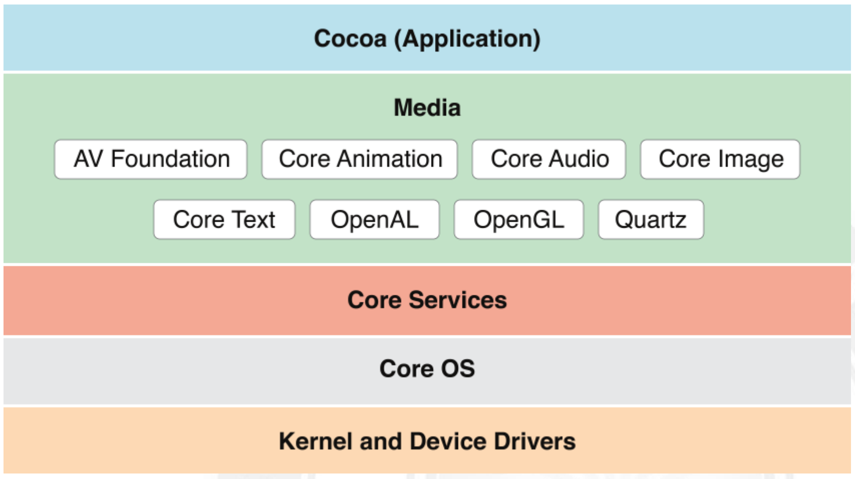
IOS - MEDIA LAYER

Beautiful graphics and high-fidelity multimedia are hallmarks of the OS X user experience. Take advantage of the technologies of the Media layer to incorporate 2D and 3D graphics, animations, image effects, and professional-grade audio and video functionality into your app.



Supported Media Formats

OS X supports more than 100 media types, covering a range of audio, video, image, and streaming formats. Table 3-1 lists some of the more common supported file formats.

Table 3-1 Partial list of formats supported in OS X

Image formats	PICT, BMP, GIF, JPEG, TIFF, PNG, DIB, ICO, EPS, PDF
Audio file and data formats	AAC, AIFF, WAVE, uLaw, AC3, MPEG-3, MPEG-4 (.mp4, .m4a), .snd, .au, .caf, Adaptive multi-rate (.amr)
Video file formats	AVI, AVR, DV, M-JPEG, MPEG-1, MPEG-2, MPEG-4, AAC, OpenDML, 3GPP, 3GPP2, AMC, H.264, iTunes (.m4v), QuickTime (.mov, .qt)
Web streaming protocols	HTTP, RTP, RTSP

Graphics Technologies

A distinctive quality of any OS X app is high-quality graphics in its user interface. And on a Retina display, users are more aware than ever of your app's graphics.

The simplest, most efficient, and most common way to ensure high-quality graphics in your app is to use the standard views and controls of the AppKit framework, along with pre - rendered images in different resolutions. In this way, you let the system do the work of rendering the app's UI appropriately for the current display.

Occasionally, you might need to go beyond off-the-shelf views and simple graphics. In these situations, you can take advantage of the powerful OS X graphics technologies. The following sections describe some of these technologies; for summaries of all technologies see Media Layer Frameworks.

Graphics and Drawing

OS X offers several system technologies for graphics and drawing. Many of these technologies provide support for making your rendered content look good at different screen resolutions. To learn how to make sure that your app looks good on a high-resolution display, see *High Resolution Guidelines for OS X*.

Cocoa Drawing

It supports drawing in both standard and custom color spaces and it supports content manipulations using graphics transforms. Drawing calls made from Cocoa are composited along with all other Quartz 2D content. You can even mix Quartz 2D drawing calls (and drawing calls from other system graphics technologies) with Cocoa calls in your code.

The AppKit framework is described in AppKit. For more information on how to draw using Cocoa features, see *Cocoa Drawing Guide*.

Metal

Metal provides the lowest-overhead access to the GPU, enabling you to maximize the graphics and compute potential of your app. With a streamlined API, precompiled shaders, and support for efficient multi-threading, Metal can take your game or graphics app to the next level of performance and capability.

The Metal framework is described in *Metal Programming Guide*, *Metal Shading Language Guide*, and the associated references.

Other Frameworks for Graphics and Drawing

In addition to AppKit (specifically, its Cocoa drawing interface), there are several other important frameworks for graphics and drawing. By design, Cocoa drawing integrates well with the other graphics and drawing technologies listed next.

- **Core Graphics** (CoreGraphics.framework). Core Graphics (also known as *Quartz 2D*) offers native 2D vector- and image-based rendering capabilities that are resolution- and device-independent. These capabilities include path-based drawing, painting with transparency, shading, drawing of shadows, transparency layers, color management, antialiased rendering, and PDF document generation. The Core Graphics framework is in the Application Services umbrella framework.

- **Core Animation.** Core Animation enables your app to create fluid animations using advanced compositing effects. It defines a hierarchical view-like abstraction that mirrors a hierarchy of views and is used to perform complex animations of user interfaces. Core Animation is implemented by the Quartz Core framework (QuartzCore.framework) (for more information, see Core Animation).

- **SpriteKit** (SpriteKit.framework). SpriteKit provides the tools and methods for creating and rendering and animating textured images, or sprites. You use graphical editors for creating sprites, and then use those sprites in scenes that simulate game physics. In addition to sprites, you can add lights, emitters, and different kinds of fields to scenes. SpriteKit animates your scene and calls back to your code for events such as collisions. To learn more, see Sprite Kit.

- **Scene Kit** (SceneKit.framework). Scene Kit provides a high-level, Objective-C graphics API that you can use to efficiently load, manipulate, and render 3D scenes. Powerful and easy-to-use Scene Kit integrates well with Core Animation and SpriteKit, allowing you to use built-in materials or custom GLSL shaders to render your 3D scenes (for more information, see Scene Kit).

- **Metal** *Metal.framework* provides extremely low-overhead access to the capabilities of modern GPUs and enables high-performance 2D and 3D graphics, and parallel computational tasks. A more flexible and efficient alternative to OpenGL and OpenCL, Metal is intended for use by games and graphics-intensive applications that require fine-grained control over the GPU. To learn more about Metal, see *Metal Programming Guide* and *Metal Shading Language Guide*

- MetalKit MetalKit.framework provides libraries of commonly needed functions and classes to reduce the overall time for developing a Metal application. For more information, see *MetalKit Framework Reference* and *MetalKit Functions Reference*.

- **OpenGL** (OpenGL.framework). OpenGL is an open, standards-based technology for creating and animating real-time 2D and 3D graphics. It is primarily used for games and other apps with real-time rendering needs. To learn more about OpenGL in OS X, see OpenGL.

- **GLKit** (GLKit.framework). GLKit provides libraries of commonly needed functions and classes that reduce the effort required to create shader-based apps or to port existing apps that rely on fixed-function vertex or fragment processing provided by earlier versions of OpenGL ES or OpenGL. To learn more about the GLKit framework, see GLKit.

Text, Typography, and Fonts

OS X provides extensive support for advanced typography for Cocoa apps. With this support, your app can control the fonts, layout, typesetting, text input, and text storage when managing the display and editing of text. For the most basic text requirements, you can use the text fields, text views, and other text-displaying objects provided by the AppKit framework.

There are two technologies to draw upon for more sophisticated text, font, and typography needs: the Cocoa text system and the Core Text API.

- **Cocoa text system.** AppKit provides a collection of classes, known as the *Cocoa text system*, that together provide a complete set of high-quality typographical services.
- **Core Text** (CoreText.framework). The Core Text framework contains low-level interfaces for laying out Unicode text and handling Unicode fonts.

Images

Both AppKit and Quartz let you create objects that represent images (NSImage and CGImageRef) from various sources, draw these images to an appropriate graphics context, and even composite one image over another according to a given blending mode. Beyond the native capabilities of AppKit and Core Graphics, you can do other things with images using the following frameworks:

- **Image Capture Core** (ImageCaptureCore.framework). The Image Capture Core framework enables your app to browse locally connected or networked scanners and cameras and images.
- **Core Image.** Core Image is an image processing technology that allows developers to process images with system-provided image filters, create custom image filters, and detect features in an image.
- **Image Kit.** Image Kit is built on top of the Image Capture Core framework.
- **Image I/O** (ImageIO.framework). The Image I/O framework helps you read image data from a source and write image data to a destination. Sources and destinations can be URLs, Core Foundation data objects, and Quartz data consumers and data providers.

Color Management

ColorSync is the color management system for OS X. It provides essential services for fast, consistent, and accurate color reproduction, proofing, and calibration. It also provides an interface for accessing and managing systemwide settings for color devices such as displays, printers, cameras, and scanners.

Printing

OS X implements printing support using a collection of APIs and system services that are available to all app environments.

Table 3-2 Features of the OS X printing system

Feature	Description

AirPrint	Users can print to an AirPrint-enabled printer on their network without having to use a third-party driver.
CUPS	Common UNIX Printing System (CUPS), an open source architecture used to handle print spooling and other low-level features, provides the underlying support for printing.
Desktop printers	Desktop printers offer users a way to manage printing and print jobs from the Dock or desktop.
Fax support	Fax support means that users can fax documents directly from the Print dialog.
Native PDF support	PDF as a native data type lets any app easily save textual and graphical data to the device-independent PDF format.
PostScript support	PostScript support allows apps to use legacy third-party drivers to print to PostScript Level 2–compatible and Level 3–compatible printers and to convert PostScript files directly to PDF.
Print preview	The print preview capability lets users see documents through a PDF viewer app prior to printing.
Printer discovery	Printer discovery enables users to detect, configure, and add to printer lists those printers that implement Bluetooth or Bonjour.
Raster printers (support for)	This support allows apps to print to raster printers using legacy third-party drivers.
Speedy spooling	Speedy spooling enables apps that use PDF to submit PDF files directly to the printing system instead of spooling individual pages.

Audio Technologies

OS X includes support for high-quality audio recording, synthesis, manipulation, and playback. The frameworks in the following list are ordered from high level to low level, with the AV Foundation framework offering the highest-level interfaces you can use. When choosing an audio technology, remember that higher-level frameworks are easier to use and, for this reason, are usually preferred. Lower-level frameworks offer more flexibility and control but require you to do more work.

- **AV Foundation** (AVFoundation.framework). AV Foundation supports audio playback, editing, analysis, and recording.
- **OpenAL** (OpenAL.framework). OpenAL implements a cross-platform standard API for 3D audio.
- **Core Audio** (CoreAudio.framework). Core Audio consists of a set of frameworks that provide audio services that support recording, playback, synchronization, signal processing, format conversion, synthesis, and surround sound.

Video Technologies

When choosing a video technology, remember that the higher-level frameworks simplify your work and are, for this reason, usually preferred.

The frameworks in the following list are ordered from highest to lowest level, with the AV Foundation framework offering the highest-level interfaces you can use.

- **AVKit** (AVKit.framework). AV Kit supports playing visual content in your application using the standard controls.
- **AV Foundation** (AVFoundation.framework). AV Foundation supports playing, recording, reading, encoding, writing, and editing audiovisual media.
- **Core Media** (CoreMedia.framework). Core Media provides a low-level C interface for managing audiovisual media. With the Core Media I/O framework, you can create plug-ins that can access media hardware and that can capture video and mixed audio and video streams.
- **Core Video** (CoreVideo.framework). Core Video provides a pipeline model for digital video between a client and the GPU to deliver hardware-accelerated video processing while allowing access to individual frames.