OpenKODE

An Open Standard for Mobile Application Portability

KHRONOS GROUP

February 2008

Neil Trevett, OpenKODE Working Group Chair

The new OpenKODE™ specification from the Khronos™ Group is a royalty-free, cross-platform open standard that bundles a set of native APIs to provide increased source portability for rich media and graphics applications. This document discusses the ecosystem of Khronos standards, motivations behind the development of OpenKODE standard within that ecosystem and the high-level architecture of OpenKODE 1.0 specification.

More information on Khronos and any of the APIs mentioned here can be found at **www.khronos.org**

1. Mobile Application Platforms and Fragmentation

Mobile devices are evolving into increasingly sophisticated, general purpose computers. Many handsets support application development platforms, such as Brew, Symbian UIQ, Android, LIMO, ALP, Qtopia or WIPI that provide programming resources for native and Java applications. Today's phones are expected to run a growing range of software such as internet browsers, navigation packages, games and music/video players. Application platforms are built over lower-level kernel operating systems (OS) such as Rex, Linux or compact real-time OS's such as Nucleus. Software running on the device accesses platform resources through a set of application programming interfaces or APIs.

Mobile developers typically develop for multiple platforms to maximize their available market. This can be a difficult and time-consuming task as multiple platforms use different API calls for common OS operations and accessing OS resources such as accessing memory and files. These OS differences are a major source of application source-level fragmentation.

Also, many platforms are still developing their support for media acceleration, and even if open standard APIs are used, such as OpenGL® ES for 3D graphics, it is often not defined how multiple media APIs interoperate. For example, it can be difficult to mix 2D and 3D rendering with full performance, or to process video through a 3D graphics accelerator, or to use a high-quality accelerated vector graphics UI over a 3D game or video stream - and almost certainly impossible to achieve such mixed-media acceleration in a platform-portable way. This is a significant barrier to the development of rich-media applications and user interfaces that need to freely mix media types to create compelling end-user experiences.

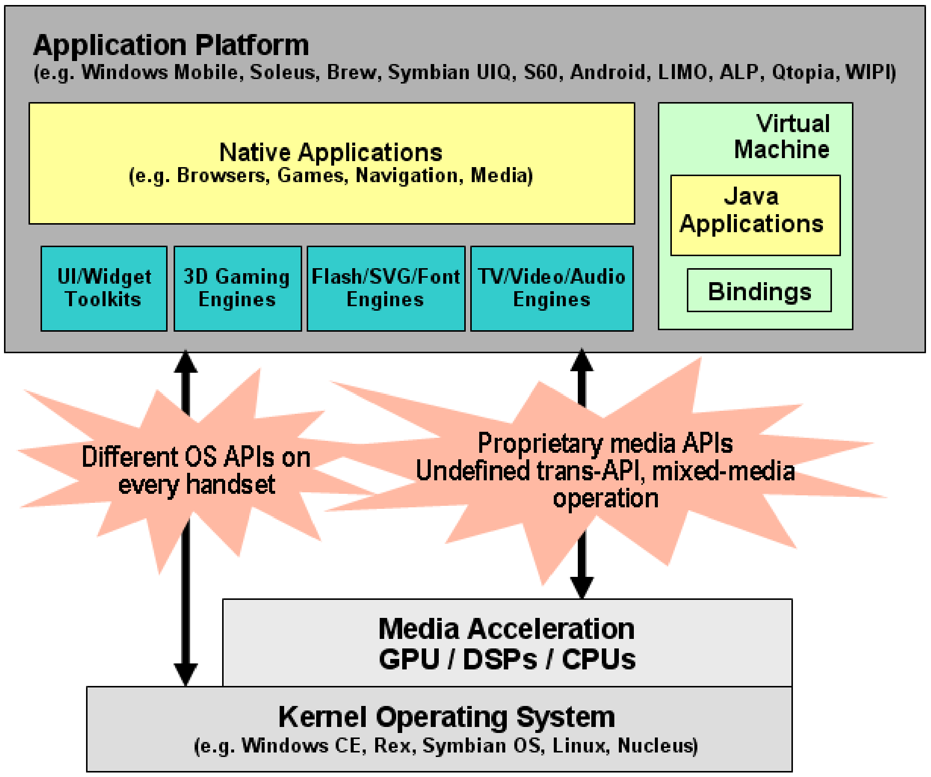

Application Platform
(e.g. Windows Mobile, Soleus, Brew, Symbian UIQ, S60, Android, LIMO, ALP, Qtopia, WIPI)
Native Applications
(e.g. Browsers, Games, Navigation, Media)
Virtual Machine
Java Applications
Bindings
UI/Widget Toolkits
3D Gaming Engines
Flash/SVG/Font Engines
TV/Video/Audio Engines
Different OS APIs on every handset
Proprietary media APIs
Undefined trans-API, mixed-media operation
Media Acceleration
GPU / DSPs / CPUs
Kernel Operating System
(e.g. Windows CE, Rex, Symbian OS, Linux, Nucleus)

Figure 1: Multiple mobile platforms result in application fragmentation

As the mobile phone market continues to expand, the diversity of handsets and application platforms has become a critical problem for the software development community. At the time of writing there are close to 1,000 different handsets shipping in the market – many with unique software environments.

Indeed, according to a recent report from Informa, the mobile games developer community typically spends 2/3 of mobile development costs to overcome platform fragmentation.

> The typical rule of thumb for a developer is that one-third of the total cost of development will go towards porting, one-third towards development of the game itself and one-third to quality assurance (QA). Since most of the QA costs are related to porting, the costs associated with this issue equals two-thirds of the total cost of game development.
> *Informa, Mobile Entertainment: The Future of Wireless Content 4th Edition*

OpenKODE aims to reduce mobile platform fragmentation by increasing source code portability of applications that access operating system resources and sophisticated media acceleration.

2. Khronos and the Family of Mobile Media APIs

2.1 Khronos Membership

The Khronos Group is a member-funded, international industry consortium focused on the creation of royalty-free open standards to enable the authoring and acceleration of dynamic media on a wide variety of platforms and devices. All Khronos members are able to contribute to the development of Khronos specifications, are empowered to vote at various stages before public deployment, and are able to accelerate the delivery of their cutting-edge media platforms and applications through early access to specification drafts and conformance tests.

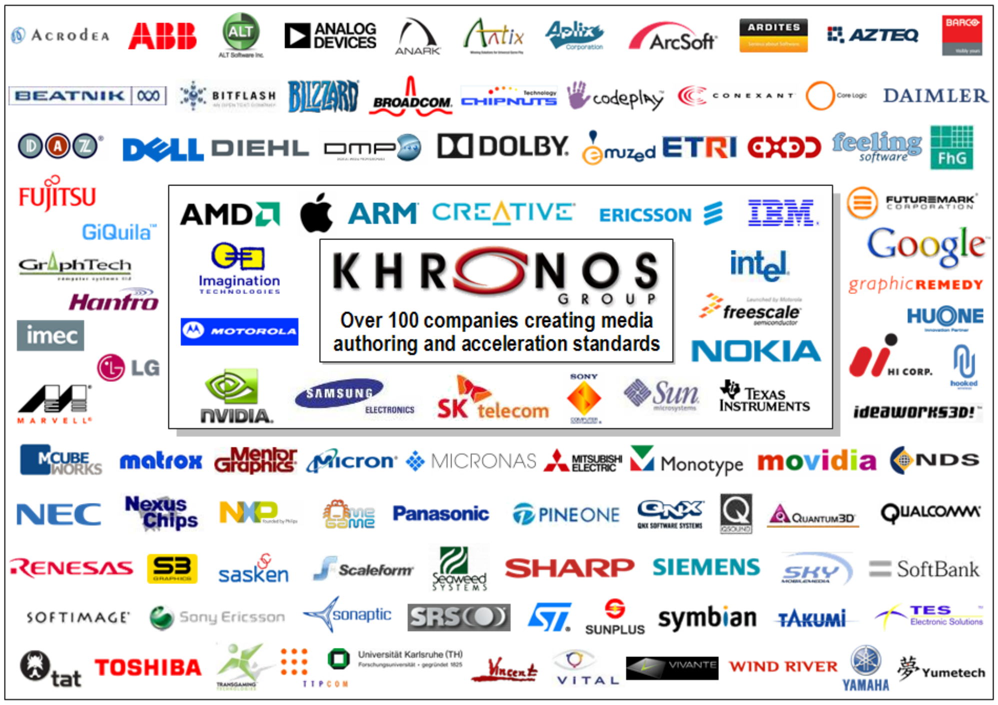
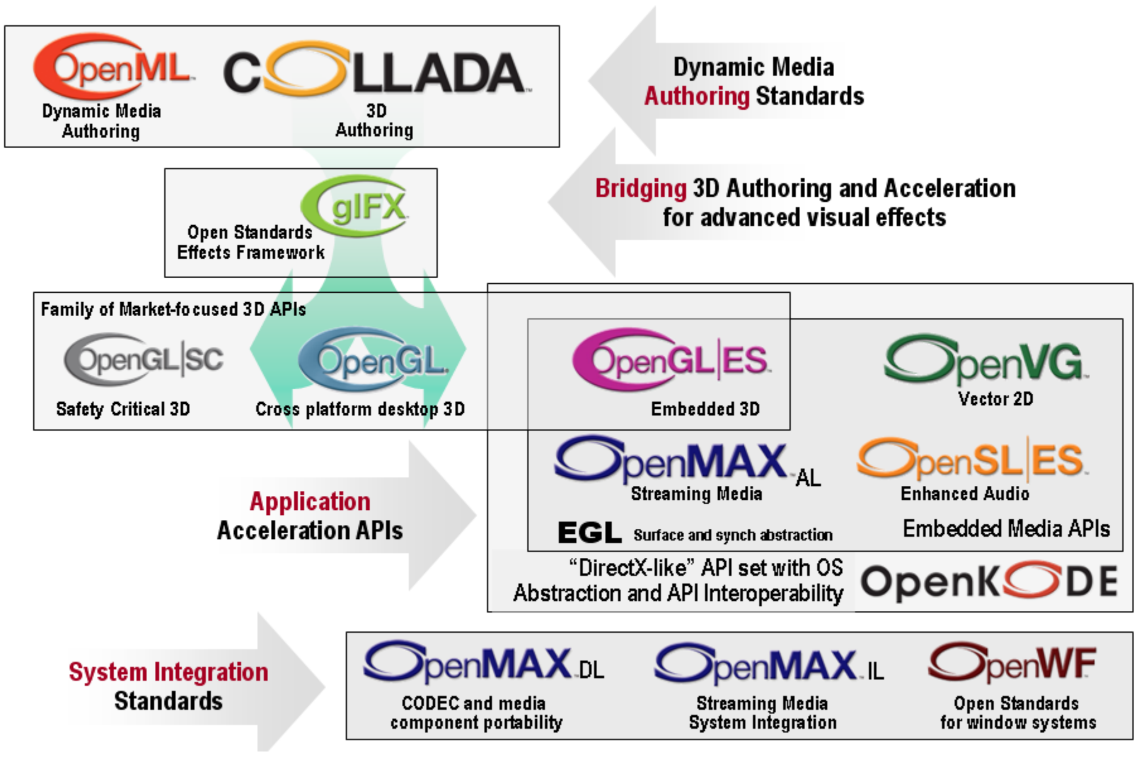


Figure 2: Khronos Membership – the center companies have a seat on the Board of Promoters

Khronos has significant industry participation with over one hundred members, including handset OEMS, carriers, silicon vendors, OS vendors, middleware vendors and application developers. The Board of Promoters controls strategy and budget, but all members have an equal vote in standardization activities. Any company is welcome to join Khronos.

2.2 The Khronos Ecosystem for Media Authoring and Acceleration

Khronos standards include both acceleration APIs and authoring standards to encourage sophisticated tools pipelines for authoring advanced content. As of February 2008, Khronos has fourteen API specifications being developed and maintained to form a complete media authoring and acceleration ecosystem for desktop, embedded and mobile platforms.



OpenML
Dynamic Media Authoring
COLLADA
3D Authoring
Dynamic Media Authoring Standards
glFX
Open Standards Effects Framework
Bridging 3D Authoring and Acceleration for advanced visual effects
Family of Market-focused 3D APIs
OpenGL SC
Safety Critical 3D
OpenGL
Cross platform desktop 3D
OpenGL ES
Embedded 3D
OpenVG
Vector 2D
OpenMAX AL
Streaming Media
OpenSL ES
Enhanced Audio
EGL Surface and synch abstraction
Embedded Media APIs
Application Acceleration APIs
"DirectX-like" API set with OS Abstraction and API Interoperability
OpenKODE
System Integration Standards
OpenMAX DL
CODEC and media component portability
OpenMAX IL
Streaming Media System Integration
OpenWF
Open Standards for window systems

Figure 3: The Family of Khronos Standards

The mobile industry has a particular focus on the family of mobile media acceleration APIs: OpenGL ES for 3D Graphics, OpenVG™ for Bezier-based vector 2D graphics, OpenMAX™ for streaming video and audio acceleration and OpenSL ES™ for enhanced audio. These mobile APIs are being increasingly inter-connected through the EGL™ standard for surface and synchronization abstraction.

The Khronos family of media APIs is in active development. The OpenGL ES specification was first released in mid-2004 and is now widely deployed becoming the de facto standard for embedded and mobile 3D. The OpenVG and OpenMAX IL specifications were first released in 2005 and 2006, respectively, and are now starting to be widely implemented. OpenSL ES and OpenMAX AL have been just released in provisional form, with final specifications and initial implementations expected in 2008.

The strong commercial momentum of the Khronos standards is largely due to the significant investment by the silicon community to define and implement a standardized set of interfaces to expose state-of-the-art media acceleration to the software community. Whichever higher-level application development platform is on a device, operating systems, middleware and applications can tap into the state-of-the-art silicon capabilities through these foundation–level, non-proprietary APIs.

Khronos has channeled hundreds of man years of design and development by the industry's leading media acceleration experts into this evolving ecosystem; and yet all these standards are made available, royalty-free, to the industry to enable and encourage market growth for the benefit of Khronos members and the industry as a whole.

3. OpenKODE

3.1 A Cross-Platform Set of Native APIs to Reduce Source Fragmentation

Any product can utilize one or more Khronos media acceleration APIs, and an increasing number of products are now shipping that use these standards. OpenKODE takes an additional step to reduce fragmentation by defining a standardized *set* of native media acceleration APIs – and defining precisely how those APIs work together. OpenKODE even defines conformance tests to ensure that this trans-API functionality is implemented correctly.

OpenKODE also includes the new OpenKODE Core API that abstracts operating system resources to minimize source changes when porting applications between Linux, Rex/Brew, Symbian, Windows and RTOS-based platforms. OpenKODE Core will be familiar to POSIX and C programmers and is a small and light abstraction layer that provides access to low-level operating system functionality. An OpenKODE Core implementation is typically less than 100KB in size and adds very minimal performance overhead. OpenKODE Core provides advanced functionality, such as multi-threading under an event-driven architecture, while being carefully designed to provide real-world portability to a wide variety of mobile platforms.

Bundling a set of APIs is a similar approach to the successful DirectX on desktop Windows that defines Direct3D for 3D, DirectShow for video, DirectInput for input etc. - to provide the functionality required by rich media applications. OpenKODE takes a similar approach, but unlike DirectX, OpenKODE is an open, cross-platform standard that has been carefully designed to be implementable on a wide range of operating systems from Symbian, Windows and Linux to Rex and smaller RTOS kernels.

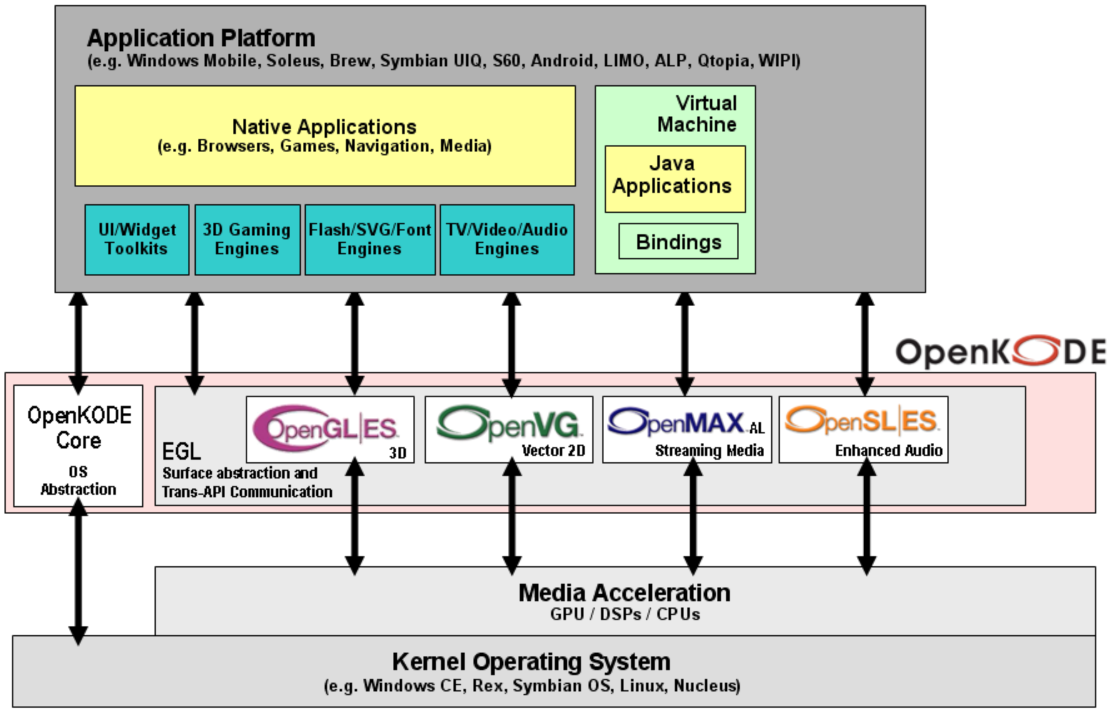


Figure 4: The OpenKODE set of native APIs

Through being designed to be deployable on almost any handset, the tested OpenKODE set of APIs can provide a reliable and cross-platform set of functionality for rich-media applications, significantly reducing application source fragmentation.

3.2 OpenKODE Implementations

OpenKODE Core must always be present in any OpenKODE implementation, hence the name of this new API.

OpenKODE 1.0 brings together the OpenGL ES 1.1 and OpenVG 1.0 media APIs with EGL 1.3 plus defined EGL extensions to provide state-of-the-art acceleration for mixed vector 2D and 3D graphics. EGL provides common rendering surfaces that both OpenGL ES and OpenVG can manipulate directly.

Upcoming versions of OpenKODE will use future versions of EGL to integrate synchronization and data processing of streaming media using the OpenSL ES and OpenMAX media APIs to provide accelerated video and audio that is fully integrated with graphics processing. OpenKODE will also integrate OpenGL ES 2.0 as soon as the conformance tests for that version for that API are released. Extensions will be available to integrate these APIs in advance of the next full version of the OpenKODE specification.

OpenKODE 1.0 enables an implementer to select which media APIs are appropriate for a particular platform. The OpenKODE specification precisely defines the level of EGL functionality that is mandated to provide trans-API operation of the selected APIs.

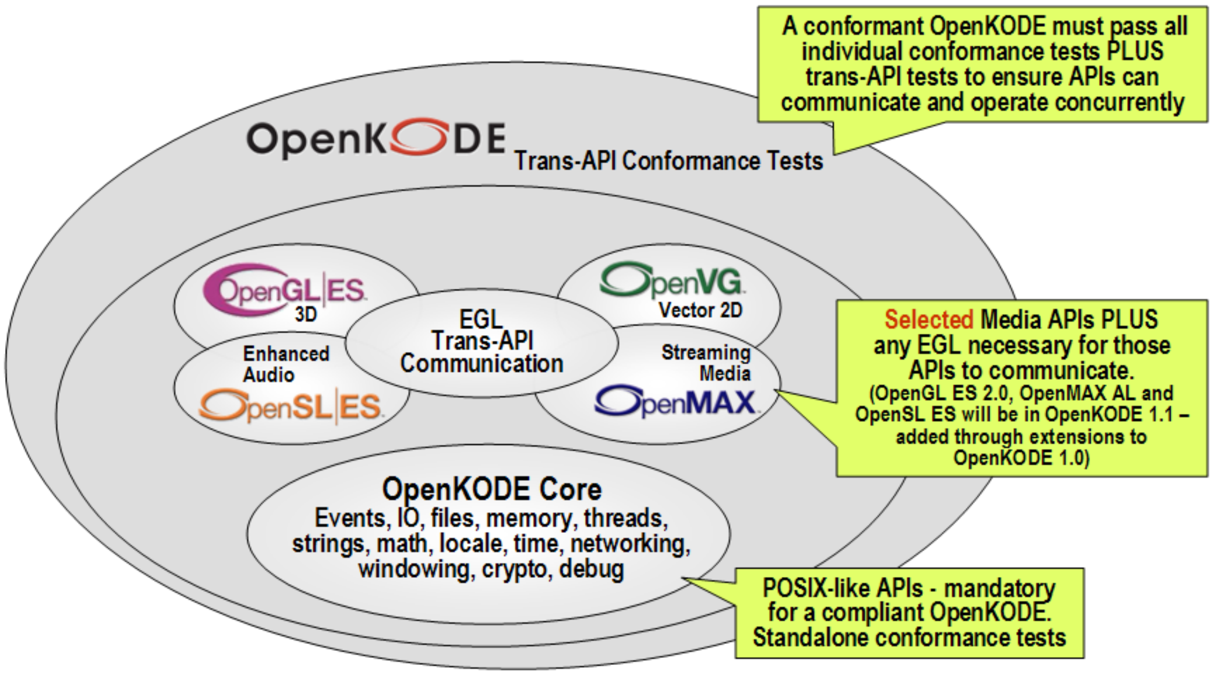

A conformant OpenKODE must pass all individual conformance tests PLUS trans-API tests to ensure APIs can communicate and operate concurrently
OpenKODE
Trans-API Conformance Tests
OpenGL ES
3D
OpenVG
Vector 2D
EGL Trans-API Communication
Enhanced Audio
OpenSL ES
Streaming Media
OpenMAX
Selected Media APIs PLUS any EGL necessary for those APIs to communicate. (OpenGL ES 2.0, OpenMAX AL and OpenSL ES will be in OpenKODE 1.1 – added through extensions to OpenKODE 1.0)
OpenKODE Core
Events, IO, files, memory, threads, strings, math, locale, time, networking, windowing, crypto, debug
POSIX-like APIs - mandatory for a compliant OpenKODE. Standalone conformance tests

Figure 5: Components within an OpenKODE implementation

All APIs included in a conformant OpenKODE implementation must pass the individual conformance tests defined for that API – including OpenKODE Core. In addition, OpenKODE defines a set of trans-API conformance tests to ensure that the media APIs and EGL correctly provide the specified mixed media functionality. Ensuring that the mixed-media functionality in OpenKODE is reliably implemented by multiple vendors is an important factor in genuinely reducing platform fragmentation for application developers.

3.3 OpenKODE Deployment

OpenKODE does not mandate whether the set of APIs is shipped from a single vendor or multiple vendors, as long as the final complete implementation passes all conformance tests.

The selected set of media APIs, together with associated EGL functionality, will often be provided by the silicon vendor. On the other hand, OpenKODE Core has been carefully defined to have no direct implementation dependencies on the media APIs, making it straightforward for the OpenKODE Core API to be deployed onto a platform independently from the media APIs.

Many middleware vendors will include OpenKODE Core implementations within their products to provide portability even if OpenKODE Core is not made available by the platform vendor, enabling rapid industry deployment of this new standard. Over time it is expected that platform vendors will provide OpenKODE Core as part of a complete OpenKODE implementation to enable easy porting of OpenKODE applications to their products.

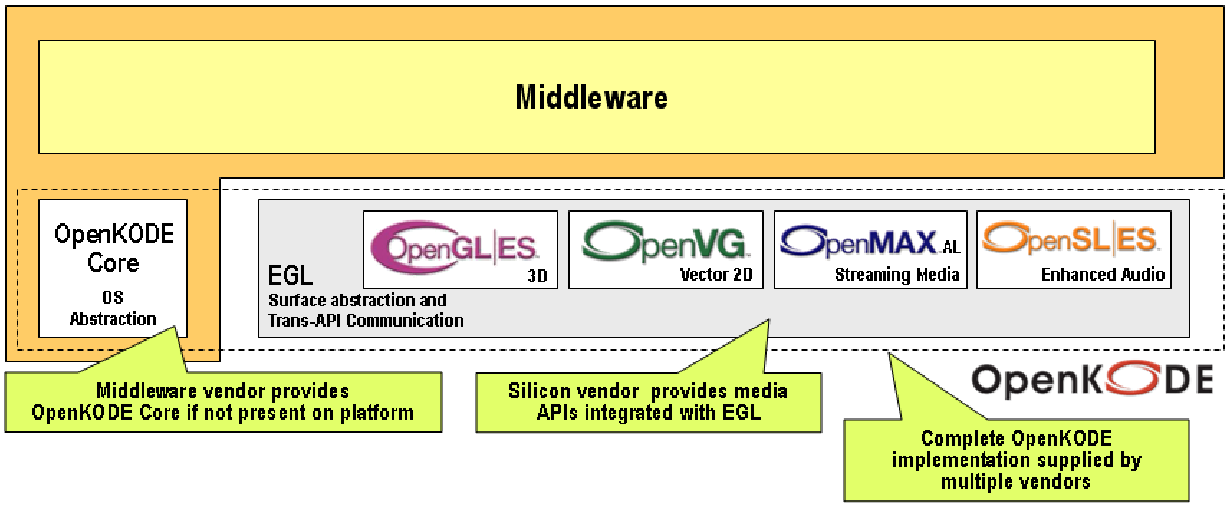

Middleware
OpenKODE Core
OS Abstraction
EGL
Surface abstraction and Trans-API Communication
OpenGL ES
3D
OpenVG
Vector 2D
OpenMAX AL
Streaming Media
OpenSL ES
Enhanced Audio
OpenKODE
Middleware vendor provides OpenKODE Core if not present on platform
Silicon vendor provides media APIs integrated with EGL
Complete OpenKODE implementation supplied by multiple vendors

Figure 6: A complete OpenKODE Implementation can ship from multiple vendors

3.4 OpenKODE as a Foundation-Level Media-stack

The set of OpenKODE APIs provides a robust set of functionality that can provide acceleration for a wide range of applications, middleware and engines, regardless of the type of media acceleration needed, including: 3D games, Flash and SVG players, imaging and camera software, TV and video applications, widget toolkits and audio/video players. OpenKODE provides a fully capable media stack architecture that can form the low-level media processing foundation for sophisticated software platforms such as Brew, Symbian UIQ, S60, Android, LIMO, ALP, Qtopia and WIPI. Through the relevant JSRs, OpenKODE can provide acceleration for Java as well as native applications.

Additionally, OpenKODE has been designed to enable multi-process capable implementations, so that multiple applications can be accelerated even when they are executing simultaneously. This is a vital evolution in media architecture design as it enables handsets to provide a media-rich, multi-tasking environment. Significantly, a multi-process OpenKODE media stack also enables the full power of media acceleration to be used by the window system or launcher applications - in parallel to applications – opening the possibility for accelerated, composition-based user interfaces.

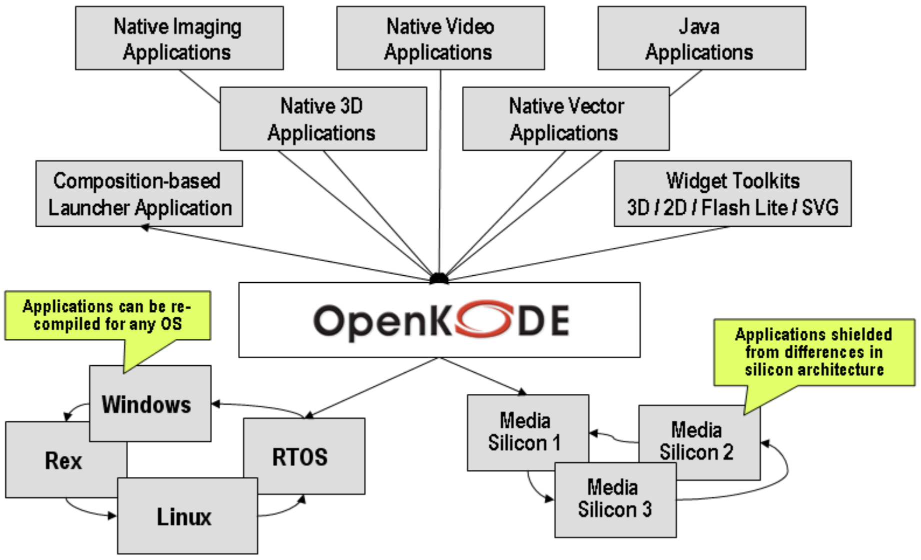

Native Imaging Applications
Native Video Applications
Java Applications
Native 3D Applications
Native Vector Applications
Composition-based Launcher Application
Widget Toolkits 3D / 2D / Flash Lite / SVG
OpenKODE
Applications can be re-compiled for any OS
Applications shielded from differences in silicon architecture
Windows
Rex
RTOS
Linux
Media Silicon 1
Media Silicon 2
Media Silicon 3

Figure 7: OpenKODE can accelerate all media applications on a device

Also, being able to reliably mix-and-match media types within a single application will catalyze the development of new classes of mixed rich-media applications and user interfaces, for example:

- Accelerated Flash UI over 3D and video applications;
- Advanced user interfaces with accelerated 3D inter-process navigation and transitions using fully accelerated scalable vector text;
- Augmented reality applications with real-time accelerated image processing to control real-time 3D overlaying a video stream;
- Advanced navigation applications that combine 3D terrains and models with high-quality vector maps and fonts;
- Avatar telephony that processes an audio stream to generate a lip-synched 3D model.

3.5 OpenKODE in the Mobile Ecosystem

OpenKODE provides the low-level functionality and access to acceleration needed on all media-capable mobile devices. As an open standard it enables key players in the mobile ecosystem communicate requirements and functionality more efficiently and effectively – growing the market opportunity for the entire industry.

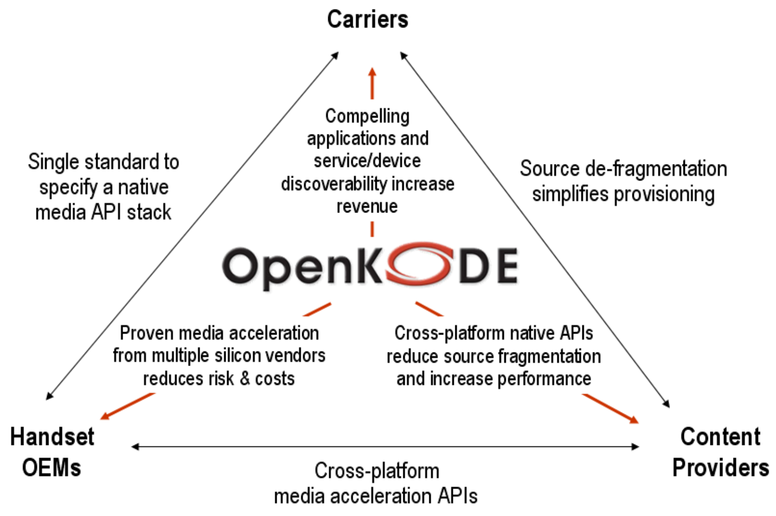

Carriers
Compelling applications and service/device discoverability increase revenue
Single standard to specify a native media API stack
Source de-fragmentation simplifies provisioning
OpenKODE
Proven media acceleration from multiple silicon vendors reduces risk & costs
Cross-platform native APIs reduce source fragmentation and increase performance
Handset OEMs
Content Providers
Cross-platform media acceleration APIs

Figure 8: OpenKODE can be valuable to all parties in the mobile ecosystem

4. APIs Included in OpenKODE

4.1 OpenKODE Core

OpenKODE Core provides source-level abstraction of common operating system services in an event-driven environment. OpenKODE Core is based on POSIX and also draws on the C89, C99 standards and has been carefully designed to be implementable over any common mobile OS currently in the market. OpenKODE Core also adds an event system and an extensible IO framework. The main areas of OpenKODE Core functionality include:

- Thread creation and synchronization based on a subset of POSIX threads, plus unnamed semaphores;
- An event system which abstracts the event system of the platform's OS - e.g. supplying quit, pause and resume, window resize, input change, timer, and socket ready to read or write events. An OpenKODE application may be written as either loop-in-application, where it contains the top-level loop to process an event in each iteration, or a loop-in-framework application where the framework calls an event handler for each event;
- Utility functions including conversions from string to number and vice versa, random number generation, memory allocation, memory and string copying, comparison and scanning, and assertions and logging;
- A math library including 32-bit floating point operations and analogs of many of the C standard math library functions. The math library can be extended to support Vectors, Matrices and Quaternions;
- Analogs of C time functions, as well as OpenKODE-specific functions for more accurate timekeeping, and for timers which generate events;
- An abstraction of platform’s file system into a virtual file system so that applications access only certain well-known locations (such as “the files that came with the application”) to be written portably. The file functions are analogs of familiar C and POSIX functions;
- A networking API similar to BSD/POSIX sockets, but with API semantics to use the event system to notify when a socket is ready to send to or receive from;
- Input/output devices (such as buttons) and outputs (such as vibrate) in an extensible framework, while specifying a small range of inputs and outputs that are likely to be present, such as game keys. This input API can be extended to include support for advanced IO such as accelerometers and touch screens with multi-touch screens with gesture processing;
- Windowing abstraction including support for multiple non-full-screen windows with simple manipulation of such windows including resizing and maximizing.

There are already multiple implementations of OpenKODE Core shipping on multiple operating systems, including Windows and Mac OS to enable applications to be developed on desktop machines and simply recompiled on the target mobile hardware. There are independent projects underway to create open source versions of OpenKODE Core.

4.2 EGL

EGL is an interface between Khronos rendering APIs such as OpenGL ES or OpenVG and the underlying native platform window system. It handles graphics context management, surface/buffer binding, and rendering synchronization to enable high-performance, accelerated, mixed-mode 2D and 3D rendering using Khronos APIs.

EGL provides mechanisms for: creating rendering surfaces into which client APIs such as OpenGL ES and OpenVG can draw, creating graphics contexts, and synchronizing drawing operations. This enables seamless, high-performance, accelerated, mixed-mode 2D and 3D rendering using both OpenGL ES and OpenVG.

OpenKODE requires the lock surface extension to EGL which provides limited direct blitting to the screen which is required for software rendering applications.

EGL 1.3 was released in December 2006. EGL 1.4 will enhance support for OpenVG and software renderers by integrating the lock surface extension and is due in spring 2008.

EGL 1.5 will integrate OpenMAX to enable video streams to be manipulated and processed by the graphics APIs, and for graphics output to be fed into a media stream, while eliminating unnecessary copies for power, memory and processing efficiency. EGL 1.5 is expected by mid-2008. The details of OpenMAX's integration into EGL are still be developed by Khronos so the figure below is conceptual only.

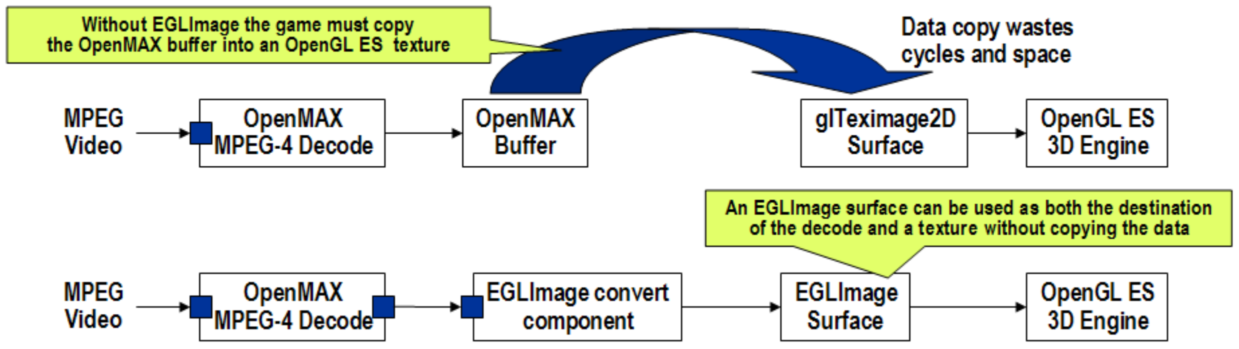

Without EGLImage the game must copy the OpenMAX buffer into an OpenGL ES texture
Data copy wastes cycles and space
MPEG Video
OpenMAX MPEG-4 Decode
OpenMAX Buffer
glTeximage2D Surface
OpenGL ES 3D Engine
An EGLImage surface can be used as both the destination of the decode and a texture without copying the data
MPEG Video
OpenMAX MPEG-4 Decode
EGLImage convert component
EGLImage Surface
OpenGL ES 3D Engine

Figure 9: EGL 1.5 will enable efficient integration of video and graphics processing

4.3 OpenGL ES

OpenGL ES is a royalty-free, cross-platform API for full-function 2D and 3D graphics on embedded systems - including gaming consoles, mobile phones, appliances and vehicles. It consists of well-defined subsets of desktop OpenGL, creating a flexible and powerful low-level interface between software and graphics accelerator silicon. OpenGL ES 1.1 defines a fixed function 3D graphics pipeline which is widely adopted across multiple industries. OpenGL ES 2.0 complements OpenGL ES 1.1 by enabling fully programmable 3D graphics.

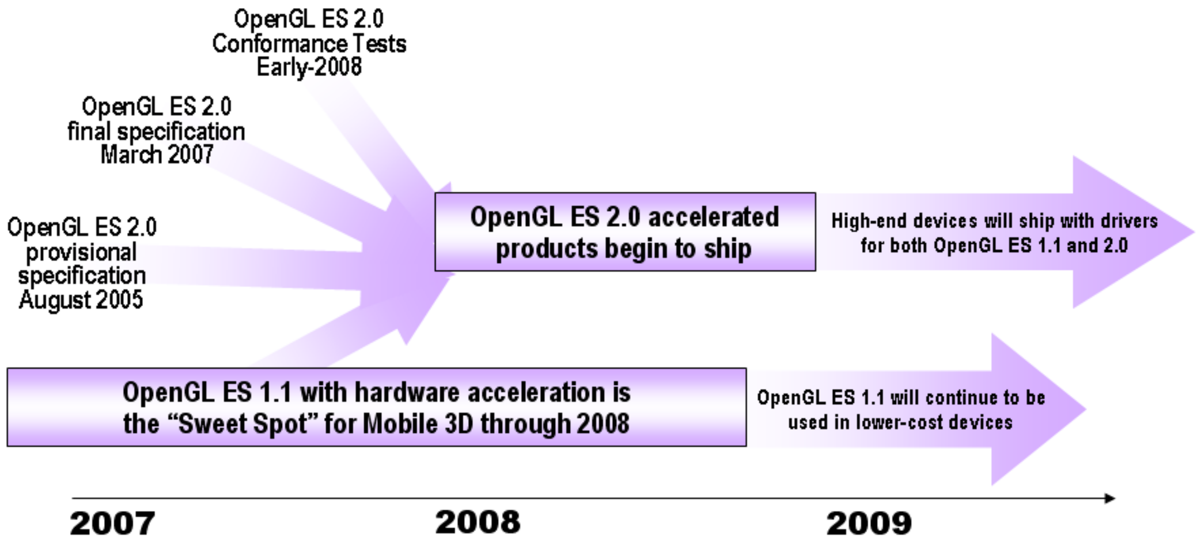

OpenGL ES 2.0 Conformance Tests Early-2008
OpenGL ES 2.0 final specification March 2007
OpenGL ES 2.0 provisional specification August 2005
OpenGL ES 2.0 accelerated products begin to ship
High-end devices will ship with drivers for both OpenGL ES 1.1 and 2.0
OpenGL ES 1.1 with hardware acceleration is the "Sweet Spot" for Mobile 3D through 2008
OpenGL ES 1.1 will continue to be used in lower-cost devices
2007
2008
2009

Figure 10: Industry adoption of OpenGL ES 1.1 and OpenGL ES 2.0

Both OpenGL ES 1.1 and OpenGL ES 2.0 are publicly released. Conformance Tests for OpenGL ES 1.1 are released. Conformance Tests for OpenGL ES 2.0 will ship in the first half of 2008. There are open source implementations of OpenGL ES 1.1.

OpenGL ES 2.0 introduces GLSL ES, a close derivative of the desktop OpenGL shading language, to bring full shader programmability to mobile devices for significantly enhanced realism and graphics programming flexibility. OpenGL ES devices will begin shipping at the beginning of 2008. OpenGL ES 1.1 devices will typically be less expensive than OpenGL ES 2.0 devices, at least until OpenGL ES 2.0 volume increases, and so it is likely that OpenGL ES 1.1 devices will continue to ship for some time. However, almost every OpenGL ES 2.0 device will also ship with OpenGL ES 1.1 drivers to ensure older content continues to run without modification.

As well as providing comprehensive 3D functionality, OpenGL ES is a sophisticated 2D API with sub-pixel accuracy, anti-aliasing and comprehensive alpha-blending. Consequently OpenGL ES is an excellent 2D composition API that is also capable of supporting 3D user interface effects and transitions.

4.4 OpenVG

OpenVG is a royalty-free, cross-platform API that provides a native hardware acceleration interface for Bezier vector graphics libraries such as Flash and SVG. OpenVG is targeted primarily at handheld devices that require portable acceleration of high-quality vector graphics for compelling user interfaces and text on small screen devices - while enabling hardware acceleration to provide fluidly interactive performance at very low power levels.

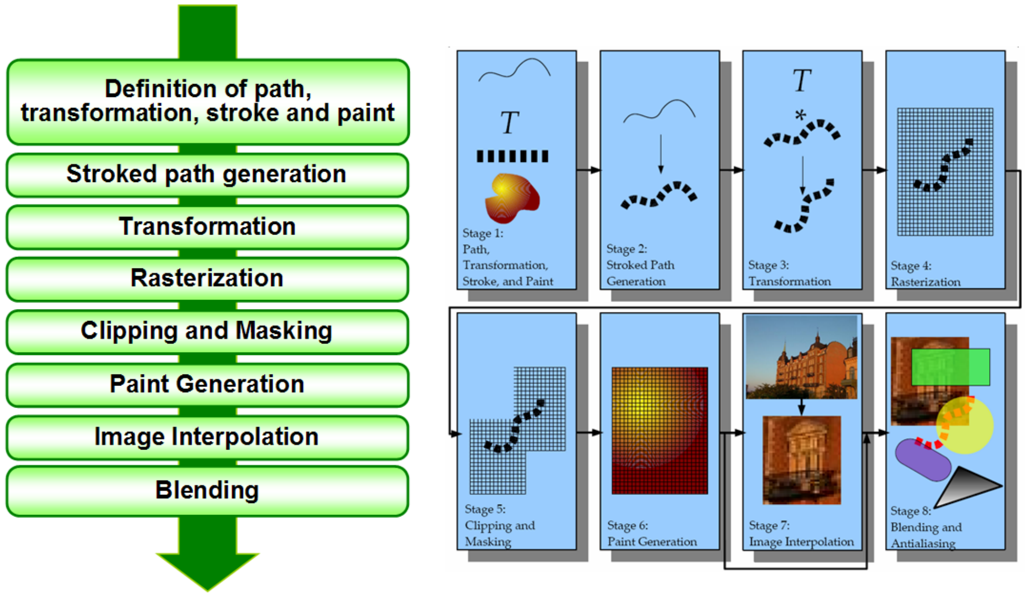

Definition of path, transformation, stroke and paint
Stroked path generation
Transformation
Rasterization
Clipping and Masking
Paint Generation
Image Interpolation
Blending
Stage 1: Path, Transformation, Stroke, and Paint
Stage 2: Stroked Path Generation
Stage 3: Transformation
Stage 4: Rasterization
Stage 5: Clipping and Masking
Stage 6: Paint Generation
Stage 7: Image Interpolation
Stage 8: Blending and Antialiasing

Figure 11: The OpenVG Rendering Pipeline

OpenVG 1.0 was targeted primarily at accelerating SVG Tiny 1.2. OpenVG 1.1 will add seamless acceleration for Flash Lite, enabling OpenVG to provide efficient acceleration for the majority of vector-base user interfaces and content.

Additionally, OpenVG 1.1 adds specific support for glyph acceleration and can significantly increase the interactivity and quality of sophisticated multi-lingual font engines, useful for state-of-the-art mobile web-browsers with dynamic zooming and panning capability. OpenVG is already being adopted by open source widget and browser engines such as GTK+ and Cairo which is used by WebKit.

OpenVG 1.0 was released in August 2005. OpenVG 1.1 is expected to be released in the first quarter of 2008 with OpenVG 1.1 parts shipping almost immediately after.

4.5 OpenMAX

OpenMAX is a multi-layered standard that provides comprehensive streaming media codec and application portability by enabling accelerated multimedia components to be developed, integrated and programmed across multiple operating systems and silicon platforms.

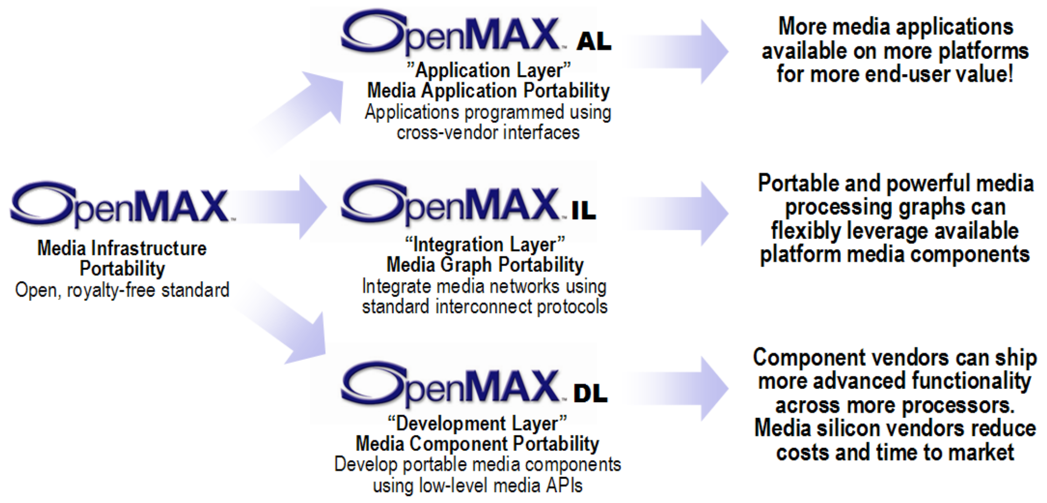

OpenMAX AL
"Application Layer"
Media Application Portability
Applications programmed using cross-vendor interfaces
More media applications available on more platforms for more end-user value!
OpenMAX
Media Infrastructure Portability
Open, royalty-free standard
OpenMAX IL
"Integration Layer"
Media Graph Portability
Integrate media networks using standard interconnect protocols
Portable and powerful media processing graphs can flexibly leverage available platform media components
OpenMAX DL
"Development Layer"
Media Component Portability
Develop portable media components using low-level media APIs
Component vendors can ship more advanced functionality across more processors. Media silicon vendors reduce costs and time to market

Figure 12: OpenMAX defines three holistically designed media open standards to provide complete media infrastructure portability

4.6 OpenMAX IL

OpenMAX IL is the heart of the OpenMAX streaming media solution, and defines standardized media component interfaces to enable developers and platform providers to flexibly integrate and communicate with multimedia processing components implemented in hardware or software.

OpenMAX IL 1.1 was released in February 2007 and adds significant functionality to OpenMAX 1.0, which was released in January 2006, including:

- Standardized components, interfaces and controls for common media functionality to make most streaming applications easier to construct and more portable;
- Enhanced video encode and decode controls for more record and playback flexibility;
- Enhanced camera controls including sophisticated focus control, continuous and single shot control and auto exposure control;
- Abstracted access to synchronous content enabling flexible media components with the ability to process content from a variety of sources;
- Extended buffer payload information such as video quantization data to enable sophisticated adaptive applications;
- Extended color format support;
- Creation and parsing of metadata in the media stream to enable intelligent media components and applications;
- Enhanced resource management for more robust operation on constrained systems.

OpenMAX IL enables the construction of processing components with standardized interfaces that may use any CODEC internally. Processing components may be replaced or inserted into the processing graph at any time without disrupting the overall structure of the application, enabling straightforward integration of new CODEC technologies.

An open source implementation of OpenMAX IL called Bellagio, initially developed by ST Microelectronics, is available on Sourceforge.

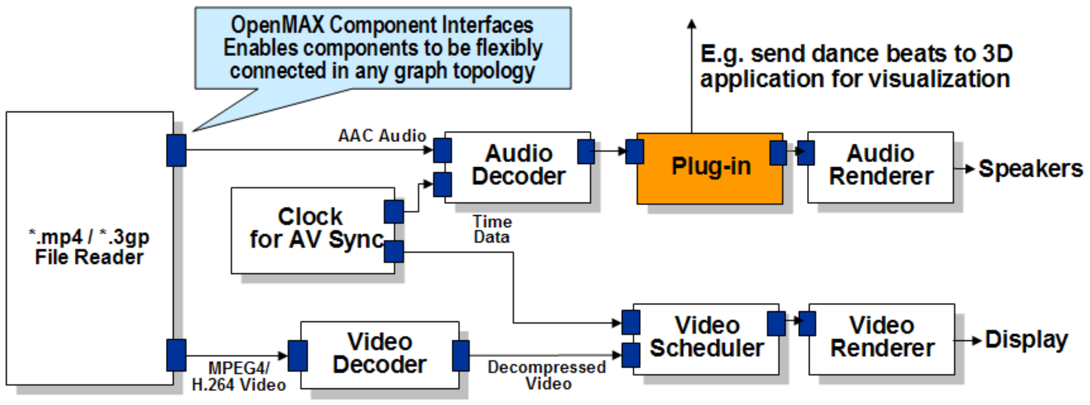

OpenMAX Component Interfaces
Enables components to be flexibly connected in any graph topology
E.g. send dance beats to 3D application for visualization
*.mp4 / *.3gp File Reader
AAC Audio
Audio Decoder
Plug-in
Audio Renderer
Speakers
Clock for AV Sync
Time Data
Video Scheduler
Video Renderer
Display
MPEG4/ H.264 Video
Video Decoder
Decompressed Video

Figure 13: An Example OpenMAX IL Streaming Media Graph
MPEG-4 video synchronized with AAC audio decode with dynamic network configuration

4.7 OpenMAX AL

OpenMAX AL is a royalty-free, cross platform open standard and forms the highest tier of the OpenMAX family of multimedia interfaces. OpenMAX AL provides simplified, operating system-agnostic, programmer-friendly interfaces to developers for accelerating the majority of streaming media applications, including accelerating the capture and presentation of audio, video and images.

OpenMAX AL enables high-level control of streaming media processing including synchronization between video and audio streams. OpenMAX AL includes the ability to create and control player and recorder objects and to connect them to configurable inputs and output objects including content readers/writers, audio inputs/outputs, display windows, cameras, analog radios, LEDs, and vibra devices. In addition, OpenMAX AL supports extensive controls for various digital camera settings and RDS/RBDS functionality for analog radio.

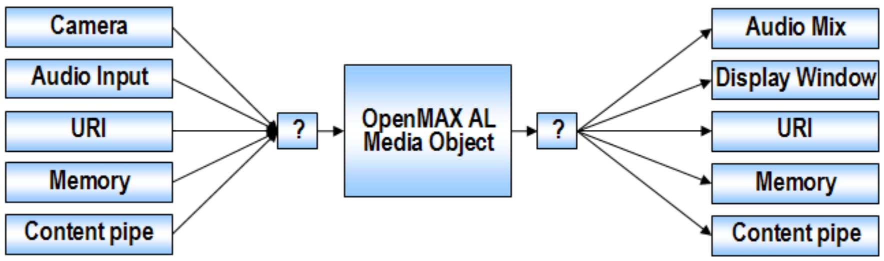


Figure 14: OpenMAX AL Object Oriented Streaming Media Programming Model

Play controls include: seek, rate, display region and metadata extraction. Record controls include: camera, video encoder, audio encoder, metadata insertion, radio and MIDI. OpenMAX enables straightforward development of sophisticated imaging applications by enabling camera to be easily controlled and the image data to be routed into the OpenMAX pipeline.

While OpenMAX AL is an independent, stand-alone API standard, it has also been designed to be efficiently implemented over the lower-level OpenMAX IL API that provides more detailed configurability of multimedia chains and is intended primarily for use by system integrators.

OpenMAX AL 1.0 was provisionally released in October 2007 and is expected to be finalized in mid-2008.

4.8 OpenSL ES

OpenSL ES is a royalty-free, cross-platform open standard for advanced audio processing on embedded and mobile devices to enable highly portable applications that integrate audio functionality including UI sounds, music playback, ring-tones and full 3D games. OpenSL ES simplifies the development of sophisticated audio-enabled applications with a comprehensive feature-set including sampled audio, SP-MIDI, Mobile XMF, metadata extraction, and equalization, as well as more advanced functionality such as MIDI messaging, 3D positional audio, reverberation and virtualization.

OpenSL ES enables application portability by providing a consistent interface to a wide variety of audio architectures on multiple operating systems, and defines phone, music and game profiles to enable diverse devices to implement relevant audio functionality, while minimizing functional fragmentation. OpenSL ES is compatible with, and can provide acceleration for, higher-level audio standards, including Java API JSR-234 (Advanced Multimedia Supplements).

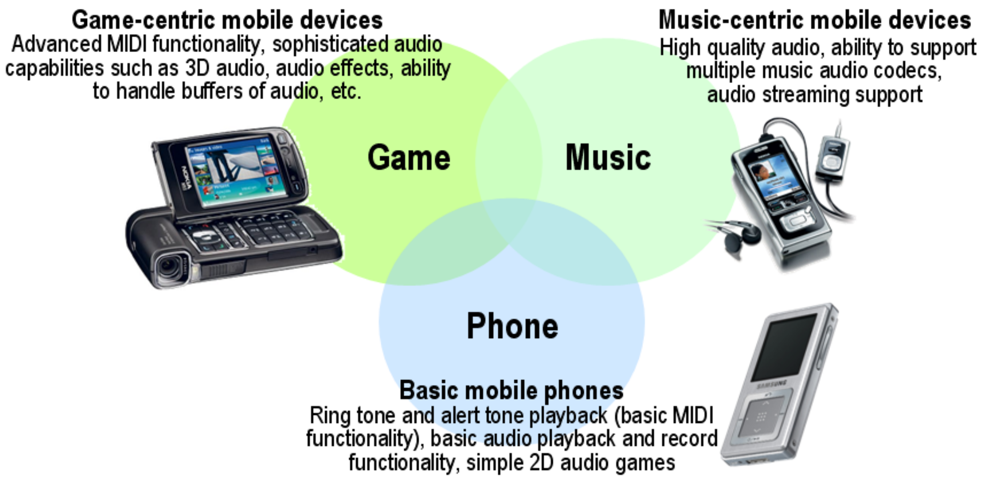

Game-centric mobile devices
Advanced MIDI functionality, sophisticated audio capabilities such as 3D audio, audio effects, ability to handle buffers of audio, etc.
Music-centric mobile devices
High quality audio, ability to support multiple music audio codecs, audio streaming support
Game
Music
Phone
Basic mobile phones
Ring tone and alert tone playback (basic MIDI functionality), basic audio playback and record functionality, simple 2D audio games

Figure 15: Three OpenSL ES Profiles reduce audio functionality fragmentation

OpenSL defined three profiles to meet the market needs of different segments of the mobile market while reducing fragmentation. An OpenSL ES conformant device can ship with one or more of the profiles. OpenSL ES 1.0 was provisionally released in October 2007 and is expected to be finalized in mid-2008.

OpenSL ES and OpenMAX AL were designed in close conjunction such that the two APIs expose common functionality, e.g. basic audio playback, in a consistent way, enabling applications to easily use whichever API is available.

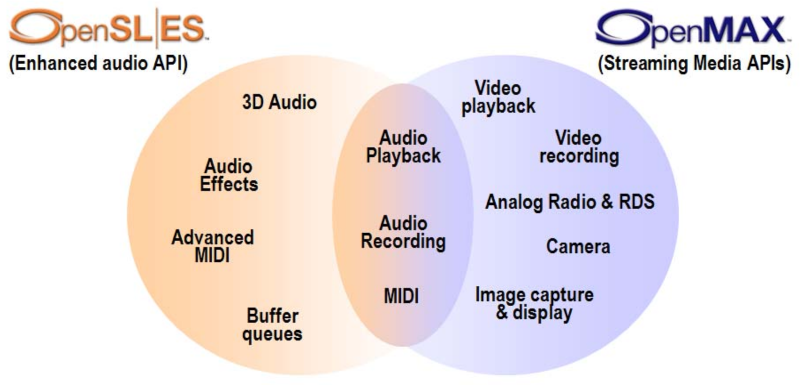


Figure 16: Overlap streaming audio functionality uses consistent APIs

5. Frequently-asked Questions

Here are some answers to the most commonly asked questions about OpenKODE:

Q: Is OpenKODE an operating system?

A: No – definitely not. OpenKODE Core is a lightweight OS abstraction layer and sits over the platform kernel OS to provides portable access to OS functionality. If there is functionality missing from the underlying OS, then OpenKODE Core would have to implement those missing pieces, but OpenKODE Core is not an OS specification.

Q: Is OpenKODE yet another mobile application development platform or does it compete with application platforms?

A: Definitely not. An application development platform defines significant high-level functionality such a virtual execution machines, provisioning, security, UI schemes etc. etc. By contrast, OpenKODE provides extremely low-level access to OS resources and media acceleration. In fact, OpenKODE is extremely complementary to application platforms such as Symbian UIQ, LIMO, Qtopia, Windows Mobile and Android as it provides the low-level access to media acceleration that all these platforms need in order to deliver compelling user experiences.

Q: Does OpenKODE prevent companies from using Khronos specifications individually or in any combinations that they wish?

A: No, OpenKODE is not mandated in any way, and implementers can ship any combination of individual or multiple Khronos APIs without restriction. OpenKODE is intended to add value over and above the individual Khronos APIs by defining and testing how they work together.

Q: Can you ship OpenKODE Core by itself without any media APIs?

A: Yes, standalone OpenKODE Core is a valid OpenKODE configuration. In fact it is possible to have OpenKODE Core sitting next to OpenGL ES and OpenVG that are not part of OpenKODE, but this means that the platform is not guaranteed to have the standardized level of EGL functionality and so OpenGL ES and OpenVG will not necessarily have been tested to work together.

Q: Why doesn't OpenKODE 1.0 include OpenGL ES 2.0, OpenMAX AL 1.0 and OpenSL ES 1.0?

A: These specifications either do not yet have conformance tests, or are still provisional. Khronos is working hard to develop EGL 1.5 that will integrate OpenMAX video into EGL – and enable synchronization between audio/video and graphics operations. When these new specification are finalized, OpenKODE will be updated to include them. We also expect to release extensions that integrate these APIs in advance of the next version of the complete OpenKODE specification.

Q: Will OpenKODE integrate OpenMAX AL or OpenMAX IL?

A: OpenMAX AL, as that is the primary application-level API in the OpenMAX family. It is possible that the integration of OpenMAX AL into EGL will enable OpenMAX IL to be integrated as well, but we currently do not believe that OpenMAX IL should be included in future versions of OpenKODE, although some more advanced applications may elect to use OpenMAX IL directly.

Q: Why did you let implementers select which media APIs they ship, isn't that re-introducing fragmentation?

A: It was a pragmatic decision to encourage rapid adoption of OpenKODE while the media APIs were still gaining market traction themselves. Subsequent versions of OpenKODE will probably be stricter about mandating media APIs.

Q: On a platform with OpenKODE, are applications prevented from calling directly to the OS if they wish?

A: No, OpenKODE applications are not excluded from using native OS APIs, but to do so will of course potentially sacrifice portability. Applications that are written exclusively to OpenKODE APIs can be re-compiled to run on any OpenKODE-conformant platform with no source changes.

Q: Why create another OS abstraction API, why not just use POSIX?

A: There were a number of reasons to define the OpenKODE Core API and not just use POSIX: POSIX is too large for mobile devices with significant desktop and server functionality that is not relevant to the mobile market; the C standards have in some cases updated POSIX in ways that are more familiar to most programmers; POSIX is missing some key functionality such as input processing; and POSIX functionality simply will not run on many of the most pervasive mobile operating systems shipping today. OpenKODE Core follows POSIX and the C standards as closely as possible – while making the API genuinely implementable on all key mobile OS and providing the functionality needed for mobile rich media applications.

Q: Does OpenKODE compete with Open C?

A: No, Open C is a POSIX compatibility layer that enables rapid porting of existing POSIX applications. This is a complementary objective to OpenKODE, and in fact OpenKODE Core is a trivial port to any platform that already supports Open C.

Q: Portability always costs something; does using OpenKODE introduce a performance overhead?

A: OpenKODE reduces fragmentation by enabling media acceleration to be implemented reliably and predictably across multiple platforms. OpenKODE does not introduce any layers between the application and the media APIs. OpenKODE Core is a layer above the OS – but it is extremely lightweight – and the cost of one extra function call per OS access is typically un-measurable in practice.