OpenJDK
Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
JDK GA/EA Builds
Mailing lists
Wiki · IRC
Bylaws · Census
Legal
Workshop
JEP Process
Source code
Mercurial
GitHub
Tools
Git
jtreg harness
Groups
(overview)
Adoption
Build
Client Libraries
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
IDE Tooling & Support
Internationalization
JMX
Members
Networking
Porters
Quality
Security
Serviceability
Vulnerability
Web
Projects
(overview, archive)
Amber
Audio Engine
CRaC
Caciocavallo
Closures
Code Tools
Coin
Common VM Interface
Compiler Grammar
Detroit
Developers' Guide
Device I/O
Duke
Font Scaler
Galahad
Graal
Graphics Rasterizer
IcedTea
JDK 7
JDK 8
JDK 8 Updates
JDK 9
JDK (..., 21, 22)
JDK Updates
JavaDoc.Next
Jigsaw
Kona
Kulla
Lambda
Lanai
Leyden
Lilliput
Locale Enhancement
Loom
Memory Model Update
Metropolis
Mission Control
Modules
Multi-Language VM
Nashorn
New I/O
OpenJFX
Panama
Penrose
Port: AArch32
Port: AArch64
Port: BSD
Port: Haiku
Port: Mac OS X
Port: MIPS
Port: Mobile
Port: PowerPC/AIX
Port: RISC-V
Port: s390x
Portola
SCTP
Shenandoah
Skara
Sumatra
Tiered Attribution
Tsan
Type Annotations
Valhalla
Verona
VisualVM
Wakefield
Zero
ZGC
ORACLE

JEP 382: New macOS Rendering Pipeline

Owner	Philip Race
Type	Feature
Scope	JDK
Status	Closed / Delivered
Release	17
Component	client-libs / 2d
Discussion	lanai dash dev at openjdk dot java dot net
Reviewed by	Ajit Ghaisas, Alexey Ushakov, Iris Clark, Jayathirth D V, Kevin Rushforth, Sergey Bylokhov
Endorsed by	Kevin Rushforth
Created	2020/01/31 21:40
Updated	2022/04/05 17:26
Issue	8238361

Summary

Implement a Java 2D internal rendering pipeline for macOS using the Apple Metal API as alternative to the existing pipeline, which uses the deprecated Apple OpenGL API.

Goals

- Provide a fully functional rendering pipeline for the Java 2D API that uses the macOS Metal framework.
- Be ready in the event Apple removes the deprecated OpenGL API from a future version of macOS.
- Ensure transparency of the new pipeline to Java applications.
- Ensure functional parity of the implementation with the existing OpenGL pipeline.
- Provide performance as good or better than the OpenGL pipeline in select real applications and benchmarks.
- Create a clean architecture that fits into the existing Java 2D pipeline model.
- Co-exist with the OpenGL pipeline until it is obsolete.

Non-Goals

- It is not a goal to remove or disable the existing OpenGL pipeline.
- It is not a goal to add any new Java or JDK APIs. This is all internal implementation.

Motivation

Two major factors motivate the introduction of a new Metal-based rendering pipeline on macOS:

- Apple deprecated the OpenGL rendering library in macOS 10.14, in September 2018. Java 2D on macOS is completely reliant on OpenGL for its internal rendering pipeline, so a new pipeline implementation is needed.
- Apple claims that the Metal framework, their replacement for OpenGL, has superior performance. For the Java 2D API, this is generally the case with some exceptions.

Description

Most graphical Java applications are written using the Swing UI toolkit, which renders via the Java 2D API. Internally, Java 2D can use software rendering plus a blit to the screen or it can use a platform-specific API, such as X11/Xrender on Linux, Direct3D on Windows, or OpenGL on macOS. These platform-specific APIs typically offer much better performance than software rendering, and generally off-load the CPU. Metal is the new macOS platform API for such rendering, replacing the deprecated OpenGL API. (The name has nothing to do with the Swing “Metal” Look and Feel; that is just a coincidence.)

We created a substantial amount of new internal implementation code to use the Metal framework, just as we already had for the other platform-specific APIs. Whilst easily fitting into the existing framework the new code is much more modern in its use of graphics hardware, making use of shaders rather than a fixed function pipeline. The changes are confined to macOS-specific code and even there only a minimal amount of code shared between Metal and OpenGL is updated. We did not introduce any new Java APIs, nor did we change any existing API.

The Metal pipeline can co-exist with the OpenGL pipeline. When a graphical application starts up, one or the other is chosen. For now, OpenGL remains the default. Metal is used only if it is specified on startup or if the initialization of OpenGL fails, as would happen in a future version of macOS with no OpenGL support.

At the time of integration of this JEP, Apple have yet to remove OpenGL. Until that happens an application can opt-in to Metal by specifying -

`Dsun.java2d.metal=true` on the `java` command line. We will make the Metal rendering pipeline the default in a future release.

Prior to integration in the JDK, we conducted work on this JEP in Project Lanai.

Testing

Testing the functionality of the new pipeline did not require new functional test development, since no Java 2D APIs were changed. Existing tests and real-world applications sufficed. These included:

- JDK jtreg regression tests,
- JCK Tests,
- Java 2D and Swing Demos, and
- IDEs such as Intellij IDEA and Netbeans, as examples of large-scale real world applications.

To test performance, we used:

- J2DBench, a Java 2D benchmarking application included in JDK,
- RenderPerfTest, a custom stress test that renders multiple objects of the same primitive type and measures frames per second (FPS) developed in Project Lanai, and
- IntelliJ IDEA IDE performance.

Performance results for the final planned early-access release are here.

To further verify the new pipeline, we used macOS Xcode instrumentation tools to check for leaks and for correct Metal API usage.

Risks and Assumptions

- We tested on a variety of hardware and macOS versions which are presumed to be representative, but not all combinations were available. Since we could not account for all scenarios, it is possible that performance limitations remain.
- We did very limited (sanity) testing of the current x64 binaries on Apple Silicon. No port of the JDK to Apple Silicon is yet available to support native testing.
- Metal does not support the XOR operation, so we had to accept lower performance in that niche case. That is likely to remain so until such time as Metal provides direct support for XOR.