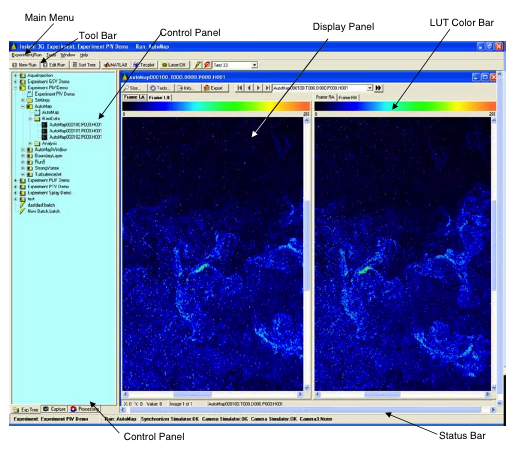
Design principles of the new 3D-PTV software

1. Open source, Python/C + GUI (wx or Qt or Tk)
2. Multi-platform
3. Used for the three-dimensional particle tracking velocimetry (3D-PTV) measurements
4. Software analyzes images of two kinds: calibration images, measurement images (sometimes measurement images serve as calibration images)
5. Images are obtained by N digital cameras (typically N = 1 – 4, we need to leave options for more than 4 cameras)
6. User should have instant feedback on the process and change the parameters and the setup in order to optimize the measurement output
7. We shall leave an option to add the acquisition part, but only as a part of the future development, not as a part of the present project
8. Additional tools should be able to plug-in, e.g. Matlab or Mathematica or Fortran. The additional modules will work through the pre-defined interface.

GUI

We will follow the design of Insight 3G from TSI Inc. Roughly the software should look like:



1. Software deals with “Experiments”. Each experiment is the folder in which we save:
   1. Calibration images and input files (e.g. position of dots that are shown in calibration images)
   2. Calibration data (result of calibration process) - the main thing that should pass to the following processes.
   3. Images
   4. Experimental settings (distances, refraction index, camera type, number of cameras, image size, etc.)
   5. Processing parameters (image processing parameters, such as intensity thresholds, size thresholds, sequence numbers, tracking parameters, e.g. type of the tracking algorithm, etc.)
   6. Some log-file that will store all the important information about the way data processed and which parameters are used
   7. It’s desired to keep the present structure to ensure “back-compatibility”, i.e. at least the same names of the folders:

/cal /img /parameters

and same names of the files: /cam1. Xxxx /cam1.ori, etc.

Software should give us an option to:

1. Open “New Experiment” – automatically creates the folders and necessary files, allowing user to fill it in manually, like: Choose images, choose files, edit parameters
2. “Edit old experiment” – uploading to memory all the necessary files from the experimental directory.
3. Open “New run” , e.g. adding new images under the same “Experiment” or making new calibration for the same measurement images
4. Edit “Old run” – again, changing something in the Experiment, e.g. adding new calibration (only /cal1 /cal2 folders will appear, the rest will remain the same)

Window menu should provide us an option to see our images in the following modes:

1. Cascade
2. Horizontal tile
3. Vertical tile
4. See All or Arrange Icons (see all 4 images in the square format)

There are three major panels:

1. Acquisition (future, optional)
2. Processing (incl. calibration, image processing, stereo-matching, tracking)
3. Data visualization and post-processing (visualize trajectories, rotate, measure simple things, e.g. velocity statistics, acceleration statistics, changing the interpolation polynom, etc.)

**Getting started**

1. Create New Experiment -> user chooses the folder and the name of experiment, and a short “readme” note
2. Create New Run -> user chooses the name of the new run -> folder structure is created automatically: /cal /img /parameters + short readme note + Tree of Experiments is updated.
3. User can later: rename, delete, copy/paste – affecting immediately the Tree and the folder structure. Not all is necessary now, but some sort of reading the folder structure or keeping some file with the structure is necessary, e.g. for the case user moves the folder to some unknown location - it should disappear from the Tree.
4. Under the Experiment – Run tree branch should appear the structure: /img /cal … and image names, so the user can simply double-click one of the images and observe it in the window. This should be carefully considered as the number of images is sometimes extremely high (100,000 images)
5. Setting general parameters:
   1. Number of images
   2. Image size
   3. Size and material of the experiment: thickness of water layer, air layer, glass layer, their index of refraction values. At present software deals with 3 layers at most. We shall leave an option to add more layers in the future.

**Calibration** -> should work in a separate window or panel

User needs GUI and algorithms to do the following:

1. load calibration images
2. basic image processing: remove background, high-pass filter, blur, threshold
3. click on pixels and get their information: pixel location in x,y and intensity (in order to process wisely, e.g. creating threshold) -> processed images can be saved in some /temp folder or in /cal folder under temprorary names
4. choose the name of the calibration target file (x,y,z locations of the known points on the calibration body)
5. insert a table of values, e.g. control points
6. select control points by clicking and leaving a sign (e.g. cross), removing and shifting these control points ( see Matlab registration of images demo for the reference)
7. Insert a table of major camera position parameters: x,y,z, three Euler angles, back focal distance and the vector of position of the glass interface) -> files like cam1.ori are created. If the files exist, the table is already populated by values, user should be able to modify them in the table, automatically saving to the file. The files, cam1.ori, cam2.ori etc. will be the major output and will serve the rest of the processing. All the 2D <-> 3D transformations are using these files. These files are crucial and should be “backed up” every time the user changes it. There should be some way to Undo changes and Revert to original.
8. Button “project target file” that takes the x,y,z locations of all the calibration dots (given by some text file, see 4 above) and projects them on the images, using the ORI files.
9. Sliders for every single value in the major parameters (see 7) allowing for dynamic change of the projection (in 8) on the image. E.g. user slides the value of ‘z’ or ‘focal distance’ and observes the visual change of the position of the projected points. So called manual adjustment. Of course, user can go back to 7 and open the table and type the number (which is already updated by the slider) and get an updated projection.
10. Sort target points button -> some automatic way of finding approximate location of target points based on the four control points that were manually clicked
11. Orientation using manual adjustment or Orientation using sort grid option -> check one of the options or choose from the list of options.
12. Button “Optimize camera parameters” or “Calibrate”. Solves the equations of transformation to get the positon, angles of the cameras, user should have an option to constrain some of the parameters (see our present Calibration parameters) by checking on/off the parameters to optimize.
13. Additional buttons that should open additional panels and allow to perform absolutely different type of calibration:
    1. Dynamic calibration using measurement images
    2. Dynamic calibration using some structured images that should appear in a separate folder /db
    3. Optional, future, calibration algorithm that our users should be able to plug-in here. Inputs to their calibration algorithms are free, output is always ORI files. User should be basically be able to type some commands or provide a function or something like this to create ORI files. Not yet clear why the GUI is necessary here, maybe just let user to create ORI files manually and add them to the folder in the right place? Maybe here we shall add few options of “Translate the orientation from pin-hole model to our ORI files” or “Translate Zhou model into our ORI files” and provide some example functions that do this.

More on 13a:

User should be able to choose the images to be processed and few more parameters. Then the button “Calibrate” should open another panel of “Processing” and user should choose all the rest of the parameters for the processing (see below) and use another algorithm that creates ORI files. See “shaking” option in our Calibration software.

13b is similar but user should be able to choose other type of images (some specific kind of images in /db folder) and set up their parameters -> See dumbbell calibration in our software

1. Should be some kind of Test calibration button -> uses the calibration images and runs them through Processing algorithm (identify points, locate them, project them) and compares the result (x,y,z of the identified points) to the given text file of the known points (see 4 above) – shows in some graphical way the differences between those. Today we show it by red lines that connect the identified point with the ‘theoretical’ position (again, through forward/backward projections from 2D to 3D and back). This gives an immediate feedback to the user about the good of fit for the calibration. Some r.m.s. numbers will be also helpful (we need to add some Output console at the bottom of the main window).

**Processing**

Four major stages of processing:

1. Image processing – pre-processing of images before starting identification, etc. now has only high-pass filter and threshold. We shall enable user to add more sophisticated image processing routines through some known interface. E.g. user can simply copy/paste images into another folder /img -> /img1 by writing a Matlab function that takes all images in /img, makes this and that and saves the result in /img1 -> huge overhead of reading/writing files. If possible, we should be able to process the image on-flight and not even save the result, just run through the function and continue with the processed image. Images should be saved for the later use, we do not re-use it many times after that, but at least for the identification routines.
2. Identification of particles: several algorithms exist and sometimes we need a sophisticated ones, so it should be kind of “choice of the function” and option to “create custom function” -> opens editor with the given template in which the user can only put lines in between a header and footer which are known. Like:

def myidentification(image):

from scipy.misc.ndimage import gaussian

out = Gaussian(image\*\*2) # 🡨- change here

return out

The output of the identification of particles are files (one file per image, per camera or some database, but careful one, that doesn’t collapse the computer or destroys itself after 100,000 x 4 data units) that have only a known type of name, e.g. cam1.tif\_targets (as it is today) in which we have N as a first line and then N lines of the particle information: x,y, in pixels, n pixels in x, n pixels in y directions (kind of shape of the particle), n pixels in total (area) and sum of intensity values. Of course somewhat more sophisticated type of information is possible, but we should keep the simple things first. These files are another major output and should be kept for the rest of the processing – many algorithms use it later.

1. Stereo-matching and 3D position reconstruction – using particles identified in each one of the 4 images, for example, the software find the same particles in 4 images and find their 3D position. Here, few parameters are necessary as a table and one major development is to add the option for the user to select ROI in one image and get the particles which only fit into the ROI. It’s relatively simple task but requires some interaction between the mouse drawing, image information (pixels along the drawn lines) and 2D -> 3D transformation to know the limits of the particles that fit into the ROI. The rest is automatically performed for the selected images – user should be able to select which time interval s/he wants to analyze. Output of this step is rs\_is.xxx files that store particles matched in different images. There are particles that are identified in 4 images, called quadruplets, particles identified in 3 images: triplets, 2 images -> pairs and so on. Some smart database could save a lot of text writing/reading, but meanwhile it’s not our problem. We could live with the ASCII files meanwhile, as long as we keep the things clear and open. E.g. rewriting only one function like readPartilcesFile from using ASCII to use HDF5 or PyTables could be very useful.
2. Tracking -> also automatic process, with few parameters defined as table, no interaction, just option to add new algorithm and use same interface, e.g.:

def mytracking(tstart,tend,max\_displacement=None):

for t in range(tstart,tend):

particlest = readTargetFile(t)

particlestplus1 = readTargetFile(t+1)

traj = nearestNeighbour(particlest,particlestplus1)

return traj

Output of this step is xuap.xxx files – one file per time step in which we store the particles in 3D that are linked to each other. The link is by the number of the particle in the previous frame and the next frame in the given file.

**Post-processing – separate panel**

1. Trajectories are smoothed using some spline or polynomial or radial basis function – we need to let user to choose one of the known algorithms, this could be found in post\_process folders under SVN. We can simply use some very simple files in Python, for example as a demo, later filling in smarter functions. The input is xuap.xxx files that user can select as All or some and process it to get new set of files (it’s impossible to add new lines to ASCII file which is already too much) with smooth linked trajectories
2. Quantitative information like velocity, acceleration, etc. is estimated using the smoothed trajectories
3. Trajectories can be re-linked together (some are broken for no obvious reason)
4. Trajectories can be removed if fail on some tests
5. We need to plot histogram of velocity, of acceleration, x,y,z, of some selected trajectories, 3D trajectories, maybe some simple animations, scatter plots of distribution of identified particles and linked trajectories in 2D and 3D, etc.
6. Main thing here is flexibility to add new functions and get them plotted in the post-processing window or data could be shown as table and analyzed by copy/paste and plot.

**Image Windows**

1. zoom in/out, pan, fit-in-window (automatic zoom to the size of the window). Separate zoom/pan and synchronized zoom (e.g. zooming in one gives zoom in all, around the same point – requires 2D<->3D transform)
2. clicking -> picking pixels and observing pixel values for the image processing, etc.
3. identified particles get crosses at their identified positions in pixels, (color of the cross defines if it’s pair, triplet or quadruplet).
4. If calibration is available and 2D<-> 3D transform is known, at some stages, e.g. interactive stereo-matching, user can click on the identified particle and get the epipolar lines (projection of the imaging axis) on other images. – provides feedback on the calibration and other properties.
5. In tracking mode we shall be able to see in different colors:

* Identified particles,
* start of the trajectory
* end of the trajectory
* particles that belong to the trajectory (i.e. linked)**Main C functions and procedures from the /src\_c folder**

from “documentation.doc” file:

**Documentation of 3D Particle Tracking Velocimetry**

Technical aspects, Installation, Handling, Software Architecture

by Jochen Willneff, October 2003

**1. Installation and Handling of the PTV-Software**

The ptvmanual.pdf contains information to use the software and gives a description of the input data file which have to be provided.

The test data set contains the following:

Cam1 Cam3.addpar man\_ori.dat

Cam1.addpar Cam3.ori parameters/

Cam1.ori Cam4 ptvmanual.pdf

Cam2 Cam4.addpar res/

Cam2.addpar Cam4.ori start.bat

Cam2.ori calFieldApril.txt

Cam3 img/

Images for calibration, camera orientation data, files for additional parameters as well as the coordinate file of the points on the reference. In man\_ori.dat the manually measured image coordinates for the pre-orientation (for calibration purpose) are stored.

- subdirectory /img contains the image sequences

- subdirectory /res for storage of results

- subdirectory /parameters contains the parameter files

The data mentioned above are the data for the experiment itself. To avoid confusion this data should be kept separated to the software data!

The software for PTV is stored under /tk84ptv. To start the software >> click double on start.bat, which establishes the link to the software. Project and software data should not be confused. To start the software it is sufficient that a start.bat file, thus the software (and code) can be stored independently.

**2. Tcl/Tk-Installation**

The install executable for Tcl/Tk is ActiceTcl8.4.2-win32-ix86.exe in this directory. Or can be downloaded from a webpage. Download under:

http://downloads.activestate.com/ActiveTcl/Windows/8.4.2/

After installing Tcl/Tk 8.4.2 all files with extension #.tcl should appear with the Tcl/Tk-symbol (feather). Otherwise repeat installation. Make sure that the flag for the extensions (\*.tcl) is included in the path. Otherwise not all needed dll-files can be found from arbitrary directories on the PC.

**3. Compilation of TIFF library**

The handling of TIFF images requires a according library libtiff.lib and the files tiff.h, tiffio.h and tiffvers.h. If these files are not available, they can be generated as described in the following. Download file tiff-v3.5.7.zip from the webpage www.libtiff.org. It’s also possible to use the newer version tiff-v3.6.0.zip. Both versions were tested for Windows2000 and WindowsXP. After unzip change to /libtiff, where you will find the file named makefile.vc.

In the header of this file the command (nmake /f makefile.vc all) to compile the library with nmake is given. It is also possible compile with Microsoft Visual C++. To open the makefile in the right way, change the filename from makefile.vc to makefile.mak. Only the files libtiff.lib, tiff.h, tiffio.h and tiffvers.h are necessary for PTV (and may be copied in the directory /tk84ptv/src\_c, if done so no paths have to be adjusted for the compilation of PTV).

**4. Compilation of the source code of PTV**

In the /tk84ptv directory You will find the following data:

index script to generate tclIndex (/ might be missing in the generated tclIndex!)

ptv.tcl main script to start graphical user interface (Windows)

ptvunix.tcl dito for Unix

start start file for Unix

tclIndex Index with relative paths to Tcl functions

/src\_c source code directory

/src\_tcl tcl script directory

The contents of the /src\_c:

change\_parameter.c jw\_main.c segmentation.c

checkpoints.c jw\_ptv.c sortgrid.c

correspondences.c libtiff.lib unixmakefile

demo.c lsqadj.c tiff.h

draw.c mousefunction.c tiffio.h

epi.c multimed.c tiffvers.h

globals.h orientation.c tools.c

homemakefile peakfitting.c track.c

homemakefile.mak pointpos.c trafo.c

image\_processing.c ptv.c ttools.c

imgcoord.c ptv.h typedefs.h

intersect.c ray\_tracing.c vrml.c

jw\_ImgFmtTIF.c rotation.c

The contents of the /src\_tcl:

button.tcl display.tcl mainpar.tcl

calpar.tcl draw.tcl trackpar.tcl

The source code is written in C in combination with Tcl/Tk. The directory /src\_c contains a makefile (homemakefile.mak) which can be open with Microsoft Visual C++. During opening this file, a new workspace will be generated.

**NOTICE!** The following paths to the libs in the makefile have to be adjusted:

INC\_DIR1 = C:\Tcl\include\

TCL\_LIB = C:\Tcl\lib\tcl84.lib

TK\_LIB = C:\Tcl\lib\tk84.lib

TIFF\_LIB = H:\tk84ptv\src\_c\libtiff.lib

An alternative is the compilation with nmake in the DOS prompt. For compilation in the DOS prompt first perform vcvars32.bat for initalization, after that:

nmake -f homemakefile.mak

**IMPORTANT!** Before running the software some paths have to be set. start.bat may contain:

G:/tk84ptv/src\_c/jw\_prog G:/tk84ptv/ptv.tcl

This has to be modified to the actual position on the PC. First the path to the jw\_prog.exe followed by the path to the ptv.tcl script file.

In addition, change path in first line of ptv.tcl (use # for comments):

set auto\_path "G:/tk84ptv . $auto\_path"

Change to according path on PC! The start.bat should be copied to the project data directory. Start with double click!

**5. 3D PTV source code**

Authors: Hans-Gerd Maas / Jochen Willneff

Address: Institute of Geodesy and Photogrammetry

ETH – Hoenggerberg

CH - 8093 Zurich

Functions in ***jw\_ptv.c***:

***init\_proc\_c***

initialization, allocation of memory for image data, reading of

parameter files

- parameters/ptv.par

- parameters/criteria.par

- parameters/sequence.par

call of ***parameter\_panel\_init***, to fill Tcl/Tk sheets

***start\_proc\_c***

reading of parameters/ptv.par

create file names for low and high pass image and ori/addpar data

Reading orientation data from file, reading image

Allocation of tracking structure

***pre\_processing\_c***

reading of parameters/unsharp\_mask.par, default value 12, if not existing, call of ***highpass***, with optional display

***detection\_proc\_c***

reading of parameters/pft\_version, switch pft for peak fitting

three cases:

- 3, call of ***peak\_fit\_new***

- 0, call of ***simple\_connectivity***

- 1, call of ***targ\_rec***

mostly ***peak\_fit\_new*** was used recently

call of ***quicksort\_target\_y***, to sort detected particles

***correspondences\_proc\_c***

Transformation from pixel to metric coordinates with call of

- ***pixel\_to\_metric***

- ***correct\_brown\_affin***

sort coordinates for binary search with ***quicksort\_coord2d\_x***

reading of parameters/criteria.par

calculation of look up table for multimedia radial displacement

- ***init\_mmLUT***, performed only once!

search for correspondences with most important function!

- ***correspondences\_4***

create #\_targets data for each image, writing data to \*\_targets files

***determination\_proc\_c***

create res/dt\_lsq file, call of function for 3D coordinate determination

- ***det\_lsq***, writing data to res/dt\_lsq file,

sort coordinates for binary search in epi line segment drawing

- ***quicksort\_coord2d\_x***

***sequence\_proc\_c***

reading of parameter file parameters/sequence.par, first and last

time step of sequence

create file names, res/rt\_is.#

processing of each time step with

- ***read\_image***

- ***pre\_processing\_c***

- ***detection\_proc\_c***

- ***correspondences\_proc\_c***

- ***determination\_proc\_c***

***calibration\_proc\_c***

reading of parameters/unsharp\_mask.par, default value 12, if not

existing

8 cases, switch set by sel:

- 1, read calibration parameter file parameters/cal\_ori.par

create file names, call of ***read\_image***

- 2, detection procedure, call of ***pre\_processing\_c***, target

recognition by call of ***targ\_rec***, reading

parameters/detect\_plate.par, writing #pix files for each image

- 3, manual orientation, read parameters/man\_ori.par, interactive

measurement of four reference points in each image, writing

pixel coordinates to file man\_ori.dat, can be used for further

orientation calculation, see case 4

- 4, read points numbers from parameters/man\_ori.par, read pixel

coordinates of older pre-orientation from man\_ori.dat, display

- 5, sort grid, read coordinate from reference body,

read orientation and addpar files, calculation of raw orientation

with 4 points by call of ***raw\_orient***, write orientation data,

call of ***write\_ori***, sorting of detected points by

back-projection by call of ***sortgrid\_man***, with display,

if examine is set to 4, create files for dump dataset,

allowing layerwise calibration

- 6, orientation, if examine set to 4, reading files for layerwise calibration, else calculation of resection by call of ***orient***, if examine set to 4, read dumped data sets first, then call orient, write results in ori and addpar files

- 7, ***checkpoint\_proc***, to display residuals of checkpoints

- 8, draw additional parameter figures, display regular grid and

(exaggerate) distorted grid

***quit\_proc\_c***

free memory, delete unneeded file and quit

**6. PTV argument examine**

The start.bat file without using the examine option is similar to:

H:/prog/tk84ptv/src\_c/jw\_prog H:/prog/tk84ptv/ptv.tcl

Different examine options can be set by the start of

PTV (e.g. .../jw\_prog .../ptv.tcl 4)

1 (or any number) more details on output during orientation

(calibration), double zoomfactor, creates #\_pix files

3 to save low pass image,

doesn't work on Windows system, sorry!

4 creates ASCII output with 3D object point list and

referring image coordinates for calibration with dumped

data sets, detailed descriptions see below.

5 more detailed output for statistical examinations

of Qvv, Qxx at determination of particle positions

and during orientation (calibration)

10 gives information about average and expected number of

touch events and the average number of pixel per target,

at detection of particles

Option 4 is used for the calibration with different z-positions of the reference body. The examine option appears in the following source code files:

draw.c: if ( examine && zoom\_f[nr] > 2 )

jw\_ptv.c: int examine = 0; /\* for more detailed output \*/

jw\_ptv.c: valp = Tcl\_GetVar(interp, "examine", TCL\_GLOBAL\_ONLY);

jw\_ptv.c: examine = atoi (valp);

jw\_ptv.c: if (examine == 4)

jw\_ptv.c: if (examine == 4)

jw\_ptv.c: if (examine) for (i=0; i<n\_img; i++)

jw\_ptv.c: if (examine == 4)

jw\_ptv.c: if (examine == 4)

jw\_ptv.c: if (examine != 4)

jw\_ptv.c: if (examine == 4)

mousefunction.c: if (examine) zf \*= 4;

orientation.c: if (examine) for (i=0; i<16; i++)

peakfitting.c: if (examine==10)

pointpos.c: if (examine == 5)

segmentation.c: if (examine == 3)

segmentation.c: if (examine == 3)

**How to calibrate with different z-positions of the reference body?**

Set examine = 4!

After the **Detection** (case 2, under calibration) #\_pix files are generated, which lists the detections of each view (no correspondences, only image coordinates), was used for template matching outside of PTV, not used for further processing steps (could be removed or commented in the code).

--->> continue as usual!!!

During **Sortgrid** (case 5, under calibration) a file dump\_for\_rdb is created. This is a list of the 3D points with the according 2D image coordinates of each view. Is not used for further steps.

--->> continue as usual!!!

**Important!!!** In **Orientation** (case 6, under calibration). For each z-position an according 3D coordinate file has to be provided (File of Coordinates on Plate, at **Calibration Parameters** has to be adjusted for that!!!)

During **Orientation** with examine set to 4, please consider the DOS prompt. You will be asked:

Resection with dumped datasets? (y/n)

Answer with n in the DOS prompt to write data to disk, which will later be used for the common calibration. For each camera you have to answer individually (up to four times depending on the number of cameras)!

The files, which are created have the following name structure:

resect\_#.crd, containing the corrected metric image coordinates (e.g. resect\_Cam1.crd).

resect\_#.fix, containing the 3D coordinates of the active reference body file (e.g. resect\_Cam1.fix).

After the generation of these files, the different file names have to be adjusted to a sequential structure:

resect.fix0 3D reference body file of z-position 0

resect.fix1 3D reference body file of z-position 1

resect.fix2 3D reference body file of z-position 2

... and so on.

For each camera and each z-position a set of metric image coordinates

has to be provided with the following names:

resect\_0.crd0 metric image coordinates of camera 0 at z-position 0

resect\_0.crd1 metric image coordinates of camera 0 at z-position 1

resect\_0.crd2 metric image coordinates of camera 0 at z-position 2

resect\_1.crd0 metric image coordinates of camera 1 at z-position 0

resect\_1.crd1 metric image coordinates of camera 1 at z-position 1

resect\_1.crd2 metric image coordinates of camera 1 at z-position 2

resect\_2.crd0 metric image coordinates of camera 2 at z-position 0

resect\_2.crd1 metric image coordinates of camera 2 at z-position 1

resect\_2.crd2 metric image coordinates of camera 2 at z-position 2

resect\_3.crd0 metric image coordinates of camera 3 at z-position 0

resect\_3.crd1 metric image coordinates of camera 3 at z-position 1

resect\_3.crd2 metric image coordinates of camera 3 at z-position 2

... and so on.

After the generation of this file structure, it's recommended to save these files first before continuing with the calibration. Restart PTV, press **Show Calib. Image** and continue directly with **Orientation**, consider the DOS prompt. Now answer to

Resection with dumped datasets? (y/n)

with y for each individual camera (again up to four times depending on the number of cameras), #.ori and #.addpar file are generated according to the given cameraname (e.g. Cam1.ori, Cam1.addpar). The orientation and additional parameters for each individual cameras are calculated in a common adjustment of all z-positions.

**7. Using C code in connection with Tcl/Tk**

The source code of PTV, which is written in C is connected to a graphical user interface realized in Tcl/Tk. This requires the declaration and initilization of commands.

In jw\_main.c the C function ***main*** is called. In this function only the initialization for the Tcl/Tk application is performed. The ***main*** just calls the function ***Tk\_Main(argc, argv, Tcl\_AppInit)***, also included in jw\_main.c, which starts the function ***Tcl\_AppInit(interp)***, (in jw\_main.c). ***Tcl\_AppInit(interp)*** calls the function ***jw\_Init*** (again in jw\_main.c).

With the call of ***jw\_Init*** the additional Tcl/Tk commands are defined. For example:

Tcl\_CreateCommand(interp, "start\_proc\_cmd", start\_proc\_c,

(ClientData)NULL, (Tcl\_CmdDeleteProc \*)NULL);

The name of the defined Tcl/Tk command is ***start\_proc\_cmd***. This command can be used in the Tcl/Tk script. The related C function ***start\_proc\_c*** is evaluated, when ***start\_proc\_cmd*** is called.

The C function is defined as Tcl/Tk command in ptv.h like this:

extern Tcl\_CmdProc start\_proc\_c;

In addition the function itself has to be defined in global.h:

int start\_proc\_c();

For the actual implementation the function needs some Tcl/Tk related arguments:

int start\_proc\_c(ClientData clientData, Tcl\_Interp\* interp, int argc, const char\*\* argv)

To call the function from the Tcl/Tk script (in this case by pressing the button "**Start**", see in ptv.tcl):

button .mbar.start -text "Start" –command "start\_proc\_cmd; bindingsstart "

The option -command also allows the combination with other commands or function calls. In this example with **bindingsstart**, which defines the current mouse functions.

If these C functions are called from some other C function and not from the

Tcl/Tk script do like this:

pre\_processing\_c (clientData, interp, argc, argv);

**Exchange of variables between C and Tcl/Tk**

In the C source code and the Tcl/Tk script variables have to be exchanged. This is the case for reading and writing the input data from the input files and the Tcl/Tk sheets.

The according variable is defined in the C code as well as in Tcl/Tk (e.g. n\_img in C refers to mp(ncam) in Tcl/Tk). The reading of the input files is realized in C. The exchange to Tcl/Tk is done by the commands:

Tcl\_SetVar2 and Tcl\_GetVar2

Remarks:

- Tcl/Tk treats the variables as string (therefore char type in C)

- Tcl\_SetVar2, Tcl\_GetVar2 (not Tcl\_SetVar, Tcl\_GetVar) are used

to be able to define variables in vector structure

Example in src\_tcl/mainpar.tcl:

global mp

mp is the structure for all main parameters. mp(ncam) represents a specific parameter

The exchange works the following way, from C to Tcl/Tk (see in function ***parameter\_panel\_init*** in src\_c/change\_parameter.c):

int parameter\_panel\_init(Tcl\_Interp\* interp)

{

char val[256];

/\* read 20 parameters from ptv.par \*/

fp1 = fopen\_r ("parameters/ptv.par");

fscanf (fp1, "%s", val);

Tcl\_SetVar2(interp, "mp", "ncam", val, TCL\_GLOBAL\_ONLY);

fclose (fp1);

return TCL\_OK;

}

The parameter is read from the input file to C variable val, its value is transferred (Tcl\_SetVar2) to Tcl/Tk variable mp(ncam) and is available in the Tcl/Tk scripts. From Tcl/Tk to C (see in function ***done\_proc\_c*** in src\_c/change\_parameter.c):

int done\_proc\_c(ClientData clientData, Tcl\_Interp\* interp, int argc, const char\*\* argv)

{

const char \*valp;

/\* rewrite all parameter files \*/

fp1 = fopen ("parameters/ptv.par", "w");

valp=Tcl\_GetVar2(interp, "mp", "ncam", TCL\_GLOBAL\_ONLY);

fprintf (fp1, "%s\n", valp);

fclose (fp1);

return TCL\_OK;

}

Parameter mp(ncam) from Tcl/Tk is transferred (Tcl\_GetVar2) to C variable valp and written to file.

**ATTENTION!!!** valp is a char type. For later use in C (for calculations) this variable may has to be converted, for e.g. to integer:

valp = Tcl\_GetVar(interp, "examine", TCL\_GLOBAL\_ONLY);

examine = atoi (valp);

**8. 3D PTV data acquisition and setup parameters**

The processing of the image sequences from PTV experiments requires some information about the hardware configuration. From the used cameras at least a rough estimation of the focal length, size of the pixel in x- and y-direction and the image size (in pixel) itself has to be specified.

To be able to model the multimedia geometry the refractive index of the fluid and glass plate as well as the glass plate thickness has to be known.

For calibration purposes the coordinates of the used reference body have to be specified. The boundary of the glass plate and fluid defines the x-y-plane (z = 0). The coordinate file of the reference body has to be edited according to its actual position, which requires the knowledge of the distance of the reference body to the inner glass plate side.

The image sequences have to be provided according to the following convention. For the sequence of each camera the filename is a combination of a base name and a number of the current time step. No extensions after the current number of the time steps are possible.

**Example**, basename: cam1., time steps from 1 to 1001, the files have to be provided with the following names:

cam1.001 … cam1.009, cam1.010 … cam1.099, cam1.100 … cam1.999, cam1.1000, cam1.1001