Obligatorisk innlevering 3 høsten 2014, INF3331

Nicolai Skogheim <nicolai.skogheim@gmail.com>

November 19, 2014

Contents

Oversikt

Programmene som brukes til bildeprossesering ligger i MyImgTool-mappen. Tester ligger i tests-mappen. I resources-mappen ligger filer hentet fra student-resources.
Denne filen, og tilhørende .tex, ligger i report-mappen, men kjøreeksempler i denne rapporten er alle ment å kjøres fra oblig03-mappen. For enkle kjøreeksempler kan det anbefales å ta en titt på README.md i oblig03-mappen.

Denne rapporten kan for øvrig genereres med følgende kommandoer. Obs. Må kjøres fra oblig03/report hvor denne filen ligger.

Terminal

```
> python ../../oblig02/PreTeX/prepro.py report.xtex report.tex
> python ../../oblig02/PreTeX/compile.py ./report.tex
```

1 Implementasjon i Python, Numpy og Weave

Hoveddelen i denne oppgaven var denoisingen. Dette er python-løsningen:

```
def denoise(data,h,w, kappa=0.1, iter=10):
    """
    Runs a denoise algorithm on a given arary.
    Returns denoised channel.
    """
    logging.info("Running denoise.")
    for _ in xrange(0,iter):
        for i in xrange(1, w-1):
            for j in xrange(1, h-1):
                data[j*w+i] += \
                              kappa*(data[(j-1)*w+i]
                                   + data[j*w+(i-1)]
                                   - 4*data[j * w + i]
                                   + data[j*w+(i+1)]
                                   + data[i+ w * (j+1)]
                                   )
    return data
```

Og her er løsningen med weave:

```
def denoise(data,h,w, kappa=0.1, iter=10):
    """
    Runs a denoise algorithm on a given arary.
    Returns denoised channel.
    """
    logging.info("Running denoise.")
    data_new = data.copy()
    tmp = data.copy()
    in_vars = ["data","data_new","tmp","kappa", "iter", "h", "w"]
    code=r"""
        for (int round=0; round<iter; round++)
        {
            for (int i=1; i<h-1; i++)
            {
                for (int j=1; j<w-1; j++)
                {
                    data_new(i,j) = data(i,j) + kappa*(data(i-1,j)
                    +data(i,j-1) -4*data(i,j) +data(i,j+1)
                    +data(i+1,j));
                }
            }
            tmp = data;
            data = data_new;
            data_new = tmp;
        }
    """

    comp=weave.inline(code,
                      in_vars,
                      type_converters=weave.converters.blitz)
    return data
```

2 Bruk av profilering

Profilering av disse, i tilleg til fargeversjonen som nevnes senere, kan kjøres slik:

Terminal

```
> python MyImgTool/profiling.py
```

Og resultatet vil se omtrent slik ut:

Terminal

```
> python MyImgTool/profiling.py

Wed Nov 19 02:23:46 2014    python_timing
Wed Nov 19 02:23:46 2014    weave_timing
Wed Nov 19 02:23:48 2014    color_timing

         191170 function calls (191084 primitive calls) in 3.728 seconds

   Ordered by: cumulative time
   List reduced from 302 to 20 due to restriction <20>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.002    0.002    2.148    2.148 denoise_colors.py:181(run)
        1    0.021    0.021    1.833    1.833 denoise_colors.py:30(rgb2hsi)
        1    0.095    0.095    1.782    1.782 function_base.py:1843(__call__)
   187501    1.678    0.000    1.678    0.000 denoise_colors.py:45(hfunk)
        3    0.000    0.000    1.531    0.510 <string>:1(<module>)
        1    0.000    0.000    1.531    1.531 denoise.py:46(run)
        1    1.476    1.476    1.476    1.476 denoise.py:28(denoise)
        1    0.192    0.192    0.198    0.198 denoise_colors.py:64(hsi2rgb)
        3    0.000    0.000    0.096    0.032 inline_tools.py:133(inline)
        3    0.073    0.024    0.073    0.024 {apply}
        2    0.000    0.000    0.071    0.036 denoise_colors.py:125(denoise)
        2    0.000    0.000    0.068    0.034 inline_tools.py:361(attempt_function_call)
        1    0.000    0.000    0.050    0.050 denoise_weave.py:59(run)
        3    0.000    0.000    0.035    0.012 Image.py:1615(save)
        1    0.000    0.000    0.029    0.029 denoise_weave.py:26(denoise)
        1    0.000    0.000    0.028    0.028 denoise.py:19(list2img)
       26    0.011    0.000    0.028    0.001 {numpy.core.multiarray.array}
        3    0.000    0.000    0.027    0.009 JpegImagePlugin.py:566(_save)
        3    0.000    0.000    0.027    0.009 ImageFile.py:449(_save)
        1    0.005    0.005    0.027    0.027 denoise.py:8(img2list)
```

For lettere å kunne sammenligne hastigheten på de tre programmene kan det være greit å greppe etter "run".

Terminal

```
> python MyImgTool/profiling.py | grep run --color=always
```

Terminal

```
python MyImgTool/profiling.py | grep run


        1    0.002    0.002    2.107    2.107 denoise_colors.py:181(run)
```

```
     1    0.000    0.000    1.530    1.530 denoise.py:46(run)
     1    0.000    0.000    0.044    0.044 denoise_weave.py:59(run)
```

Jeg har valgt å ikke bruke timeit-modulen da jeg ikke ser hva den kan gi meg som ikke cProfile kan.

3 Utvidelse til farger

Jeg må klare opp en ting helt først her. Jeg misforstod oppgaven. I stedet for å utvide weave-implementasjonen lagde jeg en egen versjon i en egen fil for dette. Som det kommer fram senere i dette dokumentet har ikke det hatt noe å si for sluttbrukeren om man bruker frontenden.

I denne oppgaven har jeg brukt akkurat samme funksjonalitet som i weave-versjonen av svarthvitt denoising. Her kunne jeg faktisk ha importert funksjonen fra denoise_weave.py.

Når det kommer til konvertering mellom fargerom har jeg valgt å bruke numpy til noe den er veldig god på, nemlig å jobbe med store lister med data. Numpy gjør operasjonene i C og Fortran så det skal være vel så raskt.

Før jeg går litt nærmere inn på koden vil jeg nevne en svakhet programmet har på nåværende tidspunkt. Utregningen av H i HSI gjøres ved å kjøre hvert element i rgb-listen gjennom en funksjon, som returnerer verdien for h. Dette er utrolig tregt, og er det eneste som gjør at dette programmet ikke kjører fortere enn denoising av svarthvitt i ren python.

Det som stort sett foregår i konverteringsprosessen er at jeg lager en ny array ved å kopiere en som har riktig størrelse og form (her er det i(ntensity) som brukes som mal for s(aturation)). Så setter jeg et predikat, eller en maske som det heter i numpyverdenen, og ved å indeksere på masken kan jeg tilegne listen, her s, verdier for de elementene hvor predikatet stemmer.

```
s = np.copy(i)
  m = i == 0
  s[m] = 0
```

4 Lineær manipulering

Her er det ikke så mye å si, annet enn at det kan være lurt å kjøre programmene med h-flagget for å se hvordan du setter verdier. For eksempel brukes -u for hue siden h-flagget var tatt for help. De andre kanalene bruker flagg man ville ha gjettet, som r for rød, s for saturation osv.

En kjøring kan gjøres direkte på denoise_colors.py, eller gjennom frontend som vist her:

Terminal

```
> python MyImgTool/frontend.py -r 30 -u -10
```

Kjører man med feil verdier vil programmet avbryte.

Terminal

```
> python MyImgTool/frontend.py -r 30 -s -10

The saturation channel can be adjusted up or down by maximum 1
Your value was -10
```

Legg til d-flagget for å kjøre denoise.

5 Frontend

Alene støtter de tre implementasjonene alle flaggene med unntak av profilering, som jeg har valgt å legge i en egen fil (profiling.py).

Gjennom frontenden derimot, er alt mulig i henhold til oppgaven. Som sagt så har jeg sett bort i fra timeit-modulen, men man kan fortsatt sende med et t-flagg. Da vil cProfile kjøre og man kan teste programmet med de parametere man ønsker.

Ved å kjøre frontend med h-flagget vil man få informasjon om alle muligheter.

Jeg har valgt å se bort fra eps.

6 Testing og dokumentasjon

Alle funksjoner er dokumentert. Når det kommer til tester har jeg dessverre ikke levert på det nivået som var forventet. Det er derimot lagt stor vekt på at alt av funksjonalitet skal være lett å teste. Det lille jeg har av tester viser hvor lett det kan være. Law of Demeter har, og vil alltid være, en viktig rettesnor i så måte.

Testene kan kjøres med pytest, fra oblig03-mappa, på følgende måte:

Terminal

```
> py.test
========================== test session starts ==========================
platform darwin -- Python 2.7.5 -- py-1.4.25 -- pytest-2.6.3
plugins: cov
collected 4 items

tests/denoise_color_test.py ....

======================== 4 passed in 0.25 seconds =======================
```

Eller med nosetests om ønskelig:

Terminal

```
> nosetests
....
----------------------------------------------------------------------
Ran 4 tests in 0.157s

OK
```

Hva gjelder doctesting har vurdert det dithen at det ikke er noe av funksjonaliteten i programmet som er hensiktsmessig å teste på denne måten.

Epilog

Det er mye som kunne vært skrevet om hvordan ting er gjort, men som vanlig stoler jeg på at programmet er oversiktlig nok til at leser uansett vil få mer utbytte av å gå rett til kilde(kode)n. Det vil alltid være mitt fremste mål at koden er så selvdokumenterende som det lar seg gjøre. Tester er en del av denne dokumentasjonen, så det var dumt at jeg ikke fikk til å gjøre den delen bedre.