Generación de PDFs a partir de documentos escaneados

Casuística, algoritmos y limitaciones

Jesús Cea Avión
Twitter: @jcea
jcea@jcea.es
http://www.jcea.es/

8 de marzo de 2012

Origen y motivación:

- Preservar la historia.

- Revista en papel editada en 1990. Se publicaron nueve números.

- Los originales informáticos están en muy mal estado, incompletos, requieren de un emulador, etc.

- Tenemos la versión en papel.

Los pasos son:

- Escaneado de los originales, en alta resolución y color o grises.

- Procesado de esas imágenes: girado, paso a bilevel, limpieza de defectos, …

- Generación de los PDFs.

Automatizar y documentar todo lo posible.

Escaneado de los originales:

- El original está en formato A5, así que se puede usar un escaner de mesa.
  - Son A4 plegados, cuatro A5 por papel físico.

- Hay que ser cuidadoso en el orden de escaneo de páginas, para facilitar el procesado automático posterior.

- Escaneo a 600ppp (DPI), en 256 grises (8 bits por píxel)
  - Cada imagen A5 mide unos 10Megabytes.
  - Cada revista son 28 páginas.
  - Formato PNG, sin pérdidas.

- Este paso es manual, crítico e irreversible.

Procesado de imágenes:

Esta parte se automatiza por completo.

- En cada escaneo digitalizamos un A4, que son dos A5. Troceamos cada imagen en dos.

- Las páginas están giradas. Las ponemos en posición vertical.

- Convertimos a Bi-Level
  - Por cada imagen A4 de 10Mbytes, obtenemos dos imágenes A5 de unos 100Kbytes.

Convertimos a Bi-Level:

- La fuente original es Bi-Level.

- Los algoritmos conocidos más habituales (dithering, haftoning y floyd-Steinberg) están pensadas para imágenes en niveles de grises.

- Necesitamos un algoritmo de “Thresholding”.
  - Los algoritmos más obvios no tienen calidad.
  - Utilizo Otsu Thresholding.
    - Difícil de encontrar.
    - Proporciona un Thresholding global.
  - La clave del asunto:
    - jcea@ubuntu:/tmp/z/gimp-2.6.12$ find . -name "*.c" -exec grep -i otsu {} \; -print
      * N. Otsu, "A threshold selection method from gray-level histograms,"
      ./app/base/gimphistogram.c

Generación de los PDF:

- Utilizo ReportLab.

- Cada página del PDF es una imagen A5.

- El programa monta las páginas en el orden correcto, si hemos escaneado en un orden consistente.

- Cada revista de 28 páginas es un PDF de unos 10Mbytes.
  - ¡Parcheando!.
  - Aceptable para los estándares actuales.

- ¡Dios, no quiero dedicar más tiempo a esto!.

El parcheo ese:

- ReportLab reconoce dos tipos de imágenes: JPEG y RBG. Cualquier otro formato es convertido a RGB.
  - Incluyendo tonos de gris, bi-level y paleta.
  - Los PDF miden tres veces más de lo que debieran.

- La única compresión posible en una imagen no JPEG es Deflate (ZIP).
  - El PNG usa Deflate con un preprocesado muy efectivo.

- No hay soporte para sistemas de compresión avanzados, como JBIG2.

El parcheo ese: (II)

```
jcea@ubuntu:/usr/lib/python2.6/dist-packages/reportlab/pdfgen$ diff -u pdfimages.py.OLD pdfimages.py
--- pdfimages.py.OLD  2009-02-03 22:26:43.000000000 +0100
+++ pdfimages.py 2012-01-03 16:39:32.235298312 +0100
@@ -100,21 +100,30 @@
         if image.mode == 'CMYK':
             myimage = image
             colorSpace = 'DeviceCMYK'
-            bpp = 4
+            bpp = 4*8
+        elif image.mode == '1' :
+            myimage = image
+            colorSpace = 'DeviceGray'
+            bpp = 1
+        elif image.mode == 'L' :
+            myimage = image
+            colorSpace = 'DeviceGray'
+            bpp = 1*8
         else:
             myimage = image.convert('RGB')
             colorSpace = 'RGB'
-            bpp = 3
+            bpp = 3*8
         imgwidth, imgheight = myimage.size

         # this describes what is in the image itself
         # *NB* according to the spec you can only use the short form in inline images
         #imagedata=['BI /Width %d /Height /BitsPerComponent 8 /ColorSpace /%s /Filter [/Filter [ /ASCII85Decode
/FlateDecode] ID]' % (imgwidth, imgheight,'RGB')]
-        imagedata=['BI /W %d /H %d /BPC 8 /CS /%s /F [/A85 /Fl] ID' % (imgwidth, imgheight,colorSpace)]
+        imagedata=['BI /W %d /H %d /BPC %d /CS /%s /F [/A85 /Fl] ID' %
+                (imgwidth, imgheight,1 if bpp<8 else 8,colorSpace)]

         #use a flate filter and Ascii Base 85 to compress
         raw = myimage.tostring()
-        assert len(raw) == imgwidth*imgheight*bpp, "Wrong amount of data for image"
+        assert len(raw) == imgwidth*imgheight*bpp/8.0, "Wrong amount of data for image"
         compressed = zlib.compress(raw)   #this bit is very fast...
         encoded = pdfutils._AsciiBase85Encode(compressed) #...sadly this may not be
         #append in blocks of 60 characters
```

El código:

```
#!/usr/bin/env python2.6

# Otsu Thresholding
# http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html

import sys, math
from PIL import Image
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import portrait, A5
from reportlab.lib.utils import ImageReader
#reportlab.lib.utils.Image = Image

a=canvas.Canvas("z.pdf", pagesize=portrait(A5))
```

[. . .]

```
for fichero in sys.argv[1:] :

    im = Image.open(fichero)
    im = im.transpose(Image.ROTATE_90).convert("L")
    histogram = im.histogram()
    width, height = im.size

    umbral = otsu_2(width, height, histogram)
    print fichero, umbral

    im = Image.eval(im, lambda x : 0 if x<umbral else 255)
    im = im.convert("1")
    #im.save("z.png")
    #im.show()

    im=im.crop((0,0, width/8*8, height))
    im.load()
#    a.drawImage(ImageReader(im), 0, 0, width=419, height=419*math.sqrt(2),
#             preserveAspectRatio=True)
    a.drawInlineImage(im, 0, 0, width=419, height=419*math.sqrt(2),
            preserveAspectRatio=True)
    a.showPage()

a.save()
```

El código: (II)

```
def otsu_1(width, heigh, histogram) :
    l = width*height
    umbral = 0
    varianza_umbral = 1e+100
    for i in xrange(257) :
        cb = float(sum((v for j,v in enumerate(histogram[:i]))))
        cw = float(sum((v for j,v in enumerate(histogram[i:], i))))
        if (cb == 0) or (cw == 0) : continue
        wb = cb/l
        ww = cw/l
        mb = sum((j*v for j,v in enumerate(histogram[:i])))/cb
        mw = sum((j*v for j,v in enumerate(histogram[i:], i)))/cw
        vb = sum(((j-mb)*(j-mb)*v for j,v in enumerate(histogram[:i])))/cb
        vw = sum(((j-mw)*(j-mw)*v for j,v in enumerate(histogram[i:], i)))/cw

        varianza = wb*vb + ww*vw

        if varianza < varianza_umbral :
            umbral = i
            varianza_umbral = varianza

    return umbral
```

El código: (III)

```
# Esta rutina, para mis escaneos, es 2-3 veces mas
# rapida, y el umbral calculado es el mismo.
def otsu_2(width, heigh, histogram) :
    l = width*height
    umbral = 0
    varianza2_umbral = -1
    for i in xrange(257) :
        cb = float(sum((v for j,v in enumerate(histogram[:i]))))
        cw = float(sum((v for j,v in enumerate(histogram[i:], i))))
        if (cb == 0) or (cw == 0) : continue
        wb = cb/l
        ww = cw/l
        mb = sum((j*v for j,v in enumerate(histogram[:i])))/cb
        mw = sum((j*v for j,v in enumerate(histogram[i:], i)))/cw

        varianza2 = wb*ww*(mb-mw)*(mb-mw)

        if varianza2 > varianza2_umbral :
            umbral = i
            varianza2_umbral = varianza2

    return umbral
```

Conclusiones:

- Preservar la historia.

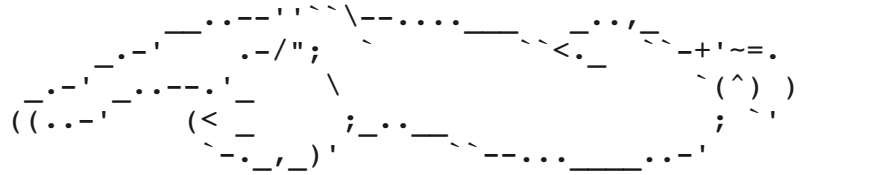
- Para escanear documentos de más de A4, usar fotografías o escaneo por trozos y luego fusionarlos.

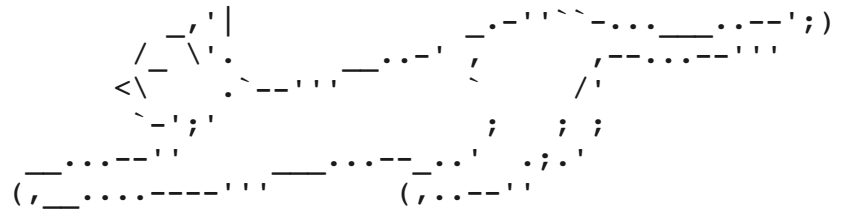
- Considerar que las operaciones son irreversibles.

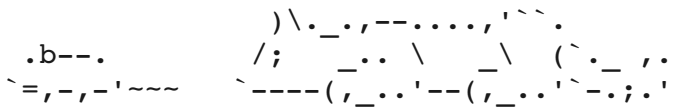
- Automatizar nos permite empaquetar operaciones y algoritmos de alta calidad. Generar una revista nueva supone menos de cinco segundos.

- Para revistas con color, segmentar la página en zonas de fotos y zonas de texto, como hace djvu.

Las fotos de gatitos de rigor...







¿Y las demos?