment layered above the IRIS Graphics Library (IRIS GL). It provides high-level support for visual simulation, interactive entertainment, virtual reality, and graphics-intensive tasks. Applications that require real-time visuals and high-performance rendering benefit from using IRIS Performer.

The main components of IRIS Performer are the two libraries *libpr* and *libpf*.

- *libpr* is a low-level library that provides optimized rendering functions, state control, and other functions that are fundamental to real-time graphics. It provides highly optimized rendering loops for rendering a wide variety of geometric primitives.
- *libpf* is a visual simulation development environment that layers a multiprocessing database traversal and rendering system on *libpr*. It supports multiprocessing, hierarchical scene construction, multiple channels, culling to each channel's field-of-view, and frame-rate control.

Figure 6-6 shows the relationship between the IRIS Performer libraries and the IRIX system software.

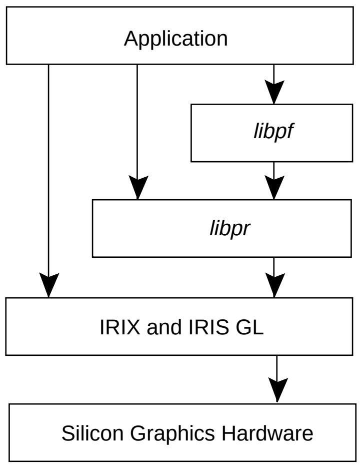

Figure 6-6 IRIS Performer Library Hierarchy

You can choose the IRIS Performer libraries that best suit your needs. You may want to build your own toolkits on top of *libpr*, the low-level, high-performance library, or you may choose to take advantage of the visual simulation environment that *libpf* provides. Note that functions from *libpf* make calls to *libpr* functions so you don't necessarily have to use the *libpr* functions directly.

IRIS Performer doesn't define a file format; it imports files from many standard database formats at run time. Some of the database formats supported by IRIS Performer are shown in Table 6-1.

Table 6-1 Database Formats Supported by Performer

Format Name	**Description**
BIN	Silicon Graphics format
DWB	Designer's Workbench format
DXF	AutoCAD® format
FLT	MultiGen™ FLIGHT format
IV	Inventor format
OBJ	Wavefront Technologies *Model* format

ImageVision Library

Unlike the graphics libraries, which build a display from a series of geometric objects, image processing applications start with an image consisting of pixel information stored in a file. These images can originate as a photographic image that's scanned or obtained from a Kodak™ CD, data obtained from medical imaging equipment, satellite data, or numerous other sources. An image processing application can manipulate this image in ways that are meaningful to the user of the application.

The ImageVision Library (IL) is a set of tools designed for developers of image processing applications. The IL is written in C++ but has interfaces for C and Fortran. You can use this library to import, manipulate, display, and store images.

The IL is an object-oriented toolkit whose modularity provides an easy and efficient means to create and maintain programs that use it for image manipulation and display. This modular structure also makes it easy to extend the IL—for example, to augment the image operators supplied by the IL or to design new ones.

The ImageVision Library contains objects and methods that allow an image processing application to:

- Import images created in a variety of different file types. The supported file formats include TIFF, GIF, Kodak Photo CD™, SGI, and FIT.
- Process images using any sequence of the image processing operators supported by the IL. Image processing functions include color conversion, arithmetic operations on pixel data, radiometric and geometric transformations, generation of statistical data for an image, spatial and non-spatial domain transformations, and edge, line, and spot detection.
- Display one or more images in an X Window. The IL provides many ways to control image displays, including stacking images or aligning them side by side, roaming a large image, or doing a wipe to move one edge of an image to reveal what is stacked beneath it.
- Store processed images on disk.

Figure 6-7 and Figure 6-8 illustrate an ImageVision Library operation. Figure 6-7 shows an image imported from a Kodak Photo CD.

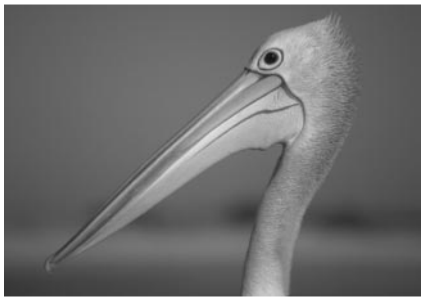

Figure 6-7 A Display Created by ImageVision Library

Figure 6-8 shows the image created when the ImageVision library performs a geometric transformation called a warp on the image in Figure 6-7.

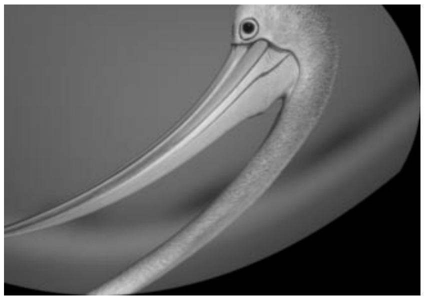

Figure 6-8 Using the ImageVision Library to Transform an Image

Figure 6-9 illustrates a simple ImageVision application that reads an image from disk, applies a rotation transformation, displays both images on a monitor, and writes the rotated image to disk.

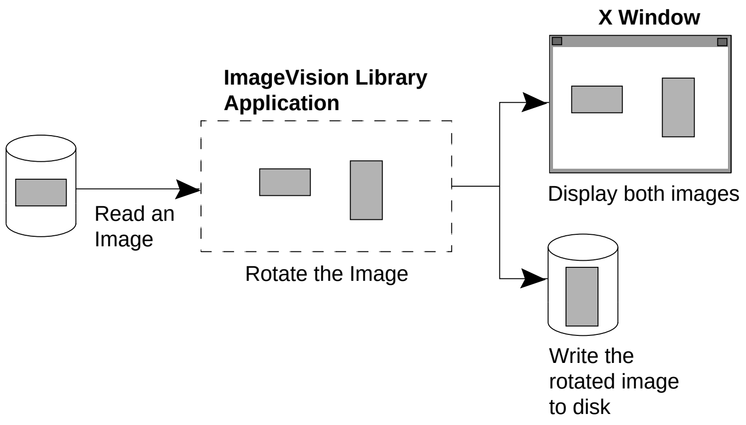
X Window
ImageVision Library Application
Read an Image
Rotate the Image
Display both images
Write the rotated image to disk

Figure 6-9 A Simple ImageVision Library Application

The ImageVision Library toolkit provides an application program interface (API) that is common across all Silicon Graphics workstations. The IL uses Xlib for window management and allows you to use either IRIS GL or X to render into an X window.

Figure 6-10 shows the architecture of an IL application.

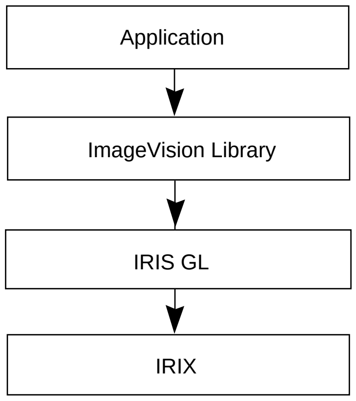

Figure 6-10 Architecture of an ImageVision Library Application

The IL implements an execution model that optimizes memory usage and performance as image data is processed. This execution model

- supports the parallel processing features of Silicon Graphics workstations
- caches image data to minimize file access
- is demand-driven so that only image data needed for output is processed
- chains operations together, which saves time because intermediate results don't have to be stored
- uses hardware acceleration of graphics operations whenever possible to improve performance of IL operations

IRIS Digital Media Development Environment

The IRIS Digital Media Development Environment provides digital media libraries and tools for developers of media applications.

IRIS Digital Media Libraries

The IRIS Digital Media Libraries provide programming support for digital media development on Silicon Graphics platforms. These libraries are included with the IDO. The term *digital media* describes digitally sampled audio and video (including still image) data, MIDI event streams, and other associated information such as time codes. Sampled audio/video data can be digitally encoded in a variety of uncompressed and compressed formats

The IRIS Digital Media Development Environment provides programming support for digital audio, digital video, and MIDI applications. The libraries provide programming interfaces to:

- audio, video, and MIDI I/O subsystems
- data format conversion (including compression)
- digital media file importation/exportation
- high-level playback functions

The five libraries that comprise the IRIS Digital Media Development Environment are briefly described in the following paragraphs.

Digital Audio Libraries

The digital audio libraries provide a device-independent programming interface to the digital audio I/O subsystems built into Silicon Graphics workstations. You can use the digital audio libraries individually or in combination. Table 6-2 describes the libraries contained in the digital audio library set.

Table 6-2 Digital Audio Libraries

Library	Library Function
Audio	Provides an interface for configuring the audio system, managing audio I/O between the application program and audio hardware, specifying attributes of digital audio data, and facilitating real-time programming.
Audio File	Provides an interface for reading and writing the standard digital audio file formats AIFF and AIFF-C standards.
CD Audio	Provides an interface for optional Silicon Graphics SCSI CD-ROM drives. This interface features a special mode that allows it to read audio CD format as well as CD-ROM format.
DAT Audio	Provides an interface for optional Silicon Graphics SCSI DAT drives.

Figure 6-11 diagrams the interaction between an audio application and the audio libraries, the device drivers, the IRIX filesystem, the audio hardware, and the optional SCSI devices.

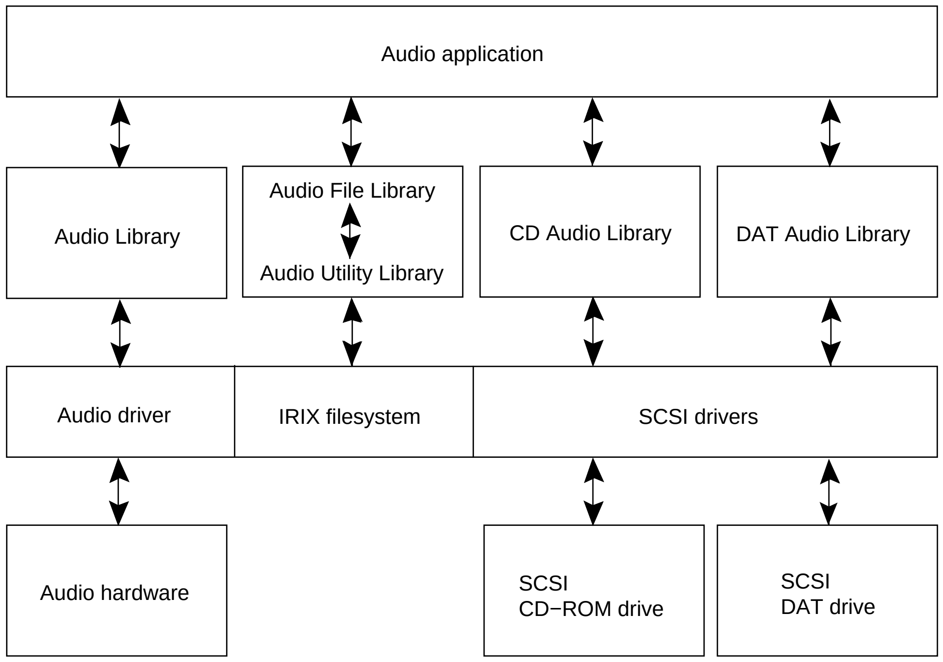

Figure 6-11 Interaction of Digital Audio System Components

MIDI library

The Musical Instrument Digital Interface (MIDI) Library provides a programming interface for timestamped MIDI input/output via serial ports.

Video Library

The Video Library provides both device-independent and device-dependent interfaces to the on-board Indy VINO, and to video options such as Indy Video™, Galileo Video™, Indigo^2 Video™, and Sirius Video™.

Compression Library

The Compression Library (CL) provides an algorithm-independent, extensible interface for compressing and decompressing animation, video, audio, and image data. The CL interface supports the Cosmo Compress JPEG codec (available for Indigo R4000, Indy, Indigo 2) in addition to several software codecs, including software JPEG. Cosmo Compress connects to a Galileo-family video device to allow realtime JPEG video capture and playback. In this configuration, Cosmo operates as a component of the video I/O system.

Movie Library

The Movie Library is a collection of routines that provides a C language API for creating, reading, writing, editing, and playing movie files. Supported file formats include Silicon Graphics Movie File (SGIMF) format and the Apple® QuickTime™ movie file format.

Digital Media Tools

IRIX includes several interactive media tools that are built on the IRIS Digital Media Development Environment and make it easy to perform basic media functions. These tools have command-line interfaces that allow you to incorporate them into your digital media application. Table 6-3 lists the digital media tools.

Table 6-3 Digital Media Tools

Tool	Tool Function
Capture	Record audio, video input, audio, and still images on your system disk and import them into multimedia applications.
Movie Maker	Combine audio, video, and still images to create a movie you can play on your workstation.
Movie Player	Play movies on your workstation using controls that allow you to play, stop, reverse, and fast forward.
Sound Editor	Record sound into and edit audio files.
Sound Filer	Play audio files on your system and convert audio files to different formats and sizes.

Table 6-3 Digital Media Tools

Tool	Tool Function
CD Manager	Play and record from a compact disc drive attached to your workstation SCSI port.
DAT Manager	Play and record to and from DAT tapes on a DAT drive attached to your workstation SCSI port

Read the *Media Tools User's Guide*, available in IRIS Insight, or the reference page for each tool to learn more about these media tools.

Printer/Scanner Management

Silicon Graphics provides a printing and scanning environment for IRIS workstations. This environment, Impressario, has features for a wide range of IRIX audiences: printer and scanner driver developers, application program developers, and end users. Impressario allows files of different types to be printed on a wide variety of printers and allows images to be scanned from a scanning device, an IRIS screen, or a Silicon Graphics image file.

Impressario provides the following libraries for application developers:

libspool	Provides an API to the IRIX printer spooling system and functions, allowing you to submit print jobs, query their status, and so on.
libprintui	Provides a graphical interface for printing that's compatible with IRIS IM.
libpod	Provides a network-transparent interface to the Printer Object Database (POD). Each printer has a POD that contains configuration, status, and other information about that printer.
libscan	Provides an interface to the Impressario scanning system.
libstiff	Allows you to read and write Stream TIFF (STIFF) files.
libimp	Allows you to read and write Silicon Graphics image files in RGB format.

Impressario is built on top of the System V print spooling system. It provides model files, filters, and drivers to convert ISO text files, SGI images files, PostScript® documents, and a wide variety of other file formats to a format suitable for both raster and PostsScript printers. Figure 6-12 shows the relationship between an application program, the Impressario libraries, and the spooling system in IRIX.

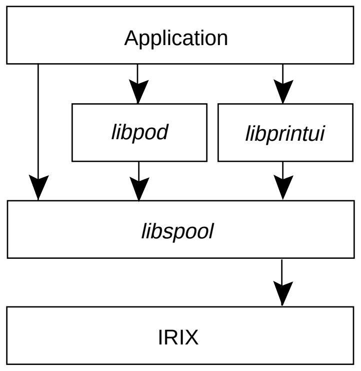

Figure 6-12 Interface to the Spooling System

Refer to the *Impressario Programming Guide* for information about writing a driver for a printer or scanner device for which no driver is available.

Index

Tell Us About This Manual

As a user of Silicon Graphics products, you can help us to better understand your needs and to improve the quality of our documentation.

Any information that you provide will be useful. Here is a list of suggested topics:

- General impression of the document
- Omission of material that you expected to find
- Technical errors
- Relevance of the material to the job you had to do
- Quality of the printing and binding

Please send the title and part number of the document with your comments. The part number for this document is 007-2476-001.

Thank you!

Three Ways to Reach Us

- To send your comments by **electronic mail**, use either of these addresses:
  - On the Internet: techpubs@sgi.com
  - For UUCP mail (through any backbone site): *[your_site]*!sgi!techpubs
- To **fax** your comments (or annotated copies of manual pages), use this fax number: 650-932-0801
- To send your comments by **traditional mail**, use this address:

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