Solaris VISUAL Geometry Interfaces 3

Geometry Interfaces

Geometry applications are among the most demanding in terms of computation and display resources. Application requirements for geometry rendering also vary from market to market, and resulting in a wide array of third-party application program interfaces.

Solaris VISUAL provides for the requirements of both geometry applications and APIs through its XGL foundation library, PEXlib API and associated PEX protocol. Third-party API developers will find that XGL and PEXlib both provide a rich feature set to support their interface, while providing access to the PEX protocol.

Geometry Applications

Mechanical Computer-Aided Design (MCAD) and Architecture, Engineering and Construction (AEC)

A likely user of geometry graphics is the implementor of 2-D and 3-D Mechanical CAD applications. MCAD applications typically place the greatest demands on graphics hardware through the need to manipulate floating point coordinate data and the computation of transformations. Frequently, representation of shaded, lighted objects in 3-D modeling is desirable for realistic visualization prior to manufacturing.

Applications in the AEC market are typically associated with drafting and design of 2-D and 3-D drawings and plans. Because this market relies upon accurate coordinate representation, geometry-based graphics interfaces are usually the best choice to provide the precision and performance necessary to manipulate and display data.

Animation

Animation markets rely heavily on fast, high-quality rendering of 3-D objects. Computation of object movement also places significant demands on graphics hardware in animation. Animation applications and tools demand broad and sophisticated graphics rendering primitives and features such as antialiasing, texture-mapping and other advanced features.

Electronic Design Automation (EDA)

Another significant application area for geometry graphics is Electronic Design Automation. Historically, the requirements of this market have been integer or fract coordinate data and rich 2-D primitive support.

Geographic Information Systems (GIS)

The general attributes of GIS applications are broad variation in data format and large amounts of data. Most commonly, GIS applications utilize either the industry-standard GKS interface or low-level interfaces such as Xlib.

Where data format or quantity is particularly important, SunSoft's geometry graphics APIs offer a flexible choice by providing broader coordinate and data type support and higher-level functionality.

Earth Resources

Like Geographic Information Systems, the Earth Resources market typically deals with large volumes of data. Earth resources market requirements include 2-D and 3-D geometry data and frequently 2-D image data. With such varied requirements, Earth Resources applications need flexible geometry interfaces with a range of coordinate type support and efficient image manipulation capabilities.

Molecular Modeling

Molecular modeling applications are typically 3-D in nature and require fast manipulation (3-D transformation) and rendering of molecules. To facilitate quality visual display of data, molecular modeling applications often rely on advanced features such as antialiasing, texture-mapping and lighting/shading.

The XGL Foundation Library

XGL is the Solaris VISUAL foundation geometry library providing the functionality and performance required by applications requiring geometry manipulation and display. XGL provides for optimization of 2-D and 3-D rendering, including high-quality lighting and shading, advanced primitives (such as NURBS and meshes, texture mapping, antialiasing and transparency) and a flexible geometry pipeline. XGL includes sophisticated state-of-the-art features such as dynamic tessellation of NURBS surfaces, as well as flexible pipelines for lighting and transformation.

XGL has an object-oriented interface that hides the low-level details of implementation from the application developer. As such, it is architected so that it may evolve as an interface, permitting new algorithms to be implemented within XGL without incompatibly changing the application program interface bindings. Figure 3.1 illustrates how XGL works with applications and Solaris or third-party APIs.

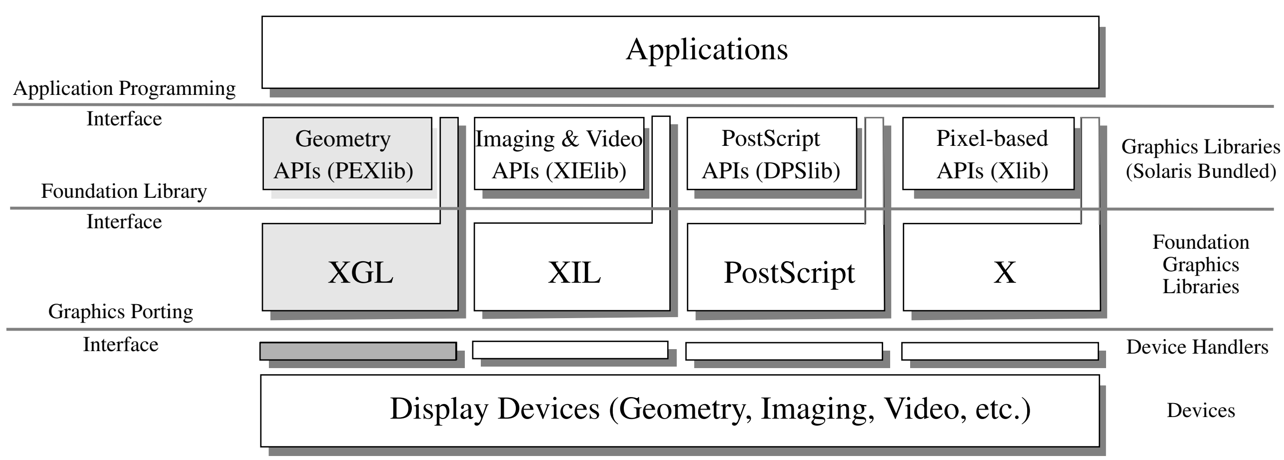


Figure 3.1 Solaris VISUAL Geometry Foundation Library and Layered Interfaces

The XGL Library contains two primary interfaces: a software programming interface specification for API and application developers and a standard graphics porting interface, which enables third-party hardware developers to readily support XGL applications.

The XGL Library delivers high functionality and performance to applications and graphics APIs that wish to maintain their own display lists or data structures. It complements other graphics interfaces by providing similar but separate 2-D and 3-D graphics pipelines, very broad primitive and coordinate type support, and direct control of graphics display (updates).

The XGL graphics interface is also specifically designed to emphasize immediate-mode rendering. Immediate-mode refers to immediate rendering of graphics primitives without the requirement for placing the data in a display list, and using predefined subroutines to manipulate, edit, or search the data set. Using this approach, XGL lets the application organize the data in a manner most appropriate to the application or API. The immediate-mode approach is especially well suited for applications requiring highly interactive characteristics, or maintaining specialized data, including applications in mapping, animation, and simulation.

XGL Core Library and Rendering

At the most basic level, XGL has two components: the XGL Core and the XGL Rendering Pipelines. The Core functions as the interface between the applications program and the Rendering Pipelines. The Rendering Pipelines turn a geometric data description of an object and its state attributes into pixel data that is displayed on a graphics hardware device or written into memory.

The XGL Core contains device-independent class hierarchies for internal objects and for API objects and abstract pipeline class interfaces to the underlying pipeline operations. The Core takes care of a variety of device-independent tasks including creation and management of API graphics objects, and processing of state changes for its pipelines.

The Rendering Pipelines include one or more device pipelines and at least one software pipeline. The pipelines are divided into three layers, each responsible for interfacing at specific pipeline functions. The default pipeline is the software pipeline which XGL provides for devices that have only minimal acceleration capabilities. The software pipeline consists of functions that implement the geometry pipeline (LI-1), and scan conversion (LI-2) functions in software. A device pipeline can choose to override the default software implementation for any primitive, choosing instead to accelerate the primitive's rendering through the accelerator's own facilities. Figure 3.2 provides a high-level view of the XGL architecture.

XGL is structured hierarchically through a series of abstract data types called *classes*. Applications can create *instances* of classes, where an instance of a class is an *object*. The class of an object determines the attributes it possesses and which operators can act on it. A class defined hierarchically below an existing one is known as a *subclass* (child) of the existing (parent) class. Although XGL is class-based, an application can only define instances of predefined classes (*objects*), and cannot extend the hierarchy by creating new classes. Figure 3.3 illustrates XGL's Class hierarchy.

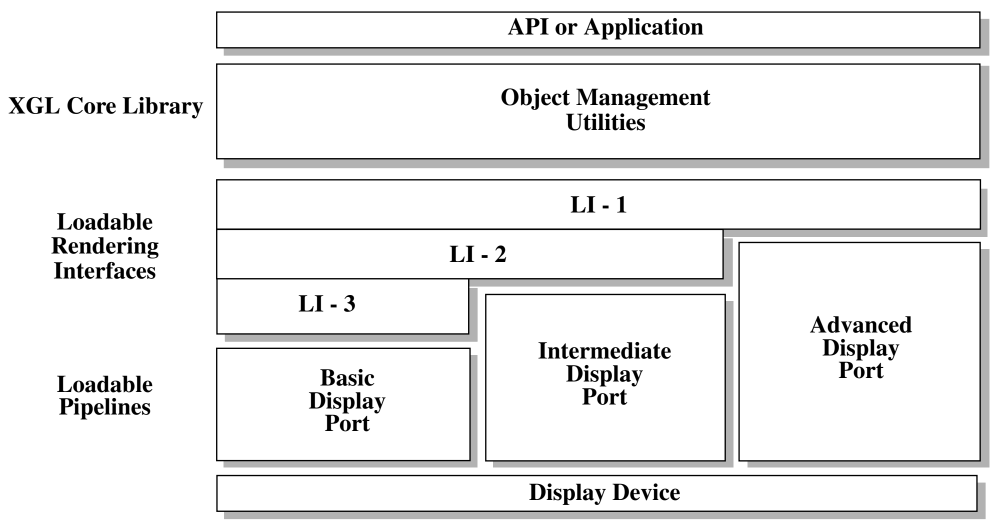


Figure 3.2 Solaris VISUAL: XGL Architecture Detail

XGL Objects

In XGL, an *object* is an abstraction of a graphics resource. Objects consist of *state information* (known as attributes in XGL) and a set of operators. Operators perform predefined actions on objects of their related class.

The primary XGL objects available for the manipulation of 2-D and 3-D data are *System State*, *Context*, *Device*, *Gcache*, *Color Map*, *Dmap Texture*, *MipMap Texture*, *Transform*, *Stroke Font*, *Line Pattern*, *Light*, and *Markers*. The following paragraphs provide a brief discussion of XGL objects. Figure 4.3 illustrates XGL's object hierarchy.

System State Object and Generic Operators

The XGL System State object maintains information pertaining to all operations occurring during a single XGL session. Operators of the System State object open and close XGL and adjust the degree of error checking XGL performs.

The 2D and 3D Context Objects

The 2D and 3D Context objects provide 2-D and 3-D drawing primitives and graphics attributes for basic rendering. Several Context objects can exist simultaneously and can share resources associated with other XGL objects. The Context object maintains both graphic and environment information on Coordinate Systems.

The Device Object

The XGL Device object can be either a raster or non-raster device. If the device belongs to the raster class, it can be either a *Window Raster* (displayed on a screen) or a *Memory Raster* (consisting of an off-screen array stored in system memory).

Non-raster devices provide protocol-independent stream-I/O pipelines, for creating formatted output such as PostScript or Computer Graphics Metafile (CGM).

The Geometry Cache (Gcache) Object

The XGL Gcache Object reduces the complexity of application geometry by allowing XGL to process the geometry into simpler forms. It reduces single complex primitives into many simple primitives, storing the relevant attribute values. The Gcache object additionally provides a mechanism to store graphics primitives that are rendered frequently, thus yielding higher performance

The Color Map Object

XGL Color Map objects specify colors in a device-independent manner. The XGL application can work within its own color space, even if the hardware employs a different one. The Color Map object attached to the device ensures optimal correspondence between the application's color model and the color model of the device. XGL supports both indexed and RGB color models.

The Transform Object

The XGL Transform Object provides attributes and operators for geometric transformations performed on graphics primitives. These include translation, scaling, and rotation, as well as conversion from one coordinate system to another. A single 2D Context or 3D Context typically uses several Transform objects, which in turn can be shared (as defined by the application) between several Context objects.

The Stroke Font Object

The XGL Library provides a Stroke Font object that can be shared between Contexts. An XGL application may use Stroke Fonts to render stroke text in a variety of font types on the Device attached to a Context.

The Line Pattern Object

The XGL Library provides a Line Pattern objects that are sharable between Contexts, and may be referenced simultaneously. Line Pattern objects are used to render non-solid lines on a Device. The application may explicitly define the Line Pattern or may use one of several predefined Line Patterns XGL creates internally.

The Light Object

The Light object allows a 3D Context to render graphical primitives with *ambient*, *positional*, *directional*, and *spot* light sources. Light objects can be shared by different Contexts while remaining independently controllable on a per-context basis.

The Marker Object

The Marker object allows applications to describe their own markers or symbols and share them between different Contexts. A user-defined Marker can contain as many as 128 vectors.

The Dmap and MipMap Texture Objects

The Dmap Texture object allows an application to specify the control parameters needed to do texturing of 3-D surface primitives when rendering through a 3D Context.

The MipMap Texture object allows the application to specify texture images, which are generally predefined raster images. The MipMap Texture object also provides filtering capabilities to generate MipMaps.

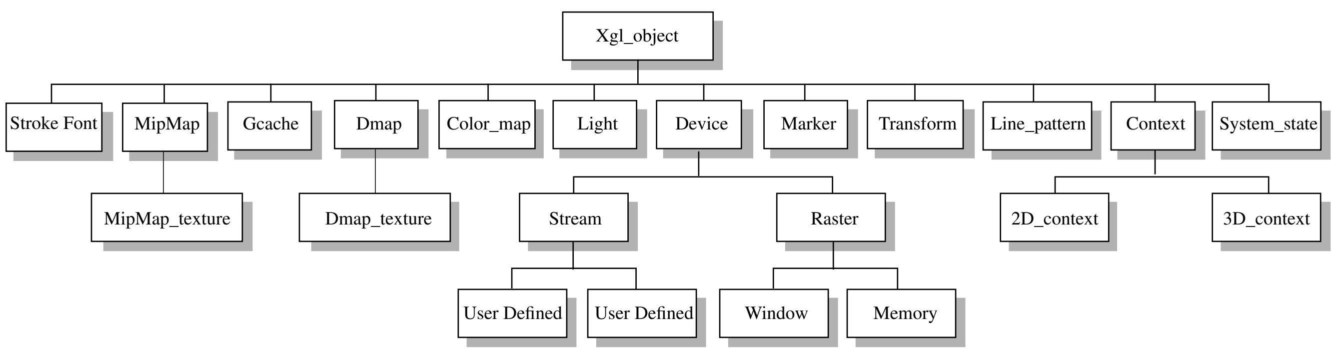


Figure 3.3 XGL Class Hierarchy

XGL Primitives

The XGL Library accepts a large number of different representations of geometrical data, providing applications with flexibility in choosing an appropriate implementation. Data representation includes coordinate type and attributes associated with the rendered geometry. Figure 3.4 illustrates the most commonly used XGL primitives (rendering operators) and some of their associated attributes. Most primitives are renderable through both 2-D and 3-D pipelines.

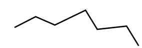
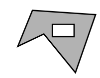
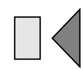
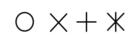
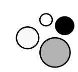
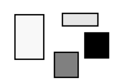
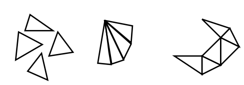
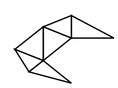
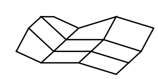
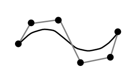
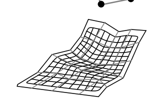
Rendering Operator	Description	Example Attributes	Example
xgl_multipolyline	Set of connected lines	Line Color, Style, Width	
xgl_polygon	Polygons	Polygon Color Fill Pattern Edge Style, Width Illumination Type	
xgl_multi_simple_polygon	Polygons (single boundary, 3 or 4 vertices)	Polygon Color Fill Pattern Edge Style, Width Illumination Type	
xgl_multimarker	Marker symbols	Marker Color Scale Factor	
xgl_multiarc	Arc	Arc Color	
xgl_multi_elliptical_arc		Edge Style, Width	
xgl_multicircle	Circle	Circle Color Edge Style, Width Fill Pattern	
xgl_multirectangle	Rectangle	Rectangle Color Edge Style, Width Fill Pattern Illumination Type	
xgl_triangle_list	Series of Triangles (triangle strip, triangle star or independent triangles)	Triangle Color Fill Pattern Edge Style, Width Illumination Type	
xgl_triangle_strip	Triangle strip (interconnected triangles)	Triangle Color Fill Pattern Edge Style, Width Illumination Type	
xgl_ quadrilateral_mesh	Quadrilateral mesh (interconnected quads)	Quad Color Edge Style, Width Fill pattern Illumination Type	
xgl_nurbs_curve	Non-Uniform Rational B-Spline Curve	Line Color Line Style, Width	
xgl_nurbs_surface	Non-Uniform Rational B-Spline Surface	Surface Color Edge Style, Width Fill Pattern Illumination Type	
xgl_stroke_text	Stroke text	Text Font Name Character Size, Width Character Direction	TEXT

Figure 3.4 XGL Rendering Operators

The PEX Extension to the X Protocol

A limitation of the Xlib interface is its inability to provide a scheme for distributed graphical databases. There is no method to distribute graphical data outside of the client. This restriction is very troublesome for hardware which is architected with an embedded hardware display list, or for software oriented around a display list. It is equally important to geometry applications which manage 2-D or 3-D floating point data, and which wish to take advantage of hardware transformation capabilities of the remote display hardware.

The designers of the X Window System were aware of the limitations of X in managing 3-D data, and determined that the best way to address it would be through the portable extension mechanism, allowing for adding capabilities to the server, and extending the communications protocol used between the server and its clients. X Consortium work on the requirement for a 3-D protocol led to the development of PEX.

PEXlib

The availability of PEX originally provided clear benefits to developers who wished to provide better integration of PHIGS programs into to the X Window System. At the same time, some implementors have perceived a need to access and use the PEX protocol directly, without using the PHIGS library, but with high-level bindings more appropriate to application development than the raw protocol.

PEXlib was designed as an extension to Xlib using the same transport and error mechanisms found in Xlib. In a manner analogous to Xlib and the X protocol, PEXlib provides developers with a mapping of C language call access to the PEX protocol.

SunSoft has extended the XGL foundation library to support PEX protocol as well. In doing so, applications and particularly APIs which use XGL can easily be extended to support network-distributed 3-D graphics to other PEX-based systems and terminals.

Remote Rendering using PEX

An application developer is not required to concern himself with network location when programming in PEXlib. The developer may however, want to optimize his application for the network, assuming remote rendering will occur. Design of the client's display list, and user-interface event handling are areas of particular interest in architecting applications to best use PEX. Figure 3.5 illustrates remote rendering of a PEXlib client application.

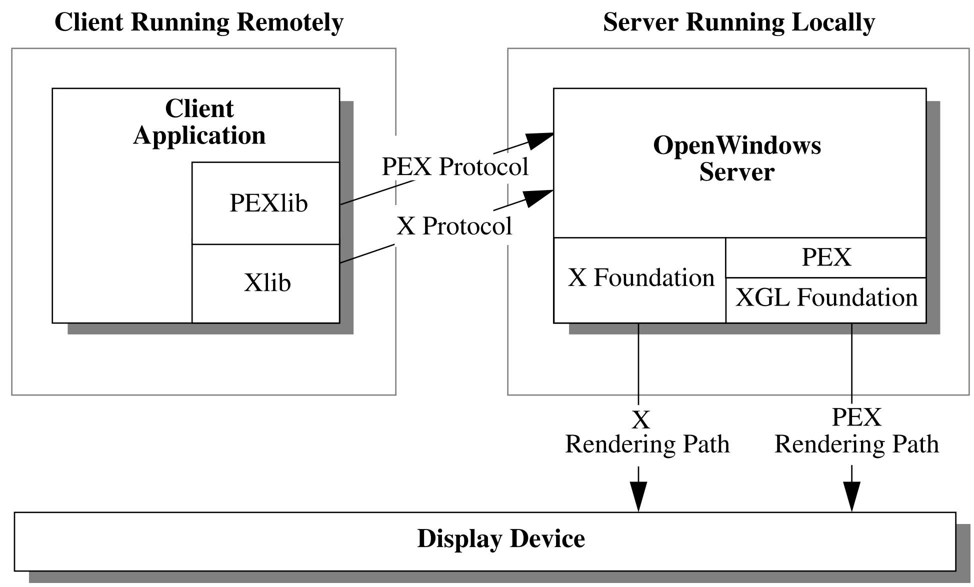


Figure 3.5 X/PEX Protocols and the X Windows Environment

Third-Party Geometry Interfaces

The Solaris VISUAL architecture is designed to support third-party geometry libraries through the XGL foundation library and the PEXlib API.

A variety of geometry APIs, both third-party and committee-defined are available today, but lack the optimized features of either a common display hardware-optimized foundation library, or a network distributed protocol. The Solaris VISUAL environment provides both of these through XGL, PEXlib, and the PEX protocol extension.

Examples of popular 2-D and 3-D geometry rendering libraries available on Solaris include various implementations of PHIGS (SunPHIGS, ATC, Liant, etc.), GKS (ATC, Prior Data Sciences, Etc.), HOOPS™ (Ithaca), OpenGL™ and Iris GL™ (Nth Graphics, Dupont Pixel Systems) and many others. These libraries take advantage of the Solaris VISUAL environment to operate locally or remotely as appropriate.