Extracting JP2 code-stream properties using Kakadu

Johan van der Knijff, KB/National Library of the Netherlands

Problem statement

In Kakadu's 'kdu_show' viewer application it is possible to view an image's code-stream information using the 'Properties' item in the 'File' menu. This short note shows how to extract this information automatically, without any user interaction

Solution: use *kdu_expand* with *–record* switch

The following is a quick and dirty solution, but it may nevertheless be useful.

Basic idea

The basic idea is to use Kakadu's 'kdu_expand' application, which is actually meant to be used for decompressing JPEG 2000 image data. However, the application has an optional switch that will "generate a log file containing all parameter attributes associated with the compressed code-stream". The basic usage is (note how the below command-line defines an input image without defining any output image):

```
kdu_expand -i <inputImage> -record <logFile>
```

E.g.:

```
kdu_expand image0123.jp2 –record properties.txt
```

The file 'properties.txt' now lists all code-stream properties. This is identical to the information given by 'kdu_show', except that code-stream comments[1] and properties of individual tiles are not listed.

Additional performance optimisation

A disadvantage of the above method is that it can be very slow, because it causes 'kdu_expand' to decompress the whole image (even though it doesn't 'do' anything with the decompressed image data, since we didn't define any output image). This can be tweaked using the '-rate' switch, which forces the decoder to only read a portion of the code-stream that corresponds to a user-defined bit-rate. An example:

```
kdu_expand –i image0123.jp2 –rate 0.1 –record properties.txt
```

[1] To extract code-stream comments, use the -com switch ('requests the printing of textual codestream comments'), which will print code-stream comments to screen.

In the above example, the decoder will read that part of the code-stream that corresponds to a bit-rate of 0.1 bits per sample (pixel). By keeping this value sufficiently small, only a small portion of the code-stream is parsed, which greatly speeds up the decoder. For *very* small values, it may not even be possible to read the code-stream headers. If that happens, the decoder exits with an error message[2].

So the generic command line becomes:

```
kdu_expand -i <inputImage> -rate <bitRate> -record <logFile>
```

With <*bitRate*> set to some arbitrary, small value.

[2] I only did some very limited testing on this, but in one of my tests this happened for a value of 0.00001.

Sample output

```
Sprofile=PROFILE2
Scap=no
Sextensions=0
Ssize={7370,5768}
Sorigin={0,0}
Stiles={1024,1024}
Stile_origin={0,0}
Scomponents=3
Ssigned=no,no,no
Sprecision=8,8,8
Ssampling={1,1},{1,1},{1,1}
Sdims={7370,5768},{7370,5768},{7370,5768}
Cycc=yes
Cmct=0
Clayers=10
Cuse_sop=yes
Cuse_eph=yes
Corder=RPCL
Calign_blk_last={no,no}
Clevels=5
Cads=0
Cdfs=0
Cdecomp=B(-:-:-)
Creversible=no
Ckernels=W9X7
Catk=0
Cuse_precincts=yes
Cprecincts={256,256},{256,256},{128,128},{128,128},{128,128},{128,128}
Cblk={64,64}
Cmodes=SEGMARK
Qguard=2
Qderived=no
Qabs_steps=0.000230,0.000228,0.000228,0.000225,0.000458,0.000458,0.0004
54,0.000
934,0.000934,0.000939,0.001956,0.001956,0.002019,0.003862,0.003862,0.00
3755

>> New attributes for tile 0:

>> New attributes for tile 1:

::
::

>> New attributes for tile 47:
```