IPL (Image Processing Language)

This program is used to execute the 3D-segmentation and different morphometric evaluations, e.g. MIL (Mean Intercept Length), DT (Distance Transformation), TRI (Triangulation), SMI (Structure Model Index) etc.

The concept of this program is: You work with multiple objects in memory. One object is usually the input for a procedure, whereas the result will be the next object. You can delete objects (internally, in memory only).

You can also read and write objects from/to disk. Each object has to be given an internal name, such as *input*, *output*, *segmented* or just *a,b,c* etc.

Once launched in interactive mode, you can execute the commands below. Each command takes a few arguments which usually have a default value. By pressing the return key, the value is entered. If you have to enter values for all coordinates (x, y and z), you can either give all three values separated by a space or just enter one value, which is then applied to all coordinates. If the order of the arguments is known by heart, you can supply them in the correct order together with the command, e.g. '`gauss in out`' and then press return to be asked the next arguments.

IPL commands may be abbreviated. The procedure names list is searched against your input, and IPL takes the first command that matches the abbreviation (execute `'help'` to see the list and and search order). There is no 'ambigous command' warning.

ATTENTION: IPL only accepts lower case input!

ATTENTION: In the standard evaluation (Start Evaluation 3D...), IPL is operated in batch mode. Then the two dots (..) in the last IPL line of the batch-command file are very important. DO NOT DELETE OR FORGET THEM! At the IPL startup in 'batch mode', the whole string is read up to the two dots, and then commands in the string are executed one by one. **On the other hand, in interactive mode IPL executes the commands immediately. To exit interactive IPL, use ôquitô.**

help

Gives the list of all available procedures or a short description of a procedure if `'help xxxx'` is typed.

quit

Quits the program

```
ipl> quit (or q)
```

list

Lists all objects currently in memory or lists all procedures, optionally only items starting with *beggining_with*.

```
ipl> list
 -objects               [true] >
 -procedures            [false] >
 -beginning_with        [] >
```

read

Reads an object from disk. The file type has to be *.aim. By reading an object from disk, you have to give it an internal name.

```
ipl> read
  -name         [in] > a
  -filename     [default_file_name] > DISK3:[DATA]C0000372.AIM
  -type         [aim] >
  -uncompress   [true] >
```

aim_read

Shortcut for reading an AIM from disk. The file type has to be *.aim. By reading an object from disk, you have to give it an internal name. Same function as `read`, but *type* and *uncompress* are assumed to be *aim* and *true*.

```
ipl> aim
  -name         [in] > a
  -filename     [default_file_name] > DISK3:[DATA]C0000372.AIM
```

or on one line only:

```
ipl> aim in DISK3:[DATA]C0000372.AIM
```

isq_to_aim

Reads an .ISQ from disk and puts the requested volume of interest into memory as an aim object. The volume of interest is given by *pos* (upper left corner) and *dim* (dimension of aim object).

```
ipl> isq
 -aim_name              [in] >a
 -isq_filename          [default_file_name] >DISK3:[DATA]C0000372.ISQ
 -pos                   [0 0 0] >150 240 0
 -dim                   [-1 -1 -1] >210 210 210
```

write

Writes an object to disk. The file type should be *.aim. You can compress segmented files using *run_length* or *binary* compression. Default is *binary*, but for gray-scale images the compress type is always switched internally to *none*.

```
ipl> write
  -name           [default_name] > a
  -filename       [default_file_name] > DISK3:[DATA]XYZ.AIM
  -compress_type  [bin] > r
```

delete

This command is used to delete an internal object, i.e. to free memory.

```
ipl> delete
  -name     [] > a
```

examine

Use this command to examine different things of an object and show the results on the screen. Possible values are: *geometry, histogram, statistics, number, log, and z_mean_max* (mean and max values of each slice along z-axis).

```
ipl> examine
  -input    [default_name] > a
  -item     [geometry] >


!> dim      504 122 223
!> off      0 0 0
!> pos      266 308 0
.
.
```

sup_divide

This command is used to subdivide an object into smaller objects. This is helpful in examining only a small part of the complete object or for segmenting the whole object using small subobjects to save memory. You can either define the subvolume(s) by entering the number of subvolumes (*su**p**dim,* in all directions) or their size in pixels (*su**b**dim*). *Suppos* is the offset of the object in pix-

els in local coordinates. If you leave the value at -1, the object(s) will be centered. *testoff* is used for overlapping the small subobjects in case of a procedure such as a gauss-filter. If *supdim_numbers* is not a divisor of the original object's dimensions, a few voxels may be discarded, at most *subdim_number* voxels per direction x, y or z.

```
ipl> sup_divide
  -input                [in] > a
  -supdim_numbers       [-1 -1 -1] > 4
  -testoff_pixels       [0 0 0] > 2
  -suppos_pixels_local  [-1 -1 -1] >
  -subdim_pixels        [-1 -1 -1] >
```

sub_get

Extracts a sub volume that has the position and size as entered by pos and dim. The flag *global_pos_flag* controls whether pos is the global position (given by the original measurement) or the local position in the input object.

```
ipl> sub_get
 -input     [in] >a
 -output    [sub] >s
 -pos       [0 0 0] >10 30 10
 -dim       [0 0 0] >120
 -global_pos_flag       [false] >
```

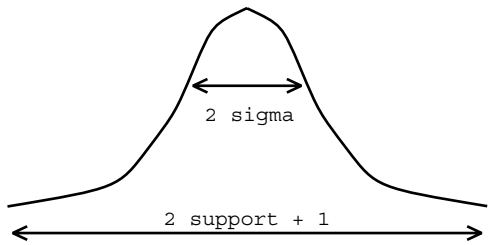
sub_pick

Is an older version of `sub_get`, that needs previous execution of `sup_divide` to enter the positions and sizes in the grid. *subpos_numbers* is the (0-based) position in numbers in the suppos-grid.

```
ipl> sub_pick
 -input     [in] >a
 -output    [sub] >s
 -subpos_numbers        [0 0 0] >
```

gauss_lp

This is the command to gauss-filter an object. The shape of the filter can be given using *sigma* and *support.* You have to give an internal name for the output object.




```
ipl> gauss
  -input    [in] > a
  -output   [gauss] > b
  -sigma    [1.000000] > 1.2
  -support  [2] >
```

threshold

Using this command, you can binarize an object. All voxels below *lower_in_perm* and above *upper_in_perm* will be set to value 0, the rest (the object) will be given a value of *value_in_range,* usually 127. The values for the lower and upper threshold are given in 1/1000, covering the whole range of the data values (0 to 32767 for gray-scale AIMs, -128 to 127 for *char* AIMs).

```
ipl> thresh
  -input                [gauss] > b
  -output               [th] > c
  -lower_in_perm        [300] > 220
  -upper_in_perm        [1000] >
  -value_in_range       [127] >
```

gauss_seg and seg_gauss

Use this command to do the *gauss_lp* and the *threshold* in one step. This saves memory and reduces the computing time. However, you have no access to the intermediate object.

```
ipl> gauss_seg    (or seg)
  -input                [in] > a
  -output               [seg] > b
  -sigma                [1.000000] > 1.2
  -support              [2] >
  -lower_in_perm        [300] > 220
  -upper_in_perm        [1000] > 1000
  -value_in_range       [127] >
```

adaptive_threshold

Using this command, you can find out which threshold leads to the best segmentation of the bone. The program examines the BV/TV parameter at different threshold values (given by *first_threshold*, *last_threshold* and *nr_steps*). It then tries to find the threshold with the least change, which it considers to be the best choice. At the end, this value is applied to the input and written to the output object using the *value_in_range* as in threshold.

```
ipl> adaptive_thresh
  -input                [gauss] > a
  -output               [th_adp] > c
  -first_threshold      [100] > 200
  -last_threshold       [400] > 300
  -nr_steps             [30] > 50
  -value_in_range       [127] >
```

fft_laplace_hamming

Segmentation of an object based on zero crossing of second derivative. The second derivative is calculated in the fourier domain by applying a w^2 filter, for noise smoothing a Hamming filter is simultaneously applied. To ensure that regions with high attenuation are segmented as foreground, the original image is added with variable weight *1–laplace_epsilon*. The result is an object of type 'float' that has to be normed to 'short' again with `norm_max`.

Redim_pow2 can be used to simultaneously interpolate the object to a new voxel-size by fourier interpolation in powers of 2 (*redim_pow2 = 2* -> 4 times interpolation). *Laplace_epsilon* controls the weight of the curvature image, the higher the epsilon, the more 'edge-enhanced' the image appears.
Lp_cut_off_freq is the Hamming filter low-pass frequency in units half the Nyquist frequency: 0.5 is a filter rolling off to 0 just at the Nyquist boundary.
Hamming_amp is the amplitude of the Hamming filter, its default is best not varied.

```
ipl> fft_laplace
 -input      [in] >in
 -output     [lh] >out
 -redim_pow2            [0 0 0] >
 -laplace_eps           [0.900000] >
 -lp_cut_off_freq       [0.400000] >
 -hamming_amp           [1.000000] >
```

norm_max

`Norm_max` has to be applied after `fft_laplace_hamming` to convert the 'float' values to 'short' again. *Max* in the input will be set to the maximal possible value of the chosen data output type *type_out* (32767 for short), as will all values above *max*. The values from zero to *max* will be scaled to the data range available for the output data type. With `'examine lh histo'` the histogram of the `fft_laplace` output can be viewed, and a suitable *max* can thus be chosen, normally a value at the upper end of the main distribution. The range of the `fft_laplace` output can vary greatly for different types of objects, since the curvature can be very different. It is advised, however, to use the same value for *max* for one type of sample, i.e. within one study population. The value is not very crucial, as long as it is above the respective value that is thresholded afterwards to produce a binary image.

```
ipl> norm
 -input      [lh] >
 -output     [norm] >
 -max        [50000.000000] >
 -type_out   [short] >
```

gobj_maskaimpeel_ow

Performs a masking of an object with a GOBJ (or a AIM-mask, needs more memory and time with AIM, though). Voxels outside the mask are set to Zero. *Peel_iter* gives the number of two-dimensional peel iteration applied to the contours. The operation is working directly on the object, i.e. it is overwriting Zeros on it. NB: `Gobj_maskaimpeel_ow` may be repeatedly used on the same object, but using a smaller *peel_iter* number does not restore the region set to Zero anymore, i.e. while trying out different *peel_iters*, go from small numbers upwards. The relative volume of the mask is stored in the proceeding log, which may be shown with '`examine in log`' afterwards.

```
ipl> gobj_mask
 -input_output          [in] >in
 -gobj_filename         [default_file_name] >U0001977.GOBJ
 -peel_iter             [0] >
```

cortex_maskoff

Expert function.

Combines a GOBJ contour (e.g. the outer bone contour) with a cortex-segmentation mask. The resulting AIM-mask is then the region inside the GOBJ, but without the cortex, i.e. only the trabecular region. The input is a GOBJ file and a volume that should represent the cortex: *input_mask.* This *input_mask* has to be produces with `seg_gauss` beforehand (e.g. with a high sigma 10.0 and support 6, and an approriate threshold 150 or 300.) *cortex_peel_iter* gives a minimal cortex thickness, i.e. the minimal boundary of the outer contour that is excluded in the mask. A component labeling is performed for every slice for the cortex (-> *cl_cortex* in percent) and then for the inner mask (-> *cl_inner* in percent), in the example below, fro every slice only parts of the cortex that take up more than 50% are considered for the slicewise exclusion, and then only parts making up more than 60% are taken for the mask.

```
ipl> cortex_mask
 -input_mask            [in] >
 -gobj_filename         [default_file_name] >u0001234.GOBJ
 -output    [out] >
 -cortex_peel_iter      [5] >
 -cl_cortex [10.000000] >50.
 -cl_inner  [10.000000] >60.
```

gobj_to_aim

Produces a solid, filled volume from a GOBJ file, i.e. the inside of the GOBJ mask. This function is needed if the element-size (voxelsize) of the original volume or segmented volume has been changed with `scale_el_size`, e.g. to make the voxels cubic. Then the GOBJ produced on the original ISQ file does not match anymore, but the AIM-mask produced with `gobj_to_aim` can be scaled to the appropriate voxel size (`scale` with *integrate false)*, written to disk and then used in `gobj_maskaimpeel_ow`. Also any other IPL commands can be applied to the AIM-mask like to any other binary volume.

```
ipl> gobj_to_aim
 -gobj_filename         [default_file_name] >U0001977.GOBJ
 -output    [out]       >out
 -peel_iter             [0] >
```

cut2d_shape_ow

Cut a two-dimensional ellipse, circle, or rectangle extended along the z-axis from an object and set all points lying outside to zero. The size of the shape *type_of_shape* is adjusted to the biggest size lying within the x-y plane, if -1 is chosen for *halfaxes_x_y*, otherwise to the given size (may overlap the boundary). *Cutborder* controls whether the border of the shape is set to zero (true) or not (false). The operation is overwriting, i.e. the input object is altered !

```
ipl> cut2d_shape
 -input_output          [in] >
 -type_of_gobj          [circle] >
 -halfaxes_x_y          [-1 -1] >
 -midpos_x_y            [-1 -1] >
 -cutborder             [false] >
```

cl_ow_rank_extract

Component labeling (CL) of a segmented image. Faces have to touch for voxels to be considered connected, i.e. a voxel can have 6 connected neighbours. Extracts the regions with the given rank in the size-ordered table. Overwrites the input. Usually used with *rank 1 to 1* to remove any small noisy speckels not connected to the main structure. *Connect_boundary* controls whether the whole boundary of the box containing the object acts as a connector of surface points.

```
ipl> cl
 -input_output          [seg] >a
 -first_rank            [1] >
 -last_rank             [1] >
 -connect_boundary      [false] >
 -value_in_range        [127] >
```

cl_rank_extract

Same as above, but not overwriting the input. Uses more memory.

```
ipl> cl_rank
 -input                 [seg] >
 -output                [cl] >
 -first_rank            [1] >
 -last_rank             [1] >
 -connect_boundary      [false] >
 -value_in_range        [127] >
```

cl26_rank_extract

Similar as above, but also voxels only touching by an edge are considered connected.

```
ipl> cl26_rank
 -input                 [seg] >
 -output                [cl] >
 -first_rank            [1] >
 -last_rank             [1] >
 -connect_boundary      [false] >
 -value_in_range        [127] >
```

cl_extract

Same as cl_rank_extract, but extracting according to fractional volume of components. Not available with *connect_boundary* flag.

```
ipl> cl_extract
 -input                 [seg] >
 -output                [cl] >
 -lo_vol_fract_in_perc  [1] >
 -up_vol_fract_in_perc  [1] >
 -value_in_range        [127] >
```

cl_nr_extract

Same as cl_rank_extract, but extracting according to number of voxels of components. Not available with *connect_boundary* flag.

```
ipl> cl_nr_extract
 -input                  [seg] >
 -output                 [cl] >
 -min_number             [10] >
 -max_number             [0] >
 -value_in_range         [127] >
```

cl_image

Gives the component labeled image as output, with value 127 for biggest connected part, 126 for second biggest etc.

```
ipl> cl_image
 -input                  [seg] >
 -output                 [cl] >
```

db_scanco_activate

Activates (or deactivates) the database for writing all subsequent evaluation results into it. It stays activated until `db_scanco_activate` *false* is entered.

```
ipl> db_scanco_activate
-write      [true] >
```

tri_da_metric_db

This command triangulates a segmented object and calculates object volume and surface as well as the structure model index (SMI). Using the plate model, trabecular number, thickness and separation are derived. `Tri_da_metric_db` uses the contours of a GOBJ or AIM-mask if it finds the GOBJ filename in the proceedings log. Otherwise, the whole box region is evaluated. The results are written into the database (->db), if the database was activated, see `db_scanco_activate`.

You should usually not modify the default values of the arguments *ip_sigma, ip_support, ip_threshold, interpolate, nr-ave_iter, t_dir_radius, epsilon*.

The output is an object with *nr_views* different 3D views of the triangulated object. This output may be written to disk with `'msq_from_aim'` and viewed with the 3D-Disply program.

```
ipl> tri
 -input                  [th] >seg
 -output                 [tri] >
 -gobj_filename          [gobj_from_log] >
 -peel_iter              [-1] >
 -ip_sigma               [2.000000] >
 -ip_support             [1] >
 -ip_threshold           [64] >
 -interpolate            [true] >
 -nr_ave_iter            [0] >
```

```
 -t_dir_radius          [2] >
 -epsilon               [1.200000] >
 -size_image            [512 512] >
 -scale_image           [0.700000] >
 -edges                 [false] >
 -nr_views              [0] >
```

dt_object_param

Calculates the mean thickness of the structure with the distance transformation (DT) method by filling largest spheres into the object and calculating their mean diameter (volume weighted mean). Mean thickness and standard deviation are written to the database, if activated. The output object shows the spheres fitting inside the structure with the voxel values being their diameter in voxel units. `Dt_object_param` uses the contours of a GOBJ or AIM-mask if it finds the GOBJ filename in the proceedings log. *Roi_radius_factor* controls whether only a sphere is evaluated or the whole region (default). DT also works with GOBJ-masked objects. The *epsilons* are used for suppressing artefacts due to rough surfaces. You should ususally not modify the default values. (For very coarse voxel sizes above 100 µm, *assign_epsilon 0.9* may be chosen.) *Histogram_or_screen* controls whether a histogram of the thickness distribution is written to a textfile and/or shown on the screen.

```
ipl> dt_obj
 -input     [in] >
 -output    [out] >
 -gobj_filename         [gobj_from_log]
 -peel_iter             [-1]
 -roi_radius_factor     [10000.0] >
 -ridge_epsilon         [0.900000] >
 -assign_epsilon        [1.800000] >
 -histofile_or_screen   [none] >samp012_thickness.tab
```

dt_background_param

Calculates the mean separation of the structure with the distance transformation (DT) method by filling largest spheres into the background of the object and calculating their mean diameter (volume weighted mean). Mean sparation and standard deviation are written to the database, if activated. The output object shows the spheres fitting inside the background with the voxel values being their diameter in voxel units. `Dt_background_param` uses the contours of a GOBJ or AIM-mask if it finds the GOBJ filename in the proceedings log. *Roi_radius_factor* controls whether only a sphere is evaluated or the whole region (default). DT also works with GOBJ-masked objects. The *epsilons* are used for suppressing artefacts due to rough surfaces. You should ususally not modify the default values. (For very coarse voxel sizes above 100 µm, *assign_epsilon 0.9* may be chosen.) *Histogram_or_screen* controls whether a histogram of the thickness distribution is written to a textfile and/or shown on the screen.

```
ipl> dt_back
 -input     [in] >
 -output    [out] >
 -gobj_filename         [gobj_from_log]
 -peel_iter             [-1]
 -roi_radius_factor     [10000.0] >
 -ridge_epsilon         [0.900000] >
```

```
 -assign_epsilon        [1.800000] >
 -histofile_or_screen   [none] >
```

dt_mat_param

Calculates the mean trabecular number of the structure with the distance transformation (DT) method by filling largest spheres into the background of the mid axis transformed object (->mat) and calculating their mean diameter (volume weighted mean). Mean number and standard deviation are written to the database, if activated. The output object shows the spheres fitting inbetween the mid axis structure with the voxel values being their diameter in voxel units. `Dt_mat_param` uses the contours of a GOBJ or AIM-mask if it finds the GOBJ filename in the proceedings log.*Roi_radius_factor* controls whether only a sphere is evaluated or the whole region (default). DT also works with GOBJ-masked objects. The *epsilons* are used for suppressing artefacts due to rough surfaces. You should ususally not modify the default values. (For very coarse voxel sizes above 100 µm, *assign_epsilon 0.9* may be chosen.) *Histogram_or_screen* controls whether a histogram of the thickness distribution is written to a textfile and/or shown on the screen.

```
ipl> dt_mat
 -input     [in] >
 -output    [out] >
 -gobj_filename         [gobj_from_log]
 -peel_iter             [-1]
 -roi_radius_factor     [10000.0] >
 -ridge_epsilon         [0.900000] >
 -assign_epsilon        [1.800000] >
 -histofile_or_screen   [none] >
```

dt_mat_output

Performs the mid axis transformation of the structure and saves the mid axes in the output.

```
ipl> dt_mat_output
 -input     [in] >
 -output    [out] >
 -ridge_epsilon         [0.900000] >
```

connectivity

Calculates the connectivity density of a segmented object, according to Odgaard and Gundersen (Bone **14**:173-182;1993). The insure that only one connected object and one connected background is evaluated, a component labeling and extraction of the dominant component is performed for both the foreground and the background. Use `conn_nocl` and `conn_bgcl` if you wish no component labeling or only a background component labeling/extraction, e.g. if a component labeling is already incorporated into the segmentation.

Attention: the component labeling and extraction changes the input volume !

```
ipl> connectivity
 -in_out    [in] >
```

conn_nocl

Calculates the connectivity density of a segmented object without prior component labeling. If the object consists of more than one unconnected part, you would have to add the number of components minus one to the connectivity, and then calculate the connectivity density. Likewise, if the background is made up of more than one part, you would have to add the number of background components minus one.

```
ipl> conn_nocl
 -in_out    [in] >
```

conn_bgcl

Calculates the connectivity density of a segmented object with only component labeling and extraction of the background. If the object consists of more than one unconnected part, you would have to add the number of components minus one to the connectivity, and then calculate the connectivity density.

Attention: the component labeling and extraction changes the input volume !

```
ipl> conn_bgcl
 -in_out    [in] >
```

voxgobj_scanco_param

Evaluates a segmented image for the fraction of voxels set and writes result to the database (if activated). For gray scale (original) images, mean attenuation coefficient is written to database. The object is masked again with the given GOBJ or AIM-mask. By default, the GOBJ filename is taken from the proceedings log (last occurence of a GOBJ filename). If the GOBJ file is not found, `vox_scanco_param` (see below) is applied. *Peel_iter* controls the number of 2D peel iterations for the mask. By default, peel_iter is taken from the proceedings log.

```
ipl> vox
 -input                 seg
 -gobj_filename         gobj_from_log
 -peel_iter             -1
```

vox_scanco_param

Evaluates a segmented image for the fraction of voxels set and writes result to the database (if activated). For gray scale (original) images, mean attenuation coefficient is written to database. If the proceedings log of the object contains a GOBJ mask operation, the relative volume of the masked region is incorporated into the calculation.

```
ipl> vox_scanco seg
```

mil_param

This command executes the MIL-calculations (Mean Intercept Length, Parfitt). You should usually not modify the default values. If the database is activated, the result of the calculations are written to the evaluation-database.

```
ipl> mil_param
  -input                [default_name] > a
  -ray_plane_scale      [2.000000] >
  -roi_radius_factor    [1.000000] >
  -t_dir_ortho          [no] >
  -t_dir_ortho_nr       [8] >
  -t_dir_radius         [2] >
  -fabric_tensor        [yes] >
```

milv1_param

This command executes the MIL-calculations in its first version (different border handling). You should usually not modify the default values. If the database is activated, the result of the calculations are written to the evaluation-database.

```
ipl> milv1
  -input                [default_name] > a
  -ray_plane_scale      [2.000000] >
  -roi_radius_factor    [1.000000] >
  -t_dir_ortho          [no] >
  -t_dir_ortho_nr       [8] >
  -t_dir_radius         [2] >
  -fabric_tensor        [yes] >
```

histo

Gives a histogram of an object. With the default arguments same effect as `'examine in histo'`. Optionally, the histogram is shown from *from_val* to *to_val*, default from minimal value to maximal value found in input volume. Optionally, a tabulated version of the histogram is printed to a file (given with *fileout_or_screentab)* or the screen (*fileout_or_screentab* screen, in addition to the bar-column representation) with the number of bins given with *nr_bins_in_tab*. The number of bins in the bar-column representation is not affected by *nr_bins_in_tab* !

```
ipl> histo
 -input                 [in] >rat
 -fileout_or_screentab  [none] > my_histo.tab
 -from_val              [-1] >1500
 -to_val                [-1] >3000
 -nr_bins_in_tab        [-1] >
```

scale_elsize

Using this command you can change the voxelsize of your object. *down_scale* enlarges the voxelsize (reduces the resolution), *up_scale* makes the voxelsize finer. Non-integer scaling can either be given directly, e.g. with *down_scale* 1.5; or with a combination of *down_scale* 3 and *up_scale* 2. The effect is exactly the same, the latter may just save you a division by hand. The *integrate*

flag controls whether averaging of voxels is done in downscaling (and potential interpolation of partial voxels) and whether interpolation is done in upscaling. It is suggested to use *integrate true* for gray-scale images, but *integrate false* for binary segmented images to preserve the binary status. Otherwise a subsequent thresholding operation may be needed to ensure a binary image again.

`Scale_elsize` automatically selects among the below described procedures `ipscale_elsize` and `noipscale_elsize`, according to whether there are fractional voxels who may then be interpolated (->ip): For integer down-scalings `noipscale` is chosen, for non-integer down-scalings and any up-scalings `ipscale` is chosen. The exception is non-averaging scaling of binary objects always with `noipscale` to maintain the binary status, if the *integrate* flag is put on false.

If a GOBJ mask is used for the evaluation of an object, the GOBJ mask has to be transformed into an AIM mask with `gobj_to_aim`, and this AIM mask can then be scaled to the same voxelsize as the original object. Use *integrate* false for the mask scaling to preserve the binary status of the mask (alternatively, after masking with *integrate* true, perform a threshold operation to produce a binary mask again).

```
ipl> scale
  -input                [in] > a
  -output               [sca]> x
  -down_scale           [2.000 2.000 2.000] >
  -up_scale             [1.000 1.000 1.000] >
  -integrate            [true] >
```

ipscale_elsize

One constituent of `scale_elsize`. If the new voxel-grid is not a integer multiple of the original voxel-grid, then the new values lying between old voxels points are interpolated (-> ip), and depending on the *integrate* flag also averaged over the voxelvolume. The *center* flag controls whether the center of the object is still in the center of the scaled object, or whether the upper left top corner is also the beginning of the scaled object (*center false*). The computation time and memeory needed is larger than for `noipscale_elsize`.

```
ipl> ipscale_elsize
 -input     [in] >a
 -output    [sca] >b
 -down_scale[2.000 2.000 2.000] >1
 -up_scale  [1.000 1.000 1.000] >2
 -center    [false] >
 -integrate [true] >
```

noipscale_elsize

Scales the objects without interpolating non-integer voxel locations in the scaled object, but takes nearest neighbour approach. Computationally faster and uses less memory, but nearest neighbour choice may create less accurate scaling. Does not account for fractional voxels for non-integer scaling, even if *average* flag is true, e.g. for *down_scale* 2.6 and *average true*, only 2 voxels are averaged (per direction).

```
ipl> noipscale
  -input                [in] > a
```

```
  -output                [sca]  x
  -down_scale            [2.000 2.000 2.000] >
  -up_scale              [1.000 1.000 1.000] >
  -average               [true] >
```

scale_ow_elsize_noip

Same as `noipscale_elsize`, but overwrites input. Saves memory. Does not work for increasing resolution, since output memory would have to be bigger than input memory.

```
ipl> noipscale
  -input_output          [in] > a
  -down_scale            [2.000 2.000 2.000] >
  -up_scale              [1.000 1.000 1.000] >
  -average               [true] >
```

set_value

With this command you can set the value of all non-zero voxels of a segmented object to a given value and all zero voxels to another. Useful for inverting foreground/background of the object and for producing a negative image (value_object -127).

```
ipl> set_val
 -input     [in] >
 -value_object           [127] >
 -value_background       [0] >
```

concat

Concatenates two objects. *Common_region_only:* if two objects with unequal dimensions or unequal positions are chosen, this flag controls whether the region common to both objects is taken; or whether the box region surrounding both objects is produced. *Add_not_overlay:* Controls whether voxel values of second object are added to the first object - with overflow taken care of, i.e. 60+90=127+50=127 for *char* images; or wheter non-zero values are layed over the first object, useful for mapping the segmented image onto the original gray-scale data. *Make_edge:* controls whether edges are made of second object (or first, if second is gray-scale and first is *char)*. The second object can be shifted with *shift_ofin2* and turned in the x-y plane with *turnangle*. *Turnpoint_global* is tried to be set to the original rotation center of the scanner, i.e. for 512x512 images to 256,256 etc. The turned image2 is mapped into the same box as image2, thus for larger angles the image may be cut at the boundary if there is not enough free space left around the object to contain the turned image2. Subtraction of two binary images can be performed by a combination of /set_value of input2 to -127 and then concat it with input1, flag *add true*.

```
ipl> concat
 -input1                [in]  >a
 -input2                [in1] >b
 -output                [out] >c
 -common_region_only    [true] >
 -add_not_overlay       [true] >
```

```
 -make_edge             [false] >
 -shift_ofin2           [0 0 0] >
 -turnangle             [0.000000] >
 -turnpoint_global      [-1 -1] >
```

join_uncompress

This operation joins two segmented volumes directly while reading and uncompressing from the disk, thus only needs the memory of the output volume once. It may be useful if `concat` does not work because of too small virtual memory. The compressed files are read from disk and the uncompressing is done directly into the correct memory location in the output. The volumes may be overlapping, but then the respective part of file2 is overwritten (including zeros) onto file1.
NOTE: `Join_uncompress` only works for compressed files on disk and only works for volumes with the same dimensions in x and y direction !

```
ipl> /join
 -file1     [file1] >rat_seg_part1.aim
 -file2     [file2] >rat_seg_part2.aim
 -output    [out] >
 -shift_ofin2_z         [0] >
```

bounding_box_cut

This operation determines the smallest box around the object of non-zero voxels. An additional boundary can be chosen with *border*. *z_only* controls if only the bounding box in z direction is determined and cut out with a given *border*.

```
ipl> /bounding_box_cut
 -input1    [in]  >
 -output    [out] >
 -z_only    [false] >
 -border    [0 0 0] >
```

flip_aim

Flips an object to a side, i.e. x y and z direction are switched according to *new_xydir* (NB: it is not necessarily a proper rotation of the object in 3D space !). The new x direction is the first letter given in *new_xydir*, the new y direction is the second letter, the new z direction is the remaining direction.

```
ipl> flip_aim
 -input     [in] >a
 -output    [out] >f
 -new_xydir [yz] >xz
```

offset_add

Changes the offset around the object. The offset is the same for the beginning and the end for one direction (voxels run from +off.x to dim.x-off.x), but may

be different for the different directions x, y and z. The offset is then disregarded in subsequent image processing or statistical examinations.

```
ipl> offset_add
 -input                 [in] >a
 -add_offset            [0 0 0] >0 0 4
```

offset_set

Changes the offset around the object, independent of what it was before.

```
ipl> offset_set
 -input                 [in] >b
 -new_offset            [0 0 0] >2 2 1
```

clear_offset

Sets all voxels in the (optionally new) offset to zero.

```
ipl> offset_set
 -input                 [in] >b
 -new_offset            [-1 -1 -1] >2 2 1
```

fill_offset_mirror

Fills the offset of an object with the mirrored values just to the inside of the offset. May be used after operations that 'steal' from the image, e.g. gauss-filter. Be aware that voxel information is just duplicated and that values calculated afterwards can be biased.

```
ipl> fill_offset_mirror
 -input                 [in] > in
```

convert_to_type

This converts objects from one input type to another, e.g. a 'char' image to a 'short' image. The output type is given by *out_type*.

```
ipl> convert_to_type
 -input     [in] >
 -output    [out] >
 -out_type  [short] >
```

xray

Produces a virtual (linear) xray image of the object along the z-axis (flip the object with `'flip_aim'` first for other desired orientations). The output is a 'char' image. The voxel values (for every x and y) are added along the z-axis from *startslice* for *number_of_slices*, and the sum is either normalized with the largest occuring voxelsum (*fixed_norm_char* false) or to a fixed norm for binary input volumes (*fixed_norm_char* true).

```
ipl> xray
 -input     [in] >
 -output    [out] >
 -startslice            [0] >
 -number_of_slices      [16] >
 -fixed_norm_char       [true] >
```

msq_from_aim

Write an object in memory to disk in the .MSQ format, e.g. to view it with the 3D-Display program.

```
ipl> msq
 -aim_name  [out] >
 -msq_filename          [default_file_name] >
```

from_aim_to_isq

Write an object in memory to disk in the .ISQ format, e.g. to produce a GOBJ with the Evaluation program again. The object will be at its original (global) position.

```
ipl> from_aim_to_isq
 -aim_name  [out] >
 -isq_filename          [default_file_name] >
```

For an extensive discussion of the obtained structural indices see Appendix (Explanation of Structural Indices, p.100).

Use this program to recreate a 3D-result-sheet including the 3D-image and the histomorphometry-values calculated by IPL.