Script used for bearing adjustment in FontForge

For bearing adjustment

import fontforge

import os

# === SETTINGS ===========================

COMPRESSION\_FACTOR = 1.0 # Change this to 1.2, 1.0, 0.5 as needed

FORCE\_SQUISH\_VALUE = -100

SQUISH\_ZONE\_MIN = -5

SQUISH\_ZONE\_MAX = 5

MAX\_COMPRESSION\_FOR\_ALEF\_HAMZA = 1.2 # Capping for Alef+Hamza if factor is too high

# List of Alef-Hamza variants (أ)

alef\_hamza\_variants = {

'uni0627', 'uni0627.fina', 'uni0622', 'uniFE82', 'uni0623', 'uniFE84', 'uni0625', 'uniFE88'

}

# === ACTIVE FONT ========================

font = fontforge.activeFont()

desktop = os.path.join(os.path.expanduser("~"), "Desktop")

log\_path = os.path.join(desktop, "final\_bearings\_adjusted\_log\_120.txt")

# === COMPRESSION LOGIC ==================

def compress(value, factor, glyphname):

# Squish zone override

if SQUISH\_ZONE\_MIN <= value <= SQUISH\_ZONE\_MAX:

return FORCE\_SQUISH\_VALUE

# Use capped factor if glyph is Alef-Hamza and factor > max

if glyphname in alef\_hamza\_variants and factor > MAX\_COMPRESSION\_FOR\_ALEF\_HAMZA:

factor = MAX\_COMPRESSION\_FOR\_ALEF\_HAMZA

return round(value + (-abs(value) \* factor))

# === EXECUTION ===========================

found\_glyphs = []

adjusted\_glyphs = []

with open(log\_path, "w", encoding="utf-8") as f:

f.write(f"🧠 Final Bearing Adjustment Log\n")

f.write(f"Compression Factor: {COMPRESSION\_FACTOR}\n")

f.write(f"Alef-Hamza Cap: {MAX\_COMPRESSION\_FOR\_ALEF\_HAMZA}\n")

f.write(f"Squish Zone: [-5, 5] → Forced to {FORCE\_SQUISH\_VALUE}\n")

f.write("===========================================\n\n")

for glyph in font.glyphs():

name = glyph.glyphname

found\_glyphs.append(name)

try:

old\_lb = glyph.left\_side\_bearing

old\_rb = glyph.right\_side\_bearing

new\_lb = compress(old\_lb, COMPRESSION\_FACTOR, name)

new\_rb = compress(old\_rb, COMPRESSION\_FACTOR, name)

glyph.left\_side\_bearing = new\_lb

glyph.right\_side\_bearing = new\_rb

adjusted\_glyphs.append(name)

f.write(f"{name}\n")

f.write(f" LB: {old\_lb} → {new\_lb}\n")

f.write(f" RB: {old\_rb} → {new\_rb}\n\n")

except Exception as e:

f.write(f"⚠️ Failed to adjust {name}: {str(e)}\n")

# === SUMMARY BLOCK ===================

f.write("===========================================\n")

f.write(f"Total glyphs adjusted: {len(adjusted\_glyphs)}\n")

f.write(f"Total glyphs in font: {len(found\_glyphs)}\n")

f.write("===========================================\n")

print("✅ Done! Bearings adjusted with smart compression. Log saved to Desktop.")

Removing Anchors:

import os

import fontforge

font = fontforge.activeFont()

desktop = os.path.join(os.path.expanduser("~"), "Desktop")

anchor\_log\_path = os.path.join(desktop, "nuked\_anchors\_log.txt")

nuked = []

skipped = []

with open(anchor\_log\_path, "w", encoding="utf-8") as f:

f.write("🚫 Glyphs with anchors removed:\n")

f.write("================================\n\n")

for glyph in font.glyphs():

name = glyph.glyphname

# Skip special system glyphs

if name in [".notdef", "NULL", "nonmarkingreturn", "space"]:

skipped.append(name)

f.write(f"⚠️ Skipped non-adjustable glyph: {name}\n")

continue

try:

anchor\_names = glyph.anchorPoints

if anchor\_names:

nuked.append(name)

f.write(f"{name}: {len(anchor\_names)} anchor(s) removed\n")

for anchor in anchor\_names:

try:

glyph.removeAnchorPoint(anchor[0])

except Exception as e:

f.write(f" ⚠️ Failed to remove anchor {anchor[0]}: {e}\n")

# Always unlink references (harmless if already unlinked)

glyph.unlinkRef()

# Force update so FontForge doesn’t retain ghost anchors

if not glyph.foreground.isEmpty():

glyph.changed = True

else:

glyph.clear()

except Exception as e:

skipped.append(name)

f.write(f"⚠️ Failed to process {name}: {e}\n")

# Summary

f.write("\n================================\n")

f.write(f"✅ Total glyphs with anchors removed: {len(nuked)}\n")

f.write(f"ℹ️ Glyphs skipped or non-applicable: {len(skipped)}\n")

f.write("================================\n")

print("✅ Anchor purge complete! Log saved to Desktop.")