White Paper

# OpenGL Performer™
# Real-Time 3D Rendering for High-Performance and Interactive Graphics Applications

## Table of Contents

## 1.0 Introduction

OpenGL Performer is a powerful and comprehensive programming interface for developers of real-time performance-oriented 3D graphics applications. OpenGL Performer dramatically simplifies development of complex applications made for visual simulation, virtual reality, simulation-based design, interactive entertainment, broadcast video, manufacturing, scientific visualization, architectural walk-through, and computer-aided design.

OpenGL Performer provides the advanced features and innovative techniques that enable you to achieve peak performance and make optimal use of system capabilities and sophisticated 3D graphics features. It gives you the capability to scale easily to multiple processors and multiple graphics pipelines, deploy to a wide range of systems and price points, and be ready-made for the graphics systems of today and the future.

## 2.0 Toolkit Overview

The basis of OpenGL Performer is the performance-rendering library libpr, a low-level, object-oriented library providing high-speed rendering functions, efficient graphics state control, and other application-neutral 3D graphics functions.

Layered above libpr is libpf, a real-time visual simulation environment providing a high-performance scene graph and extensible multiprocess rendering system designed to take best advantage of SGI® scalable graphics systems.

The modular libpfv library provides a high-level infrastructure for rapid construction and extension of OpenGL Performer applications. It is based on the pfvViewer interface and ships with an extensible collection of ready-to-use modules for display control, model manipulation, navigation, picking, and customization.

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

The database library libpfdb contains file importers for many popular industry-standard database formats including Alias® Maya®, Dassault CATIA®, OpenFlight™, Open Inventor™ .iv, OpenGL Optimizer™ .csb, and more than 70 others. A run-time look-up mechanism is used to invoke loaders for requested files that, on loading, can be combined into a single database. Additionally, OpenGL Performer provides the fast-loading PFB file format for use in run-time database paging. These loaders also serve as a guide to developers creating new database importers.

The libpfui library contains user interface building blocks for creating manipulators and user-interface components common to many interactive applications.

Completing the suite of libraries is libpfutil, the OpenGL Performer utility library. It provides a collection of convenience routines implementing tasks such as multiprocessing configuration, multichannel support, texture management, graphical user interface tools, input event collection and management, and scene graph traversal functions.

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

**3.0 Write Once, Deploy Anywhere: The Perfect Fit for Your Software Development Workflow**

OpenGL Performer supports the IRIX® operating system and both 32-bit and 64-bit versions of the Linux® operating system, as well as Microsoft® Windows® 2000 and Windows® XP. The API is built atop the industry-standard OpenGL® graphics library, includes both ANSI C and C++ bindings, and is completely source-code compatible across all platforms.

**4.0 Rendering Architecture: App, Cull, and Draw**

OpenGL Performer provides a pipelined rendering architecture designed to detect and take full advantage of the capabilities of the underlying system. At its core lie high-performance, multithreaded, parallel rendering stages for per-frame scene management and image generation output to one or more graphics pipelines. The software architecture is split into three major stages to handle critical path operations:

- APP: Simulation processing, which includes reading input from control devices, simulating the dynamics of moving models, updating the visual database, and interacting with other libraries or simulation stations.

- CULL: Traverses the visual database and determines which portions of it are potentially visible (a procedure known as culling), selects a level of detail (LOD) for each model, sorts objects and optimizes state management, and generates a display list of objects to be rendered.

- DRAW: Traverses the display list and issues graphics library commands to a geometry pipeline in order to create an image for subsequent display. The user has full control over the configuration of the App, Cull, and Draw tasks, including the ability to combine multiple tasks into a single process or divide them among multiple processes and processors on the system. OpenGL Performer can also automatically make process configuration decisions at run time based on the hardware platform.

In addition to the main App/Cull/Draw pipeline, several asynchronous processes are available for user customization and optional tasks such as:

- DBASE: database paging, for asynchronously loading files and adding data to or deleting data from the scene graph

- ISECT: intersection testing, to intersect line segments with the database for operations such as collision detection and line-of-sight determination

- COMPUTE: general asynchronous computations, such as those used for dynamic geometry evaluation and morphing

- INPUT: input handling

- CULL_SIDEKICK: advanced culling modes, for per-polygon cull operations and occlusion culling tests

The multiprocess pipeline of OpenGL Performer is largely transparent to the user because the toolkit manages the difficult multiprocessing issues for you, such as interprocess communication, process timing, synchronization, data exclusion, coherence, and real-time control.

### 4.1 Running Real Time

In situations where a guaranteed fixed frame rate is required, OpenGL Performer uses extensions to the operating system to control process scheduling and process priority management, as well as real-time system profiling. This guarantees real-time predictable behavior from the IRIX operating system by restricting and isolating processes to specific processors and maintaining nondegrading priorities.

### 4.2 Performance Monitoring

OpenGL Performer provides a full suite of diagnostic statistics, including graphics pipeline hardware statistics for extremely accurate measurements of rendering time. These statistics are used for tuning and real-time monitoring of full system performance for load management and for direct use with other system monitoring tools.

OpenGL Performer also includes an event analysis tool called EventView, which traces events generated internally by the toolkit as well as user-generated time events on a logic-analyzer-style display. EventView is particularly useful for measuring how the duration of various OpenGL Performer execution blocks varies across time and as a function of user application events.

### 5.0 The Scene-to-Screen Path: Scene Graph, Channels, Pipes, Windows

The OpenGL Performer scene graph holds the data that defines your scene or virtual world. The scene graph includes low-level descriptions of object geometry and their appearance, as well as higher-level, spatial information, such as transformations, animations, levels of detail, environmental elements, and special effects, as well as additional application-specific data. OpenGL Performer and your application act on the scene graph to perform rendering, culling, paging, intersection, and other functions.

A channel is equivalent to a camera moving about the scene. Whereas the scene graph encapsulates all of the visual data in the scene, the channel sees only the visual information that is visible to the viewer; the channel shows a slice of the scene from a specified perspective. Each channel is associated with a single viewport in the final display configuration.

The pfPipe is the heart of all processing done by OpenGL Performer. It performs the per-frame App/Cull/Draw stages, thereby rendering each channel to the windows on the display. Under the hood, the pfPipe is a high-performance abstraction of the graphics pipeline, designed to manage performance, realism, image quality, and the sense of immersion for the end user; while ensuring that the system hardware and operating system capabilities are fully and efficiently utilized.

The pfPipeWindow is the mechanism by which a pfPipe manages the windows to which it is to render, the size of the render area, and the configuration of the frame buffer. OpenGL Performer uses this information for proper viewport and frustum management and for any features affected by frame buffer configuration, such as anti-aliasing, transparency for fade LOD, layers for decal geometry, and so on.

### 6.0 Multichannel and Multipipe Flexibility at Peak Performance

Multiple channels can be arranged side-by-side with multiple offset views for fully synchronized panoramic and tiled displays with optional overlap for insets and edge blending. Performance can be scaled even further with the use of multiple synchronized InfiniteReality®, InfinitePerformance™, UltimateVision™, or Silicon Graphics Prism™ graphics pipelines. OpenGL Performer maximizes multipipe throughput by assigning a dedicated Cull/Draw pair for each hardware pipeline and automatically manages stress for each channel.

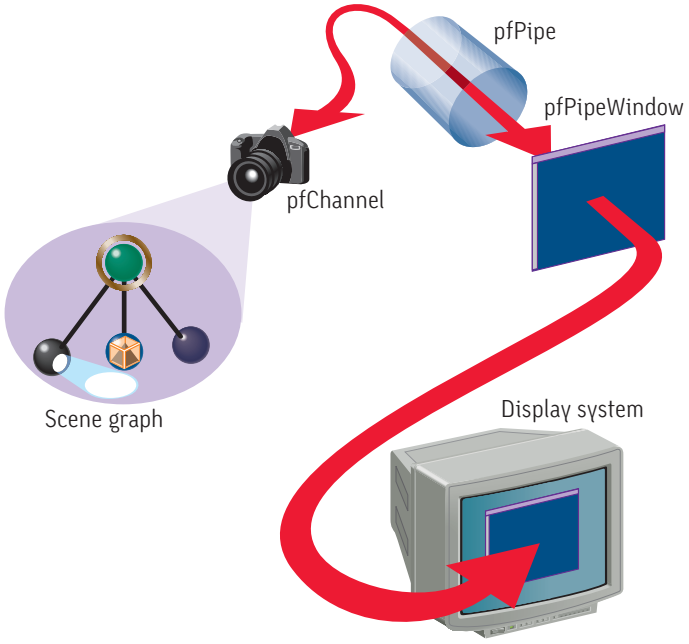


Fig. 1. The scene-to-screen path

Fig. 2. Image-based rendering of a human figure

### 6.1 Increasing Frame Rate by Using DPLEX (IRIX Operating System Only)

On InfiniteReality systems with multiple rendering pipes and a Digital Video Multiplexer Option (DPLEX), OpenGL Performer provides support for time-multiplexing the output of the different pipes into a single screen. For example, a five-pipe system that could render a complex model at 12 Hz can render the same model at 60 Hz with DLPEX. Each one of the five pipes starts drawing its frame at a different time, and the resulting images are multiplexed into the output screen. The result is that the output screen sees a new image 60 times per second even though each one of the pipes can produce only 12 new images per second.

### 6.2 Compositor Control for Scalable Performance

Using the Scalable Graphics Compositor option on Silicon Graphics Prism™, Onyx4™ UltimateVision™, and Onyx InfinitePerformance systems, the output from multiple graphics pipelines can be directed into a single composite output. Each pipe renders only a portion of the overall display, thereby directly scaling performance. The composition can be tiled spatially on the final display or blended for anti-aliasing and other effects. OpenGL Performer also performs cross-pipe load balancing on a frame-by-frame basis by adjusting the relative size and position of all pipes using the compositor.

## 7.0 Increasing Visual Fidelity with Image-Based Rendering

One of the major challenges in the creation of visually realistic scenes is the rendering of organic forms such as trees and people. A single tree, for example, can require thousands of polygons, and a forest can overload the real-time rendering capabilities of even the fastest graphics system. Traditional visual simulation systems provide a partial solution with billboard objects or by neglecting the use of rich scene elements altogether.

OpenGL Performer provides a technique called image-based rendering (IBR) to dramatically increase the photographic realism of a scene while simultaneously freeing polygonal rendering power to increase scene complexity. Instead of creating an image from geometric primitives or rendering a billboard that always shows the same texture, IBR combines a series of images of the desired form into a seamless depiction of the object that can be viewed from any angle or from any distance.

Image-based rendering can also be applied to the texture maps of arbitrary geometric figures, enabling a simple object with a low polygon count to appear to be an extremely complex model with features that correlate with the view angle. OpenGL Performer takes this idea even further with the inclusion of a simplification utility that preprocesses complex models into a set of IBR textures and an IBR proxy containing only a fraction of the polygon count of the original.

Fig. 3. Examples of programmable shading

### 8.0 Enhanced Realism through GPU Programming

The ability to reprogram sections of the graphics pipeline gives users far greater flexibility in producing dramatically compelling scene content. For example, shaders can be created to resemble real-world materials ranging from highly realistic metals, wood, plastics, skin, leather, and cloth to water, glass, dirt, dust, and smoke (see Figure 3). OpenGL Performer takes much of the complexity out of GPU programming by encapsulating the low-level number crunching and enabling users to work at a higher level with simpler-to-use interfaces.

OpenGL Performer provides direct support for vertex programs, which enable the customization of geometry transforms, lighting, and other texture coordinate-generation functions. It also supports a fragment programs, which enable users to replace texturing and atmospheric effects functions in the graphics pipeline with a user-defined program that performs special or unusual tasks. These can include advanced pixel shaders that render fog effects in a specific way. Additionally, OpenGL Performer provides two new classes, pfShaderProgram and pfShaderObject, which encapsulate the functionality associated with the vertex and fragment programs used by the GL Shading Language from OpenGL 2.0.

Another high-level rendering feature supported by OpenGL Performer GPU programming is the ability to emulate clip texture functionality. This feature enables users to implement clip-texturing [see Section 13.0] on hardware systems that do not provide native support for it. Developers can also use GPU programmability to process subdivision surfaces within the graphics pipeline and render them directly to the window.


Fig. 4. Example of clip texture emulation

### 9.0 Curves and Surfaces

OpenGL Performer incorporates the parametric curves and surfaces features of OpenGL Optimizer, widely used in the manufacturing industry. They can be used to describe a variety of 2D and 3D curves and surfaces, including plane, sphere, cone, torus, or NURBS surfaces. In all cases users are able to apply crack-free surface tessellation.

OpenGL Performer supports loop subdivision and Catmull-Clark subdivision algorithms to subdivide coarser surface structures into finer, more smoothly curved surfaces. Loop subdivision can be programmed into the GPU, which enables users to send coarse data to the graphics pipe for very fast rendering into smooth surfaces.

### 10.0 Culling in OpenGL Performer: Powerful, Flexible, and Programmable

The culling operations done by OpenGL Performer process the scene graph to construct a list of visible objects, which is then used to render the scene. By eliminating objects that are occluded or outside the user's field of view, the load on the underlying graphics hardware is greatly reduced, thereby enabling all of the rendering power of the system to be directed to those objects that do contribute to the final display. OpenGL Performer also provides the facility to use spare CPU resources to perform extra culling operations such as per-polygon culling, backface removal on the host, and occlusion culling.

The OpenGL Performer culling operations are fully programmable. Many custom rendering tasks and advanced multipass special effects, for example, require certain elements of the scene to be reprocessed several times per frame with different appearances, environmental effects, and rendering modes. The user can use the built-in flexibility of programmable culling to facilitate all of these operations.

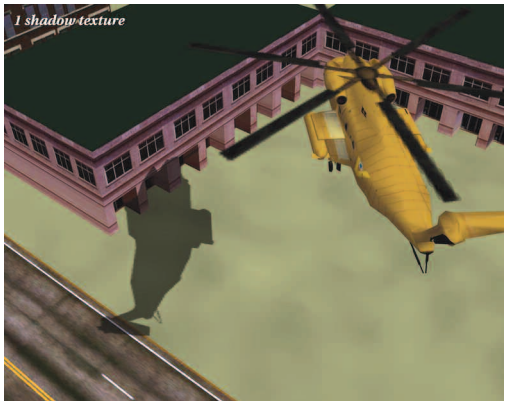

Fig. 5. Example of a real-time shadow

Small feature culling can automatically remove scene objects below a user-defined visibility threshold so that these objects are not sent to the graphics hardware for processing. This feature is particularly useful for models composed from many smaller objects, such as automobiles.

### 11.0 Real-Time Shadows

OpenGL Performer provides the facility to visualize the projection of an object's true shadow on any and all other objects in the scene, in real time. Each shadow is generated by projecting the objects as seen from one or more light sources and performing Image-Based Rendering to visualize the result. The user can specify any number of objects that cast shadows and can even perform image-processing operations before rendering to create soft shadow edges.

### 12.0 Frame and Load Management: LOD and DLB

OpenGL Performer provides two mechanisms for automated real-time load management. The first mechanism, level-of-detail (LOD) control, adjusts object complexity in accordance with scene quality and performance considerations set by the database or user. This mechanism allows objects with low contribution to scene quality (far from the eye point, small in scene, low in priority, or based on custom parameters) to be rendered at a lower level of complexity—thus reducing polygonal and graphics state loads. The second mechanism is targeted at pixel-fill or raster-load management. To manage the raster load, each display channel can be rendered with fewer pixels (determined on a per-frame basis based on per-channel load) using dynamic video resize (DVR) on the InfiniteReality series of graphics subsystems, or dynamic load balancing (DLB) on Onyx4 UltimateVision and Silicon Graphics Prism graphics systems using the SGI Scalable Graphics Compositor. The resulting image is displayed at full output resolution through bilinear interpolation, without added latency or loss of performance.

### 13.0 Texturing with Large Imagery: Virtual Clip-Texturing

Traditionally, large geographic areas were textured with separate tiled textures. This required significant modeling effort, complex application management of the texture paging, and a substantial amount of texture memory. Supported by OpenGL Performer on the InfiniteReality and Prism series of graphics subsystems, clip-texturing (clipmapping) is a superior alternative because it virtualizes the texture and allows the entire texture to be specified in a single coordinate system. Only a small fixed amount of these virtualized textures, called clipmaps, need to be kept in hardware texture memory. OpenGL Performer provides functionality that can map texture coordinates from the original virtual

space into this “clipped” texture space. This allows texture and geometry of large textures to be defined more independently than is possible with texture paging. With clipmapping, large-area geospecific imagery, such as satellite and aerial photographs, can be easily mapped onto terrain geometry with minimal database-creation effort. This clipped part of the texture is actually a subset of the clipmapping pyramid usually associated with MIPmapping and is centered at a point of interest in the virtual texture.

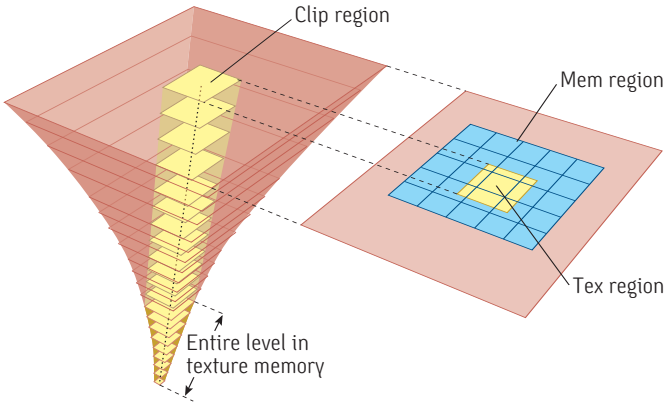


Fig. 6. Clipmapping pyramid

The size of the clipped area needs to be only as big as the number of high-resolution texels that can fit on the screen at one time and is completely decoupled from the size of the virtual texture. For a virtual texture of size 8 million x 8 million texels, typically less than .0000003% (under 45MB) of the actual potential full virtual texture is kept in hardware texture memory. The user can choose a smaller clipped space and thus use even less hardware texture memory. The virtual texture space can also be very sparsely populated with high-resolution insets. Lower-resolution versions of the image data will automatically be used where high-resolution data is unavailable.

OpenGL Performer manages the virtualization of clip-mapped textures, the update of the center of interest based on viewer position, and the automatic paging of texture data to keep the clipped space up-to-date. A two-level look-ahead caching scheme is employed in order to minimize disk-paging latency and improve download bandwidth into texture. Load management controls are provided to control the texture and paging resources. Utilities are provided to convert image data to clipmapped texture files for optimal texture paging speed.

### 14.0 Rendering Large Geometric Surfaces: Active Surface Definition

The rendering of very large or heavily tessellated surfaces presents many image-quality and load-management challenges. OpenGL Performer solves these problems using an approach called active surface definition (ASD). ASD provides an efficient, multiprocessed framework for the evaluation and paging of geometry over precomputed levels of detail based on user-specified evaluation, quality, and load-management constraints. Transitions between different levels of detail are made smoothly, on a per-triangle basis, eliminating spatial and temporal artifacts.

### 15.0 Dynamic Data Buffers: Engine and Flux

OpenGL Performer includes several features for the representation and evaluation of dynamic data. Engines allow the description of operations, such as morphing, blending, and bounding box computation, to be performed on specific objects or buffers of data. Fluxes are dynamic evaluated objects, the contents of which can be computed by engines and used as geometry or transformations any place where fluxed data is allowed. Asynchronously generated data is rendered when available in a frame-accurate manner.

### 16.0 Double-Precision Coordinate Systems

The use of standard single-precision floating-point numbers provides the best performance but sometimes does not provide enough precision to represent position information for objects at an extreme distance from the origin of the database. This is a problem when rendering very large scenes (e.g., terrain of the whole earth). Current OpenGL hardware does not support double-precision values for vertex coordinates and matrices; therefore, the solution to the precision problem must come from a higher-level layer.

OpenGL Performer provides the ideal solution to this problem by allowing a double-precision transformation to be used to represent the camera position and overall transformation of each scene graph subbranch (e.g., a single terrain tile), while maintaining the use of high-performance single-precision values in the rest of the scene graph. This enables the correlation of the origin of the database to the camera position, thereby eliminating any precision artifacts.

### 17.0 Rendering Features for Realistic Visual Simulation

OpenGL Performer includes a large number of environmental and other advanced special effects for use in virtual reality and visual simulation systems. One of these features is the high-performance atmospheric model that includes fog, haze, and an earth/sky model with graduated sky color and horizon glow. OpenGL Performer also implements high-fidelity and volumetrically accurate layered fog, ground fog, and patchy fog and clouds effects.

Fig. 7. Examples of a light shaft

Such features can also be combined for specific simulation effects such as light shafts, with a patchy fog defining the shape of the light cone, and a layered fog modulating the cone density and color with elevation (see Figure 6). The use of programmable culling in OpenGL Performer further enables these advanced multipass features to operate at peak performance.

OpenGL Performer also offers sophisticated lighting effects, including projected shadows, image-based rendering shadows, and local projected lights to simulate objects such as landing lights and vehicle headlights.

OpenGL Performer also supports a rotorwash effect for helicopter simulation applications. The effect is overlaid upon the geometry of the scene graph and may vary its appearance depending on the material properties of the geometry. By automatically detecting the underlying material properties and geometry, the rotor-wash effect adjusts the color and appearance of the dynamic texture to achieve the appropriate visual effect.

OpenGL Performer incorporates wide support for visible, nonilluminating light points—essential for accurate renderings of a given view that might include such lights as stars, runway lights, visual approach slope indicators, precision approach path indicators, and even street lights when viewed from a great distance. The computation for these lights can be done in a separate process in parallel with the main rendering process, which can be multithreaded. OpenGL Performer contains full support for calligraphic light points and the management of calligraphic hardware.

### 18.0 API Interoperability

OpenGL Performer is interoperable with each of the advanced rendering toolkits and utilities created by SGI, enabling you to share data with programs using these tools and, in many cases, to directly utilize the features and capabilities of the tools in your own application based on OpenGL Performer.

OpenGL®—The industry-standard high-performance 3D graphics interface. OpenGL Performer uses OpenGL for all of its rendering functions and enables users to customize their application with objects and callbacks, where they can invoke OpenGL functions directly.

OpenGL Volumizer™—A volume-rendering API that targets the needs of the sciences and oil and gas markets, supports volume roaming, TLUTs, clip planes, volume paging, and many other volume-visualization operations. OpenGL Volumizer operations are supported in OpenGL Performer via the pfVolume object.

OpenGL Vizserver™—A tool that enables Visual Area Networking and allows universal access to applications based on OpenGL Performer from any IRIX, Linux, Solaris™, or Microsoft Windows client. OpenGL Vizserver is directly compatible with OpenGL Performer applications.

OpenGL Multipipe™ SDK—A software toolkit for creating OpenGL API-based multichannel and multi-pipe applications. OpenGL Performer can import OpenGL Multipipe SDK configuration files and use them to configure its own pipes and channels.

OpenGL Optimizer—A scene graph library that targets the needs of CAD and manufacturing applications; also includes the Cosmo3D™ library. OpenGL Performer can directly import OpenGL Optimizer and Cosmo3D .csb data sets.

Open Inventor—A popular object-oriented toolkit and scene graph library for interactive 3D graphics applications. OpenGL Performer can directly import Open Inventor .iv datasets.

Image Format Library—IRIX OS-based 2D image and texture file loaders for a variety of industry-standard formats including RGB, GIF, JPEG, TIFF, and many others. Use of the Image Format Library enables OpenGL Performer for IRIX to load these formats directly.

## 19.0 Scene Graph Types and Selected Objects of Interest

### 19.1 Libpf Node Types

**ASD** Active surface definition evaluates continuous level of detail of terrain

**Billboard** Rotates geometry to face the viewer for efficient rendering of symmetric geometry

**DCS** Dynamic coordinate system: applies transformation to its children

**FCS** Fluxed coordinate system: for asynchronous (fluxed) transformations

**Geode** Contains geometry described with GeoArrays, GeoSets, GeoStates, or Shaders

**Group** Groups with zero or more children

**IBRNode** Contains the textures and geometry to enable image-based rendering

**Layer** Renders co-planar geometry (e.g., stripes on a road or pictures on a wall)

**LightSource**
Light source to illuminate geometry in the scene

**LOD** Level of detail: selects one or more children based on distance from viewer

**Scene** Root node of a visual database

**SCS** Static coordinate system: applies static transformation to its children

**Sequence** Sequences through its children for sequenced animation effects

**Switch** Enables/disables traversal of children nodes in a group

### 19.2 Additional Libpf Objects

**Channel** Camera moving about the scene, defining which objects are visible

**Compositor**
Controller for multipipe composite displays

**FrameStats**
Holds per-frame timing, computation, and rendering statistics

**LightShaft** A light shaft cone special effect with volumetrically correct attenuation

**LODState** Represents custom LOD parameters and priority classes for LOD nodes

**MPClipTexture**
Manages the paging of a clip-texture for a graphics pipe

**Pipe** Software rendering pipeline implementing the App/Cull/Draw stages

**PipeWindow**
Controls the configuration and behavior of a window on the screen

**Rotorwash** Encapsulates the helicopter rotor wash effect

**Shader** Containers for multipass rendering specifications

**Shadow** Real-time shadows projected on the scene

**VolFog** Encapsulates layered and patchy fog rendering algorithms

### 19.3 Libpr Objects

**Calligraphic**
Configures/manages a calligraphic channel for rendering calligraphic light points

**Engine** Encodes a standard or custom user-defined operation on a fluxable object

**Flux** An object or buffer for dynamic and optionally asynchronous update

**GeoArray & GeoSet**
The containers for all geometric primitives in OpenGL Performer

**GeoState** Holds state description for geometry: texture, material, lighting, transparency, and fog

**LpointState** Holds description of light-point illumination behavior

19.4 Libpfv Objects

**Viewer** Modular execution structure for easy construction of OpenGL Performer applications

**World** Representation of the visual database with associated modules

**View** Encapsulation of camera positions and associated modules

**DisplayManager**
Manages the configuration of pipes, windows, and channels

**InputManager**
Manages keyboard, mouse, and other input

19.5 Libpfv Sample Modules

**Trackball** Module for trackball-like on-screen model navigation

**Loader** Module for model loading and placement

**Logo** Module to add a logo overlay to the display

**WorldSwitcher**
Module to enable seamless transitions from one world to another

**Earthsky** Module to control the earth and sky backdrop

**DrawStyle** Module to control the render mode of objects in the world

**Navigator** Module to navigate the eyepoint within the world

**Picker** Module for picking and selection of objects in the world

**Snapshot** Module to capture screen snapshots of the scene

**Motif®** Module for GUI integration with applications based on Motif

19.6 Supported Curves and Surfaces Types

**Base Classes**
pfRep, pfCurve2d, pfCurve3d, pfParaSurface, pfDVector

**Surface Classes**
pfPlane, pfSphere, pfCone, pfCylinder, pfTorus, pfNurbSurface, pfPieceWisePolySurface, pfRuled, pfCoons, pfHsplineSurface, pfSweptSurface, pfFrenetSweptSurface

**2D Curves** pfLine2d, pfNurbCurve2d, pfPieceWisePolyCurve2d, pfCircle2d, pfHsplineCurve2d, pfSuperQuadCurve2d

**3D Curves** pfLine3d, pfNurbCurve3d, pfPieceWisePolyCurve3d, pfCircle3d, pfHsplineCurve3d, pfSuperQuadCurve3d, pfCompositeCurve3d

**Untrimmed Surface Actions**
pfTesselateAction, pfTessParaSurfaceAction, pfTessNurbSurfaceAction

**Trimmed Surface Objects**
pfEdge, pfDisCurve2d, pfDisCurve3d, pfDisSurface

**Stitched Trimmed Surface Objects**
pfBoundary, pfJunction, pfTopo, pfSolid

## 20.0 OpenGL Performer and the Silicon Graphics Prism™ Family: A Powerful Combination for Real-Time Rendering

OpenGL Performer and the Prism family of visualization systems combine to provide the features, tools, and capabilities you need to build a complete real-time solution, including:

- Unsurpassed performance, optimized to achieve peak system throughput—by the company that invented OpenGL
- Flexible multiprocessor and multichannel system management, with the automatic use of multiple CPUs, video channels, and graphics pipelines

- Fully configurable for any display environment, including head-mounted, walk-in, desktop, flat and curved wall, or dome displays

- Image-based rendering for dramatically increased image quality, realism, and performance

- Volume rendering using OpenGL Volumizer, and Visual Area Networking using OpenGL Vizserver

- GPU programmability, including custom shaders, custom effects, clip-texturing, and GLSL support

- The ability to spatialize arbitrary data sets to improve the efficiency of operations such as culling and drawing

- Support for composite combined output on Prism systems to scale multipipe performance into a single display

- Low-latency, real-time fixed frame-rate control to maximize immersion

- Automatic scene and load management features, view frustum culling, level of detail evaluation, and dynamic video resolution

- Per-polygon frustum culling, backface removal, and occlusion culling

- Run-time and real-time profiling for use in debugging, load management, and performance tuning

- Real-time fly-through of geospecific terrain, database geometry, and imagery

- Active surface definition for automatic paging of terrain with continuous LOD evaluation

- Asynchronous database intersection and user data-base processing

- Dynamic animated geometry and morphing

- Atmospheric effects for visual simulation, including fog and haze, and support for range/angle-correct layered fog and patchy fog

- Visual simulation effects, including real-time shadows, light shafts, light points, rotor wash, and calligraphic lights

- Double precision coordinate systems for large area database support

- Interoperability with more that 70 industry-standard data storage formats

## 21.0 Release and Compatibility Information

### 32-bit Linux:

Qualified for Red Hat® Linux® versions 8.0 and above and SuSE® Linux version 9.1

### 64-bit Linux:

Qualified for SGI ProPack™ for Linux®

### IRIX:

All SGI systems running IRIX® 6.5 and later
Supports O32, N32, and N64 MIPS® ABIs
Includes backwards-compatibility environment for OpenGL Performer™ 2.0 and above

### Windows:

Microsoft Windows 2000
Microsoft Windows XP

## 22.0 Further Information

For additional information on OpenGL Performer, visit the Web site at *www.sgi.com/software/performer/.*



Corporate Office
1500 Crittenden Lane
Mountain View, CA 94043
(650) 960-1980
www.sgi.com

North America +1 800.800.7441
Latin America +55 11.5509.1455
Europe +44 118.912.7500
Japan +81 3.5488.1811
Asia Pacific +1 650.933.3000



3637 [10.05.2005]

J15003