Pyctools

A picture processing algorithm development kit

Jim Easterbrook, ex BBC R&D

Some history – mid 1980s

- Very limited RAM and disc capacity
- Monolithic Fortran programs
- Ran as batch jobs overnight – not enough RAM to run simultaneously
- Used “Kingswood picture files”
- Custom video stores for display, e.g. 50MByte in 6 foot tall 19 inch rack

Some history – late 1980s

- More RAM, more disc and more power
- Still centralised – tech memos refer to “the MicroVAX”
- Started breaking programs down into smaller parts with intermediate files
- Experimented with Transputers and Occam – discovered that parallelism is hard

Some history – early 1990s

- UNIX arrived and we got multiple computers
- Started using shell pipes to connect tools running in parallel
- “pic-tools” name first appeared
- “picsubs_tng” C library replaced Fortran library – started using Pascal and C as well as Fortran, also used Modula-2 and Ada later on

What's wrong with pic-tools?

- Slow! Picture data piped from program to program, 4KBytes at a time
- Using “makefile” or shell scripts to connect programs is not very user friendly
- Simple pipelines easy, but anything more complex can easily deadlock
- Very, very non-standard and non-portable
- A mess of different programming languages & dialects

What's right with pic-tools?

- Power and flexibility of connecting simple parts to make complex simulations
- Simple parts are easy to define, test and reuse
- “Audit trail” – every file includes metadata giving its life history
- Pipelining programs gives easy parallelism

Shopping list for a pic-tools replacement

- Exploit existing libraries – build on others' work
- Use standard file formats
- Easy to use, but powerful and flexible
- Easy to extend, both “shared” and “private”
- Need parallelism to benefit from multi-CPU, multi-core computers

Recent history

- Multi-camera capture system used Python with existing C++ “vplibs”
- Used Michael Sparks' Kamaelia to create networks of parallel components
- Since retiring I thought I'd try writing a pic-tools replacement using this experience

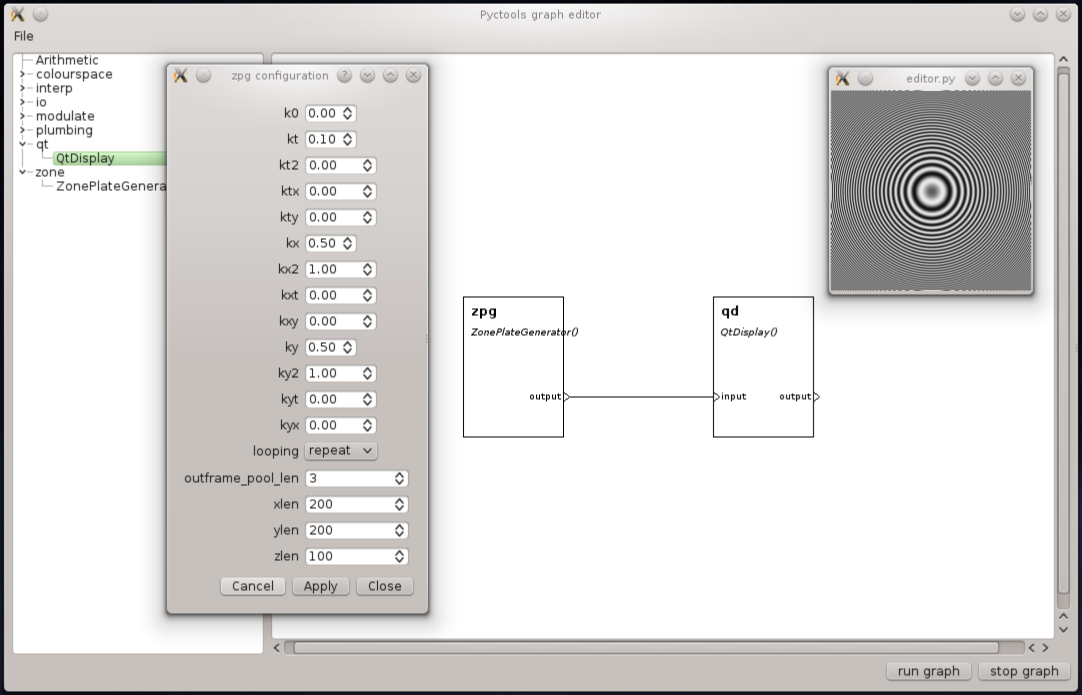
Pyctools, pronounced pic-tools

- Py because it's written in Python
- Uses NumPy, OpenCV, SciPy, Python Imaging Library to process data
- Uses gexiv2 to handle metadata
- Uses ffmpeg to read & write video files
- Open source – get it from GitHub: https://github.com/jim-easterbrook/pyctools

Pyctools uses “components”

- Based on Michael Sparks' Kamaelia replacement Guild
- Components have named input(s) and output(s)
- Unit of work is a single frame
- Connect components to make a pipeline or network
- Make any configuration adjustments
- Press “go” and wait for last components to finish

Pyctools has a GUI!

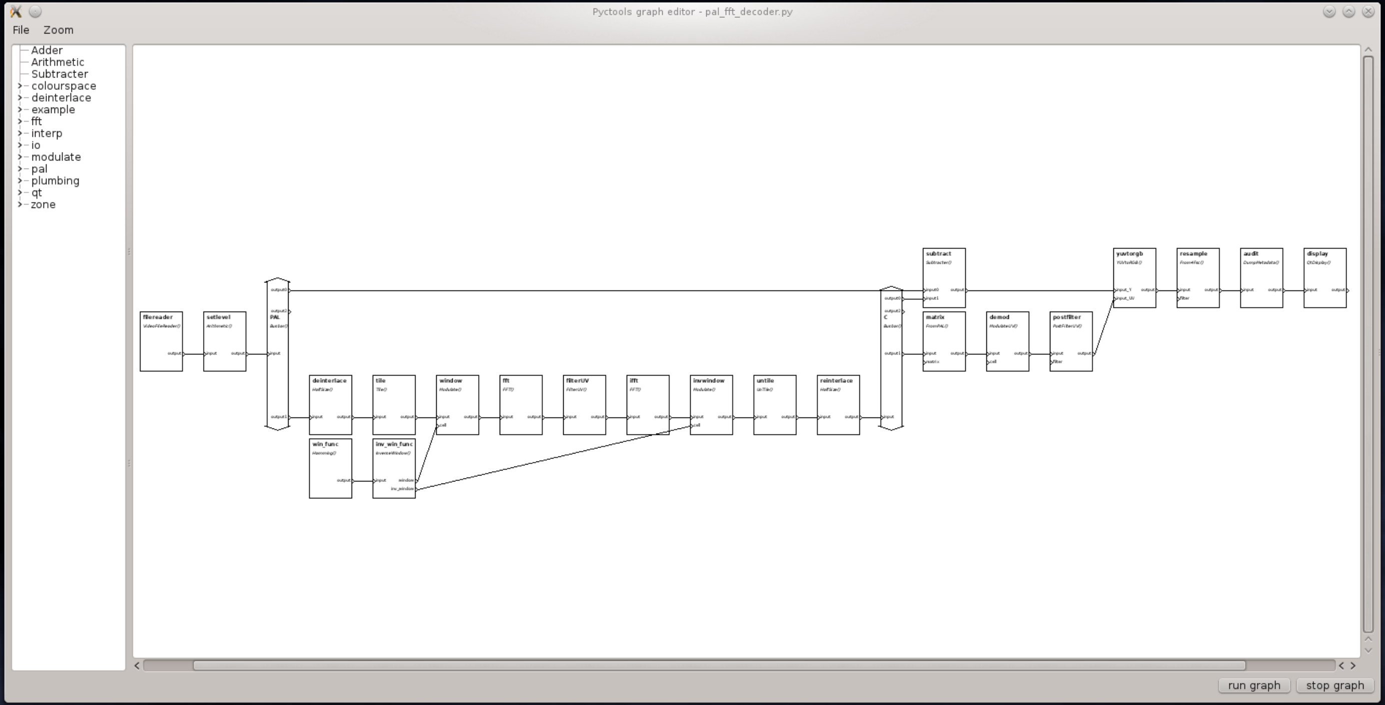


k0 0.00
kt 0.10
kt2 0.00
ktx 0.00
kty 0.00
kx 0.50
kx2 1.00
kxt 0.00
kxy 0.00
ky 0.50
ky2 1.00
kyt 0.00
kyx 0.00
looping repeat
outframe_pool_len 3
xlen 200
ylen 200
zlen 100

Pyctools GUI is easy to use

- All components installed on the computer are available
- Connect components by click & drag
- Configure components with simple dialogs
- Press “run graph” to run it
- Network can be saved as a runnable program
- Demonstration!

Networks can get complicated



But isn't Python slow?

- Python is mostly used for “plumbing”
- Hard work is done in libraries (e.g. NumPy) written in C/C++
- Where no library function is suitable can use Cython – looks like Python but up to 1000x faster
- Write Python first, then gradually convert to Cython

Parallelism

- All components are running in one Python interpreter
- Python is infamously non-parallel – “Global Interpreter Lock” (GIL)
- Components run in Python threads, but only one thread runs at a time *in the Python interpreter*
- C/C++/Cython functions can surrender GIL to allow other threads to run while number crunching continues
- C/C++/Cython can also use OpenMP

Load balancing

- Need to stop any component getting too far ahead
- Original pic-tools used limited capacity pipes
- Each Pyctool component has a finite “pool” of output frames – cannot generate a new output frame until the consuming component releases an earlier one
  - Uses Python “weakref” to notify pool when frame is released (i.e. deleted)

Components are event driven

- “Events” include:
  - An input frame arriving
  - An output frame being available from the “pool”
  - The configuration changing
- Choice of event loops:
  - Normal Python thread
  - Qt event loop – allows component to send Qt signals

Input / Output

- Headerless “raw” files – use separate metadata file to store dimensions and format (UYVY, YUYV, YV12, RGB, BGR, etc.)
- Other formats – use ffmpeg to read and write
  - 8-bit or 16-bit data – ffmpeg does its own conversion
  - Lossless(?) formats FFV1 or H264 with qp=0
  - Can use any encoder options that ffmpeg will accept
- Always create separate metadata file for audit trail

Extending Pyctools

- Core – fork on GitHub, submit pull request
- Components – separate projects add to existing hierarchy, thanks to “namespace packages”
- Can be published (GPL) or private, including partners under NDA

Demonstrations

- Temporal aliasing
- De-interlacing
- PAL decoding (time permitting)