

GRABUI,

the ultimate GUI multi-purpose photograbber

Grabui has been developed in order to ease the control of two Canon cameras that were attached to a stereo microscope. Both were supposed to be fired at the same time and official Windows only drivers weren't capable to perform such a simple action.

Thus we tried Linux and its *gphoto2* and *gphoto2* libraries in order to control the cameras like we wanted. Although we weren't actually able to explicitly express our wish to fire the two cameras simultaneously, we were able to launch two instances of the *gphoto2* at the same time (one in background). However, this is non-trivial (ie. you have to specify port numbers precisely and they change each time you unplug/switch off your hardware, etc.) and one of our requirements was user-friendliness. *gtkam* interface was limited in the same fashion as *gphoto2* itself. So this is how the decision to write another software was made.

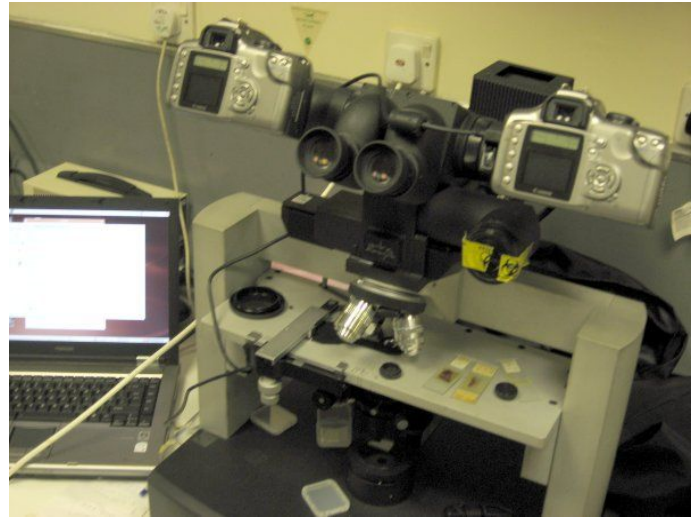
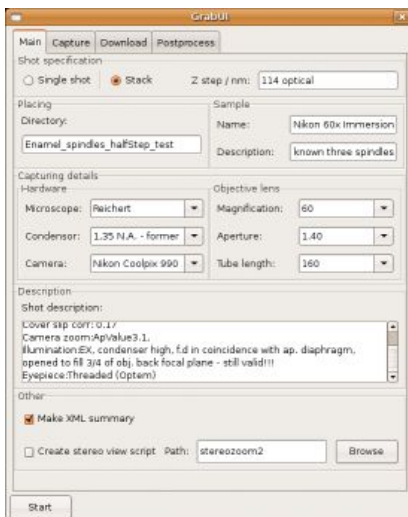


Illustration 1: Edge stereo microscope with dual Canon Eos Digital setup taking advantage of grabui

Due to not very good documentation of *libgphoto2* we have gave up studying the API and we went on another direction – programming a front-end to *gphoto2*. The GUI library *wxWidgets* was chosen.



The application has these features:

- Detects connected cameras automatically and displays their ports
- Optionally generates some user shot documentation
- Generates (and saves :-) all scripts that it is going to launch
- Fires the cameras and displays some sort of a progress dialog
- Downloads the photos (provides a feature to download them from the card later than rather from the camera)
- Applies some post processing (like typical image conversion) and image manipulation

Benefits of **grabui**:

- Easy to use and providing advanced features at the same time
- Every operation is traceable and reproducible
- Open and documented source code

How to use it:

- Main screen:
Here you can specify the directory where pictures are going to be saved, information describing the shot and optionally request their saving and generation of script that should launch a viewing utility.
Info file is saved into the current directory as a XML file
- Capture screen:
Here you can specify shot preferences. The most important thing here is Update ports button – the application will probe your computer and find any cameras connected.
You can specify “additional commands” here – they will be executed as shell commands from the application directory each time after a shot. Useful for turning stepper motor, dynamic aperture etc.
- Download screen:
Filename has this form: <prefix><common name><number (if any)>.<suffix (if any)>
Download files later from the card reader doesn't work yet, sorry.
- Postprocess screen:
Image handling options are useful only and only if you are using RAW image format on the camera.
GIMP connection: The script on the right will be run on every downloaded file (if you wish). Sadly enough, *GIMP* is not able to handle comments inside scripts, so do not enter any... The basic template just fills basic variables and then saves the picture. Place your code above “(gimp-file-save... “ line. There are some function templates with some kind of guidance in the combo box.
If you want ffmpeg to hopefully produce an animation from shots, just tick the checkbox :-)

What does it do:

First of all, the directory where the files will be stored is created. Then, XML file documentation is saved and stereo view script is created if requested. Then the capture script is created from the information provided on capture tab. Then, it is launched (and errors will emerge if you forgot to update ports). Output is used to determine the extension of the files that will be downloaded from the camera afterwards.

Once the capture finishes, the download script (using the data obtained during capturing) and post-processing script are created. Then the download script is executed. The downloaded file is then renamed and placed into the appropriate directory and passed to the post-process script (that one may be empty).

When everything finishes, things should be (hopefully) the same as in the beginning.

The advantage of scripts is that you can launch them any time (from the program executable directory), possibly modify them and thus having an absolute control over the process. One has just to be careful of interface of other applications (like *gphoto2* – many cameras ceased to work when we've “upgraded” from Ubuntu Dapper to Feisty).