Analysis of PEX 5.1 and OpenGL 1.0

Allen Akin

Silicon Graphics Computer Systems

August 3, 1992

1. Introduction

PEX and OpenGL are two 3D graphics systems contending for the position of de facto standard in the workstation market. While they are similar in many ways, they also differ in important aspects of their application programming interfaces, functionality, performance, portability, openness, and responsiveness to changes in markets and in technology.

To our knowledge, no detailed comparisons of PEX and OpenGL have been published. Perhaps this is because the two systems are evolving and few individuals have experience with both. With the release of PEX 5.1 and OpenGL 1.0, we believe the systems are relatively stable and it is now time to undertake a comparison.

This paper analyzes some of the significant differences between PEX and OpenGL, with attention to issues faced by users as well as implementors. We hope it will be a positive contribution to the debate.

2. Programming Interfaces

There is an essential difference in scope between PEX and OpenGL:

> OpenGL is an *application programming interface.* The OpenGL specification defines the data structures and procedure calls application programs must use to produce graphical output, as well as the semantics of the drawing process.
>
> PEX is a *wire protocol*. That is, PEX specifies how graphics requests are to be encoded (as if for transmission over a network), and what the semantics of those encoded requests must be.

PEX does *not* specify an application programming interface. One of the primary goals of PEX is efficient support for PHIGS. The original PEX designers planned to use the ANSI and ISO programming language bindings for PHIGS, and they explicitly refused to define a competing API. Their decision was reflected in the name PEX, which is often said to be an acronym for <u>P</u>HIGS <u>E</u>xtensions to <u>X</u>.

This choice generated considerable controversy. Although PHIGS has the advantages of international standardization and an existing software base, it is quite complex and some software developers have claimed that using it would force them to restructure their applications. (For example, standard PHIGS lacks support for immediate-mode graphics.) As a result, many PEX proponents favor the use of a "PEXlib" which would expose the PEX protocol more completely. The X Consortium has proposed such an interface. It offers greater device-specific control and in many cases better performance than PHIGS. However, it also exposes PEX implementation dependencies (such as subsets) that must be handled by special-case code in applications rather than in the API.

The deficiencies of PHIGS and PEXlib have led to proposals for additional PEX-compatible APIs. While supporting a variety of APIs improves the chance that at least one will be appropriate for any given application, the proliferation of interfaces has slowed the acceptance of PEX. Many hardware vendors and software developers have chosen to wait for one API to dominate, rather than risk investing in one that fails. The possibility that incompatible protocol changes in PEX 6.0 will force incompatible API changes in PEXlib has also caused concern.

The situation with OpenGL is much simpler. There is a single well-defined OpenGL API. To guarantee interoperability in heterogeneous networks, there is an OpenGL extension (GLX) and wire protocol for use with the X Window System. In other environments (e.g. Microsoft Windows) where currently there is no need for a wire protocol, OpenGL has none.

In recent months some in the PEX community have advanced the view that the PEX protocol is a general-purpose foundation for supporting *all* 3D graphics APIs. IRIS GL is arguably the market-leading API for 3D, but PEX 5.1 is missing functionality that is crucial for supporting it. (See below.) In contrast, it has been demonstrated that GL has sufficient functionality to support PHIGS because there are commercial PHIGS implementations built on IRIS GL. Furthermore, IRIS GL has also been used to support higher-level APIs including IRIS Inventor, IRIS Explorer, and Ithaca Software's HOOPS. These examples lead us to believe that the GLX protocol is actually a better candidate than PEX to support a wide variety of APIs on an equal footing.

3. Functionality

It is difficult to compare PEX and OpenGL functionality because of a major difference in design philosophy:

> Nearly all PEX functionality is *optional:* it is not present in all PEX subsets, or some aspect of it is "implementa-

tion-dependent." The PHIGS API or PEXlib-based applications must query the PEX extension before using any implementation-dependent feature, and adjust their behavior accordingly. For example, a PEX implementation might include just the "PHIGS Workstation" subset, and thus be incapable of immediate-mode rendering. Alternatively, it might include just the "Immediate Rendering" subset, and thus be incapable of anything else! As another example, PEX provides an interface to select Phong shading, but there is no requirement that Phong shading actually be supported (and to our knowledge, no current implementations of PEX support it). In short, it is not easy to say precisely what functionality PEX includes, because it varies so much from implementation to implementation.

In contrast, all OpenGL implementations are required to support *all* the functionality described in the specification. Even the advanced rendering features are guaranteed to be present.

Despite this fundamental difficulty, some functionality comparisons can be made, and in such areas it is important to understand the differences between the two systems. Some examples follow.

3.1. Text

PEX has extensive support for text, including a PEX font mechanism that is independent of X Window System fonts. (Access to X11 fonts is optional.) Ordinary PEX stroke-precision text is derived from a vector font, and is fully 3D-transformable.

OpenGL provides routines that convert fonts from its environment (the X Window System or Microsoft Windows) into OpenGL display lists. The only font mechanism is that provided by the environment. If the original fonts are in outline form (e.g. TrueType), then the display lists will contain fully transformable geometry. If the original fonts are in bitmap form, then the display lists will contain bitmaps.

PEX has the advantage that a fully transformable vector font is always available. OpenGL has the advantage that it guarantees all of the environment's existing fonts may be used, and there is no need to create, license, or distribute new fonts specifically for 3D.

3.2. Display List Editing

Both PEX and OpenGL are used in networked environments, where it may be expensive to transport drawing requests from the client application to the graphics server. Both provide a server-side storage mechanism (called "display lists" in OpenGL, "structure store" in PEX) that allows applications to cache drawing requests on the server and avoid transport costs for requests that are reused.

PEX supports nearly all of the PHIGS structure store editing functionality, including the ability to label sections of structures; insert, replace, or delete elements of structures; copy structures; and search through structures according to labels, hierarchical organization, or spatial position of the primitives in the structures.

OpenGL display lists may only be created, replaced, and deleted. However, these operations can be performed on structures with very fine granularity. For example, in OpenGL it is possible to create display lists containing the data for independent vertices of a polygon, and to define the polygon as a set of calls to those display lists. A given vertex of the polygon may then be "edited" by replacing its display list. This is not possible in PEX; individual vertices cannot be edited, and primitives must be replaced in their entirety. (See **Performance** below for some consequences of this difference.)

3.3. Complex Polygons

Both OpenGL and PEX support single- and multi-contour polygons with convex and nonconvex shapes. PEX supports self-intersecting contours and polygon lists with shared geometry, while OpenGL does not.

3.4. Feedback

OpenGL has provisions for tapping the graphics pipeline, so that the results of geometric transformations and lighting computations can be "fed back" to the application. PEX has no analogous capability.

3.5. Alpha Blending

OpenGL supports an alpha (coverage) buffer and an explicit alpha component in colors. These are used for antialiasing and image composition. PEX does not support alpha blending.

3.6. Image Operations

OpenGL includes routines to read, write, and copy pixels. These routines may be directed to the framebuffer or to bitmap data structures in the application. A number of operations including scaling, luminance transformations, and type conversion may be applied to the pixel data as it is moved. This functionality can be used to implement ordinary 2D raster operations, but is also used to render text and is needed to support image manipulation for texture mapping.

PEX has no support for image operations.

3.7. Texture and Environment Mapping

OpenGL includes a complete implementation of texture mapping. The application controls the exchange of rendering time for image quality, and it may use its own texture prefiltering if that provided by OpenGL is inadequate. Texture coordinates may be specified explicitly or generated automatically, and they may be transformed by an arbitrary 4x4 matrix. Textures may be applied to a surface as if they were decals, or they may be blended with the surface in a number of ways.

PEX has no equivalent capability.

3.8. Antialiasing

PEX has no official support for antialiasing. Some implementations have provided it as a nonstandard extension.

OpenGL has explicit support for antialiasing points, lines, and polygons. In addition, the accumulation buffer may be used to implement full-scene antialiasing.

3.9. Progressive Refinement

The OpenGL accumulation buffer permits straightforward implementation of progressive refinement, the gradual improvement of image quality over a period of time. PEX has no equivalent.

3.10. Capping Planes

OpenGL supports a multipass capping-plane algorithm using its stencil buffer and other auxiliary buffers. PEX as currently defined cannot support this functionality.

3.11. Constructive Solid Geometry

The OpenGL stencil, accumulation, and auxiliary buffers can be used to render objects defined by the regularized Boolean operations of constructive solid geometry. PEX does not have this ability.

3.12. Motion Blur

The OpenGL accumulation buffer can be used to simulate motion blur. PEX has no support for it.

3.13. Depth-of-Field

With appropriate change in viewing parameters, the OpenGL accumulation buffer can be used to produce an image with finite depth-of-field. This function is not available in PEX.

4. Performance

Interactive 3D graphics demands much higher performance than 2D graphics, and therefore affects more aspects of system design. The architecture of a successful 3D graphics system must encompass not only 3D transformations, shading, and scan conversion, but also CPU load, cache behavior, main memory bandwidth requirements, main memory capacity requirements, bus transaction latency and throughput, address translation costs, and graphics processor request queueing, among other things. These issues must be considered for the graphics software, operating system, window system, and even the applications programs.

There are numerous existence proofs that both PEX and OpenGL can support high-performance implementations. However, the two systems do not place equal emphasis on some of the design issues mentioned above. We believe that these differences in design would lead to differences in performance should PEX and OpenGL be implemented on the same hardware, and we believe that trends in semiconductor technology will tend to widen the performance gap.

4.1. Data Reformatting

It is nearly always necessary for applications to maintain private data structures that are specialized for their particular problem domains. Graphics data is just one component of these structures.

Current PEX APIs (and the PEX protocol) use their own data structures for describing graphics primitives. For example, a polygon consists of a counted list of vertices, each of which contains coordinates in floating-point format, plus optional data such as colors and surface normals.

When an application wishes to make an immediate-mode graphics request, it must extract data from its private data structures, reformat it to match the PEX interface requirements, write it to a temporary buffer, and then invoke the PEX API to process it. This has several consequences:

> The CPU must spend cycles to reformat and copy the data. These cycles do no useful work, yet they are lost to the application and to other processes in the system that might have made use of them. Furthermore, since they are in application code outside the scope of the graphics system, no improvement in the graphics hardware or software can recover the loss.
>
> For high-performance interactive 3D graphics applications, where a considerable amount of data must be processed for each frame, the reformatting process sweeps the system's data cache repeatedly. This can drastically reduce the effectiveness of the cache, and thus degrade performance for the entire workstation.
>
> During reformatting, the CPU is generating memory transactions at a high rate. The bus traffic resulting from these transactions interferes with other users of the system bus. One critical user of the bus that may be adversely affected by this traffic is the graphics processor.

To quantify this problem, consider a hypothetical workstation (simplified, but similar to many in use today). It has a 32-byte cache line, a 64-bit-wide bus to memory, and main memory with a 240ns random-access cycle time and an 80ns page-mode-access cycle time. A cache line occupies four sequential addresses, so it can be read in (240ns + 80ns + 80ns + 80ns), or 480ns. We'll ignore the cost of reformatting. Writing the cache line to memory takes another 480ns. Therefore we can copy 32 bytes in a total of 960ns, for an average copy rate of 33MB/s. An independent 3D triangle is composed of three vertices, each having three coordinates, three color components, and three surface normal components. If all of these are specified by single-precision floating-point values, then the total size of the triangle is 108 bytes. Therefore our 33MB/s copy rate is sufficient to copy a little more than 300K triangles per second. At this rate, our workstation is completely saturated by the data reformatting process; there is no time for drawing the triangles, much less running the application. *This maximum performance level is already below the target drawing rate for comparable machines that exist today.* And any additional reformatting or copying (for example, to build PEX protocol packets) reduces the maximum drawing rate still further.

OpenGL uses a different approach. The OpenGL API allows graphics data to be extracted directly from the application's private data structures, without reformatting. It is also possible to support this interface in hardware in such a way that the data need never be transferred into the CPU, so it will not sweep the CPU's cache. This is impractical for PEX on the hardware with which we are familiar.

For years, CPU cycle times have improved more rapidly than main memory cycle times. We expect this trend to continue, with the result that PHIGS and PEXlib-style APIs (as well as any new PEX APIs that reformat data to build PEX protocol packets) will become increasingly limited by memory access. Since so many cycles will be spent copying data, systems based on PEX will need to be faster and more expensive to sustain the same overall level of application performance as systems based on OpenGL.

4.2. Data Duplication

There are two ways developers of PEX-based applications can avoid the dilemma described in the previous section. The first is to rewrite all data-structure-handling code so that the graphics data is stored in precisely the form required by PHIGS or PEXlib. In the extreme this involves dispensing with private data structures entirely, i.e. using PEX structure store for all graphics data and PEX structure extensions ("GSEs") for private non-graphics data. Not only would this force applications to be rewritten, but it would also require the use of interprocess (and sometimes network) communication to access the application database. In any case, eliminating the application's specialized data structures is usually not feasible, since those structures are instrumental in providing the added value that makes the application successful commercially.

The second approach is to duplicate the graphics data, keeping a copy in the proper form for a PEX API. This can be done by maintaining parallel data structures in the application or in PEX structure store. In practice, we have found that these methods rarely work. In the case where the graphics data is static, and well-separated from other application data, maintaining a parallel graphics database can be effective. Usually, though, the graphics data is being referenced frequently, is edited, or is intermixed with other data. In such cases maintaining parallel data structures causes the real memory requirements of the application to expand dramatically. Typical CAD/CAM graphics databases are 20MB-40MB in size today, and are expected to grow. Customers are reluctant to purchase that much real memory to add to their systems, especially if they know that it is only being used for duplicated data! Applications developers are reluctant to add the code needed to keep the duplicated data in sync with the main database.

OpenGL avoids all of these problems by making it unnecessary to duplicate or reformat graphics data.

4.3. Display Lists

Once a PEX structure is created, it may be edited by inserting or deleting new structure elements. Once an OpenGL display list is created, it may only be deleted or replaced; it may not be edited.

OpenGL makes this restriction to allow its display lists to be "compiled" into a more efficient form at the time they are created. This is particularly effective for entities like texture maps and material definitions that may involve a significant amount of preprocessing. It can also be used to perform data type and format conversions, or to move drawing requests into special display list memory for fast access by the graphics hardware.

PEX structures may be compiled, too. However, because they are editable, a number of complications arise. For example, since there is no provision in PEX for "opening" and "closing" a structure, the PEX extension cannot determine when edits to a structure are complete. Recompiling a structure after each edit could be a great waste of time, since another edit might arrive shortly thereafter. To avoid this, recompilation must be delayed until the structure is actually invoked. This increases implementation complexity and on systems with graphics accelerator hardware it can increase overhead in the "fast path" case for drawing.

Another issue arises when a PEX structure is in main memory, and the graphics hardware accesses it by DMA. Since a given structure may be read by the graphics hardware and written by the structure editor at the same time, access to it must be interlocked. This interlock must be fairly fine-grained; one cannot simply lock out edits until structure traversal is completed, because the PHIGS model for graphics requires support for continuous traversal and "pending" edits. Usually the structures being changed are precisely those that need to be accessed by the graphics hardware to update the display, so conflicts are frequent. Furthermore the required locks are interprocessor locks, so they are expensive on many machines.

On a machine with a similar architecture, OpenGL display lists would also require interlocks. However, the interlocks are needed for just three operations (display list replacement and deletion must be interlocked with display list lookup), and continuous traversal is not an issue. Fewer synchronization operations are needed than for PEX, and the critical sections complete more quickly (often in a single instruction).

4.4. Display List Editing Granularity

PEX structures are edited by inserting, deleting, or replacing "structure elements," which consist of geometric primitives or attribute-setting commands. Thus PEX structure editing can be applied to individual primitives, but not to any entity at a finer granularity. This can cause performance problems when editing large primitives. For example, to change the location of one vertex in a large quadrilateral mesh, the entire mesh must be re-transmitted to the server. In many cases this eliminates the advantage of PEX structure store.

As noted above in **Functionality**, OpenGL display lists may invoke other display lists even within a single primitive. Although display lists may not be edited, they may be replaced. Together these two features allow primitives to be edited at a very fine granularity. For example, a quadrilateral mesh can be defined as a set of calls to display lists that define its individual vertices. Replacing the display list for a vertex effects a vertex-level edit. Only the data for the new vertex need be transmitted to the server.

4.5. State Save and Restore

PEX semantics require that all rendering pipeline attributes be saved when a structure is traversed, and restored when the traversal is completed. This demands an expensive "push and pop" of the complete state, or a complex copy-on-write scheme that adds cost to attribute-setting commands.

OpenGL state is saved and restored selectively, and only when the client application requests it, so the process can be optimized or even eliminated if it is not needed.

4.6. Indirect Attributes

In PEX, attributes like the color and style of a primitive may be stored in "bundles." A flag associated with each primitive determines whether that primitive's attributes are provided directly (along with the geometry for the primitive) or indirectly (from a bundle). In immediate-mode drawing the flag must be tested, and that slows down the fast drawing path for the primitive. In both immediate and structure modes, the use of the bundle forces a memory access to a nonsequential address, which often causes a TLB miss or a random-access penalty for the memory subsystem. Since the bundle tables may be large, they cannot always be stored on the graphics subsystem; this makes it difficult to optimize the graphics pipeline for some important cases (like polygons, where the style of the primitive has a substantial effect on the computations needed to display it).

OpenGL pushes attribute lookup back to the level of the application, so the cost of indirection is incurred only when it is strictly necessary. The display list mechanism can be used to implement attribute indirection straightforwardly, for arbitrary sets of attributes.

5. Portability

The single most important reason to standardize a graphics system is to guarantee portability for the applications that are based on it. PEX and OpenGL may well succeed or fail depending on how effectively their APIs and wire protocols support this goal.

5.1. Subsetting

PEX has three mutually-incompatible subsets: immediate-mode, structure-mode, and "PHIGS Workstation." Vendors of PHIGS for PEX must provide three significantly different implementations layered on these subsets to insure network transparency and application portability. A single PEXlib is sufficient to support all three subsets, but at the cost of pushing portability issues all the way up into the application. Real software portability does not yet exist in the PEX world because of the wide variations in commercial versions of PEX.

The existence of these PEX subsets may be indicative of the difficulty of building a full PEX implementation. If true, this has negative implications for the quality, availability, and timeliness of PEX.

In order to combat application portability problems caused by subsetting and optional functionality, a group of PEX vendors has found it desirable to fund a private organization, the PEX Interoperability Center, just to provide a PEX porting and testing environment.

OpenGL may not be subsetted. Application developers can be assured that all OpenGL features will be available on all implementations. Unlike PEX today, there exists an OpenGL sample implementation with complete functionality. The need for interoperability testing is greatly reduced because OpenGL implementations are much more consistent than PEX implementations.

5.2. Optional Functionality

Many of the features described in the PEX protocol specification are optional. For example, the protocol contains provisions for scanline rendering, Phong shading, and CIE LUV color computations, but none of these features is guaranteed to be available in any particular PEX implementation. To use any optional feature, an application must query the server to determine if the feature is available. If not, the application must include code for a fallback strategy. This approach is very general, but it yields portable programs only by imposing a large burden on every application developer.

In contrast, OpenGL has a much larger set of core features that are universally supported.

5.3. Conformance Testing

To our knowledge, there are no comprehensive test suites available to validate PEX implementations. This makes it difficult to determine whether a particular implementation conforms to the specification. It also reduces confidence that an application that runs on one implementation of PEX will also run correctly on another.

A suite of conformance tests for OpenGL 1.0 ships with every release kit.

5.4. Window-System Dependency

PEX as it is defined today is very closely coupled to the X Window System. The semantics of X resources, drawables, atomicity/sequentiality requirements, and events are integral to the design of PEX.

However, there are other commercially-significant window systems. Microsoft Windows is an example. In order to accommodate these systems, OpenGL separates the parts of the interface that are window-system dependent (about ten routines under X11) from those that are window-system independent. Adding a small new set of window-system dependent routines allows the remainder of OpenGL to be used without change. This capability is being exploited to port OpenGL to Microsoft Windows/NT, with the expectation that Windows/NT platforms will offer a large new market for 3D graphics applications.

6. Business Issues

Although the primary focus of this paper is on the technical differences between PEX and OpenGL, it is also necessary to consider the most critical business issues: licensing, processes for changing the systems, market acceptance, and availability.

6.1. Licensing

The PEX specifications and sample implementation are available as part of the X11 distribution, at no additional fee. Vendors may release their commercial-quality implementations of PEX as bundled or layered products without royalty. To cover costs, the members of the X Consortium make direct financial contributions for support of X and PEX; for example, the PEX Sample Implementation cost contributors a total of roughly $750K.

Anyone can license OpenGL from Silicon Graphics at nominal prices. Manual pages are available by anonymous ftp at no cost. A Level 1 license (including the OpenGL specifications, manual pages, C bindings, GLU utilities, demo applications, compliance suite, and the right to redistribute an implementation) is $25K. A Level 2 license (all of Level 1 plus reference implementation, optimized portable sample implementation, X11 server extension, and porting documentation) is $100K. (This was chosen to be less than the fully-burdened cost of one engineer for a year.) A university license is available for $500. There is a royalty of $5 per revenue copy. These fees allow SGI to recover expenses for distribution and maintenance of OpenGL, in much the same way consortium membership fees allow the X Consortium to provide maintenance for PEX.

6.2. Mechanisms for Change

In the case of PEX, multiple independent entities (ANSI, ISO, the PHIGS-PLUS committee, and the X Consortium) are responsible for defining various parts of the APIs, rendering semantics, and wire protocol. This has led to some problems of timeliness and compatibility, particularly with respect to PEX support for PHIGS-PLUS. It has also contributed to the proliferation of PEX APIs. However, the process is open to essentially anyone.

A single organization (Silicon Graphics) defined the original specification for OpenGL, and a single entity (the OpenGL Architecture Review Board) is now responsible for all aspects of the API and wire protocol. The bylaws ensure that each of the six companies on the ARB has an equal vote on all API and governance issues. All decisions require a two-thirds majority vote. Compared to the organization used by PEX, we believe this structure will be more responsive to the needs of OpenGL users and more likely to maintain the technical integrity of the design as it evolves.

SGI has full-time engineering resources assigned to implement the decisions of the OpenGL ARB. This allows new code to be written and tested quickly, and thus brought to market earlier or with more user-requested features. The X Consortium funds two full-time positions for PEX; other development work is provided by sponsoring companies on a volunteer basis.

6.3. Vendor Support

Presently we know of five commercial-quality PEX products. Twenty companies are participating in the PEX Interoperability Center in Dallas, Texas, and therefore many should be expected to release commercial PEX products sometime in the next year.

Today SGI, IBM, Nth Graphics, and DuPont Pixel Systems are shipping products compatible with IRIS GL, the predecessor to OpenGL. Twenty personal computer and workstation companies have licensed OpenGL, and commercial implementations will be available sometime in the next year.

A considerable number of vendors have stated their intent to support both PEX and OpenGL.

The most compelling difference in vendor support is the endorsement of OpenGL by Microsoft and Intel. The OpenGL API will be bundled with future versions of Microsoft's Windows/NT environment, providing applications based on OpenGL with a personal computer market at least an order of magnitude larger than the workstation arena.

6.4. Software Applications Base

PEX per se has a small base of applications, because it has not been widely available until recently. The existing base of PHIGS code, much of which is found in CAD/CAM, government, and European markets, has a logical porting path to PEX. PEXlib is an alternate API for developers who find PHIGS inappropriate, but it requires substantial recoding because it is essentially a new API. There is some concern that ISVs will delay ports to PEXlib, because the PEX designers have chosen to make incompatible changes to the protocol for the next PEX release, and this could force incompatible changes to the PEXlib interface.

IRIS GL has the majority of applications in markets requiring high-performance 3D graphics — CAD/CAM (particularly finite element and structural analyses), animation, creative graphics, visual simulation, and scientific visualization. Since OpenGL is not backward-compatible with IRIS GL, existing IRIS GL applications must be ported. The majority of the current base of 1500 IRIS GL applications are expected to port to OpenGL, given its similarity to IRIS GL, the need to stay current with SGI and its product release schedule, and the wider market that is provided by OpenGL. According to a poll taken at the last SGI Developers Conference, 85% of the current IRIS GL ISVs have already decided to port to OpenGL. Because OpenGL evolved from IRIS GL, much of the conversion process can be automated; SGI demonstration programs were ported at the rate of 5000 lines per engineer-day. SGI has a vested interest in simplifying the move from IRIS GL to OpenGL, because OpenGL is the native interface for future SGI hardware.

6.5. Availability

The PEX 5.1 specification review process will be complete very shortly, and we can expect PEX 5.1 implementations to be available in the Fall of 1992.

The OpenGL 1.0 specification and sample implementation shipped on June 30, 1992. The first commercial implementations are expected in the first quarter of 1993.

The PEX design committee has chosen to make the next update of PEX incompatible with previous versions. The specification for PEX 6.0 will include some of the advanced rendering features of OpenGL, and should be complete near the middle of 1993. First implementations could appear as early as 1994.

7. Conclusions

We believe diversity in computer graphics interfaces can be desirable if it encourages real innovation in hardware, system software, and applications. However, if the computer graphics community feels it must standardize on one graphics system, OpenGL is a better choice than PEX:

OpenGL has a single well-defined API. There are two major competing APIs for PEX, and one of them (PHIGS) is not under the control of the organization defining PEX.

OpenGL has more rendering functionality than PEX. In addition, more OpenGL functionality is guaranteed to be present in all implementations than is the case for PEX.

When the entire computing system is analyzed, OpenGL has several important performance advantages over PEX. These advantages apply to both immediate-mode and structure-mode rendering.

OpenGL applications can be made portable with less effort than is needed for applications based on PEX, because OpenGL implementations are more consistent functionally and are subject to validation by a conformance test. OpenGL is also less dependent on the window system under which it is run.

Both OpenGL and PEX are available on generous terms, and both have a good deal of support in the industry. The X Consortium process for defining PEX is more democratic and participatory, but OpenGL's governance process will likely result in better technology coming to market more quickly.

Both systems offer porting paths from existing code bases: from stand-alone PHIGS to PHIGS on PEX, and from IRIS GL to OpenGL. However, the initial OpenGL application base will be substantially larger.

OpenGL will be a standard part of Microsoft Windows/NT. This provides OpenGL applications a vastly larger potential market than is available to PEX applications, and will make OpenGL more attractive to independent software developers.

We welcome your comments and suggestions.

References

Goldfeather, Jack, et. al., "Fast Constructive Solid Geometry Display in the Pixel-Powers Graphics System", Computer Graphics 20:4 (SIGGRAPH 1986 Conference Proceedings), August, 1986, pp. 107-116.

Haeberli, Paul, and Akeley, Kurt, "The Accumulation Buffer: Hardware Support for High Quality Rendering," Computer Graphics 24:4 (SIGGRAPH 1990 Conference Proceedings), August, 1990, pp. 309-318.

Mammen, Abraham, "Transparency and Antialiasing Algorithms Implemented with the Virtual Pixel Maps Technique," IEEE Computer Graphics and Applications, July, 1989.

Segal, Mark, and Akeley, Kurt, The OpenGL™ Graphics System: A Specification (Version 1.0), June 30, 1992, Silicon Graphics, Inc.

Womack, Paula (ed.), PEX Protocol Specification Version 5.1P - X Public Review Draft, May 11, 1992, Massachusetts Institute of Technology.

The Author

Allen Akin is an Engineering Manager in the OpenGL group at Silicon Graphics. He is currently integrating OpenGL with Microsoft Windows/NT. His graphics experience includes a year at MacroMedia, where he completed the MacroMind Three-D animation system for the Apple Macintosh, and five years at Digital Equipment Corporation, where he served as Engineering Manager for the first commercial implementation of PEX and as Graphics Software Architect for RISC workstations. He can be reached at the Internet address akin@sgi.com.