High Quality Movie Encoding

High Quality Movie Encoding

Once you have rendered frames, there are two steps to encoding a high quality movie. First, the frames may need some last minute adjustments (anti-aliasing, adjusting frame size, applying logos, etc.). Second, one should use the right encoder with the right parameters.

Preparing Frames

Often it is desirable to have anti-aliased images. Jaggies can be distracting, and the distraction can be aggravated during animation. The simple approach is to render frames that are between 2x and 3x larger in each direction than the desired final frame, and then average these down to the correct size.

One approach is to use ImageMagick's command line interface:

```
convert -geometry 50% big_frame.png final_frame.png
```

or

```
convert -geometry 1920x1080 big_frame.png final_frame.png
```

. The first just reduces the big frame by a fixed percentage. If you rendered at 2x, then 50% is perfect. The second is when you don't use an integer scaling number to produce your oversized frames. In this case, whatever the big frame size is, ImageMagick will scale the whole image (preserving the aspect ratio) so that it fits within a box that's 1920x1080. This means that if you used a different aspect ratio, the final frame could be smaller than the vertical or horizontal dimension requested.

Another approach is to write a short script in python, using the Python Imaging library. In practice, this is cleaner and more convenient for large scale conversion than the ImageMagick approach. Put the following code in a file called

```
label.py
```

, and make it executable via

```
chmod u+x label.py
```

```
#!/bin/env python

import sys

##############################################################################
# image stuff and convenience functions

import Image

def width(im):
  return im.size[0]

def height(im):
  return im.size[1]

def aspect(im):
  return 1.0*width(im)/height(im)

##############################################################################

if __name__ == "__main__":

  ########################################
  # handle command line args

  if len(sys.argv) != 1+2:
    print "usage: %s background.png outfile.png" % sys.argv[0]
    exit()
```

```
bgFilename= sys.argv[1]
outFilename= sys.argv[2]

########################################
# load image and logo

bg = Image.open(bgFilename)
if bg.mode != "RGBA":
  bg = bg.convert("RGBA")

lg = Image.open("SLAC_Logo_hires.png")
if lg.mode != "RGBA":
  lg = lg.convert("RGBA")


########################################
# make frame the desired size

scale= 1280/float(width(bg))

bg_width  = int(scale*width(bg))
bg_height = int(scale*height(bg))
bg = bg.resize((bg_width, bg_height), Image.ANTIALIAS)

########################################
# apply the logo

# make logo the desired size
logo_height = int(200*scale)

logo_width = int(aspect(lg)*logo_height)
logo = lg.resize((logo_width, logo_height), Image.ANTIALIAS)

# figure out pixel offsets
xo= width(bg)-width(logo) -10
yo= height(bg)-height(logo) -10

# slap the logo onto the background
bg.paste(logo, (xo, yo), logo)


# uncomment the show() to see image directly on the screen.
# this is useful for debugging.  Be sure it's commented out
# again before running a batch job from the makefile, because you'll
# be drowned in images popping up!
#bg.show()

bg.save(outFilename)
```

When dealing with thousands of frames, it helps to be able to take advantage of multiple cores to do the work. It also helps to be able to interrupt and resume the conversion process without having to start over. A simple Makefile can handle this nicely, regardless of whether one uses ImageMagick or Python Imaging. As an example, if oversized unlabled filenames are of the form

```
scatterts_t000000009470000fs.png
```

and the labled, antialiased frame name is of the form

```
f_scatterts_t000000094700000fs.png
```

, the logo image is in a filed called

```
SLAC_Logo_hires.png
```

, then the following makefile:

```
SOURCES:= $(wildcard scat*.png)
TARGETS:= $(SOURCES:%=f_%)

f_%.png: %.png
        ./label.py $< $@

default: $(TARGETS)
```

should let you run

```
make -j4
```

to build all frames in parallel.

Encoding

The best (in terms of image quality, small file sizes, and portability of the final movie across differing platforms) movies come from H.264 encoding (which is another name for MPEG-4 Part 10. Although the actual encoding is standardized, the result is "packaged" in ways that are often not standardized. For example, there is an x.264 implementation of the standard that makes great preview movies under Linux, but is not honored by Microsoft's player or Apple's player. By staying with the Apple Quicktime Wrapper around Apple's H.264 implementation, we get portability to any place able to play quicktime movies (and that includes Microsoft and Linux platforms).

Apple provided a nice way to get at the H.264 parameters in the Pro version of Quicktime 7. Quicktime 7 is free, and the Pro upgrade is about $30. More recently, Quicktime no longer allows general adjustment of the encoding parameters, instead providing a few defaults that work for portable devices. Now they want you to buy Final Cut to do more interesting compression. But, for now, Quicktime 7 with the Pro upgrade are still available, and serve our purposes very well.

In QuickTime 7 Pro,

Open "Image Sequence", and select the first frame of the sequence. Quicktime will spot the pattern and load all other frames in that sequence as well. For previewing purposes, select the frame rate of your final movie (shoot for 30fps, but use 15 fps if you know that you have to).

Then Export your movie. Check the options carefully. You want:

- H.264
- Multi Pass encoding.
- Automatic Frame Rate.
- Quick Start for streaming
- Current Frame size (don't let the encoder change frame size; it's usually much faster to do this via the makefile/script combo mentioned above).
- Quality should be set to High as a first guess.
  More than this tends to look no different, but create a larger file. Less creates a smaller file, but can start to look bad. Almost always, "High" or slightly less turns out to be best. For laptop presentations, you probably want to keep movie size under 15 or 20 MB. For nice HD movies, you probably want to stay under about 100MB.

save as .mov