What 3D API for Java should I use and why?

Organizer
Dave Nadeau (San Diego Supercomputer Center)

Panelists
Brad Grantham (Silicon Graphics, Inc.)
Colin McCartney (Microsoft Corporation)
Mitra (Mitra Internet Consulting)
Henry Sowizral (Sun Microsystems Inc.)

The Java^{tm} slogan "Write once, run anywhere" has attracted a great deal of attention. Today, 3D graphics professionals are wondering if the slogan can apply to 3D graphics application development as well. This panel brings together representatives from groups developing 3D APIs for Java and challenges them to compare and contrast their products, discussing product features, differences, performance, portability, and limitations.

Brad Grantham, Silicon Graphics Inc.

Silicon Graphics provides a number of solutions to application developers and users desiring the Java language and high-performance interactive 3D graphics.

Application programmers may use Java bindings to OpenGL++, a scene graph toolkit for OpenGL^{tm}, to render and to provide interaction with 3D objects and scenes. OpenGL++ provides interaction features similar to the Inventor interactive toolkit and features derived from Silicon Graphics' experience with the high-performance Performer visual simulation toolkit. Developers have control over their 3D application from as high a level as "load and render this VRML 2.0 database" to as low a level as "draw these polygons with these colors, viewed from this location".

The Cosmo Worlds^{tm} VRML 2.0 authoring application in conjunction with the multi-platform Cosmo Player^{tm} VRML 2.0 navigator provides VRML 2.0 content authors with powerful modeling features. Embedded Java script nodes can directly control and manipulate the VRML 2.0 scene graph and access standard Java packages and language features. Java applets may control VRML 2.0 content through the External Authoring Interface (EAI).

Java bindings to OpenGL on both Silicon Graphics workstations and Windows 95/NT PCs can be used in a variety of domains where the overhead of frequent Java method calls is acceptable, including education, experimentation, and the prototype conversion of OpenGL code from other languages.

A developer's choice of 3D API for Java must be based on the developer's requirements and the available features.

Java bindings for OpenGL and OpenGL++ provide access to advanced features on workstations which may be important to a developer requiring, for example, high polygon counts, advanced texture capabilities, or machine-specific extensions. On the other hand, the use of these advanced features may preclude using a Windows NT/95 PC. Java3D, as a required part of Java, can be relied upon to be ubiquitous. Any operating system that provides the JavaMedia APIs will provide Java3D, and that may be important to users from games developers to teachers to intranet application developers.

The VRML 2.0 EAI and scripting nodes provide an efficient interface to controlling active VRML 2.0 content that exists on any VRML 2.0 browser. The EAI provides simple application control of databases. This may be important, for example, to customers providing kiosk applications which must direct user interaction with the database. Scripting nodes, on the other hand, provide intelligent content that can be readily referenced and included as URLs in larger databases, which may be important for vendors building multi-user VRML worlds.

Silicon Graphics brings a substantial amount of expertise in interactive high-performance 3D graphics to the table. SGI provides a range of solutions for Java developers from detailed OpenGL pipeline control to high-level VRML 2.0 navigation.

To state that any particular solution is the best, however, is as inappropriate as claiming a Lotus is always a better choice of vehicle than a Geo Metro. OpenGL++ and alternative 3D APIs all have advantages and disadvantages. A developer's choice of 3D Java API can only be made after consideration of the merits of each solution and the developer's requirements.

Colin McCartney, Microsoft Corp.

Microsoft's commitment to 3D graphics is founded on the principals of flexibility and choice for developers, giving them the ability to select the appropriate tools for the task in hand. Microsoft offers a full range of integrated 3D graphics options from the high to low end, enabling developers to create the full spectrum of 3D applications.

The company's Java^{tm} strategy embraces these same principals of choice and flexibility, enabling developers to create applications in the languages of their choice through features such as ActiveX/Beans integration and cross-language debugging, allowing them to leverage their existing experience and code bases while taking advantage of Java. To enhance the developer's Java experience, Microsoft is focused on providing the fastest, most secure, most robust Java VM, the richest, most fully featured suite of Java class libraries and the best tools for Java development.

Microsoft's DirectX J^{tm} suite of multimedia Java class libraries gives developers access to the power of DirectX's highly optimised native code and enables them to take advantage of hardware acceleration in their Java applications. DirectX J is Microsoft's multimedia solution for Java, and includes the following technologies for comprehensive, cross-platform 3D graphics in Java:

* Direct3D^{tm} J: Direct3D J gives Java developers both low-overhead to-the-metal access via its Immediate Mode API, and interoperable access to a higher-level suite of functionality, including a full world management system,

via its Retained Mode API.
* DirectAnimation^tm: DirectAnimation is a higher level media integration API that allows developers to integrate media compoments such as 3D graphics, video, 2D graphics and audio with ease.
* VRML 97: Following Microsoft's acquisition of DimensionX, developers can not only build tools and applications using Microsoft's Liquid Reality^tm-based Java VRML viewer; they can now take advantage of Java and VRML-specific 3D capabilities that will be integrated into the Direct3D J class libraries.

Mitra, Mitra Internet Consulting

When choosing an API to use for programming 3D and Java, the most important thing is to know what you are trying to achieve. Each of the API's presented in this panel has their strengths and weaknesses.

As I see it, the biggest distinction between the API's is what I call "Who's on top".

* If you see yourself as writing a 3D application (for example a game), then one of the lower level API's (Direct3D, Java3D, or OpenGL++) might make sense. At the time of this writing, Direct3D has only just added Java bindings, and the specifications for Java3D and OpenGL++ are not available, so comparisom is not really useful.

* If, on the other hand, you see yourself as modelling a world of active, independent, or inter-dependant, objects, then you should probably be building the world in VRML, and adding behavior to the VRML objects via VRML2.0's Java scripting API (not to be confused with JavaScript scripting).

* Alternatively, if you are building a Web page, that needs a 3D image, for example to graph some results, then using VRML's Java or Javascript External Application Interface (EAI) probably makes sense.

The biggest advantage that the VRML/Java scripting API gives you is its simplicity. Essentially your Java program is mostly manipulating the fields and nodes of a standard VRML 2.0 scene graph. In Java3D or OpenGL++ I understand that you will have intimate control of the lighting, and other rendering characteristics. In VRML's API's you don't worry about these things, leaving them to the browser. This allows the behavior author to concentrate on what they do best, writing programs, leaving a modeler to do what they do best, using an authoring tool to set up a visual effect.

Of course, in the VRML API, as in anything else the author has full access to everything you expect in Java, for example to AWT classes, or to Threads, or the network.

Henry Sowizral, Sun Microsystems Inc.

The Java 3D Graphics API is a scene graph-based and Java-based API designed with graphics performance in mind. Its inheritance model removes as much graphic state information from the interior of a scene graph as possible and moves that information to the scene-graph's leaves. By placing the state information at the leaves, Java 3D can use scene graphs as carriers of graphics information. It need not treat them as a computational structure. Efficient processing of a Java 3D scene graph will require that a Java 3D renderer build and use ancillary data structures.

The Java 3D API provides programmers with three rendering modes: immediate, retained, and compiled retained. Programmers can use any one, two, or even all three modes at the same time to render a scene. The immediate mode defines a higher level abstraction of low-level functionality by providing programmers with a means to change graphic attributes and to render sets of points, lines, or triangles directly to a canvas. The retained mode allows programmers to specify a scene-graph and to inform the Java 3D renderer that it should render that scene graph. The compiled retained mode allows programmers to identify a scene graph fragment as a candidate for compilation. That fragment may have mutable components identified as such by the programmer. Such information allows a Java 3D compiler to analyze a scene-graph fragment and replace it with a opaque representation subject to the programmer specified mutability constraints.

The API includes a number of unique features including a more complete view model (one that supports more exotic viewing environments such a immersive and fish-tank VR applications), an extended input model that permits access to real-time inputs such as six-degree-of-freedom trackers and joysticks, and a behavior execution and execution-culling model that uses Java as its base language and the registration of "wakeup criteria" to permit culling. As a runtime API, Java 3D programs must construct their scene-graphs programatically. The API does not specify an external file format for scene graphs. It does however, make it quite straightforward to build object loaders and scene-graph loaders. As an example, we rapidly built a wavefront ".obj" format loader and a VRML 1.0 loader. In future, we anticipate the availability of many other loaders including a VRML 2.0 loader. In the case of VRML 2.0, the Java 3D API specifies a uniform technique for processing VRML 2.0 routes and fields. That specification relies on a Java 3D behavior that triggers at each frame and propagates any changes associated with a route. Applications that do not use VRML 2.0 routes pay no cost for runtime support of routes and fields.

The Java 3D Graphics API was initially designed by a small group of partner companies. Sun distributed the design resulting from that effort to its Java licensees as a 0.9 specification of the Java 3D API. Interested Java licensees provided feedback that resulted in changes to the 0.9 specification. Sun incorporated those updates into the specification and then distributed the updated 0.95 specification of the Java 3D API to the general public for feedback. Comments from the public will change the 0.95 specification and culminate in the 1.0 specification of the Java 3D API.