UFO Tasks Reference

Release 0.16.0

Matthias Vogelgesang

Jun 29, 2018

Contents

CHAPTER 1

Getting started

1.1 Installation

Prior to building the filter suite you have to install the base library ufo-core as well as all task-specific dependencies (e.g. *libtiff* for `read` and `write`). Once installed you can check out the source with:

```
$ git clone https://github.com/ufo-kit/ufo-filters
```

Configure the build with:

```
$ cd <source-path>
$ cmake .
```

Installation paths can be customized by passing `configure` equivalents like so:

```
$ cmake . -DCMAKE_INSTALL_PREFIX=/usr -DCMAKE_INSTALL_LIBDIR=/usr/lib64
```

Now build and install the filters with:

```
$ make && make install
```

Depending on the installation location, the second step requires administration rights.

CHAPTER 2

Reference

2.1 Generators

Generators produce data and have at least one output but no input.

2.1.1 File reader

class read

The reader loads single files from disk to produce a stream of two-dimensional data items. Supported file types depend on the compiled plugin. Raw (*.raw*) and EDF (*.edf*) files can always be read without additional support. Additionally, loading TIFF (*.tif* and *.tiff*) and HDF5 (*.h5*) files might be supported.

The nominal resolution can be decreased by specifying the `y` coordinate and a `height`. Due to reduced I/O, this can dramatically improve performance.

"path": string
: Glob-style pattern that describes the file path. For HDF5 files this must point to a file and a data set separated by a colon, e.g. `/path/to/file.h5:/my/data/set`.

"number": uint
: Number of files to read.

"start": uint
: First index from where files are read.

"step": uint
: Number of files to skip.

"y": uint
: Vertical coordinate from where to start reading.

"height": uint
: Height of the region that is read from the image.

"y-step": uint
Read every `y-step` row.

"convert": boolean
Convert input data to float elements, enabled by default.

"raw-width": uint
Specifies the width of raw files.

"raw-height": uint
Specifies the height of raw files.

"raw-bitdepth": uint
Specifies the bit depth of raw files.

"raw-pre-offset": ulong
Offset that is skipped before reading the next frame from the current file.

"raw-post-offset": ulong
Offset that is skipped after reading the last frame from the current file.

"type": enum
Overrides the type detection that is based on the file extension. For example, to load *.foo* files as raw files, set the `type` property to *raw*.

"retries": uint
Set the number of retries in case files do not exist yet and are being written. If you set this, you *must* also set `number` otherwise you would have to wait basically forever for the execution to finish. Note, that only files are considered which come after the last successful filename.

"retry-timeout": uint
Seconds to wait before reading new files.

2.1.2 Memory reader

class memory-in
Reads data from a pre-allocated memory region. Unlike input and output tasks this can be used to interface with other code more directly, e.g. to read from a NumPy buffer:

```
from gi.repository import Ufo
import numpy as np
import tifffile


ref = np.random.random((512, 512)).astype(np.float32)

pm = Ufo.PluginManager()
g = Ufo.TaskGraph()
sched = Ufo.Scheduler()
read = pm.get_task('memory-in')
write = pm.get_task('write')

read.props.pointer = ref.__array_interface__['data'][0]
read.props.width = ref.shape[1]
read.props.height = ref.shape[0]
read.props.number = 1

write.props.filename = 'out.tif'
```

(continues on next page)

(continued from previous page)

```
g.connect_nodes(read, write)
sched.run(g)

out = tifffile.imread('out.tif')
assert np.sum(out - ref) == 0.0
```

"pointer": ulong
Pointer to pre-allocated memory.

"width": uint
Specifies the width of input.

"height": uint
Specifies the height of input.

"number": uint
Specifies the number of items to read.

2.1.3 ZeroMQ subscriber

class zmq-sub
Generates a stream from a compatible ZeroMQ data stream, for example published by the `zmq-pub` task.

"address": string
Host address of the ZeroMQ publisher. Note, that as of now the publisher binds to a `tcp` endpoint, thus you have to use that as well. By default, the address is set to the local host address 127.0.0.1.

2.1.4 UcaCamera reader

class camera
The camera task uses libuca to read frames from a connected camera and provides them as a stream.

When `name` is provided, the corresponding plugin is instantiated by the camera task itself. However, an already configured UcaCamera object can also be passed via `camera`.

"name": string
Name of the camera that is used.

"number": uint
Number of frames that are recorded.

"properties": string
Property string, i.e. `roi-width=512 exposure-time=0.1`.

Note: This requires third-party library *libuca*.

2.1.5 stdin reader

class stdin
Reads data from stdin to produce a valid data stream. `width`, `height` and `bitdepth` must be set correctly to ensure correctly sized data items.

"width": uint
Specifies the width of input.

"height": uint
Specifies the height of input.

"bitdepth": uint
Specifies the bit depth of input.

"convert": boolean
Convert input data types to float, enabled by default.

2.1.6 Metaball simulation

class metaballs
Generate animated meta balls. In each time step the meta balls move by a random velocity.

"width": uint
Width of output data stream.

"height": uint
Height of output data stream.

"number-balls": uint
Number of meta balls.

"number": uint
Length of data stream.

2.1.7 Data generation

class dummy-data
Only asks for image data sized `width` times `height` times `depth` and forwards `number` of them to the next filter. The data is never touched if `init` is not set, thus it might be suitable for performance measurements.

"width": uint
Width of image data stream.

"height": uint
Height of image data stream.

"depth": uint
Depth of image data stream.

"number": uint
Number of images to produce.

"init": float
Value to initialize the output buffer.

2.2 Filters

Filters transform data and have at least one input and one output.

2.2.1 Point-based transformation

Binarization

class binarize
Binarizes an image.

"threshold": float
Any values above the threshold are set to one all others to zero.

Clipping

class clip
Clip input to set minimum and maximum value.

"min": float
Minimum value, all values lower than *min* are set to *min*.

"max": float
Maximum value, all values higher than *max* are set to *max*.

Masking

class mask
Mask the circular outer region by setting values to zero.

Arithmetic expressions

class calculate
Calculate an arithmetic expression. You have access to the value stored in the input buffer via the *v* letter in `expression` and to the index of *v* via letter *x*. Please be aware that *v* is a floating point number while *x* is an integer. This is useful if you have multidimensional data and want to address only one dimension. Let's say the input is two dimensional, 256 pixels wide and you want to fill the x-coordinate with *x* for all respective y-coordinates (a gradient in x-direction). Then you can write *expression="x % 256"*. Another example is the *sinc* function which you would calculate as *expression="sin(v) / x"* for 1D input. For more complex math or other operations please consider using *OpenCL*.

"expression": string
Arithmetic expression with math functions supported by OpenCL.

Statistics

class measure
Measure basic image properties.

"metric": string
Metric, one of `min`, `max`, `sum`, `mean`, `var`, `std`, `skew` or `kurtosis`.

"axis": int
Along which axis to measure (-1, all).

Generic OpenCL

class opencl

Load an arbitrary OpenCL `kernel` from `filename` or `source` and execute it on each input. The kernel must accept as many global float array parameters as connected to the filter and one additional as an output. For example, to compute the difference between two images, the kernel would look like:

```
kernel void difference (global float *a, global float *b, global float *c)
{
    size_t idx = get_global_id (1) * get_global_size (0) + get_global_id (0);
    c[idx] = a[idx] - b[idx];
}
```

and could be used like so if defined in a file named `diff.cl`:

```
$ ufo-launch [read, read] ! opencl kernel=difference filename=diff.cl !  null
```

If `filename` is not set, a default kernel file (`opencl.cl`) is loaded. See *OpenCL default kernels* for a list of kernel names defined in that file.

"filename": string
Filename with kernel sources to load.

"source": string
String with OpenCL kernel code.

"kernel": string
Name of the kernel that this filter is associated with.

"options": string
OpenCL build options.

"dimensions": uint
Number of dimensions the kernel works on. Must be in [1, 3].

2.2.2 Spatial transformation

Transposition

class transpose

Transpose images from (x, y) to (y, x).

Rotation

class rotate

Rotates images clockwise by an `angle` around a `center` (x, y). When `reshape` is `True`, the rotated image is not cropped, i.e. the output image size can be larger that the input size. Moreover, this mode makes sure that the original coordinates of the input are all contained in the output so that it is easier to see the rotation in the output. Try e.g. rotation with `center` equal to $(0, 0)$ and angle $\pi/2$.

"angle": float
Rotation angle in radians.

"reshape": boolean
Reshape the result to encompass the complete input image and input indices.

"center": GValueArray
Center of rotation (x, y)

"addressing-mode": enum
Addressing mode specifies the behavior for pixels falling outside the original image. See OpenCL `sampler_t` documentation for more information.

"interpolation": enum
Specifies interpolation when a computed pixel coordinate falls between pixels, can be *nearest* or *linear*.

Flipping

class flip
Flips images vertically or horizontally.

"direction": enum
Can be either *horizontal* or *vertical* and denotes the direction along with the image is flipped.

Binning

class bin
Bin a square of pixels by summing their values.

"size": uint
Number of pixels in one direction to bin to a single pixel value.

Rescaling

class rescale
Rescale input data by a fixed `factor`.

"factor": float
Fixed factor for scaling the input in both directions.

"x-factor": float
Fixed factor for scaling the input width.

"y-factor": float
Fixed factor for scaling the input height.

"width": uint
Fixed width, disabling scalar rescaling.

"height": uint
Fixed height, disabling scalar rescaling.

"interpolation": enum
Interpolation method used for rescaling which can be either `nearest` or `linear`.

Padding

class pad
Pad an image to some extent with specific behavior for pixels falling outside the original image.

"x": int
Horizontal coordinate in the output image which will contain the first input column.

"y": int
Vertical coordinate in the output image which will contain the first input row.

"width": uint
Width of the padded image.

"height": uint
Height of the padded image.

"addressing-mode": enum
Addressing mode specifies the behavior for pixels falling outside the original image. See OpenCL `sampler_t` documentation for more information.

Cropping

class crop
Crop a region of interest from two-dimensional input. If the region is (partially) outside the input, only accessible data will be copied.

"x": uint
Horizontal coordinate from where to start the ROI.

"y": uint
Vertical coordinate from where to start the ROI.

"width": uint
Width of the region of interest.

"height": uint
Height of the region of interest.

"from-center": boolean
Start cropping from the center outwards.

Cutting

class cut
Cuts a region from the input and merges the two halves together. In a way, it is the opposite of crop.

"width": uint
Width of the region to cut out.

Tiling

class tile
Cuts input into multiple tiles. The stream contains tiles in a zig-zag pattern, i.e. the first tile starts at the top left corner of the input goes on the same row until the end and continues on the first tile of the next row until the final tile in the lower right corner.

"width": uint
Width of a tile which must be a divisor of the input width. If this is not changed, the full width will be used.

"height": uint
Width of a tile which must be a divisor of the input height. If this is not changed, the full height will be used.

Swapping quadrants

class swap-quadrants
Cuts the input into four quadrants and swaps the lower right with the upper left and the lower left with the upper right quadrant.

Polar transformation

class polar-coordinates
Transformation between polar and cartesian coordinate systems.

When transforming from cartesian to polar coordinates the origin is in the image center (`width` / 2, `height` / 2). When transforming from polar to cartesian coordinates the origin is in the image corner (0, 0).

"width": uint
Final width after transformation.

"height": uint
Final height after transformation.

"direction": string
Conversion direction from `polar_to_cartesian`.

Stitching

class stitch
Stitches two images horizontally based on their relative given `shift`, which indicates how much is the second image shifted with respect to the first one, i.e. there is an overlapping region given by $first\_width - shift$. First image is inserted to the stitched image from its left edge and the second image is inserted after the overlapping region. If shift is negative, the two images are swapped and stitched as described above with shift made positive.

If you are stitching a 360-degree off-centered tomographic data set and know the axis of rotation, shift can be computed as $2axis - second\_width$ for the case the axis of rotation is greater than half of the first image. If it is less, then the shift is $first\_width - 2axis$. Moreover, you need to horizontally flip one of the images because this task expects images which can be stitched directly, without additional needed transformations.

Stitching requires two inputs. If you want to stitch a 360-degree off-centered tomographic data set you can use:

```
ufo-launch [read path=projections_left/, read path=projections_right/ ! flip␣
↪direction=horizontal] ! stitch shift=N ! write filename=foo.tif
```

"shift": int
How much is second image shifted with respect to the first one. For example, shift 0 means that both images overlap perfectly and the stitching doesn't actually broaden the image. Shift corresponding to image width makes for a stitched image with twice the width of the respective images (if they have equal width).

"adjust-mean": boolean
Compute the mean of the overlapping region in the two images and adjust the second image to match the mean of the first one.

"blend": boolean
Linearly interpolate between the two images in the overlapping region.

2.2.3 Multi-stream

Interpolation

class interpolate
Interpolates incoming data from two compatible streams, i.e. the task computes $(1-\alpha)s_1 + \alpha s_2$ where s_1 and s_2 are the two input streams and α a blend factor. α is $i/(n-1)$ for $n > 1$, n being `number` and i the current iteration.

"number": uint
Number of total output stream length.

class interpolate-stream
Interpolates between elements from an incoming stream.

"number": uint
Number of total output stream length.

Subtract

class subtract
Subtract data items of the second from the first stream.

Correlate

class correlate-stacks
Reads two datastreams, the first must provide a 3D stack of images that is used to correlate individual 2D images from the second datastream. The `number` property must contain the expected number of items in the second stream.

"number": uint
Number of data items in the second data stream.

2.2.4 Filters

Median

class median-filter
Filters input with a simple median.

"size": uint
Odd-numbered size of the neighbouring window.

Edge detection

class detect-edge
Detect edges by computing the power gradient image using different edge filters.

"filter": enum
Edge filter (or operator) which is one of `sobel`, `laplace` and `prewitt`. By default, the `sobel` operator is used.

Gaussian blur

class blur
Blur image with a gaussian kernel.

"size": uint
Size of the kernel.

"sigma": float
Sigma of the kernel.

Gradient

class gradient
Compute gradient.

"direction": enum
Direction of the gradient, can be either `horizontal`, `vertical`, `both` or `both_abs`.

2.2.5 Stream transformations

Averaging

class average
Read in full data stream and generate an averaged output.

"number": uint
Number of averaged images to output. By default one image is generated.

Reducing with OpenCL

class opencl-reduce
Reduces or folds the input stream using a generic OpenCL kernel by loading an arbitrary `kernel` from `filename` or `source`. The kernel must accept exactly two global float arrays, one for the input and one for the output. Additionally a second `finish` kernel can be specified which is called once when the processing finished. This kernel must have two arguments as well, the global float array and an unsigned integer count. Folding (i.e. setting the initial data to a known value) is enabled by setting the `fold-value`.

Here is an OpenCL example how to compute the average:

```
kernel void sum (global float *in, global float *out)
{
    size_t idx = get_global_id (1) * get_global_size (0) + get_global_id (0);
    out[idx] += in[idx];
}

kernel void divide (global float *out, uint count)
{
    size_t idx = get_global_id (1) * get_global_size (0) + get_global_id (0);
    out[idx] /= count;
}
```

And this is how you would use it with `ufo-launch`:

```
ufo-launch ... ! opencl-reduce kernel=sum finish=divide ! ...
```

If `filename` is not set, a default kernel file is loaded. See *OpenCL reduction default kernels* for a list of possible kernels.

"filename": string
Filename with kernel sources to load.

"source": string
String with OpenCL kernel code.

"kernel": string
Name of the kernel that is called on each iteration. Must have two global float array arguments, the first being the input, the second the output.

"finish": string
Name of the kernel that is called at the end after all iterations. Must have a global float array and an unsigned integer arguments, the first being the data, the second the iteration counter.

"fold-value": float
If given, the initial data is filled with this value, otherwise the first input element is used.

"dimensions": uint
Number of dimensions the kernel works on. Must be in [1, 3].

Statistics

class flatten
Flatten input stream by reducing with operation based on the given mode.

"mode": string
Operation, can be either `min`, `max`, `sum` and `median`.

class flatten-inplace
Faster inplace operating variant of the `flatten` task.

"mode": enum
Operation, can be either `min`, `max` and `sum`.

Slicing

class slice
Slices a three-dimensional input buffer to two-dimensional slices.

Stacking

class stack
Symmetrical to the slice filter, the stack filter stacks two-dimensional input.

"number": uint
Number of items, i.e. the length of the third dimension.

Merging

class merge
Merges the data from two or more input data streams into a single data stream by concatenation.

"number": uint
Number of input streams. By default this is two.

Slice mapping

class map-slice
Lays out input images on a quadratic grid. If the `number` of input elements is not the square of some integer value, the next higher number is chosen and the remaining data is blackened.

"number": uint
Number of expected input elements. If more elements are sent to the mapper, warnings are issued.

Color mapping

class map-color
Receives a two-dimensional image and maps its gray values to three red, green and blue color channels using the Viridis color map.

Splitting channels

class unsplit
Turns a three-dimensional image into two-dimensional image by interleaving the third dimension, i.e. [[[XXX],[YYY],[ZZZ]]] is turned into [[XYZ],[XYZ],[XYZ]]. This is useful to merge a separate multi-channel RGB image into a "regular" RGB image that can be shown with `cv-show`.

This task adds the `channels` key to the output buffer containing the original depth of the input buffer.

2.2.6 Fourier domain

Fast Fourier transform

class fft
Compute the Fourier spectrum of input data. If `dimensions` is one but the input data is 2-dimensional, the 1-D FFT is computed for each row.

"auto-zeropadding": boolean
Automatically zeropad input data to a size to the next power of 2.

"dimensions": uint
Number of dimensions in [1, 3].

"size-x": uint
Size of FFT transform in x-direction.

"size-y": uint
Size of FFT transform in y-direction.

"size-z": uint
Size of FFT transform in z-direction.

class ifft
Compute the inverse Fourier of spectral input data. If `dimensions` is one but the input data is 2-dimensional, the 1-D FFT is computed for each row.

"dimensions": uint
Number of dimensions in [1, 3].

"crop-width": int
Width to crop output.

"crop-height": int
Height to crop output.

Frequency filtering

class filter
Computes a frequency filter function and multiplies it with its input, effectively attenuating certain frequencies.

"filter ": enum
Any of `ramp`, `ramp-fromreal`, `butterworth`, `faris-byer`, `hamming` and `bh3` (Blackman-Harris-3). The default filter is `ramp-fromreal` which computes a correct ramp filter avoiding offset issues encountered with naive implementations.

"scale": float
Arbitrary scale that is multiplied to each frequency component.

"cutoff": float
Cutoff frequency of the Butterworth filter.

"order": float
Order of the Butterworth filter.

"tau": float
Tau parameter of Faris-Byer filter.

"theta": float
Theta parameter of Faris-Byer filter.

1D stripe filtering

class filter-stripes1d
Filter stripes in 1D along the x-axis. The input and output are in frequency domain. The filter multiplies the frequencies with an inverse Gaussian profile centered at 0 frequency. The inversed profile means that the filter is f(k) = 1 - gauss(k) in order to suppress the low frequencies.

"strength": float
Filter strength, which is the full width at half maximum of the gaussian.

Zeropadding

class zeropad
Add zeros in the center of sinogram using `oversampling` to manage the amount of zeros which will be added.

"oversampling": uint
Oversampling coefficient.

"center-of-rotation": float
Center of rotation of sample.

2.2.7 Reconstruction

Flat-field correction

class flat-field-correct
Computes the flat field correction using three data streams:

1. Projection data on input 0
2. Dark field data on input 1
3. Flat field data on input 2

"absorption-correct": boolean
If *TRUE*, compute the negative natural logarithm of the flat-corrected data.

"fix-nan-and-inf": boolean
If *TRUE*, replace all resulting NANs and INFs with zeros.

"sinogram-input": boolean
If *TRUE*, correct only one line (the sinogram), thus darks are flats are 1D.

"dark-scale": float
Scale the dark field prior to the flat field correct.

Sinogram transposition

class transpose-projections
Read a stream of two-dimensional projections and output a stream of transposed sinograms. `number` *must* be set to the number of incoming projections to allocate enough memory.

"number": uint
Number of projections.

> **Warning:** This is a memory intensive task and can easily exhaust your system memory. Make sure you have enough memory, otherwise the process will be killed.

Tomographic backprojection

class backproject
Computes the backprojection for a single sinogram.

"num-projections": uint
Number of projections between 0 and 180 degrees.

"offset": uint
Offset to the first projection.

"axis-pos": double
Position of the rotation axis in horizontal pixel dimension of a sinogram or projection. If not given, the center of the sinogram is assumed.

"angle-step": double
Angle step increment in radians. If not given, pi divided by height of input sinogram is assumed.

"angle-offset": double
Constant angle offset in radians. This determines effectively the starting angle.

"mode": enum
Reconstruction mode which can be either `nearest` or `texture`.

"roi-x": uint
Horizontal coordinate of the start of the ROI. By default 0.

"roi-y": uint
Vertical coordinate of the start of the ROI. By default 0.

"roi-width": uint
Width of the region of interest. The default value of 0 denotes full width.

"roi-height": uint
Height of the region of interest. The default value of 0 denotes full height.

Forward projection

class forwardproject
Computes the forward projection of slices into sinograms.

"number": uint
Number of final 1D projections, that means height of the sinogram.

"angle-step": float
Angular step between two adjacent projections. If not changed, it is simply pi divided by `number`.

Laminographic backprojection

class lamino-backproject
Backprojects parallel beam computed laminography projection-by-projection into a 3D volume.

"region-values": int
Elements in regions.

"float-region-values": float
Elements in float regions.

"x-region": GValueArray
X region for reconstruction as (from, to, step).

"y-region": GValueArray
Y region for reconstruction as (from, to, step).

"z": float
Z coordinate of the reconstructed slice.

"region": GValueArray
Region for the parameter along z-axis as (from, to, step).

"projection-offset": GValueArray
Offset to projection data as (x, y) for the case input data is cropped to the necessary range of interest.

"center": GValueArray
Center of the volume with respect to projections (x, y), (rotation axes).

"overall-angle": float
Angle covered by all projections (can be negative for negative steps in case only num-projections is specified)

"num-projections": uint
Number of projections.

"tomo-angle": float
Tomographic rotation angle in radians (used for acquiring projections).

"lamino-angle": float
Absolute laminogrpahic angle in radians determining the sample tilt.

"roll-angle": float
Sample angular misalignment to the side (roll) in radians (CW is positive).

"parameter": enum
Which paramter will be varied along the z-axis, from `z`, `x-center`, `lamino-angle`, `roll-angle`.

Fourier interpolation

class dfi-sinc
Computes the 2D Fourier spectrum of reconstructed image using 1D Fourier projection of sinogram (fft filter must be applied before). There are no default values for properties, therefore they should be assigned manually.

"kernel-size": uint
The length of kernel which will be used in interpolation.

"number-presampled-values": uint
Number of presampled values which will be used to calculate `kernel-size` kernel coefficients.

"roi-size": int
The length of one side of region of Interest.

"angle-step": double
Increment of angle in radians.

Center of rotation

class center-of-rotation
Compute the center of rotation of input sinograms.

"angle-step": double

Step between two successive projections.

"center": double
The calculated center of rotation.

Sinogram offset shift

class cut-sinogram
Shifts the sinogram given a center not centered to the input image.

"center-of-rotation": float
Center of rotation of specimen.

Phase retrieval

class retrieve-phase
Computes and applies a fourier filter to correct phase-shifted data. Expects frequencies as an input and produces frequencies as an output.

"method": enum
Retrieval method which is one of `tie`, `ctf`, `ctfhalfsine`, `qp`, `qphalfsine` or `qp2`.

"energy": float
Energy in keV.

"distance": float
Distance in meter.

"pixel-size": float
Pixel size in meter.

"regularization-rate": float
Regularization parameter is log10 of the constant to be added to the denominator to regularize the singularity at zero frequency: 1/sin(x) -> 1/(sin(x)+10^-RegPar).

Typical values [2, 3].

"thresholding-rate": float
Parameter for Quasiparticle phase retrieval which defines the width of the rings to be cropped around the zero crossing of the CTF denominator in Fourier space.

Typical values in [0.01, 0.1], `qp` retrieval is rather independent of cropping width.

2.2.8 General matrix-matrix multiplication

class gemm
Computes $\alpha A \cdot B + \beta C$ where A, B and C are input streams 0, 1 and 2 respectively. A must be of size $m \times k$, B $k \times n$ and C $m \times n$.

Note: This filter is only available if CLBlast support is available.

"alpha": float
Scalar multiplied with AB.

"beta": float
Scalar multiplied with C.

2.2.9 Segmentation

class segment
Segments a stack of images given a field of labels using the random walk algorithm described in[1]. The first input stream must contain three-dimensional image stacks, the second input stream a label image with the same width and height as the images. Any pixel value other than zero is treated as a label and used to determine segments in all directions.

2.2.10 Auxiliary

Buffering

class buffer
Buffers items internally until data stream has finished. After that all buffered elements are forwarded to the next

[1] Lösel and Heuveline, *Enhancing a Diffusion Algorithm for 4D Image Segmentation Using Local Information* in Proc. SPIE 9784, Medical Imaging 2016, http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=2506235

task.

"number": uint
Number of pre-allocated buffers.

"dup-count": uint
Number of times each image should be duplicated.

"loop": boolean
Duplicates the data in a loop manner `dup-count` times.

Stamp

class stamp
Writes the current iteration into the top-left corner.

"font": string
Pango font description, by default set to `Mono 9`.

"scale": float
Scales the default brightness of 1.0.

Note: This filter requires Pango and Cairo for text layouting.

Loops

class loop
Repeats output of incoming data items. It uses a low-overhead policy to avoid unnecessary copies. You can expect the data items to be on the device where the data originated.

"number": uint
Number of iterations for each received data item.

Monitoring

class monitor
Inspects a data stream and prints size, location and associated metadata keys on stdout.

"print": uint
If set print the given numbers of items on stdout as hexadecimally formatted numbers.

Sleep

class sleep
Wait `time` seconds before continuing. Useful for debugging throughput issues.

"time": double
Time to sleep in seconds.

Display

class cv-show
Shows the input using an OpenCV window.

"min": float
Minimum for display value scaling. If not set, will be determined at run-time.

"max": float
Maximum for display value scaling. If not set, will be determined at run-time.

2.3 Sinks

Sinks are endpoints and have at least one input but no output.

2.3.1 File writer

class write
Writes input data to the file system. Support for writing depends on compile support, however raw (*.raw*) files can always be written. TIFF (*.tif* and *.tiff*), HDF5 (*.h5*) and JPEG (*.jpg* and *.jpeg*) might be supported additionally.

"filename": string
Format string specifying the location and filename pattern of the written data. It must contain at most *one* integer format specifier that denotes the current index of a series. For example, `"data-%03i.tif"` produces `data-001.tif`, `data-002.tif` and so on. If no specifier is given, the data is written preferably to a single file (i.e. multi-tiff, HDF5 data set). If no filename is given the data is written as-is to stdout.

"counter-start": uint
Sets the counter that replaces the format specifier. Initially, it is set to 0.

"counter-step": uint
Determines the number of steps the counter replacing the format specifier is incremented. Initially, it is set to 1.

"append": boolean
Append rather than overwrite if `TRUE`.

"bits": uint
Number of bits to store the data if applicable to the file format. Possible values are 8 and 16 which are saved as integer types and 32 bit float. By default, the minimum and maximum for scaling is determined automatically, however depending on the use case you should override this with the `minimum` and `maximum` properties. To avoid rescaling, set the `rescale` property to `FALSE`.

"minimum": float
This value will represent zero for discrete bit depths, i.e. 8 and 16 bit.

"minimum": float
This value will represent the largest possible value for discrete bit depths, i.e. 8 and 16 bit.

"rescale": boolean
If `TRUE` and `bits` is set to a value less than 32, rescale values either by looking for minimum and maximum values or using the values provided by the user.

For JPEG files the following property applies:

"jpeg-quality": uint
JPEG quality value between 0 and 100. Higher values correspond to higher quality and larger file sizes.

2.3.2 Memory writer

class memory-out
Writes input to a given memory location. Unlike input and output tasks this can be used to interface with other code more directly, e.g. to write into a NumPy buffer:

```
from gi.repository import Ufo
import numpy as np
import tifffile

ref = tifffile.imread('data.tif')
a = np.zeros_like(ref)

pm = Ufo.PluginManager()
g = Ufo.TaskGraph()
sched = Ufo.Scheduler()
read = pm.get_task('read')
out = pm.get_task('memory-out')

read.props.path = 'data.tif'
out.props.pointer = a.__array_interface__['data'][0]
out.props.max_size = ref.nbytes

g.connect_nodes(read, out)
sched.run(g)

assert np.sum(a - ref) == 0.0
```

"pointer": ulong
Pointer to pre-allocated memory.

"max-size": ulong
Size of the pre-allocated memory area in bytes. Data is written up to that point only.

2.3.3 ZeroMQ publisher

class zmq-pub
Publishes the stream as a ZeroMQ data stream to compatible ZeroMQ subscribers such as the `zmq-sub` source.

"expected-subscribers": uint
If set, the publisher will wait until the number of expected subscribers have connected.

2.3.4 Auxiliary sink

2.3.5 Null

class null
Eats input and discards it.

"download": boolean
If *TRUE* force final data transfer from device to host if necessary.

"finish": boolean
Call finish on the associated command queue.

"durations": boolean
Print durations computed from timestamps on `stderr`.

2.4 PIV filters

Filters related to the PIV particle tracking software.

2.4.1 Ring pattern

class ring-pattern
This generator is used to create all the patterns that one wants to recognize. In this case, only ring patterns are generated and the only difference between each pattern is it's radii size. The thickness of the rings stays identical no matter the radii size.

The `ring-start` and `ring-end` represent the range of radii used for the ring generations and are given on a pixel basis. Each of these rings will have a thickness of `ring-thickness` pixels. Using a low value for the ring thickness tends to result in more rings being detected. Ideally this value should be the same as the actual ring thickness within the image.

"ring-start": uint
Gives the size of the radius of the first ring to be generated. The size is given on a pixel basis.

"ring-step": uint
Indicates by how much the radii should be increased at each iteration. The value is given on a pixel basis.

"ring-end": uint
Gives the size of the radius of the last ring to be generated. The size is given on a pixel basis.

"ring-thickness": uint
Specifies the desired thickness of the generated ring on a pixel basis.

"width": uint
Give x size of output image.

"height": uint
Give y size of output image.

2.4.2 Concatenate

class concatenate-result
For each image, there are `(ring-end - ring-start + 1) / ring-step` streams of data. Each stream represents a set of detected rings. The concatenate plugin groups these results into one big stream of ring coordinates. This stream is then passed to a set of post processing plug-ins that try to remove false positives and find the most accurate ring possible.

Input A 1D stream. This stream represents the list of all the rings detected for a certain radius and a certain image. Of course if their are 10 different radii sizes, then 10 calls to the input buffer will result into a single call to the output buffer.

Output One list of coordinates, corresponding to all the rings of the current image being processed.

"max-count": uint
Sometimes for small rings patterns hundreds of rings can be detected due to the noise. When large amounts of rings are detected, most of them tend to be false positives. To ignore those rings, set the `max-count`. Note that if it is set to a very high value (over 200) the post processing algorithms might considerably slow down the software.

"ring-count": uint
The maximum number of rings desired per ring pattern.

2.4.3 Denoise

class denoise
A temporary background image is computed from the input image. For each pixel in the input image, the neighbouring pixels are loaded into memory and then sorted in ascending order. The 30th percentile is then loaded into the background image. The input image is then subtracted by this background image. The advantage of this algorithm is to create a new image whose intensity level is homogeneously spread across the whole image. Indeed, the objective here is to remove all background noise and keep the rings whose intensities are always higher than the background noise. This filter later helps the Hough Transform because when noise will be summed up, the overall value will be close to zero instead of having a high value if we had not removed this background noise.

Input A 2D stream. The image taken by the CMOSs camera.

Output A 2D stream. This plug-in computes an average background image of the input. The output image is then created by subtracting the input image to this background image.

"matrix-size": uint
This parameter specifies the size of the matrix used when looking for neighbouring pixels. A bigger value for the matrix size means that more pixels will be compared at a time. Ideally, the size should be twice as big as the desired `ring-thickness`. The ring thickness is the number of pixels that can be seen on the rings edge. If the size is identical to or less than the effective ring thickness, pixels within rings in the image might get removed (i.e. set to 0).

2.4.4 Contrast

class contrast
It has been noticed in an empirical way that the rings always stand in the high frequencies of the images, i.e. the pixels with higher intensities. Moreover, only a small amount of the pixels, around 10%, form all the rings in the image. Hence a histogram is computed to know where most of the pixels stand. As a general rule, it was noticed that every pixels that are below the peak in the histogram are simply background noise. This is why each pixel below this peak is set to 0. To make the ring stand out a bit more a non linear mapping is made to enhance the bright pixels even more. By using the *imadjust* algorithm as described in matlab, we compute the new pixel values using the following formula : $f'(x,y) = \left(\frac{f(x,y)-low}{high-low}\right)^{\gamma}$ Where f' is the output image, f is the input image, $high$ is the maximum value and low is the smallest value. γ is a value less than 1, and is what allows to get a non linear mapping and more values near the high intensities.

Input A 2D stream. The image is the previously denoised image.

Output A 2D stream. All low intensities have been removed and the rings contrast has been increased.

"remove-high": boolean
When this parameter is set true, every pixel in the histogram that lie between half of the distance of the peak and the maximum and the maximum value are replaced by a value of 0. This can be useful when the image has lots of bright regions which cause a lot of noise and hence generating many false positives.

2.4.5 Ordfilt

class ordfilt

The plug-in matches a pattern over each pixel of the image and computes a value representing the likeliness for that pixel to be the center of that pattern. To achieve this, every pixel that lie under the pattern are loaded into memory and then sorted. Once the array is sorted two values are picked to compute the rings contrast and the rings average intensities. Currently we pick the 25th and 50th percentile pixel value. The following formula is then applied to get the new pixel value:

$$contrast = 1 - (high_p - low_p)$$
$$intensity = \frac{(high_p + low_p)}{2}$$
$$f'(x, y) = intensity \cdot contrast$$

$high_p$ is the 50 percentile pixel value. low_p is the 25th percentile pixel value. This formula is based on the fact that rings are always brighter, hence the more bright the pixels the more likely we have a ring. Moreover, the pixels forming the ring should not vary in intensity, i.e. the low and high percentile should have the same value, by computing the difference we can compute a good contrast value of the ring. The resulting image therefor takes into consideration both the contrast of the ring and its intensity.

Input 1 A 2D stream. The previously contrasted image.

Input 2 A 2D stream. An image representing a pattern to match. In our case, the pattern is a ring.

Output A 2D stream. An image where each pixel value represents the likeliness for that pixel to be the center of the current pattern passed in input1.

2.4.6 Particle filtering

class filter-particle

This algorithm is based on two-pass method to detect blobs. A blob, is a set of bright pixels that form a bright spot on the image. Each pixel in a blob has sufficiently high enough value, based on a threshold, such as that pixel is a candidate to being the center of the ring-pattern being currently searched for. For each of these blobs, a unique (x, y, r) value is computed. Where (x, y) is the center of the blob of pixels and r is the radius of the current ring-pattern being searched for.

Input A 2D stream. The image generated by the ordfilt, where each pixel value represents the likeliness of it to become the center of a ring.

Output A 1D stream. An array of (x, y, r) coordinates representing the list of currently detected rings.

"threshold": float
A value between 0 and 1 representing a threshold relative to the images maximum value. Each pixel of the image whose value is greater than $threshold \cdot \max(Image)$ is considered as a candidate to being a center of a ring.

"min": float
Gives the minimum value a pixels needs to have to be considered a possible candidate.

2.5 OpenCL default kernels

This section lists all kernel functions that are available to the *opencl* filter if no filename is specified.

void **fix_nan_and_inf**()
Sets element to 0.0 if it is NaN or Inf.

void **absorptivity**()
Computes $f(x) = -log(x)$.

void **nlm_noise_reduction**()
Smooths data within a local neighbourhood.

void **diff**()
Computes $f(x, y) = x - y$.

2.6 OpenCL reduction default kernels

This section lists all kernel functions that are available to the *opencl-reduce* filter if no filename is specified. These kernels are supposed to be used for the `kernel` argument.

void **minimum**()
Computes the minimum of each pixel in the stream.

void **maximum**()
Computes the maximum of each pixel in the stream.

void **sum**()
Computes the sum of each pixel in the stream.

These kernels are supposed to be used in the `finish` argument:

void **divide**()
Divides each pixel by the stream count. Together with `sum` this can be used to compute the average, i.e.:

```
ufo-launch .. ! opencl-reduce kernel=sum finish=divide ! ..
```

2.7 Third party contributions

The filters described in this section come from third-party groups and must be enabled explicitly during the configuration process either by calling CMake with:

```
cmake <src-dir> -DWITH_CONTRIB=ON
```

or setting the `WITH_CONTRIB` flag in the `ccmake` user interface.

2.7.1 Filters

Contributions by Serge X. Cohen

These filters were initially written with X-ray tomographic processing in mind. Still they are of a general usage as long as input are image.

Point-based transformation

Rejecting outliers in 3D

class med-mad-reject
For each pixel of a frame within a stream, makes a 3x3x3 box (that is, 3x3 box including previous, current

and following frames) and compute the median (med) and median absolute deviation (mad). If the value of the central pixel is too far from the median (relative to the mad) it is rejected and its value is replaced by the median value of the box.

"threshold": float
When abs(px-med) > threshold*mad the pixel value (noted px) is replaced by med.

Rejecting outliers in 2D

class med-mad-reject-2d
For each pixel of a frame make a square box centred on the pixel and compute the median (med) and median absolute deviation (mad). If the value of the central pixel is too far from the median (relative to the mad) it is rejected and its value is replaced by the median value of the box.

"box-size": uint
The edge size of the box to be used (in px). This should be an even number so that it can be centred on a pixel.

"threshold": float
When abs(px-med) > threshold*mad the pixel value (noted px) is replaced by med.

OpenCL one-liner computation

class ocl-1liner
The aim is to enable the implementation of simple, nevertheless multiple input, computation on the basis of one work-item per pixel of the frame. The filter accepts arbitrary number of inputs, as long as more than one is provided. The output has the same size as the first input (indexed 0) and the generated OpenCL kernel is run with one work item per pixel of the output. The user provides a single computation line and the filter places it within a skeleton to produce an OpenCL kernel on the fly, then compiles it and uses it in the current workflow.

In the kernel the following variables are defined : *sizeX* and *sizeY* are the size of the frame in X and Y directions; *x* and *y* are the coordinate of the pixel corresponding to the current work item; *in_x* are the buffer holding the 0..(n-1) input frames; *out* is the buffer holding the output of the computation.

In the computation line provided through `one-line` the pixel corresponding to the current work item is *px_index*. Also reference to the pixel values can use multiple syntax : *out[px_index]*, *in_0[px_index]*, ... *in_x[px_index]* or as shortcut (indeed macro of those) *out_px*, *in_0_px*, ... *in_x_px*. Finally if one wants to have finer control over the pixel used in the computation (being able to use neighbouring pixel values) one can use the *IMG_VAL* macro as such *IMG_VAL(x,y,out)*, *IMG_VAL(x,y,in_x)* ...

"one-line": string
The computation to be performed expressed in one line of OpenCL, no trailing semi-column (added by the skeleton). To avoid miss-interpretation of the symbols by the line parser of ufo-launch it is advisable to surround the line by single quotes (on top of shell quoting). One example (invoking through ufo-launch) would be *"'out_px = (in_0_px > 0) ? sqrt(in_0_px) : 0.0f'"* .

"num-inputs": uint
The number of input streams. This is mandatory since it can not be inferred as it is the case by the *OpenCL* task.

"quiet": boolean
Default to *true*, when set to *false* the dynamically generated kernel sources are printed to the standard output during the task setup.

Auxiliary

Producing simple statistics on a stream

class stat-monitor

Inspects a data stream in a way similar to the `monitor` task but also computing simple statistics on the monitored frame stream: min, max, mean and standard deviation of each frame is computed. To limit truncation errors the OpenCL kernel uses fp64 operations if those are supported by the used OpenCL device, otherwise it falls back to use fp32 arithmetic which might incurs significant truncation errors on images of large dimensions.

"filename": string

When provided the tabulated statistics are output the file with this filename rather than displayed to standard output.

"trace": boolean

When set to *true* will print processed frame index to standard output. This is useful if the task is placed in before a task somehow hiding the number of processed frames (in a complex workflow). Defaulting to *false*

"quiet": boolean

When set to *true* will not print the frame monitoring. Defaulting to *false* to be as close as possible to the output of the `monitor` task.

"print": uint

If set print the given numbers of items on stdout as hexadecimally formatted numbers (taken from `monitor` task).

CHAPTER 3

Examples

3.1 Examples

3.1.1 CT Pre-processing

Flat field correction

To remove fixed pattern noise that stems from the optical system caused by imperfections in the scintillator screen or an inhomogeneous beam and thermal noise from the detector sensor, you can use *flat field correction*. This assumes that you have a set of dark fields acquired with the shutter closed, a set of flat fields acquired without the sample in the beam and the projections with samples. If the beam intensity shifts over time it can be beneficial to acquire flat fields before and after the projections and interpolate between them.

In the simplest case you connect the projection stream to input 0, the dark field to input 1 and the flat field to input 2 of *flat-field-correct*:

```
ufo-launch \
    [ \
        read path=projections*.tif, \
        read path=dark.tif, \
        read path=flat.tif \
    ] ! \
    flat-field-correct !
    write filename=corrected-%05i.tif
```

If you have a stream of flats and darks you have to reduce them either by connection them to *average* or *stack* → *flatten* with the mode set to `median`. Suppose, we want to average the darks and remove extreme outliers from the flats, we would call

```
ufo-launch \
    [ \
        read path=projections*.tif, \
        read path=darks/ ! average, \
```

(continues on next page)

(continued from previous page)

```
        read path=flats/ ! stack number=11 ! flatten mode=median \
    ] ! \
    flat-field-correct !
    write filename=corrected-%05i.tif
```

If you have to interpolate between the flats taken before and after the sample scan, you would connect the first flat to input 0 and the second to input 1 of *interpolate* and set the `number` property to the number of expected projections:

```
ufo-launch \
    [ \
        read path=projections*.tif, \
        read path=darks/ ! average, \
        [ \
            read path=flat-before.tif, \
            read path=flat-after.tif \
        ] ! interpolate number=2000
    ] ! \
    flat-field-correct !
    write filename=corrected-%05i.tif
```

If you want to avoid the automatic absorption correction you have to set `absorption-correct` to FALSE and if you want to ignore NaN and Inf values in the data, set `fix-nan-and-inf` to FALSE.

Sinograms

The reconstruction pipelines presented in the following section assume sinograms as input in order to parallelize along slices. To transpose a stream of (corrected) projections connect it to *transpose-projections* and set `number` to the number of expected projections. Note, that the transposition happens in main memory and thus may exhaust your system resources for a larger number of big projections. For example, to transpose 2048 projections, each at a size of 2048 by 2048 pixels requires 32 GB of RAM.

3.1.2 CT Reconstruction

Filtered backprojection

To reconstruct from sinograms using the analytical filtered backproject method *[KaSl01]*, you have to feed the sinograms into *fft* → *filter* → *ifft* → *backproject* to obtain slices one by one:

```
ufo-launch \
    dummy-data width=$DETECTOR_WIDTH height=$N_PROJECTIONS number=$N_SLICES ! \
    fft dimensions=1 ! \
    filter ! \
    ifft dimensions=! ! \
    backproject axis-pos=$AXIS ! \
    null
```

Direct Fourier inversion

In this example we use the Fourier slice theorem to obtain slices directly from projection data *[KaSl01]* and use a sinc kernel to interpolate in the Fourier space. To reconstruct, you have to feed the sinograms into *zeropad* → *fft* → *dfi-sinc* → *swap-quadrants* → *ifft* → *swap-quadrants*

```
ufo-launch \
    dummy-data width=$DETECTOR_WIDTH height=$N_PROJECTIONS number=$N_SLICES ! \
    zeropad center-of-rotation=$AXIS ! \
    fft dimensions=1 auto-zeropadding=0 ! \
    dfi-sinc ! \
    swap-quadrants ! \
    ifft dimensions=2 ! \
    swap-quadrants ! \
    null
```

3.1.3 Data distribution

To distribute data in a compute network you can use the *zmq-pub* sink and *zmq-sub* generator. For example, to read data on machine A and store it on machine B, you would run

```
ufo-launch read path=/data ! zmq-pub
```

on machine A and

```
ufo-launch zmq-sub address=tcp://hostname-of-machine-a ! write
```

on machine B. Note that by default *zmq-pub* publishes data as soon as it receives it, thus some of the data will get lost if the *zmq-sub* is run after *zmq-pub*. You can prevent this by telling the *zmq-pub* task to wait for a certain number of subscribers to subscribe:

```
ufo-launch read path=/data ! zmq-pub expected-subscribers=1
```

References

CHAPTER 4

Additional notes

4.1 ChangeLog

4.1.1 Version 0.16.0

Enhancements:

- filter: Enable scaling in ramp_fromreal
- opencl: add *options* property to set build opts
- opencl: allow overriding **PATCH_** and SEARCH_RADIUS
- opencl: add *diff* kernel
- nlm: use sigma if passed as an option
- nlm: don't scale sigma arbitrarily
- backproject: lift angle-step and -offset limits
- read: support single plane RGB data
- write: support RGB TIFFs and JPEGs
- write: do not require fmt specifier for jpeg
- bin: support 3D binning as well
- fft: add debug message showing underlying FFT lib
- Do transpose on GPU

Fixes:

- Fix #153: handle 64 bit TIFFs gracefully
- Fix #159: add boolean *rescale* option
- Fix #161: add test to prove things work

- Fix #162: make use of new buffer layout API
- Fix #163: return raw-height correctly
- Fix #165: use current get_kernel API
- Fix #166: propagate OpenCL errors if possible

Breaks:

- detect-edge: rename “type” to “filter”

New filters:

- Add cone beam reconstructor
- Add tile task
- Add unsplit task
- Add map-color task
- Add gradient filter
- Add zmq-pub and zmq-sub tasks

4.1.2 Version 0.15.1

Fixes:

- #153: do not crash with 64 bit floating point TIFFs
- Use specific OpenCV 2 header file in an OpenCV 3 environment

4.1.3 Version 0.15.0

Enhancements:

- Added a manual section showing basic image processing examples
- Added a manual section to list default kernels usable with opencl and opencl-reduce
- backproject: unroll loop for P100, Quadro M6000, GTX 1080 TI and Tesla K20XM
- cv-show: use unique window name to allow multiple viewers
- dfi: clean up and simplify reflection code
- read: avoid file open check if successful
- read: add lazy timeout-based reading
- retrieve-phase: remove unused normalize parameter
- retrieve-phase: untangle macro and ?: mess
- stat-monitor: clean up and remove dead code
- stitch: minor cleanups and correct kernel release
- swap-quadrants: simplified code
- write: warn if no format spec is given for jpeg
- Fix #144: document swap-quadrants

Fixes:

- camera: fix linking with libuca
- cv-show: fix compilation with older g++ compilers
- dfi: fix wrong warning about even sample number
- dummy-data: lift number limit
- opencl: kernel name cannot be NULL
- Fix #149: image2d_t is always global
- Fix #146: use gnu99 instead of c99
- Fix #133: off-by-one cropping is bad

Breaks:

- Moved nlm kernel from nlm.cl to opencl.cl
- Remove unused default.cl

New filters:

- Added cv-show viewer
- Added circular mask filter
- Added opencl-reduce
- Added projection filter bh3
- Added filter to remove outliers

4.1.4 Version 0.14.1

Fixes:

- Let meson build all the tasks that CMake could before
- Check if Python is actually available in order to generate lamino kernels
- Fix install documentation
- Fix compilation with MacOS compilation and Python 3
- memory-in: cast pointer to target type
- write: fix problem with generated filenames that are not incremented

4.1.5 Version 0.14.0

Enhancements:

- Support meson build system alongside CMake
- Suppress tiff writing warnings
- dummy-data: add random-metadata flag
- interpolate: use GPU instead of OpenMP which is an order of magnitude faster
- lamino: allow setting addressing mode
- monitor: output metadata values as well
- raw-read: split offset in pre and post offsets

- write: add counter-start and counter-step
- write: add minimum/maximum to control conversion
- null: allow printing durations from timestamps

Fixes:

- lamino: prevent volume shifting in center kernel
- Fix #133: allow crop position with specifying dims

Breaks:

- Replaced stdout filter with standard write module
- write: rename quality property to jpeg-quality

New filters:

- Add rotate filter
- Add stitch task
- Add interpolate-stream task
- Add correlate-stacks task
- Add cut task
- Add stamp filter to print current iteration into output buffer

4.1.6 Version 0.13.0

Enhancements:

- Added infrastructure to "stage" filter contributions by third parties. To enable building it the WITH_CONTRIB option must be set explicitly to ON.

Fixes:

- write: call conversion only once
- read: fix segfault with start too large
- read: fix dumping to JSON
- Fix compilation and installation on MacOS
- Fix #128: prevent segfault with start parameter
- Do not compile ufo-priv.c for each task thus saving compile and link time
- Add documentation for undocumented tasks

New contributed filters by Serge X. Cohen (Synchrotron SOLEIL):

- Add MedMadReject median value rejection in 3D
- Add MedMadReject2D median value rejection in 2D
- Add Ocl1Liner to compute basic OpenCL arithmetics
- Add StatMonitor to output stream statistics

4.1.7 Version 0.12.0

Enhancements:

- Fortify source and enable large file support
- Re-arrange filter documentation

Fixes:

- Fix #127: use enums where possible
- Document the filter task
- Fix potential errors found with static analysis
- stdin: use gsize to avoid LFS problems
- dfi-sinc: do not call exit()
- raw/read: fix type translation for raw-offset

Breaks:

- metaballs: create filled balls rather than circles
- metaballs: remove run-infinitely and fps props
- filter: use enum instead of type-unsafe string
- loop: rename ::count to ::number

New filters:

- Add binarization filter
- Add basic segmentation filter

4.1.8 Version 0.11.1

Fixes:

- Fix #124: build and install oclfft optionally
- Use OLD behaviour for CMP0046
- Use G_MAXSIZE instead of ULLONG_MAX
- Include oclfft deps dir only if enabled
- filter: link FFT libs
- ifft: remove unused/wrong imports
- raw: do not ignore return value of fread
- transpose: fix warning if SSE is not possible
- Add license statements where missing
- Link against m unconditionally

4.1.9 Version 0.11.0

Enhancements:

- Add option to build Sphinx manual
- Improved filter documentation
- Increase robustness of OpenCL kernels by using correct type everywhere
- Make AMD clFFT optional
- backproject: improve performance on GTX Titan
- rescale: allow setting absolute width and height
- camera: allow passing properties to camera
- camera: simplify readout mechanism
- dummy-data: opt-in for initialization using *init*

Fixes:

- Link only to required dependencies
- Do not link oclfft unconditionally
- zeropad: fix for centers < half width
- Fix #121: use correct exit condition
- Set std=c99 only on C source files
- oclfft: link against UFO
- rescale: remove debug output
- lamino-backproject: fix for small max workgroups

Breaks:

- dummy-data: remove bitdepth property

New filters:

- Add GEMM matrix multiplication using CLBlast
- Add bin filter to bin pixel values

4.1.10 Version 0.10.0

Enhancements:

- Restructured FFT-based filters to use a common code base
- filter: Use real space ramp by default
- crop: add `from-center` property
- hdf5: whitelist .hdf5 and .nxs extensions

Fixes:

- camera: do not convert 32 bit float data
- EDF: fix problem parsing Frelon EDF data

- Fix #117: fail gracefully if file can't be written
- edf reader: Allow 512-multiple header size
- Fix reading 32 bit float raws as unsigned int

Breaks:

- read: renamed enable-conversion → convert
- null: renamed force-download → download

New filters:

- Add MemoryIn generator
- Add MemoryOut sink
- Add stdin generator
- Add stdout sink
- Add laminographic backprojection
- Add 1D stripe filter
- Add sleep task for debugging purposes

4.1.11 Version 0.9.0

Enhancements:

- backproject: reconstruct region-of-interest
- backproject: loop unroll on GTX Titan Black
- filter: generalize filter types
- read: allow overriding type detection
- read: read as many bytes as expected in raw mode
- map-slice arbitrary number of input data
- monitor: add print property to show data

Fixes:

- Fix ramp filter computation and mirroring
- Fix two dimensional batch processing of FFT and IFFT
- Fix segfault caused by double-freeing kernel
- opencl: fix copying dimension property
- read: fix segfault reading big-endian .edf
- fbp: Use number of projs to compute angle step
- dfi: add angle-step property
- blur: free allocated OpenCL buffers
- slice: slice correct number of input items
- stack: stack every [number] inputs

New filters:

- Add flip task
- Add clip task
- Add loop task
- Add refeed task
- Add merge task
- Add basic raw reader

4.1.12 Version 0.8.0

Major changes:

- Read changed "end" property back to "number"
- Renamed downsample filter to rescale
- Renamed cut-roi filter to crop
- null: added "finish" property to call clFinish()
- filter: added Faris-Byer type filter coefficients
- ifft: added crop-height property
- Removed possibility to disable building plugins

New filters:

- Add calculate task
- Add new monitor task
- Add edge detection filter
- Added HDF5 reader and writer
- Added raw writer
- Added JPEG writer

4.1.13 Version 0.7.0

This release breaks badly with all previous version because we renamed several filters and properties to clean up inconsistencies.

Major changes include:

- Integration tests have been moved to core
- writer: allow 8 and 16 bit uint output
- reader: support Multi EDF files
- reader: add y-step parameter
- reader: from:to:step selection of files
- flatfieldcorrection: add "dark-scale" property

New filters:

- Import uPIV related filters by Alexandre Lewkowicz

- Add pad to add zero padding
- Add slice mapper to arrange input as a single grid
- Add inplace flatten task for sum, min and max
- Add interpolation task to interpolate between two streams
- Add flatten task based on median sort
- Add stack task to create a volume from 2D series
- Add transpose task to rotate data
- Add measure task to measure image metrics
- Add PolarCoordinates task
- Integration of UfoIR algebraic reconstruction tasks
- Add median noise filter
- Add slice task to cut volumes into 2D data stream
- Add stripe removal task
- Add phase retrieval filter

4.1.14 Version 0.6.0

Changes

- Added angle offset parameter to backproject filter.
- Fix backprojection for NaN in input
- Fix LUT computation resulting in wrong reconstructions.
- Install kernel files into ${datadir}/ufo as required by ufo-core 0.6.

New filters

- “generate”: takes width, height and depth parameters as well as a number that is produces with the specified dimensions.
- “downsample”: reduce the size of an image by an integer

4.2 Copyright

Bibliography

[KaSl01] Kak, A. C., & Slaney, M. (2001). Principles of Computerized Tomographic Imaging (Philadelphia, PA: SIAM).

Index