Work In Progress

Work done so far :

A working prototype of basic feature-set with Java 2D Metal rendering pipeline has been implemented for -

1. 2D Primitives rendering
    a. implemented AA rendering for some primitives and painting types
    b. optimised geometry handling (no unnecessary MTLBuffer creation)
    c. improved stability: fixed some severe memory management problems in primitive handling
2. Multisampling (hardware accelerated) AA
    a. investigated multisampling AA rendering
    b. implemented AA parallelogram rendering via multisampling AA
3. Image rendering.
    a. improved stability: fixed several memory management problems
    b. native blit operations (Blit, IsoBlit) for almost all blit primitives (of MTLBlitLoops)
    c. all 32bpp raster formats with ‘opaque’ flags
    d. all alpha-composite modes (still unsupported extra-alpha)
    e. clipping, transform
    f. textures pool (to avoid reallocations for temporary textures) with restricted size (cleaned with LRU-strategy)
        i. fast search in pool
        ii. no unnecessary sync
        iii. blit without sampling if possible
    g. supported all parameters of blit operations
4. General metal pipeline workflow
    a. fixed resources leaks, added minor optimizations (in PipelineStateStorage and TexPool)
    b. refactored MTLContext (splited into several classes)
    c. fixed issues connected with changes of pipelineState
    d. added minor optimizations (don’t change state when possible)
5. Grayscale Text Rendering - Feature completion of Grayscale text rendering without cache under JDK-8225174
6. Initial implementation of Grayscale text rendering with cache under JDK-8237608
7. Closure of main performance bottleneck task JDK-8228573 after implementing proper back/front buffer drawing under JDK-8233190
8. JDK-8233233 - Implement Shape Clip (for aliased rendering)

It is available in the lanai repo https://hg.openjdk.java.net/lanai/lanai/

It can be tested with J2DDemo (available at - src/demo/share/jfc/J2Ddemo) that shows working / partially working / not-working Java 2D features with Metal rendering as compared with OpenGL rendering.
VM option -Dsun.java2d.metal=True needs to be passed in to switch rendering pipeline to Metal.
Also, antialiasing needs to be turned off (which is on by default in J2DDemo)

Work in Progress :

Current focus area is - Rendering performance evaluation :
As basic blocks of rendering have been implemented, it is a good time to check on rendering performance to evaluate how Metal rendering in Java 2D fares as compared to OpenGL rendering in Java 2D. As a first step, we are using below test benchmarkings -

1. RenderPerfTest (available at - src/demo/share/java2d/RenderPerfTest)
    a. implemented AA benchmarks
2. J2DBench Tests (available at - src/demo/share/java2d/J2DBench) for drawing 2D primitives - line, rectangle, filled rectangle, ellipse, filled ellipse.

The performance numbers for Java 2D Metal rendering pipeline are not up to the mark as compared to the Java 2D OpenGL rendering pipeline. Multiple design level optimizations are being done to address this.

1. Reducing number of MTLRenderEncoders that get created per render pass
2. Reducing the number of drawPrimitives calls by batching data whenever possible
3. Using setNeedsDisplay flag of MTLLayer and updating the layer content on AppKit thread in a callback
4. Use multisampling AA for AA primitive rendering

Current numbers from RenderPerfTest (OGL vs Metal):

Bench	OGL	Metal
FlatOval(AA)	71(18)	72(5)
FlatBox(AA)	87(90)	87(6)
Image(AA)	91(90)	88(88)
RotatedBox(AA)	88(84)	89(6)
RotatedOval(AA)	77(19)	78(8)
LinGradRotatedOval	28	74

WiredBubbles(AA)	44(14)	46(5)
WiredBox	85	88
Lines	85	85
FlatQuad(AA)	71(9)	73(1)
WiredQuad(AA)	57(13)	60(5)
TextNoAA	94	62
TextLCD	81	57
TextGray	88	58