```
package com.example.pdfviewer

import android.content.Context
import android.graphics.Bitmap
import android.graphics.Canvas
import android.graphics.Matrix
import android.graphics.Rect
import android.graphics.pdf.PdfRenderer
import android.util.AttributeSet
import android.view.GestureDetector
import android.view.MotionEvent
import android.view.ScaleGestureDetector
import android.view.View
import kotlinx.coroutines.*
import kotlin.math.max
import kotlin.math.min

/**
 * A Custom View that uses PdfRenderer to render ONLY the visible
portion of the PDF.
 * This ensures high-quality text even at high zoom levels (Vector
Zoom),
 * without running OutOfMemory by creating giant bitmaps.
 */
class VectorPdfView @JvmOverloads constructor(
    context: Context,
    attrs: AttributeSet? = null
) : View(context, attrs) {

    // Rendering Components
    private var pdfPage: PdfRenderer.Page? = null
    private var renderBitmap: Bitmap? = null
    private val renderScope = CoroutineScope(Dispatchers.Main + Job())
    private var renderJob: Job? = null

    // State
    private var currentZoom = 1f
    private var currentX = 0f
    private var currentY = 0f

    // Limits
    private val minZoom = 1f
    private val maxZoom = 10f // Allow deep zoom
    private var pdfWidth = 0
    private var pdfHeight = 0

    // Detectors
    private val scaleDetector = ScaleGestureDetector(context,
```

```
ScaleListener())
    private val gestureDetector = GestureDetector(context,
GestureListener())

    /**
     * Call this to display a page. The page must be open.
     */
    fun showPage(page: PdfRenderer.Page) {
        this.pdfPage = page
        this.pdfWidth = page.width
        this.pdfHeight = page.height

        // Reset view
        currentZoom = 1f
        currentX = 0f
        currentY = 0f

        requestRender()
    }

    override fun onSizeChanged(w: Int, h: Int, oldw: Int, oldh: Int) {
        super.onSizeChanged(w, h, oldw, oldh)
        // Create a bitmap that matches the screen/view size
        if (w > 0 && h > 0) {
            renderBitmap = Bitmap.createBitmap(w, h,
Bitmap.Config.ARGB_8888)
            requestRender()
        }
    }

    override fun onDraw(canvas: Canvas) {
        super.onDraw(canvas)
        // We draw the rendered bitmap.
        // Note: We don't apply the matrix to the canvas because the
bitmap
        // ITSELF contains the zoomed/translated content.
        renderBitmap?.let {
            canvas.drawBitmap(it, 0f, 0f, null)
        }
    }

    override fun onTouchEvent(event: MotionEvent): Boolean {
        scaleDetector.onTouchEvent(event)
        gestureDetector.onTouchEvent(event)
        return true
    }

    private fun requestRender() {
```

```
        if (pdfPage == null || width == 0 || height == 0) return

        // Debounce render calls to avoid lag during rapid gestures
        renderJob?.cancel()
        renderJob = renderScope.launch {
            // Small delay to let the user finish the gesture movement
            delay(50)
            renderNow()
        }
    }

    /**
     * The Magic: Calculate the Matrix needed to project the PDF
content
     * onto our fixed-size View based on current zoom/pan.
     */
    private fun renderNow() {
        val page = pdfPage ?: return
        val bitmap = renderBitmap ?: return

        // 1. Calculate the matrix
        // We want to map the PDF coordinates to the View coordinates.
        // Base scale fits the PDF width to the View width
        val initialScale = width.toFloat() / page.width.toFloat()
        val totalScale = initialScale * currentZoom

        val matrix = Matrix()
        matrix.setScale(totalScale, totalScale)
        matrix.postTranslate(currentX, currentY)

        // 2. Render!
        // PdfRenderer uses this matrix to rasterize the vectors.
        // This makes the text sharp even at 10x zoom.
        // We use RENDER_MODE_FOR_DISPLAY which optimizes for screen
viewing.
        try {
            // Clear bitmap before drawing (transparent background)
            bitmap.eraseColor(0xFFFFFFFF.toInt())

            // The logic: "Draw the PDF onto this bitmap, transforming
it by 'matrix'"
            page.render(bitmap, null, matrix,
PdfRenderer.Page.RENDER_MODE_FOR_DISPLAY)

            invalidate() // Force a redraw of the View
        } catch (e: Exception) {
            e.printStackTrace()
        }
```

```
    }

    // --- Gesture Handling ---

    private fun checkBounds() {
        val page = pdfPage ?: return
        val viewW = width.toFloat()
        val viewH = height.toFloat()

        val initialScale = viewW / page.width.toFloat()
        val totalScale = initialScale * currentZoom

        val contentHeight = page.height * totalScale
        val contentWidth = page.width * totalScale

        // Y-Bounds
        if (contentHeight < viewH) {
            currentY = (viewH - contentHeight) / 2 // Center
vertically
        } else {
            if (currentY > 0) currentY = 0f
            if (currentY < viewH - contentHeight) currentY = viewH -
contentHeight
        }

        // X-Bounds
        if (contentWidth < viewW) {
            currentX = (viewW - contentWidth) / 2 // Center
horizontally
        } else {
            if (currentX > 0) currentX = 0f
            if (currentX < viewW - contentWidth) currentX = viewW -
contentWidth
        }
    }

    private inner class ScaleListener :
ScaleGestureDetector.SimpleOnScaleGestureListener() {
        override fun onScale(detector: ScaleGestureDetector): Boolean
{
            val scaleFactor = detector.scaleFactor
            val px = detector.focusX
            val py = detector.focusY

            val oldZoom = currentZoom
            currentZoom *= scaleFactor
            currentZoom = max(minZoom, min(currentZoom, maxZoom))
```

```
            // Adjust X/Y to zoom towards the focal point
            val scaleChange = currentZoom / oldZoom
            currentX = px - (px - currentX) * scaleChange
            currentY = py - (py - currentY) * scaleChange

            checkBounds()
            requestRender() // Trigger re-render
            return true
        }
    }

    private inner class GestureListener :
GestureDetector.SimpleOnGestureListener() {
        override fun onScroll(e1: MotionEvent?, e2: MotionEvent,
distanceX: Float, distanceY: Float): Boolean {
            currentX -= distanceX
            currentY -= distanceY
            checkBounds()
            requestRender()
            return true
        }
    }

    override fun onDetachedFromWindow() {
        super.onDetachedFromWindow()
        renderJob?.cancel()
        renderBitmap?.recycle()
        renderBitmap = null
    }
}
```